Fachhochschule Münster
FB Elektrotechnik und Informatik
Labor für Softwareverifikation

Prof. Dr. Moritz Sinn
Julian Winter M.Sc.

WS 2025/26
Aufgabenblatt 5

Praktikum
Zuverlässige Softwaresysteme

Da Herr Winter in Elternzeit ist, wird das Praktikum Dienstag und Mittwoch von 10 bis 12 Uhr stattfinden. Bitte kommen Sie vorbereitet in den Termin, damit das Abtestat direkt erfolgen kann.

Um das Praktikum zu bestehen, müssen Sie mindestens 3 der 4 Aufgaben erfolgreich bearbeiten und Ihre Lösung erläutern bzw. Rückfragen beantworten können.

Hinweis: In diesem Praktikum wird der Dafny-Datentyp *string* verwendet. Bei *string* handelt es sich um ein Synonym für `seq<char>`. Der Sequence-Datentyp wurde in der Vorlesung ausführlich behandelt.

Aufgabe 1 Palindromprüfung

Gegeben ist die folgende Signatur einer Methode

```
method isPalindrom(mystring: string) returns (isPalindrom: bool)
```

(a) Schreiben Sie eine formale Spezifikation für die Methode: Es soll genau dann `true` zurückgegeben werden, wenn `mystring` ein *Palindrom* ist (Die Reihenfolge der Zeichen ist von links wie von rechts gelesen gleich). Spezifizieren Sie die Vor- und Nachbedingungen der Methode so, dass diese die Funktionalität der Methode definieren. Eine sinnvolle Vorbedingung kann beispielsweise die Anforderung sein, dass `mystring` mindestens ein Zeichen enthält.

(b) Geben Sie die Schleifenspezifikation an. Die Schleifenspezifikation sollte hinreichen um die Nachbedingung zu beweisen.

(c) Implementieren Sie die Schleife.

Aufgabe 2 Das größte Element

Gegeben ist die folgende Signatur einer Methode

```
method maximum(myarray : array<int>) returns (m : int)
```

(a) Schreiben Sie eine formale Spezifikation für die Methode: Es soll das größte Element in `myarray` zurückgeben werden. Spezifizieren Sie die Vor- und Nachbedingungen der Methode so, dass diese die Funktionalität der Methode definieren. Eine sinnvolle Vorbedingung kann beispielsweise die Anforderung sein, dass `myarray` mindestens ein Element enthält.

(b) Geben Sie die Schleifenspezifikation an. Die Schleifenspezifikation sollte hinreichen um die Nachbedingung zu beweisen.

(c) Implementieren Sie die Schleife.

Aufgabe 3 String-Matching

In dieser Aufgabe soll die Korrektheit der Implementierung des eines naiven String-Matching-Verfahren sichergestellt werden. *Hinweis: Verwenden Sie den Code welcher in Ilias zur Verfügung gestellt wurde, hier sind Annotierungen vorgenommen, welche Sie benötigen.*

(a) Betrachten Sie zunächst den naiven Algorithmus in Abb. 1 . Schreiben Sie eine formale Spezifikation für die *Korrektheit* der Methode (Vor- und Nachbedingungen) und verifizieren Sie diese in *Dafny*. Sie können beispielsweise fordern, dass die Zeichenkette wie auch das gesuchte Muster nicht leer sind. Die Nachbedingung sollte aussagen, dass das Muster, falls es gefunden wurde, an der gefunden Stelle tatsächlich zu finden ist. *Hinweis:* Es werden Schleifeninvarianten benötigt.

(b) Ergänzen Sie nun eine Nachbedingung für die *Vollständigkeit* der Methode und verifizieren Sie diese in *Dafny*. Diese Nachbedingung sollte aussagen, dass das Muster *nicht* in der Zeichenkette auftritt, falls es nicht gefunden wird. In Abb 2 finden Sie ein Prädikat welches Sie für die Formulierung der Nachbedingung wie auch der benötigten Schleifeninvariante verwenden können.

Aufgabe 4 String-Matching: KMP

Betrachten Sie nun die Implementierung des KMP-String-Matching-Algorithmus in Abb. 1 und verifizieren diese in *Dafny*. *Hinweis:* In Iilias finden Sie den Code mit der Spezifikation für die Methode `analysePrefix`.

```
method stringMatch(thestring: string, pattern: string) returns (start : int)
{
    if(|pattern| > |thestring|) {
      return -1;
    }
    var i : int;
    var j : int;
    i := 0;
    j := 0;

    while (i < |thestring| ∧ j < |pattern|)
    {
      if (thestring[i] = pattern[j]) {
        j := j + 1;
      } else {
        i := i - j;
        j := 0;
      }
      i := i + 1;
    }

    if(j = |pattern|) {
      return i - j;
    }
    return -1;
}
```

Abbildung 1: Implementierung eines naives String-Matching-Verfahren

```
predicate thereIsNoMatchStartingHere(thestring: string, pattern: string, here : int)
  requires 0 ≤ here
{
  if(here + |pattern| > |thestring|) then true else thestring[here .. here + |pattern|] ≠
pattern
}
```

Abbildung 2: Prädikat für den Nachweis der Vollständigkeit der Implementierung aus Abb. 1

```
method analysePrefix(pattern: string) returns (ptable : array<int>)
{
        if(|pattern| = 0) {
            return new int[0];
        }
        var N := new int[|pattern|];
        var i : int;
        var j : int;
        i := 0;
        j := -1;
        N[0] := -1;

        while(i < |pattern| - 1)
        {
            while(j ≥ 0 ∧ pattern[i] ≠ pattern[j])
            {
                j := N[j];
            }

            i := i + 1;
            j := j + 1;

            N[i] := j;
        }
        return N;
    }


method stringMatch(thestring: string, pattern: string) returns (start : int)
{
    if(|pattern| > |thestring|) {
            return -1;
    }
    var i : int;
    var j : int;
    i := 0;
    j := 0;
    var N := analysePrefix(pattern);

    while (i < |thestring| ∧ j < |pattern|)
    {
            if (thestring[i] = pattern[j]) {
                j := j + 1;
            } else if (j > 0) {
                j := N[j];
                i := i - 1;
            }
            i := i + 1;
    }

        if(j = |pattern|) {
            return i - j;
        }
        return -1;
}
```

Abbildung 3: Knuth-Morris-Pratt String-Matching-Algorithmus