UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Verificarea formală în Dafny a unei implementări greedy pentru problema bancnotelor cu 2^5 sistem de bancnote

propusă de

Alexandra Elena Contur

Sesiunea: februarie, 2023

Coordonator științific

Conf. Dr. Ciobâcă Ștefan

UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

Verificarea formală în Dafny a unei implementări greedy pentru problema bancnotelor cu 2^5 sistem de bancnote

Alexandra Elena Contur

Sesiunea: februarie, 2023

Coordonator științific

Conf. Dr. Ciobâcă Ștefan

Avizat,

Îndrumător lucrare de licență,

Conf. Dr. Ciobâcă Ștefan.

Data: ..... Semnătura: .....

Declarație privind originalitatea conținutului lucrării de licență

Subsemnatul **Contur Alexandra Elena** domiciliat în **România, jud. Iași, com. Holboca, str. Dascălilor, nr. 10**, născut la data de **04 martie 2001**, identificat prin CNP **6010304226743**, absolvent al Facultății de informatică, **Facultatea de informatică** specializarea **informatică**, promoția 2022, declar pe propria răspundere cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art. 143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul **Verificarea formală în Dafny a unei implementări greedy pentru problema bancnotelor cu** 2^5 **sistem de bancnote** elaborată sub îndrumarea domnului **Conf. Dr. Ciobâcă Ștefan**, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului ei într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diplomă sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Data: ..... Semnătura: .....

Declarație de consimțământ

Prin prezenta declar că sunt de acord ca lucrarea de licență cu titlul **Verificarea formală în Dafny a unei implementări greedy pentru problema bancnotelor cu 2^5 sistem de bancnote**, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test, etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de informatică.

De asemenea, sunt de acord ca Facultatea de informatică de la Universitatea "Alexandru-Ioan Cuza" din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Absolvent **Alexandra Elena Contur**

Data: ..... Semnătura: .....

Cuprins

Capitolul 1

Context

Dafny este un limbaj de programare și verificare, capabil să verifice corectitudinea funcțională a unui program.

Verificarea este posibilă datorită caracteristicilor specifice limbajului precum precondiții, postcondiții, invariante, ș.a.m.d. De asemenea, verificatorul Dafny are grijă ca adnotările făcute să se îndeplinească, astfel acesta ne scapă de povara de a scrie cod fără erori, în schimbul scrierii de adnotări fără erori.

Problema bancnotelor are ca scop reprezentarea unei sume într-un număr minim posibil de bancnote.

Metoda greedy face alegerea cea mai bună la fiecare pas, construind soluția finală. Verificarea formală in Dafny a problemei bancnotelor demonstrează faptul că soluția construită este optimă pentru orice sumă dată ca input.

Reprezentarea folosită pentru soluție este : $banknote_1 := 1 < banknote_2 < ... < banknote_n$.

Anterior am menționat forma generală a problemei bancnotelor.

Bancnotele posibile în problema bancnotelor cu 2^5 sistem de bancnote sunt: $[1, 2, 4, 8, 16, 32]$

.

Datorită bancnotelor care sunt puteri ale lui 2, dacă avem mai mult de o bancnotă de valoare mai mică decât 32, putem înlocui 2 bancnote de acea valoare cu o bancnotă de valoarea următoare și am obține o soluție cu cost mai mic.

Astfel, am descoperit proprietatea: $forall\ i :: 0 <= i <= 4 ==> s[i] <= 1$

Capitolul 2

Implementarea problemei în Dafny

2.1 Reprezentarea datelor de intrare

În cazul problemei discutate datele de intrare sunt reprezentate de o **sumă** pentru care se verifică dacă se produce soluția finală.

2.2 Reprezentarea datelor de ieșire

Datele de ieșire sunt reprezentate de o secvență de forma :

• soluție = $b_0, b_1, b_2, b_3, b_4, b_5$, unde $\sum_{k=0}^{5} b_k \cdot 2^k = suma$

2.3 Structuri de date folosite

Soluția este reprezentată ca o secvență de numere naturale, fiind chiar secvența descrisă la secțiunea date de ieșire. Fiecare element din soluție reprezintă numărul de apariții ale bancnotei corespunzătoare în soluție.

2.3.1 Condiții ca o soluție finală să fie optimă

• **predicatul isValidSolution**: Pentru a fi soluție optimă trebuie ca secvența să fie o soluție validă (cu 6 elemente), care produce suma corectă și care are costul cel mai mic.

• **predicatul isSolution**: Pentru a fi soluție trebuie ca secvența să fie o soluție validă (cu 6 elemente) și să producă suma corectă.

• **predicatul isValidSolution**: Pentru a fi soluție validă trebuie ca secvența să aibă 6 tipuri de bancnote, cu proprietatea că fiecare bancnotă are un număr nul sau pozitiv de apariții în soluție.

Capitolul 3

Verificarea formală a problemei

3.1 Entry point-ul problemei

Un entry point este locația din cod în care se face transferul controlului, acolo se fac apelurile altor funcții ș.a.m.d. Implementarea algoritmului propriu-zis care rezolvă problema a fost primul și cel mai ușor pas.

Am început prin crearea unei bucle care la fiecare pas alegea bancnota optimă, o adăuga în secvența de bancnote considerată soluție și o scădea din sumă, fiind un algoritm tipic metodei greedy.

```
1    var rest:= sum;
2    solution:= [0, 0, 0, 0, 0, 0];
3    while (0 < rest)
4        decreases rest
5        {
6        index:= maxBanknote(rest);
7        var banknote:= power(2, index);
8        solution:= addValueToIndex(solution,1,index);
9        rest:= rest - banknote;
10   }
```

Acest algoritm era suficient pentru a rezolva problema, dar nu era suficient pentru a demonstra că soluția produsă este optimă.

Considerăm o soluție optimă soluția de cost minim, costul fiind numărul de bancnote.

3.2 Demonstrarea optimalității

3.2.1 Schema verificării

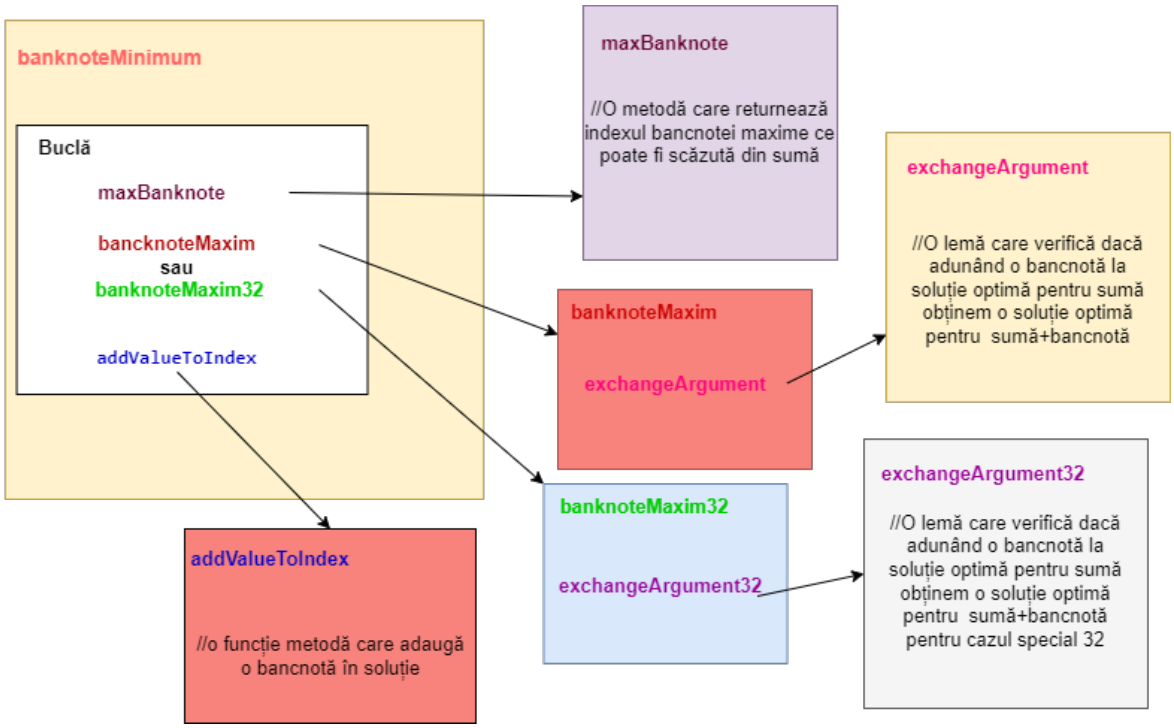

Figura 1: o schemă explicativă a apelurilor de funcții și leme în cadrul verificării

3.3 Etapele verificării

Am implementat algoritmul greedy care rezolva problema bancnotelor cu bancnote puteri ale lui 2, apoi am adăugat treptat condițiile care garantează că soluția găsită este soluție optimă.

3.3.1 Metoda banknoteMinimum

Metoda banknoteMinimum returnează soluția optimă pentru o sumă dată ca input.

Această metodă folosește o buclă, în care, la fiecare iterație, se adaugă o bancnotă în soluție.

În primă fază calculează bancnota cea mai mare ce poate fi dată ca rest, cu ajutorul funcției maxBanknote și funcției power.

Ulterior, folosește lema banknoteMaxim sau banknoteMaxim32, în funcție de

bancnota aleasă, pentru a demonstra că soluția calculată până în prezent, adunată cu bancnota aleasă și cu o soluție optimă pentru $rest - bancnota$ produce o soluție optimă pentru sumă .

Proprietățile necesare unei soluții optime se păstreaza pe parcursul buclei cu ajutorul invariantelor.

```
method banknoteMinimum(sum: int) returns(solution: seq <int> )
    requires sum >= 0
    ensures isValidSolution(solution)
    ensures isSolution(solution, sum)
    ensures isOptimalSolution(solution, sum)
{
    var rest:= sum;
    solution:= [0, 0, 0, 0, 0, 0];
    var index:= 0;
    assert isOptimalSolution(solution, sum - rest);
    while (0 < rest)
        invariant 0 <= rest <= sum
        invariant isValidSolution(solution)
        invariant addOptimRestEqualsOptimSum(rest, sum, solution)
        decreases rest
    {
        index:= maxBanknote(rest);
        var banknote:= power(2, index);
        if (index != 5)
        {
            banknoteMaxim(rest, sum, solution, index);
        }
        else
        {
            banknoteMaxim32(rest, sum, solution);
        }
        solution:= addValueToIndex(solution,1,index);
        rest:= rest - banknote;
    }
}
```

3.3.2 Metoda maxBanknote

Metoda maxBanknote este folosită pentru a returna un index cu proprietatea:

$bancnota_{index} <= rest < bancnota_{index+1}$.

```
method maxBanknote(sum: int) returns(index: int)
  requires sum > 0
  ensures 0 <= index <= 5
  ensures 0 <= power(2, index) <= sum
  ensures(index != 5 && power(2, index + 1) > sum) || index == 5
{
  index:= 5;
  if (power(2, index) > sum)
  {
    assert power(2, index + 1) > sum;
    while (power(2, index) > sum && index > 0)
      invariant power(2, index + 1) > sum
      {
        index:= index - 1;
        assert power(2, index + 1) > sum;
      }
  }
    else
  {
    assert index == 5;
  }
}
```

3.3.3 Lema banknoteMaxim

Lema banknoteMaxim este folosită pentru a demonstra faptul că dacă adăugam o bancnotă în soluția pe care o construim, în continuare suma acestei soluții cu soluția optimă pentru $rest - bancnota$ va produce soluția optimă pentru suma întreagă.

Pentru a demonstra acest lucru avem nevoie să știm că dacă în soluția curentă, optimă pentru $rest - bancnota$ adaugăm bancnota aceasta devine optimă pentru $rest$.

Acest lucru este demonstrat cu ajutorul lemei exchangeArgument.

```
lemma banknoteMaxim(rest: int, sum: int, finalSolution: seq <int> ,
   index: int)
    requires 0 <= index <= 4
    requires power(2, index) <= rest < power(2, index + 1)
    requires isValidSolution(finalSolution)
    requires addOptimRestEqualsOptimSum(rest, sum, finalSolution)
```

```
6         ensures addOptimRestEqualsOptimSum(rest - power(2, index), sum,
              finalSolution[index:= finalSolution[index] + 1])
7     {
8         var banknote:= power(2, index);
9         forall currentSolution | isValidSolution(currentSolution) &&
              isOptimalSolution(currentSolution, rest - banknote)
10            ensures isOptimalSolution(solutionsSum(solutionsSum(
                  currentSolution, finalSolution), [0, 0, 0, 0, 0, 0][index:=
                  1]), sum)
11        {
12            assert isSolution(currentSolution[index:= currentSolution[index
                  ] + 1], rest);
13            exchangeArgument(rest, currentSolution, index);
14        }
15        assert forall currentSolution::isValidSolution(currentSolution) &&
              isOptimalSolution(currentSolution, rest - banknote) ==>
              isOptimalSolution(solutionsSum(solutionsSum(currentSolution,
              finalSolution), [0, 0, 0, 0, 0, 0][index:= 1]), sum);
16    }
```

3.3.4 Lema exchangeArgument

Lema exchangeArgument presupune că nu este optimă soluția dacă adaugăm bancnota aleasă și ajunge la o contradicție. Știm că soluția e optimă pentru $rest - bancnota$, dar nu pentru rest, dacă adaugăm bancnota.

Consideră că există o altă soluție optimă pentru rest.

Verifică în presupusa soluție pentru rest că nu avem bancnota în soluție deja, altfel acea presupusă $solutie - bancnota$ are cost mai mic decât soluția noastră care e optimă pentru $rest - bancnota$, contradicție. Apoi verificăm proprietatea descrisă la observație.

Nu există două bancnote de aceeași valoare în soluție, mai mici de 32.

În acest mod știm că soluția presupusă nu poate fi optimă, deoarece, dacă respectă proprietatea suma produsă de aceasta nu poate fi egală cu rest.

Deoarece :

• $\sum_{k=0}^{index-1} 2^k = 2^{index} - 1$

• $rest >= 2^{index}$

Deci presupusa soluție pentru rest nu poate fi egală decât cu cel mult $rest - 1$, astfel nu poate fi soluție optimă și se ajunge la o contradicție.

```
lemma exchangeArgument(rest: int, currentSolution: seq <int> , index: int)
    requires 0 <= index <= 4
    requires power(2, index) <= rest < power(2, index + 1)
    requires isValidSolution(currentSolution)
    requires isOptimalSolution(currentSolution, rest - power(2, index))
    ensures isOptimalSolution(currentSolution[index:= currentSolution[index
        ] + 1], rest)
{
    var banknote:= power(2, index);
    var solution:= currentSolution[index:= currentSolution[index] + 1];
    assert isValidSolution(solution);
    assert isSolution(solution, rest);
    var i:= index;
    if (!isOptimalSolution(solution, rest))
    {
    var optimalSolution:| isValidSolution(optimalSolution) && isSolution(
        optimalSolution, rest) &&
        isOptimalSolution(optimalSolution, rest) && cost(optimalSolution) <
            cost(solution);
    if (optimalSolution[index] -1 >= 0)
    {
        var betterSolution:= addValueToIndex(optimalSolution,-1,index);
        assert isSolution(betterSolution, rest - banknote);
        assert cost(betterSolution) == cost(optimalSolution) - 1;
        assert cost(optimalSolution) - 1 < cost(currentSolution);
        assert false;
    }
    else
    {
        while (0 < i)
        invariant 0 <= i <= index
        invariant forall x::index >= x >= i ==> optimalSolution[x] <= 1
        {
        i:= i - 1;
        assert isOptimalSolution(optimalSolution, rest);
        if (optimalSolution[i] > 1)
        {
            var optimalSolution' := optimalSolution[i:=optimalSolution[i
```

```
                ]-2];
36              optimalSolution' := optimalSolution' [i + 1:= optimalSolution'[
                    i+1]+1];
37              assert isSolution(optimalSolution', rest);
38              assert cost(optimalSolution') == cost(optimalSolution) - 1;
39              assert cost(optimalSolution') < cost(optimalSolution);
40              assert false;
41          }
42          }
43          assert solutionElementsSum(optimalSolution) <= banknote - 1;
44          assert rest >= banknote;
45          assert solutionElementsSum(optimalSolution) <= rest - 1;
46          assert isOptimalSolution(optimalSolution, rest);
47          assert false;
48          }
49      }
50  }
```

3.3.5 Lema banknoteMaxim32

Lema banknoteMaxim32 demonstrează proprietatea menționată anterior.

Știm că finalSolution este soluție optimă pentru $sum - rest$.

Dacă currentSolution la care adăugăm bancnota 32 este soluție optimă pentru $rest$, atunci suma soluției optime pentru $sum - rest$ adunată cu suma soluției optime pentru rest este soluție optimă pentru sumă.

BanknoteMaxim32 se asigură că este soluție optimă pentru rest currentSolution folosind lema exchangeArgument32.

```
1   lemma banknoteMaxim32(rest: int, sum: int, finalSolution: seq <int> )
2       requires rest >= 32
3       requires isValidSolution(finalSolution)
4       requires addOptimRestEqualsOptimSum(rest, sum, finalSolution)
5       ensures addOptimRestEqualsOptimSum(rest - 32, sum, solutionsSum(
            finalSolution, [0, 0, 0, 0, 0, 1]))
6   {
7       forall currentSolution | isValidSolution(currentSolution) && isSolution
            (currentSolution, rest - 32)
8         ensures isSolution(solutionsSum(solutionsSum(finalSolution,
              currentSolution), [0, 0, 0, 0, 0, 1]), sum)
```

```
    {
      assert isSolution(solutionsSum(currentSolution, [0, 0, 0, 0, 0, 1]),
          rest);
    }

    forall currentSolution | isValidSolution(currentSolution) &&
        isOptimalSolution(currentSolution, rest - 32)
      ensures isOptimalSolution(solutionsSum(solutionsSum(finalSolution,
          currentSolution), [0, 0, 0, 0, 0, 1]), sum)
    {
      forall someSolution | isValidSolution(someSolution) && isSolution(
          someSolution, sum)
        ensures cost(someSolution) >= cost(solutionsSum(solutionsSum(
            finalSolution, currentSolution), [0, 0, 0, 0, 0, 1]))
      {
        exchangeArgument32(rest, sum,  currentSolution);
      }
    }
}
```

3.3.6 Lema exchangeArgument32

Lema exchangeArgument32 presupune că nu este optimă soluția dacă adaugăm bancnota 32 și ajunge la o contradicție.

Știm că soluția e optimă pentru $rest - 32$, dar nu pentru rest, dacă adaugăm bancnota.

Consideră că există o altă soluție optimă pentru rest.

Deoarece nu avem o bancnotă mai mare de 32 cu care am putea înlocui 2 apariții ale acestei bancnote, verificăm doar proprietatea descrisă la observație.

Nu există două bancnote de aceeași valoare în soluție, mai mici de 32.

În acest mod știm că soluția presupusă nu poate fi optimă, deoarece, dacă respectă proprietatea suma produsă de aceasta nu poate fi egală cu rest.

```
lemma exchangeArgument32(rest: int, sum: int,  optimalSolution: seq <int> )
    requires 32 <= rest
    requires isValidSolution(optimalSolution)
    requires isOptimalSolution(optimalSolution, rest - 32)
    ensures isOptimalSolution(optimalSolution[5:= optimalSolution[5] + 1],
        rest)
```

```
{
    var solution:= optimalSolution[5:= optimalSolution[5] + 1];
    var i:= 4;
    if (!isOptimalSolution(solution, rest))
    {
        if (optimalSolution[i] > 1)
        {
            var solution:= optimalSolution[i:= optimalSolution[i] - 2];
            solution:= solution[i + 1:= solution[i + 1] + 1];
            assert isSolution(solution, rest - 32);
            assert cost(solution) == cost(optimalSolution) - 1;
            assert cost(optimalSolution) - 1 < cost(solution);
            assert false;
        }
        else
        {
            while (0 < i)
                invariant 0 <= i <= 4
                invariant forall index::4 >= index >= i ==> optimalSolution
                    [index] <= 1
            {
                i:= i - 1;
                if (optimalSolution[i] > 1)
                {
                    var solution:= optimalSolution[i:= optimalSolution[i] -
                        2];
                    solution:= solution[i + 1:= solution[i + 1] + 1];
                    assert isSolution(solution, rest - 32);
                    assert cost(solution) == cost(optimalSolution) - 1;
                    assert cost(optimalSolution) - 1 < cost(solution);
                    assert false;
                }
            }
            assert solutionElementsSum(optimalSolution) <= rest - 1;
            assert isOptimalSolution(solution, rest);
            assert false;
        }
    }
}
```

Concluzie

În cadrul acestei lucrări am realizat verificarea formală a unei implementări a metodei greedy ce rezolvă problema bancnotelor cu bancnote puteri ale lui 2.

Una dintre dificultățile întâlnite în acest proces a fost crearea invariantului care menține proprietatea de soluție optimă, așa a apărut predicatul addOptimRestEqualsOptimSum.

Acest predicat afirmă că o soluție optimă pentru o sumă x adunată cu o soluție pentru suma y are ca rezultat o soluție optimă pentru suma $x + y$. Astfel, secvența ce reprezintă soluția care se construiește pe parcursul buclei are garanția că atât timp cât se adună cu soluții locale optime, în final, vom avea o soluție finală optimă.

Parcurgerea acestor etape pentru verificare m-a făcut să aprofundez noțiunea de soluție și să înțeleg în amănunt formarea acesteia prin metoda greedy.

Pe viitor îmi propun să găsesc o modalitate de a unifica metodele banknoteMaxim pentru a funcționa pentru orice caz.

Un alt lucru interesant de făcut în viitor, ar fi găsirea unei proprietăți care să se aplice pentru orice sistem de bancnote, pentru a verifica faptul că produce o soluție optimă.

Bibliografie

1. https://github.com/alexxandra21/Licenta2023
2. https://core.ac.uk/works/69828860
3. https://arxiv.org/pdf/1412.4395.pdf
4. https://jarednielsen.com/sum-consecutive-powers-2/
5. http://www.cse.unsw.edu.au/ se2011/DafnyDocumentation/Dafny