Prof. T. Nipkow
M. Eberl, J. Rädle, S. Wimmer
Technische Universität München
PROGRAMMIERABGABE **4 Punkte**
Abgabefrist: Montag, 03.06.2019, 14 Uhr

Einführung in die theoretische Informatik

Sommersemester 2019 – Hausaufgabenblatt 5

Programmierabgabe

Für die Korrektur Ihrer Programmierabgaben wird TUMjudge(`https://judge.in.tum.de/theo/public/`) verwendet. Auf dem ersten Hausaufgabenblatt wurden die für Sie relevanten Funktionen des Webinterfaces erklärt. Weitere Details finden Sie auf den Folien des Praktikums Algorithms for Programming Contests von 2017(`https://www7.in.tum.de/um/courses/theo/ss2018/materials/judge.pdf`).

AUFGABE 5.1. (*Automata Tutor*)

2 Punkte

Im Automata Tutor finden Sie wieder Übungsaufgaben sowie bepunktete Hausaufgaben. Beachten Sie, dass Sie für diese Aufgaben weiterhin nur fünf Versuche haben. Die Aufgaben finden Sie unter der Kategorie "Equivalence Classes".

AUFGABE 5.2. (*NFA* $\rightarrow$ *RE*)

1 Punkte

In dieser Aufgabe implementieren Sie die Prozedur aus Satz 3.19, um NFAs in reguläre Ausdrücke umzuwandeln. Um Ihnen eine eventuelle Fehlersuche zu erleichtern, sind Aufgabe und Templates so angelegt, dass *alle* Zwischenergebnisse auch ausgegeben und vom Testsystem überprüft werden.

- Lesen Sie sich die PDF-Angabe zu der Aufgabe 'NFAToRegex' durch (zu finden unter 'course/problems' auf TUMjudge).
- Beachten Sie, dass sich die Implementierung von regulären Ausdrücken in den Templates seit Blatt 3 geändert hat. Sie sollen reguläre Ausdrücken aufbauen mithilfe der Smart-Konstruktoren `Regex.empty()`, `Regex.concat()`, `Regex.concats()` etc.
- Laden Sie das Codegerüst für die Hausaufgabe auf `https://www21.in.tum.de/teaching/theo/SS19/materials/ha/05/templates_ha05.tar.gz` herunter. Dort finden Sie auch graphische Versionen der Beispiel-Eingabe-NFAs als PDFs.
- Betrachten Sie die Klasse `NFAToRegex` und die drei mit `TODO` markierten Methoden darin.
- Rekapitulieren Sie die entsprechenden drei Schritte (Initialisierung der Tabelle α_{ij}^0, Schritt $\alpha_{ij}^k \rightarrow \alpha_{ij}^{k+1}$ und Ablesen des Ergebnisses aus der letzten Tabelle α_{ij}^n und implementieren Sie sie in den entsprechenden drei Methoden.
- Laden Sie dann für Problem A (NFAToRegex) alle benötigten Dateien hoch – falls Sie eines unserer Templates benutzen, müssen Sie *alle* Dateien des Templates hochladen, nicht nur die, die Sie verändert haben.

AUFGABE 5.3. (*Aufzählen der Wörter einer kontextfreien Grammatik*)

1 Punkt

In dieser Aufgabe implementieren bekommen Sie eine kontextfreie Grammatik gegeben und sollen alle erzeugten Wörter der Länge maximal n ausgeben.

- Lesen Sie sich die PDF-Angabe zu der Aufgabe 'EnumerateCSG' durch (zu finden unter 'course/problems' auf TUMjudge)
- Laden Sie das Codegerüst für die Hausaufgabe auf `https://www21.in.tum.de/teaching/theo/SS19/materials/ha/05/templates_ha05.tar.gz` herunter.
- Betrachten Sie die Klassen `Production` und `ContextSensitiveGrammar`.
- Implementieren Sie die mit `TODO` markierten Methoden in diesen Klassen.
- Laden Sie dann für Problem B (EnumerateCSG) alle benötigten Dateien hoch – falls Sie eines unserer Templates benutzen, müssen Sie *alle* Dateien des Templates hochladen, nicht nur die, die Sie verändert haben.

AUFGABE 5.4. (*Restsprachen und RE-Matching*)

1+1 Punkte

In dieser Aufgabe müssen Sie sich zuerst das Konzept von "Restsprachen" (siehe Angabe) erarbeiten und sich überlegen, wie Sie dies für reguläre Ausdrücke als rekursive Prozedur umsetzen können. Dann müssen Sie dies implementieren.

- Lesen Sie sich die PDF-Angabe zu der Aufgabe 'RemainderRegex' durch (zu finden unter 'course/problems' auf TUMjudge)
- Laden Sie das Codegerüst für die Hausaufgabe auf `https://www21.in.tum.de/teaching/theo/SS19/materials/ha/05/templates_ha05.tar.gz` herunter.
- Machen Sie sich mit der geänderten Repräsentation von regulären Ausdrücken in der Klasse `Regex` vertraut. Der Hauptunterschied ist, dass `Concat` und `Alternative` nun n-*äre* Operationen sind statt *binäre* und dass reguläre Ausdrücke jetzt mithilfe von *smart constructors* aufgebaut werden, die direkt beim Erzeugen des regulären Ausdrucks diesen bereits vereinfachen (sodass z. B. $ab\varepsilon|\emptyset$ direkt zu ab wird).

- Machen Sie sich ebenfalls mit dem Interface `Regex.Visitor` vertraut, das benutzt wird, um rekursive Prozeduren auf regulären Ausdrücken zu implementieren. Beispiele für die Verwendung dieses Visitors finden Sie ebenfalls in den Templates.
- Implementieren Sie den Visitor `RemainderRegex`, der die Restsprache zu dem gegebenen regulären Ausdruck berechnet.
- Laden Sie dann für Problem C (RemainderRegex) alle benötigten Dateien hoch – falls Sie eines unserer Templates benutzen, müssen Sie *alle* Dateien des Templates hochladen, nicht nur die, die Sie verändert haben.

Wenn Sie eine korrekte Lösung für Problem C erarbeitet haben, können Sie nun noch für den zweiten Punkt Aufgabe D bearbeiten. Hier sollen Sie die `remainderRegex`-Funktion aus Aufgabe C nutzen, um eine Prozedur zu implementieren, die entscheidet, ob ein regulärer Ausdruck ein bestimmtes Wort erkennt oder nicht.

- Lesen Sie sich die PDF-Angabe zu der Aufgabe 'RegexMatcher' durch (zu finden unter 'course/problems' auf TUMjudge)
- Laden Sie das Codegerüst für die Hausaufgabe auf `https://www21.in.tum.de/teaching/theo/SS19/materials/ha/05/templates_ha05.tar.gz` herunter.
- Ergänzen Sie die Klasse `RemainderRegex` um die Visitor-Implementierung, die Sie in Aufgabe C erarbeitet haben.
- Implementieren Sie die zwei restlichen mit `TODO` versehenen Methoden.
- Laden Sie dann für Problem D (RegexMatcher) alle benötigten Dateien hoch – falls Sie eines unserer Templates benutzen, müssen Sie *alle* Dateien des Templates hochladen, nicht nur die, die Sie verändert haben.