

Fragen, Antworten und Kommentare zur aktuellen Vorlesung

Das Video zur Lösung der Aufgaben 12 und 13 finden Sie unter: <https://youtu.be/1XP2INE0Y04>

Das Video zur Lösung der Aufgaben 14 und 15 finden Sie unter: <https://youtu.be/BXVDv85ToQg>. Wie oft und gerade bei Aufgabe 15a gilt, dass es auch andere Lösungsansätze gibt, die mindestens genauso gut, wie mein Ansatz sind. Hier ohne Kommentar: $S \rightarrow aaaSb \mid aa \mid a \mid \varepsilon$

Frage: Sollen wir in der Klausur immer die vollständige Grammatik als 4-Tupel angeben?

Antwort: Bei Aufgaben der Form in denen „nur“ die Regelmenge ausgetauscht wird oder die anderen Mengen klar ersichtlich sind, reicht auf jeden Fall die Angabe der neuen Regelmenge aus. Sollten Sie eventuell an einer mündlichen Prüfung teilnehmen müssen, sollten sie aber das 4-Tupel genau erklären können.

Frage: In Aufgabe 12 habe ich mehrere Worte gefunden, für die es nur eine Ableitung gibt, was heißt das dann für die Mehrdeutigkeit?

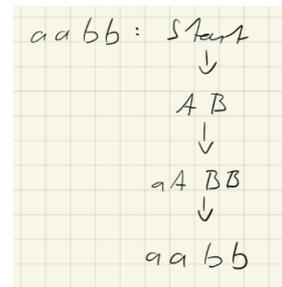
Antwort: Das wäre nur relevant, wenn alle Worte eindeutig erzeugt werden können. Um die Mehrdeutigkeit einer Grammatik zu zeigen, reicht es aus ein einziges Wort zu finden, das zwei verschiedene Linksableitungen hat.

Frage: Ich gehe gerade für die Klausurvorbereitung die Übungsaufgaben noch einmal durch.

Gegeben sei folgende Grammatik $(\{Start, A, B\}, \{a, b\}, \text{Regeln}, \text{Start})$ mit den Regeln

$Start \rightarrow AB \quad A \rightarrow aA \mid a \mid B \mid \varepsilon, \quad B \rightarrow BB \mid b$

Kann es sein, dass bei Ihrer Lösung ein Fehler vorliegt? Bei a) haben Sie aus $AB \rightarrow aAB$ gemacht. Müsste es nicht $aABB$ sein, da es keinen Übergang von $B \rightarrow B$ gibt?



Antwort: Gut, dass Sie fragen, nein, die Lösung stimmt, zumindest an dieser Stelle. Sie machen einen Denkfehler beim Ableiten, Sie wollen in jedem Schritt jedes Nichtterminal-Zeichen ableiten, das ist nicht gefordert. Es wird in jedem Schritt genau eines der Nichtterminal-Zeichen mit der rechten Seite einer passenden Regel ersetzt. D.h, aus AB kann in einem Schritt aAB oder aB oder BB oder B oder auch ABB oder Ab werden. Der Fehler fällt nicht so schnell auf, da natürlich alle Worte, die Sie ableiten auch ableitbar (in 2 oder mehr Schritten) sind.

Frage: Wie können wir zeigen, dass eine Sprache mehrdeutig ist?

Antwort: Wird in der Vorlesung nicht behandelt, da hierfür jeweils ein individueller Beweis gefordert ist. Der Ansatz sieht typisch so aus. Eine Sprache ist nicht mehrdeutig, wenn es zumindest eine eindeutige Grammatik gibt. Dann wird angenommen, dass es diese Grammatik gibt und dann konstruiert, dass dann ein Wort abgeleitet werden können muss, das nicht in der Sprache liegt. Alternativ wird gezeigt, dass es ein Wort gibt, das nicht abgeleitet werden kann. Daraus folgt dann der Widerspruch zur Existenz einer eindeutigen Grammatik. Zu dem Thema gibt es einige Sprachen

mit Beweisen im Internet, die alle recht komplex sind. Aus diesem Grund ist der Nachweis der Mehrdeutigkeit einer Sprache keine geeignete Klausuraufgabe.

Frage: Warum wird in den Tests zu den Grammatiken nicht genau geprüft, ob die Grammatik richtig ist?

Antwort: Eine solche Überprüfung ist nur in Spezialfällen vollständig möglich. Generell gilt, dass es keinen Algorithmus für zwei kontextfreie Grammatiken gibt, der überprüfen kann, dass diese Grammatiken sprachäquivalent sind. Das Problem ist unentscheidbar.

Frage: Kann man in den Lösungsvideos die Lösung noch detaillierter erklären?

Antwort: Kurz, nein. Die Lösungsvideos geben ein laut erzählten Überblick über einen möglichen Lösungsansatz basierend auf der Vorlesung. Die Videos wiederholen Details der Vorlesung nicht, deshalb muss diese vorher durchgearbeitet werden. Denken Sie immer daran, dass während der Veranstaltungszeit wir direkt über die Aufgaben reden können.

Frage: Können die Lösungsüberprüfungen auch ohne Eclipse genutzt werden?

Antwort: Generell ja, da JUnit unabhängig von einer Entwicklungsumgebung auch von der Konsole aus aufrufbar ist. Allerdings ist der Weg recht aufwendig. Dabei gibt es eine einfachere funktionierende Lösung mit Schwächen in der Ausgabe und eine komplexere und schönere Lösung. Beide werden im Folgenden vorgestellt.

Bei der einfachen Lösung wird die Command-Umgebung (cmd) von Windows genutzt. Die Konsole wird entweder über Windows direkt oder über die KleukerSEU mit StartKonsole.bat aufgerufen. Wird die KleukerSEU nicht genutzt, muss Java installiert sein. Als Beispiel wird folgende Aufgabe betrachtet.

a) Übertragen Sie Ihre Turing-Maschine aus 4a) in das Format des Simulators in die vorhandene Datei `beispiele/turingmaschinen/TMcca_nb_n.tm`. Prüfen Sie mit den JUnit-Tests aus `test.turingMaschine.TMcca_nb_nTest.java` ob Ihre Maschine die enthaltenen Tests besteht. Korrigieren Sie gegebenenfalls Ihre Turing-Maschine.

Zur Vorbereitung wird die Datei `theoriesammlung.zip` von der Veranstaltungsseite geladen und in einem beliebigen Verzeichnis, hier `C:\Internet` ausgepackt. Weiterhin wird eine Hilfsbibliothek zur einfachen JUnit-Ausführung benötigt, die unter <https://repo1.maven.org/maven2/org/junit/platform/junit-platform-console-standalone/> und der höchsten Nummer heruntergeladen werden kann. Im Beispiel wird <https://repo1.maven.org/maven2/org/junit/platform/junit-platform-console-standalone/1.9.3/junit-platform-console-standalone-1.9.3.jar> genutzt und in das Unterverzeichnis der Theoriesammlung kopiert. Das Verzeichnis sieht dann wie folgt aus, in diesem Verzeichnis erfolgen auch die folgenden Schritte:

```
C:\Internet\theoriesammlung> dir
Volume in Laufwerk C: hat keine Bezeichnung.
Volumeseriennummer: 64A2-XYZA
```

Verzeichnis von C:\Internet\theoriesammlung

```
03.04.2024 16:52 <DIR> .
03.04.2024 16:52 <DIR> ..
03.12.2022 12:39          502 .classpath
03.12.2022 11:43          391 .project
03.12.2022 11:43 <DIR> .settings
02.03.2023 20:36 <DIR> beispiele
19.02.2024 20:40 <DIR> bin
03.04.2024 16:36          2.614.186 junit-platform-console-standalone-1.9.3.jar
02.03.2023 20:27 <DIR> src
                3 Datei(en),          2.615.079 Bytes
                7 Verzeichnis(se), 15.514.030.080 Bytes frei
```

Jetzt muss zunächst die zur Verfügung gestellte Testdatei aus der Aufgabenstellung übersetzt werden. Dies erfolgt mit dem folgenden Befehl, der in einer Zeile steht.

```
javac -d out -cp junit-platform-console-standalone-1.9.3.jar;src
src\test\turingMaschine\TMcca_nb_nTest.java
```

Mit -d out wird ein Verzeichnis für die übersetzten Dateien angegeben. Werden verschiedene Tests ausgeführt, ist es sinnvoll, dieses Verzeichnis zwischenzeitlich wieder zu löschen.

Die eigentlichen Tests werden mit dem folgenden Befehl ausgeführt. Der letzte Parameter verhindert die farbige Ausgabe der Ergebnisse, da im Standard keine ANSI-Steuerzeichen unterstützt werden. Die Turing-Maschine muss in der in der Aufgabe geforderten Datei stehen. Im Beispiel befindet sich dort eine Turing-Maschine, die keine Schritte machen kann.

```
java -jar junit-platform-console-standalone-1.9.3.jar --class-path out --scan-
class-path --disable-ansi-colors
```

Der folgende Ausschnitt des Ergebnisses zeigt die Testergebnisse, weiterhin sind Ausgaben der theoriesammlung erahnbar, allerdings nicht klar lesbar. Zu jedem gescheiterten Test werden weitere Informationen ausgegeben.

```
C:\Internet\theoriesammlung>java -jar junit-platform-console-standalone-1.9.3.jar
--class-path out --scan-class-path --disable-ansi-colors
Start: (Start, #aabb<[4m#<[0m)
Start: (Start, #ccab<[4m#<[0m)
Start: (Start, #cc<[4m#<[0m)
Start: (Start, #ccaaabbb<[4m#<[0m)
Start: (Start, #ccaaaaaaaabbbbbbbbbb<[4m#<[0m)
Start: (Start, #<[4m#<[0m)
Start: (Start, #cb<[4m#<[0m)
Start: (Start, #cab<[4m#<[0m)
Start: (Start, #cccab<[4m#<[0m)
Start: (Start, #ccaacbb<[4m#<[0m)
```

Thanks for using JUnit! Support its development at <https://junit.org/sponsoring>

```
.
+-- JUnit Jupiter [OK]
| '-- TMcca_nb_nTest [OK]
|   +-- test10() [OK]
|   +-- test1() [X] expected: <true> but was: <false>
|   +-- test2() [X] expected: <true> but was: <false>
|   +-- test3() [X] expected: <true> but was: <false>
|   +-- test4() [X] expected: <true> but was: <false>
|   +-- test5() [OK]
```

```
|  +-- test6() [OK]
|  +-- test7() [OK]
|  +-- test8() [OK]
|  '-- test9() [OK]
+-- JUnit Vintage [OK]
'-- JUnit Platform Suite [OK]
```

Failures (4):

```
JUnit Jupiter:TMcca_nb_nTest:test1()
  MethodSource [className = 'test.turingMaschine.TMcca_nb_nTest', methodName =
'test1', methodParameterTypes = '']
```

Um zu einer lesbareren Ausgabe zu kommen, wird eine Erweiterung der Windows Powershell genutzt, die unter <https://github.com/microsoft/terminal/releases> geladen werden kann. Im Beispiel wird

<https://github.com/microsoft/terminal/releases/download/v1.19.10821.0/Microsoft.WindowsTerminal.1.19.10821.0.x64.zip> genutzt. Um Java temporär in der Shell zu nutzen, kann folgender Befehl genutzt werden, der zum Installationsverzeichnis zeigen muss.

```
$env:Path = 'C:\kleukersSEU\java\bin;' + $env:Path
```

Es wird das Programm WindowsTerminal.exe gestartet. Generell werden fast die gleichen Befehle genutzt, allerdings der Class-Path (-cp) muss bei der Kompilierung in Anführungszeichen stehen und bei der Ausführung wird der letzte Parameter weggelassen. Ein Ausschnitt des Ergebnisses sieht wie folgt aus.

```

Windows PowerShell
Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.

Lernen Sie das neue plattformübergreifende PowerShell kennen - https://aka.ms/pscore6

PS C:\Users\y> cd C:\Internet\theoriesammlung\
PS C:\Internet\theoriesammlung> $env:Path = 'C:\kleukersSEU\java\bin;' + $env:Path
PS C:\Internet\theoriesammlung> javac -d out -cp 'junit-platform-console-standalone-1.9.3.jar;src' src\test\turingMaschine\TMcca_nb_nTest.java
PS C:\Internet\theoriesammlung> java -jar junit-platform-console-standalone-1.9.3.jar --class-path out --scan-class-path
Start: (Start, #aabb#c)
Start: (Start, #ccab#c)
Start: (Start, #cc#c)
Start: (Start, #ccaaabbb#c)
Start: (Start, #ccaaaaaaaabbbbbbbbbb#c)
Start: (Start, ##)
Start: (Start, #cb#c)
Start: (Start, #cab#c)
Start: (Start, #cccab#c)
Start: (Start, #ccaacbb#c)

Thanks for using JUnit! Support its development at https://junit.org/sponsoring

+-- JUnit Jupiter [OK]
|   '-- TMcca_nb_nTest [OK]
|       +-- test10() [OK]
|       +-- test1() [X] expected: <true> but was: <false>
|       +-- test2() [X] expected: <true> but was: <false>
|       +-- test3() [X] expected: <true> but was: <false>
|       +-- test4() [X] expected: <true> but was: <false>
|       +-- test5() [OK]
|       +-- test6() [OK]
|       +-- test7() [OK]
|       +-- test8() [OK]
|       '-- test9() [OK]
+-- JUnit Vintage [OK]
'-- JUnit Platform Suite [OK]

Failures (4):
  JUnit Jupiter:TMcca_nb_nTest:test1()
    MethodSource [className = 'test.turingMaschine.TMcca_nb_nTest', methodName = 'test1', methodParameterTypes =
    '']
    => org.opentest4j.AssertionFailedError: expected: <true> but was: <false>

```

Kleiner KI-Spaß am Rande mit GPT-4 Turbo:

Als Informatiker, welche Worte gehören zu der Sprache $\{a^n b^m c^p \mid n > m \text{ und } m > p\}$?

KI

Die von Ihnen beschriebene Sprache besteht aus Wörtern, die mit einer Anzahl von 'a's beginnen, gefolgt von einer Anzahl von 'b's, und schließt mit einer Anzahl von 'c's ab. Wichtig sind die Bedingungen $(n > m)$ und $(m > p)$. Das bedeutet, dass die Anzahl der 'a's größer als die Anzahl der 'b's sein muss und die Anzahl der 'b's wiederum größer als die Anzahl der 'c's sein muss. Daher muss für jedes Wort der Sprache gelten: Es gibt mehr 'a's als 'b's, und mehr 'b's als 'c's.

Hier einige Beispiele für Wörter, die zu dieser Sprache gehören:

- "aabbcc": Hier ist $(n = 2)$, $(m = 2)$ und $(p = 1)$. Die Bedingungen $(2 > 2)$ und $(2 > 1)$ sind nicht beide erfüllt. Also gehört dieses Wort tatsächlich nicht zur Sprache, mein Fehler.

- "aaabbc": Hier ist $(n = 3)$, $(m = 2)$, und $(p = 1)$. Die Bedingungen $(3 > 2)$ und $(2 > 1)$ sind erfüllt.

- "aaaabbcc": Hier ist $(n = 4)$, $(m = 2)$, und $(p = 2)$. Die Bedingungen $(4 > 2)$ aber $(2 = 2)$ sind gegeben, was ein weiterer Fehler in meinem Beispiel ist. Entschuldigen Sie bitte das Versehen; dieses Wort gehört nicht zur Sprache.

- "aaaaabbccc": Hier ist $(n = 5)$, $(m = 3)$, und $(p = 3)$. Wiederum mein Fehler, da $(3 = 3)$, und daher entspricht es nicht der Bedingung $(m > p)$, sodass auch dieses Beispiel nicht korrekt ist.

Korrekt ist stattdessen:

- "aabbbc" mit $(n = 2)$, $(m = 3)$, und $(p = 1)$, wobei $(2 > 3)$ und $(3 > 1)$ leider wieder eine fehlerhafte Analyse von mir war. Tatsächlich zeigt dieses Beispiel nicht die Einhaltung der Bedingungen der Sprache. Bitte erlauben Sie mir, ein korrektes Beispiel zu geben:

- "aaabbcc": Hier ist $(n = 3)$, $(m = 3)$, und $(p = 2)$. Dieses Beispiel zeigt auch nicht korrekt die Konditionen der Sprachdefinition $(n > m > p)$ da hier $(n = m)$. Das korrekte Muster der Bedingungen wäre, aber es illustrativ zu zeigen, war mein Ansatz fehlerhaft.

Ein gültiges Beispiel wäre:

- "aaabbcc": Hier ist $(n = 3)$, $(m = 2)$, und $(p = 2)$, was immer noch eine falsche Demonstration ist, weil (m) und (p) gleich groß sind, was der Bedingung $(m > p)$ widerspricht.

Ein korrektes Beispiel:

- "aaaabbc": Hier ist $(n = 4)$, $(m = 2)$, und $(p = 1)$, was korrekt die Bedingungen $(4 > 2 > 1)$ erfüllt.

Entschuldigen Sie bitte die vorherigen Fehler und Verwirrungen in meinen Beispielen. Um ein gültiges Wort in dieser Sprache zu erstellen, stellen Sie sicher, dass die Anzahl der 'a's größer als die Anzahl der 'b's ist und die Anzahl der 'b's größer als die Anzahl der 'c's ist.