Prof. M. Leuschel Düsseldorf, 18.04.2023

Testat zur Vorlesung

Theoretische Informatik

Testat 1, 24.04.2023–05.05.2023

Hinweise:

- Bei Ausgabe der Aufgaben sind gegebenenfalls noch nicht alle Inhalte in der Vorlesung besprochen worden. Mindestens eine Aufgabe kann aber bereits vorbereitet werden.
- Termine für die Testate können über das ILIAS gebucht werden (aber nur eine aktive Buchung auf einmal). Sie finden dort auch den Raum und einen Namen, der Ihnen mittlerweile vorgekommen sein sollte. Dieser Name wird vor Ort einem Tutor oder einer Tutorin zugeordnet.
- Bitte wählen Sie bei der Buchung grundsätzlich *keine Wiederholung*, auch wenn es nicht der Erstversuch ist.
- Halten Sie für das Testat bitte Ihren Studierendenausweis und einen Lichtbildausweis bereit.
- Für das Testat bringen Sie bitte eine Lösung *aller* unten stehenden Aufgaben mit. Diese Lösung dürfen Sie in Gruppen anfertigen.
- Im Testat wird *Ihr Verständnis* des Stoffes getestet. Ihre Lösung muss nicht perfekt sein, Sie müssen den Stoff aber verstanden haben.
- Sollten Sie ein Testat nicht bestehen, dürfen Sie auch bis zum 12.5.2023 weitere Versuche (aber immer nur einen aktiven auf einmal) buchen.
- Sollte es irgendwelche anderen Probleme geben wie etwa Krankheit, weitere benötigte Versuche, etc. wenden Sie sich bitte an theoinf@cs.uni-duesseldorf.de.
- Ihren Fortschritt über bestandene Testate können Sie ab dem nächsten Tag im ILIAS einsehen. Sollte das einmal nicht der Fall sein, wenden Sie sich bitte an obige Mailadresse.

Aufgabe 1 Mehrdeutige Ableitungen
In Java (und vielen anderen Programmiersprachen) ist es möglich, geschwungene Klammern bei den Zweigen einer if-Anweisung wegzulassen, wenn nur eine Anweisung vorhanden ist. Gültige Codeschnippsel wären also beispielsweise:

```
if (true)                    if (true)
    x++;                         x++;
else
    x--;
```

Eine vereinfachte Grammatik dafür ist $G = (\{\textbf{if}, \textbf{then}, \textbf{else}, \texttt{x++;}, \texttt{x--;}, \textbf{true}, \textbf{false}\}, \{S, C\}, S, P)$ mit

$$\begin{aligned} P = \{S \to\ & \textbf{if}\ C\ \textbf{then}\ S \mid \\ & \textbf{if}\ C\ \textbf{then}\ S\ \textbf{else}\ S \mid \\ & \texttt{x++;} \mid \texttt{x--;}\ , \\ C \to\ & \textbf{true} \mid \textbf{false}\} \end{aligned}$$

Finden Sie ein Programm, das in dieser Grammatik mehrdeutig ist. Zeigen Sie das, indem Sie mindestens zwei Syntaxbäume dafür angeben.

Anmerkung: Hierbei handelt es sich um das sogenannte "dangling else"-Problem. Sie zeigen hier, dass die Zuordnung des `else`-Branches nicht über diese Grammatik lösbar ist.

Aufgabe 2 Scanner
Ziel der Aufgabe ist es, einen Teil eines Tokenizers zu schreiben. Ein Tokenizer ist ein Teil des Parsers, der eine Zeichenkette (z.B. eine Datei mit Sourcecode) in sogenannte Tokens umwandelt, auf denen später das eigentliche Parsing passiert. Auf diesen Tokens kann man dann Regeln wie

Statement → **if** *Expression* **then** *Statement* **else** *Statement* | *Statement Statement*

schreiben. Hier sind dann **if**, **then** und **else** keine Zeichenketten mehr, sondern Tokens, die direkt Elemente im Alphabet der Grammatik sind.
Für den Tokenizer verwendet man in der Regel reguläre Ausdrücke, wie etwa in der Form:

```
Literal := Number | Identifier
Number := [0-9][0-9]*
Identifier = [a-ZA-Z][a-zA-Z0-9]*
Literal_or_ws := Literal | ws ws*
```

Hier steht `ws` für Whitespace (Leerzeichen, Tabulator, Zeilenumbruch, ...) und `[0-9]` ist eine Range, in der alle Ziffern von 0 bis 9 enthalten sind. Analog sind etwa in `[a-zA-Z0-9]` alle Klein- und Großbuchstaben sowie alle Ziffern enthalten.

So ein regulärer Ausdruck wird dann in einen NFA übersetzt, der aus Performance-Gründen dann in einen DFA umgewandelt wird.

(a) Wir beginnen nur mit dem regulären Ausdruck `[0-9][0-9]`*. Generieren Sie daraus entsprechend der Konstruktion des Satzes 2.23 (Kleene) einen NFA. Sie dürfen für die Kanten ebenso die Range `0-9` verwenden, damit Sie nicht zehn Kanten zeichnen müssen.

Geben Sie *alle drei* Automaten an, die unterwegs konstruiert werden.

Kontrolle: Der finale Automat sollte 5 Zustände und 6 Kanten haben. Testen Sie auch, ob der Automat zu dem regulären Ausdruck passt.

(b) Überführen Sie den NFA in einen DFA gemäß der Konstruktion des Satzes 2.13 (Rabin-Scott). Sie dürfen Ihre (End-)Zustandsmenge und Überführungsfunktion auf erreichbare Zustände aus der Potenzmenge beschränken.

(c) Finden Sie einen kleineren Automaten, der denselben regulären Ausdruck akzeptiert?

(d) Ergänzen Sie den Automaten um Zustände und Transitionen, sodass auch Whitespaces (`ws ws*`) und Identifier (`[a-ZA-Z][a-zA-Z0-9]*`) erkannt werden. Sie müssen die Ergänzungen nicht erneut über einen NFA konstruieren.

Anmerkung: Die Unterscheidung von Tokens wie `if` von Variablenbezeichnern wird üblicherweise nicht in dem Automaten, sondern in einem nachfolgenden Schritt gemacht.

Aufgabe 3 Reguläre Ausdrücke
Häufig werden auf regulären Ausdrücken noch andere Operatoren definiert, die eine kompaktere Darstellung erlauben. Beispiele sind:

(a) α^+ — α steht beliebig oft, aber mindestens einmal, hintereinander

(b) $\alpha?$ — α tritt einmal oder gar nicht auf

(c) α^n — für eine natürliche Zahl n wird α genau n-mal wiederholt

Geben Sie jeweils eine Umschreibung an, die unserer Definition von regulären Ausdrücken genügt.

Ergänzende Anmerkung: Was in vielen Programmiersprachen als "RegEx" zur Verfügung steht, enthält oft Erweiterungen, die *nicht* mehr regulär, also *keine* Typ-3 Sprache, sind. Beispielsweise kann man mit runden Klammern Gruppen festlegen und später bspw. mit `\1` (für die erste Gruppe) referenzieren; der Ausdruck `(0*)1\1` steht dann für eine beliebige Anzahl an Nullen, gefolgt von exakt einer Eins, gefolgt von genau der gleichen Zahl von Nullen wie am Anfang.

Prof. M. Leuschel Düsseldorf, 03.05.2023

Testat zur Vorlesung
Theoretische Informatik
Testat 2, 09.05.2023-22.05.2023

Hinweise:

- Es gelten die Hinweise des ersten Testats. Der Zeitraum für erste Versuche ist oben angegeben.
- Achten Sie bei der Buchung darauf, einen Termin für Testat 2 zu buchen, nicht für Testat 1.
- Sollten Sie das Testat wiederholen müssen, dürfen Sie bis zum 29.5.2023 weitere Versuche (aber auch hier immer nur einen aktiven auf einmal) buchen.
- Sollte es irgendwelche anderen Probleme geben wie etwa Krankheit, weitere benötigte Versuche, etc. wenden Sie sich bitte an theoinf@cs.uni-duesseldorf.de.

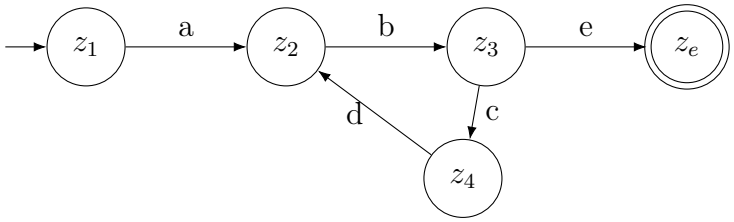
Aufgabe 1 Aufgepumpte Klammern
Wir betrachten die folgende Grammatik, die genau die korrekt geklammerten Additionen der Bezeichner a und b enthält: $G = (\{ \texttt{(}, \texttt{a}, \texttt{+}, \texttt{b}, \texttt{)} \}, \{S\}, S, P)$ mit

$$P = \{S \to \texttt{(}\ S\ \texttt{)} \mid S\ \texttt{+}\ S \mid \texttt{a} \mid \texttt{b}\}$$

Zeigen Sie mit Hilfe des Pumping Lemmas, dass die von G erzeugte Sprache $L(G)$ nicht regulär ist.

Aufgabe 2 Automaten und Pumpen

Gegeben sei der folgende DFA M:



Offensichtlich ist die von M akzeptierte Sprache $L(M)$ regulär. Also existiert nach dem Pumping-Lemma eine natürliche Zahl p, die die Zerlegung eines Wort $x = uvw$ mit $|x| \geq p$ erlaubt mit

1 $|uv| \leq p$

2 $|v| \geq 1$

3 $\forall i \geq 0 \,.\, uv^i w \in L(G)$

Eine valide Pumping-Lemma-Zahl für M ist $p = 5$. Was ist eine Zerlegung für das Wort $x = abcdbcdbcdbe \in L(M)$, die den Anforderungen oben genügt?

Aufgabe 3 Reguläre Sprachen?
Welche der folgenden Sprachen sind regulär (über einem Alphabet bestehend aus Unicode-Zeichen)?

(a) alle validen Datumsangaben in der Form TT.MM.JJJJ

(b) HTML

(c) Java

Sie müssen keinen Beweis führen. Eine intuitive Begründung ist ausreichend.

Aufgabe 4 Minimalautomaten & Myhill-Nerode
Gegeben sei der DFA $M = (\{0,1\}, \{z_0, \ldots, z_4\}, z_0, \delta, \{z_0, z_4\})$ mit δ wie folgt:

δ	z_0	z_1	z_2	z_3	z_4
0	z_3	z_2	z_2	z_4	z_3
1	z_1	z_0	z_2	z_2	z_1

(a) Bestimmen Sie mit dem Algorithmus der Vorlesung den Minimalautomaten M' zu M.

(b) Bestimmen Sie die Äquivalenzklassen der Myhill-Nerode-Relation zu M'. Sie müssen nicht die Teilsprachen der Klassen formal charakterisieren, es reicht aus, einen Repräsentanten anzugeben.

(c) Gegeben sei die folgende Tabelle aus dem Algorithmus zu Minimalautomaten:

	z_0	z_1	z_2	z_3	z_4	z_5	z_6
z_7	$\times_0$	$\times_F$		$\times_F$		$\times_0$	$\times_F$
z_6	$\times_F$		$\times_F$	$\times_1$	$\times_F$	$\times_F$	
z_5	$\times_1$	$\times_F$	$\times_0$	$\times_F$	$\times_0$		
z_4	$\times_0$	$\times_F$		$\times_F$			
z_3	$\times_F$	$\times_1$	$\times_F$				
z_2	$\times_0$	$\times_F$					
z_1	$\times_F$						

Welche Zustände kann man zusammenlegen?

Prof. M. Leuschel Düsseldorf, 17.05.2023

Testat zur Vorlesung

Theoretische Informatik

Testat 3, 24.05.2023-09.06.2023

Hinweise:

- Es gelten die Hinweise des ersten Testats. Der Zeitraum für erste Versuche ist oben angegeben.
- Achten Sie bei der Buchung darauf, einen Termin für Testat 3 zu buchen, nicht für Testat 2.
- Sollten Sie das Testat wiederholen müssen, dürfen Sie bis zum 20.06.2023 weitere Versuche (aber auch hier immer nur einen aktiven auf einmal) buchen.
- Sollte es irgendwelche anderen Probleme geben wie etwa Krankheit, weitere benötigte Versuche, etc. wenden Sie sich bitte an theoinf@cs.uni-duesseldorf.de.

Aufgabe 1 CNF
Nutzen Sie für alle Aufgabenteile die Konstruktionen der Vorlesung.

(a) Gegeben sei die Grammatik $G_1 = (\{0,1\}, N_1, S, P_1)$ mit $N_1 = \{S, A, B, C, D\}$ und P_1 wie unten definiert. Geben Sie eine äquivalente λ-freie Grammatik G_1' an.

$$\begin{aligned} P_1 = \{&S \to AD \mid DA,\\ &A \to BC,\\ &B \to S1 \mid 0,\\ &C \to 1 \mid \lambda,\\ &D \to AB \mid CCC \mid 0C\} \end{aligned}$$

(b) Gegeben sei die Grammatik $G_2 = (\{a,b,c,d\}, N_2, S, P_2)$ mit $N_2 = \{S, A, B, C, D\}$ und P_2 wie unten definiert. Geben Sie eine äquivalente Grammatik G_2' ohne Zyklen an.

$$\begin{aligned} P_2 = \{&S \to ABCS \mid a,\\ &A \to a \mid aB,\\ &B \to C \mid bA,\\ &C \to AD \mid D \mid c,\\ &D \to B \mid d\} \end{aligned}$$

(c) Gegeben sei die Grammatik $G_3 = (\{a,b,c,d\}, N_3, S, P_3)$ mit $N_3 = \{S, A, B, C, D\}$ und P_3 wie unten definiert. Geben Sie eine äquivalente Grammatik G_3' ohne einfache Regeln an.

$$\begin{aligned} P_3 = \{&S \to ABCS \mid a,\\ &A \to a \mid aB,\\ &B \to C \mid bA,\\ &C \to AD \mid D \mid c,\\ &D \to d\} \end{aligned}$$

(d) Gegeben sei die Grammatik $G_4 = (\{a,b\}, N_4, S, P_4)$ mit $N_4 = \{S, A, B\}$ und P_4 wie unten definiert. Geben Sie eine äquivalente Grammatik G_4' in Chomsky-Normalform an.

$$\begin{aligned} P_4 = \{&S \to ABAS \mid ab,\\ &A \to aA \mid a,\\ &B \to b \mid abb\} \end{aligned}$$

Aufgabe 2 CYK

Gegeben sei die Grammatik G vom ersten Testatsblatt aus Aufgabe 1. Eine äquivalente Grammatik lautet:
$G' = (\{\textbf{if}, \textbf{then}, \textbf{else}, \texttt{x++;}, \texttt{x--;}, \textbf{true}, \textbf{false}\}, \{S, S_2, C, C_2, E, E_2, I, T, T_2\}, S, P)$ mit

$$\begin{aligned} P = \{ & S \to IC_2 \mid \texttt{x++;} \mid \texttt{x--;}\ , \\ & C_2 \to CT_2, \\ & T_2 \to TS_2 \mid TS, \\ & S_2 \to SE_2, \\ & E_2 \to ES, \\ & C \to \textbf{true} \mid \textbf{false}, \\ & I \to \textbf{if}, \\ & T \to \textbf{then}, \\ & E \to \textbf{else}\} \end{aligned}$$

Nach dem Tokenizing-Schritt auf dem Testatsblatt 1 ist nun das Ziel, wie ein Parser zu entscheiden, ob ein Wort nun ein valides Programm der (Programmier-)Sprache ist.

(a) Zeigen Sie mit dem CYK-Algorithmus, dass das Wort **if true then** `x++;` **else** `x--;` in $L(G')$ enthalten ist.

(b) Zeigen Sie mit Hilfe des CYK-Algorithmus, dass das Wort **if true then** `x++;` **else** nicht in $L(G')$ enthalten ist.

Aufgabe 3 PDA

(a) Gegeben sei der PDA $(\Sigma, \Gamma, Z, \delta, z_0, \#)$ mit
$\Sigma = \{a, b, c\}$, $\Gamma = \{X, \#\}$, $Z = \{z_0, z_1, z_2\}$ und δ wie folgt:

$$\begin{aligned} z_0\, a\, \# &\to z_1\, X \# \\ z_0\, a\, X &\to z_1\, X X \\ z_0\, c\, X &\to z_2\, \lambda \\ z_1\, b\, X &\to z_0\, X X \\ z_2\, c\, X &\to z_2\, \lambda \\ z_2\, \lambda\, \# &\to z_2 \lambda \end{aligned}$$

Was ist die Sprache, die der PDA akzeptiert?

(b) Was ist der Unterschied zwischen PDAs und DPDAs?

Aufgabe 4 CfG $\rightarrow$ PDA
Gegeben sei die kontextfreie Grammatik $G_4 = (\{a, b\}, N_4, S, P_4)$ mit $N_4 = \{S, A, B\}$ und P_4 wie folgt:

$$\begin{aligned} P_4 = \{S &\rightarrow ABAS \mid ab, \\ A &\rightarrow aA \mid a, \\ B &\rightarrow b \mid abb\} \end{aligned}$$

Geben Sie einen PDA M an mit $L(M) = L(G_4)$ wie er mit dem Algorithmus der Vorlesung entsteht.

Prof. M. Leuschel Düsseldorf, 07.06.2023

Testat zur Vorlesung

Theoretische Informatik

Testat 4, Erstversuche: 14.06.2023–23.06.2023

Hinweise:

- Sollten Sie das Testat wiederholen müssen, dürfen Sie bis zum 30.06.2023 weitere Versuche (aber auch hier immer nur einen aktiven auf einmal) buchen.
- Sollte es irgendwelche anderen Probleme geben wie etwa Krankheit, weitere benötigte Versuche, etc. wenden Sie sich bitte an theoinf@cs.uni-duesseldorf.de.

Aufgabe 1 Turing-Maschine

Über dem Alphabet $\Sigma = \{\text{<div>}, \text{</div>}, \widehat{\text{<div>}}, \widehat{\text{</div>}}\}$ definieren wir die Turing-Maschine $M = (\Sigma, \Sigma \cup \{\square\}, \{z_0, z_1, z_2, z_3, z_4, z_e\}, \delta, z_0, \square, \{z_e\})$ mit δ wie folgt:

δ	z_0	z_1	z_2	z_3	z_4
	$(z_1, \widehat{\text{<div>}}, R)$	$(z_1, \text{<div>}, R)$	$(z_1, \square, R)$		$(z_1, \widehat{\text{<div>}}, R)$
$\widehat{\text{<div>}}$			$(z_4, \widehat{\text{<div>}}, R)$	$(z_e, \widehat{\text{<div>}}, N)$	
		$(z_2, \square, L)$			
$\widehat{\text{</div>}}$		$(z_3, \widehat{\text{</div>}}, L)$			
$\square$	$(z_e, \square, N)$	$(z_1, \square, R)$	$(z_2, \square, L)$	$(z_3, \square, L)$	$(z_4, \square, R)$

(a) Geben Sie die Konfigurationenfolge an, wenn die Initialkonfiguration z_0 $\widehat{\text{</div>}}$ ist. Ausnahmsweise müssen Sie hier nicht alle Zwischenschritte angeben. Es reichen diejenigen Schritte aus, in denen ein Symbol verändert oder der Zustand gewechselt wird.

(b) Welche Sprache akzeptiert die Turing-Maschine M? Eine informelle Beschreibung reicht hier aus.

(c) Was unterscheidet einen LBA von einer Turing-Maschine?

(d) Handelt es sich bei M um einen LBA?

Aufgabe 2 mod im Turing-Express

Schreiben Sie eine Turingmaschine M, die eine Funktion berechnet, die auf den natürlichen Zahlen im Dezimalsystem, also auf Σ^* mit $\Sigma = \{0, \ldots, 9\}$ definiert ist. M soll den Rest bei Division durch 3 angeben, also die Funktion $f(x) = x \bmod 3$ berechnen.

Hinweis: Sie können sich Teilbarkeitsregeln durch 3 zu Nutze machen.

Aufgabe 3 go to loop or not to loop

Gegeben seien folgende Code-Schnipsel. Welche sind jeweils valide LOOP-, WHILE-, bzw. GOTO-Programme?

```
P1: N → N
  x2 := x1 + 1;
  WHILE x2 ≠ 0 DO
    x0 := x2 + 1
  END
```

```
P2: N × N → N
  x0 := x1 + 0;
  LOOP x2 DO
    x0 := x2 + 1;
  END
```

```
P3: N → N
  x0 := x1;
  x2 := x1;
  LOOP x2 DO
    x0 := x0 + x1
  END
```

```
P4: N × N → N
  x0 := x1 + 0;
  LOOP x2 DO
    x0 := x0 + 1;
    x2 := x2 − 1
  END
```

```
P5: N → N
  LOOP x1 DO
    LOOP x1 DO
      x0 := x0 + 1
    END
  END
```

```
P6: N → N
  x2 := x1 − 2;
  WHILE x2 ≠ 0 DO
    x1 := x1 − 3;
    x2 := x2 − 3;
  END;
  x0 := x1 + 0
```

```
P7: N → N
  x0 := 0;
  x2 := x1 + 0;
  x3 := x1 + 0;
  LOOP x3 DO
    LOOP x2 DO
      x0 := x0 + 1
    END
  END
```

```
P8: N² → N
    x0 := x1 + 0;
    WHILE x2 ≠ 0 DO
      x0 := x0 + 1;
      x2 := x2 − 1;
      x3 := x0 + 0;
      x4 := 100;
      LOOP x4 DO
        x3 := x3 − 1;
      END;
      LOOP x3 DO
        GOTO L1;
      END
    END;
L1: x0 := x0 + 1;
    HALT;
```

```
P9: N → N
    x0 := 1;
L1: IF x1 = 0 THEN GOTO L2;
    x2 := x0 + 0;
    x3 := x1 + 0;
L3: IF x3 = 1 THEN GOTO L4;
    x4 := x0 + 0;
L5: IF x4 = 0 THEN GOTO L6;
    x2 := x2 + 1;
    x4 := x4 − 1;
    GOTO L5;
L6: x3 := x3 − 1;
    GOTO L3;
L4: x0 := x2 + 0;
    x1 := x1 − 1;
    GOTO L1;
L2: HALT;
```

Prof. M. Leuschel Düsseldorf, 20.06.2023

Testat zur Vorlesung
Theoretische Informatik
Testat 5, Erstversuche: 26.06.2023–04.07.2023

Hinweise:

- Sollten Sie das Testat wiederholen müssen, dürfen Sie bis zum 14.07.2023 weitere Versuche (aber auch hier immer nur einen aktiven auf einmal) buchen.
- Sollte es irgendwelche anderen Probleme geben wie etwa Krankheit, weitere benötigte Versuche, etc. wenden Sie sich bitte an theoinf@cs.uni-duesseldorf.de.

Aufgabe 1 Primitive Rekursion

(a) Berechnen Sie $mult(3, 2)$. Verwenden Sie nur die Definitionen von $mult$ und add, und vereinfachen Sie nur die Nachfolgefunktion $s(x)$ und Projektionen. Geben Sie alle Zwischenschritte an.

(b) Zeigen Sie mit den Normalschemata, dass die folgende Funktion $ite(C, T, E)$ primitiv rekursiv ist:

$$ite(C, T, E) = \begin{cases} T & \text{, falls } C \neq 0 \\ E & \text{, falls } C = 0. \end{cases}$$

Aufgabe 2 Rekursion

(a) Gegeben sei die Funktion $f : \mathbb{N}^2 \to \mathbb{N}$ mit

$$f(x, k) = \begin{cases} k - x^2 & \text{, falls } k \geq x^2 \\ 0 & \text{, sonst.} \end{cases}$$

Bestimmen Sie die Funktion μf.

(b) Zu welchen Formalismen sind partiell rekursive Funktionen äquivalent?

(c) Wie könnte man einem WHILE-Programm A ein anderes WHILE-**Programm** B als Parameter übergeben?

Aufgabe 3 Entscheidbarkeit, Aufzählbarkeit, Halteproblem

(a) Welche der folgenden Mengen sind rekursiv aufzählbar?

- Die Menge der Java-Programme, die bei Eingabe "Hummelbummel" terminieren
- Die Menge der Java-Programme, die bei Eingabe "Hummelbummel" nicht terminieren
- Die Menge aller Java-Programme
- Die leere Menge
- Die Menge der Java-Programme, die "Hummelbummel" ausgeben
- Die Menge der Java-Programme, die nicht "Hummelbummel" ausgeben
- Die Menge der Strings, die keine Java-Programme sind

(b) Formulieren Sie eines der Halteprobleme und dessen Beweisidee in eigenen Worten.

(c) Welche der folgenden Eigenschaften von beliebigen Java-Programmen kann man maschinell verifizieren?

- Das Programm ist syntaktisch korrekt.
- Das Programm terminiert.
- Es gibt keine NullpointerException.
- Alle Variablen werden vor ihrer jeweiligen Verwendung deklariert.
- Alle Codezeilen sind erreichbar.