Projet MVSI

Groupe 3 : Sarah Combe Deschaumes, Mathieu Dreyer, Antoine Jacque

Mise en place du projet

Le but de ce projet était de transformer trois algorithmes dans trois langages : PlusCal, Rodin et FramaC afin d'en effectuer la preuve partielle.

Nous avons donc décidé de nous charger chacun d'un algorithme et de le retranscrire dans les trois langages demandés.

Cependant, nous avons tous retranscrit notre algorithme dans le même sens, c'est-à-dire PlusCal, Rodin puis FramaC. Ainsi nous avons pu nous aider et nous appuyer les uns les autres.

Comme nous avions peu de connaissances sur FramaC, ceux-ci sont expérimentaux et peuvent être totalement faux. De plus comme il s'agit d'un domaine relativement fermé nous n'avons pas trouvé la documentation adaptée à nos besoin (ou du moins nous n'avons pas su où chercher).

Problèmes rencontrés

Le principal problème que nous avons rencontré est l'installation des différents logiciels. En effet, nous avons beau avoir essayé d'installer les logiciels sur nos machines, l'installation n'était jamais tout à fait correcte et tout ne fonctionnait pas. Nous avons eu beaucoup de problème avec FramaC, suite à la fermeture de l'école nous n'avons pas réussi à installer cet outil sur nos machines, nous avons donc dû nous débrouiller et écrire les programmes à l'aveugle sans pouvoir les tester.

Un autre problème a été la prise en main de ces outils. En effet, nous avons dû utiliser des structures que nous n'avions jamais utilisé, notamment les tableaux et cela s'est avéré plus compliqué que nous le pensions.

Palindrome

Objectifs du programme

Ce programme doit, à partir d'un nombre entré, trouver si ce nombre est un palindrome. Un palindrome étant un mot qui peut se lire indifféremment de gauche à droite ou de droite à gauche.

Programme annoté

```
#include <stdio .h>
int main()
{
	int n, reverse = 0, temp;
	printf("Enter a number to check if it is a palindrome or not\n");
	scanf("%d", &n);
	temp = n;
	# temp = n
	while (temp != 0)
	{
		# temp != 0
		reverse = reverse * 10;
		# reverse = reverse * 10
		reverse = reverse + temp % 10;
		# reverse = reverse * 10 + temp % 10
		temp = temp / 10;
		# temp = temp / 10
	}
	if (n == reverse)
		# n = reverse
		printf("%d is a palindrome number.\n", n);
	else
		# n != reverse
		printf("%d is not a palindrome number.\n", n);
	return 0;
}
```

Les parties en rouge du programme correspondent à des entrées/sorties utilisateur, elles ne sont pas à prouver.

Implémentation TLA+

Pour prouver la correction partielle du programme, nous avons du verifier que à la fin, si n = reverse, nous avons bien un palindrome et sinon que nous n'avons pas un palindrome. Pour se faire nous avons vérifié que quand on arrive dans l'état l6 (qui correspond à 'n est un

palindrome') nous avons bien un palindrome. Nous avons donc mis en place une fonction récursive qui transforme le nombre de départ en tableau et une seconde fonction récursive qui vérifie que le tableau est un palindrome.
TLA nous prouve que le programme est correct et se termine bien pour les valeurs testées.

Conclusion

Ce programme a été prouvé pour les préconditions suivantes : n>0 à l'aide de Rodin principalement.
La post condition est la suivante : si A est un palindrome alors n=reverse sinon n!=reverse.

Nombre premier

Objectifs du programme

Ce programme a pour objectif de retourner la liste des N premiers nombres premiers, N étant le nombre de nombre premier souhaité.

Programme annoté

```
#include<stdio .h>


int main()
{
    int n, i = 3, count , c;
    # i = 3
    printf ("Enter the number of prime numbers required\n" );
    scanf("%d",&n);
    if ( n >= 1 )
    {
        # n >= 1
        printf ("First %d prime numbers are :\n" ,n); printf ("2\n" );
    }
    for ( count = 2 ; count <= n ; )
    {
        # count >= 2 && count <=  n
        for ( c = 2 ; c <= i - 1 ; c++ )
        {
            # c >=2 && c <= i - 1
            if ( i%c == 0 )
                # i%c = 0
                break;
        }
        if ( c == i )
        {
            # c = i
            printf ("%d\n" ,i ); count++;
            # count = count + 1
        }
        i++;
        # i = i + 1
    }
    return 0;
}
```

Les parties en rouge du programme correspondent à des entrées/sorties utilisateur, elles ne sont pas à prouver.

Conclusion

Nous avons prouvé ce programme principalement à l’aide de Rodin.

Tri Sélection

Objectif du programme

Le programme a pour but de trier un tableau d'entiers.
Le programme ne prend pas de paramètres mais demande une entrée utilisateur. Il va tout d'abord demander à l'utilisateur de renseigner le nombre de cases dans le tableau puis demander les valeurs à trier.
Le programme doit ensuite afficher le tableau trié.
Un tableau est trié si et seulement si , tableau[k] =< tableau[k+1] pour tout k appartenant à [0,n-2] (n taille du tableau).

Programme annoté

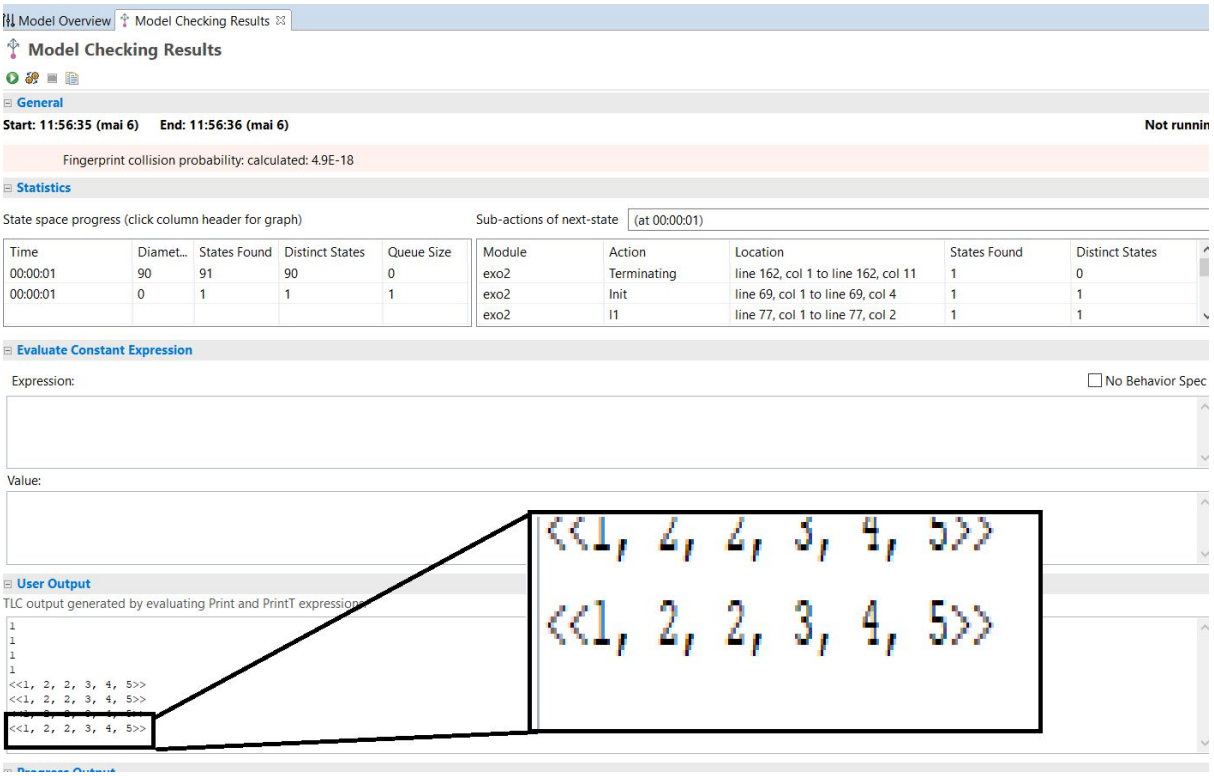
```
#include <stdio .h>

int main()
{
	int array[100], n, c, d, position, swap;
	printf("Enter number of elements\n"); scanf("%d", &n);
	printf("Enter %d integers\n", n);
	for (c = 0; c < n; c++) scanf("%d", &array[c]);
	# n<100 & n>0
	for (c = 0; c < (n - 1); c++) {
		# c>=0 & c<n-1 & n<100
		position = c;
		# c>=0 & c<n-1 & n<100 & position = c
		for (d = c + 1; d < n; d++)
		{
			# c>=0 & c<n-1 & n<100 & position = c & d >= c+1 & d<n
			if (array[position] > array[d]) position = d;
		# c<(n-1) & (array[position] <= array[d] | position = d )
		}
		# d >= n & position >=c
		if (position != c)
		{
			swap = array[c]; array[c] = array[position];
			array[position] = swap;
# array[position] > array[c] && position != c && swap = array[position]
		}
		#  array[position] >= array[c];
	}
	# tableau trié
	printf("Sorted list in ascending order:\n");
```

```
        for (c = 0; c < n; c++) printf("%d\n", array[c]);
        return 0;
}
```

Les parties en rouge du programme correspondent à des entrées/sorties utilisateur, elles ne sont pas à prouver. Cependant nous pouvons formuler des conditions sur les entrées utilisateurs (qui pourront par la suite être considérées comme précondition pour la preuve totale).

Problèmes de modélisation

Pour ce programme, nous n'avons pas réussi à modéliser la postcondition (le tableau est trié) cependant avec TLA il a été possible de vérifier la postcondition.



On remarque que pour l'entrée <<1,2,3,5,4,2>>, La sortie (imprimée à la fin du programme) vaut <<1,2,2,3,4,5>>

Par la suite nous avons mis en place une fonction récursive permettant de vérifier que le tableau est bien trié et nous l'avons utilisé pour faire la correction partielle du programme.

Conclusion

Ce programme a été prouvé pour les préconditions suivantes : n<100 & n>0 à l'aide de Rodin principalement pour la terminaison et TLA pour son entièreté .