# Überraschend komplizierte Muster in Texten

**Reguläre Ausdrücke sind ein verbreiteter Mechanismus mit dem man leicht komplizierte Muster in Texten suchen kann, ohne programmieren zu müssen. Wenn man aber genauer hinsieht stellt sich heraus, dass reguläre Ausdrücke eigentlich auch nicht leichter zu verstehen sind als Computerprogramme.**

Auch wenn Computer heutzutage in fast jedem Haushalt zu finden sind, können doch nur die wenigsten Leute ihre Computer selbst programmieren. Das ist nicht weiter überraschend, denn Programmieren ist alles andere als einfach, und es zu lernen braucht viel Zeit. Mindestens so schwer wie das Schreiben von Programmen ist das Verstehen von Programmen, ganz besonders, wenn man sie nicht selbst geschrieben hat. Das liegt unter anderem daran, dass auch winzige Änderungen in einem Programm enorme Folgen haben können.

Um bestimmte häufige Aufgaben zu erleichtern, wurden verschiedene Vereinfachungsmechanismen entwickelt, so zum Beispiel Anfragesprachen wie SQL, mit denen die Arbeit mit Datenbanken vereinfacht wird, oder die in diesem Artikel behandelten regulären Ausdrücke, mit denen Muster in Texten beschrieben werden. Diese Muster werden gewöhnlich für Suchanfragen benutzt; statt ein Programm zu schreiben, das die Suche durchführt, schreibt man einen regulären Ausdruck und verwendet ein bereits bestehendes Suchprogramm oder die Suchfunktion der Textverarbeitung (die meisten modernen Textverarbeitungen unterstützen reguläre Ausdrücke). Der Vorteil von regulären Ausdrücken ist dabei, dass sich mit ihnen viele Suchanfragen relativ einfach schreiben lassen, wie man an dem folgenden Beispiel erkennen kann:

Angenommen, wir haben eine sehr, sehr große Textdatei, in der viele Rezepte stehen unsortiert hintereinander stehen. Unser Ziel ist es, darin alle Vorkommen der Wörter Apfelkuchen, Apfelmarmelade, Kirschkuchen und Kirschmarmelade zu finden. Nun können wir nicht einfach nach "Apfel" oder "Kirsch" suchen, da wir dann auch "Apfelwein" und "Kirschtomaten" finden würden. Statt mühsam nach allen vier Wörtern einzeln zu suchen, kann man stattdessen den regulären Ausdruck **`(Apfel|Kirsch)(kuchen|marmelade)`** verwenden, der die vier Suchbegriffe beschreibt.

Aber auch komplizierte Muster sind möglich, auch wenn sie für dieses Rezeptbeispiel nicht unbedingt sinnvoll sind. So findet man mit dem Ausdruck **`S\w*[aeiou][aeiou][aeiou]`** alle Wörter im Text, die mit einem S beginnen und drei aufeinanderfolgende Vokale enthalten (z.B. “Seeelefant” oder “Spreeaal”, wenn sie denn im Text vorkommen).

Auch wenn das Schreiben von regulären Ausdrücken ein wenig Übung erfordert, ist es doch in vielen Fällen einfacher als Programmieren. Daher werden reguläre Ausdrücke heute an vielen Stellen verwendet, an denen es um das Suchen von Mustern in Texten oder Dateien geht, insbesondere auch in der Bioinformatik oder in der Computerlinguistik. Viele Programmieranfänger haben daher schon

mit regulären Ausdrücken zu tun, bevor sie das erste Mal Programme in einer "richtigen" Programmiersprache schrieben.

Leider werden auch reguläre Ausdrücke schnell kompliziert und schwer zu lesen, was besonders dann ein Problem ist, wenn man bemerkt, dass der Ausdruck irgendwo einen Fehler enthält, den man reparieren möchte. Die dabei entstehende Frustration ist vielen bekannt und wird in diesem Zitat von Jamie Zawinski recht gut ausgedrückt: "Some people, when confronted with a problem, think 'I know, I'll use regular expressions.' Now they have two problems." (Übersetzt ungefähr: "Manche Leute stehen vor einem Problem und denken 'Ich hab's, ich benutze dafür reguläre Ausdrücke.' Jetzt haben sie zwei Probleme.")

In einem Kapitel meiner Doktorarbeit habe ich mich mit der Frage befasst, wie schwer das Verstehen von regulären Ausdrücken ist, und ob sich nicht eventuell Verfahren entwickeln lassen, die einem die Arbeit mit ihnen erleichtern. Dabei habe ich mit Methoden der theoretischen Informatik bewiesen, dass die in der Praxis verwendeten regulären Ausdrücke trotz ihrer eingeschränkten Natur ähnlich schwer wie Programme zu verstehen sind. Insbesondere heißt dass, das sie nicht zuverlässig optimiert werden können.

Eigentlich stammen reguläre Ausdrücke aus der theoretischen Informatik. Sie wurden 1956 von Stephen Kleene erfunden; das erste Suchprogramm für reguläre Ausdrücke wurde 1968 von Ken Thompson vorgestellt. Während die Theorie ihre regulären Ausdrücke nicht veränderte, sondern lieber ausführlich untersuchte (auch heute noch sind nicht alle Fragen dazu beantwortet), haben sich in der Praxis nach und nach Erweiterungen durchgesetzt.

Manche dieser Erweiterungen waren nur nützliche Abkürzungen für Schreibfaule, andere hingegen fügten zusätzliche Ausdrucksmöglichkeiten ein. Daher lassen sich mit diesen erweiterten Ausdrücken Muster beschreiben, die mit den "klassischen" Ausdrücken der Theoretiker nicht möglich sind, wie das Beispiel im nächsten Absatz. Die wichtigste dieser Erweiterungen sind Variablen (je nach Dialekt auch Rückreferenzen genannt), mit denen sich der Ausdruck Wörter speichern kann. Durch dieses Speichern können Wiederholungen ausgedrückt werden.

So findet man zum Beispiel mit dem Ausdruck `\b(?P<X>\w+) (?P=X)\b` alle Stellen im Text, an denen dasselbe Wort zweimal hintereinander vorkommt (ein recht häufiger Tippfehler). Ich verzichte hier auf eine Erklärung der Funktionsweise, allerdings merken wir uns, dass der Ausdruck eine einzige Variable namens X enthält, und dass dieses Muster ohne Variablen nicht ausgedrückt werden kann. (Leser mit Vorkenntnissen bemerken auch, dass der Ausdruck im Stil der Bibliothek von Python geschrieben ist; andere Dialekte würden das gleiche Muster leicht anders ausdrücken.)

Nahezu alle Resultate aus der Theorie beziehen sich auf klassische reguläre Ausdrücke (ohne Variablen), während die Praxis häufig praktische reguläre Ausdrücke (mit Variablen) verwendet. Da das praktische Modell komplizierter ist als das klassische, können die in der Theorie entwickelten

Verfahren nur dann in die Praxis übertragen werden, wenn man auf Variablen verzichtet. Viele Praktiker sind damit nicht zufrieden.

Nun ist die Frage, was es bedeutet, einen regulären Ausdruck zu verstehen, im Grunde eine sehr philosophische. Glücklicherweise können wir dieser Frage ausweichen, da schon ein Teilproblem schwer ist. In der theoretischen Informatik sind Probleme gewöhnlich Fragen wie die folgende:

"Äquivalenzproblem für praktische reguläre Ausdrücke.

*Eingabe:* Zwei praktische reguläre Ausdrücke A und B.

*Frage:* Beschreiben A und B die gleichen Muster?"

Eine Lösung für ein Problem ist ein Programm, das für alle Eingaben die Frage korrekt beantwortet. In der Theorie ist seit langem bekannt, dass das Äquivalenzproblem für klassische Ausdrücke lösbar ist. In meiner Dissertation habe ich bewiesen, dass keine solche Lösung für das Äquivalenzproblem für praktische reguläre Ausdrücke existieren kann. Das gilt sogar dann, wenn man die praktischen Ausdrücke nur sehr sparsam erweitert, nämlich mit einer einzigen Variable (der eigentliche Beweis ist ungefähr 60 mal so lang wie dieser Artikel).

Die Unlösbarkeit des Äquivalenzproblems  erklärt auch, warum es so schwer ist, praktische regulären Ausdrücken zu verstehen. Unabhängig davon wie man "Verstehen" genau definiert ist es doch eine plausible Forderung, dass das Verstehen der Funktion eines Ausdrucks einem auch ermöglichen muss, zu entscheiden, ob zwei Ausdrücke die gleiche Funktion haben. Wenn nun das Äquivalenzproblem nicht lösbar ist, dann bedeutet das, dass zwei sehr unterschiedliche Ausdrücke die gleichen Muster darstellen können, ohne dass man ihnen das ansieht. Insbesondere heißt es auch, dass es weder ein "Patentrezept", noch ein allgemeines Verfahren, noch ein Programm zum "Entwirren" von komplizierten praktischen regulären Ausdrücken geben kann.

Dass genau diese Schwierigkeiten bei Programmen auftreten ist schon seit langem bekannt; Alan Turing, der dieses Jahr 100 geworden wäre, hat die theoretischen Grundlagen dafür bereits 1936 gelegt. Wie bei den praktischen regulären Ausdrücken ist die Frage ob zwei Programme die gleiche Funktion berechnen ebenfalls nicht lösbar. Analog kann man aus der Nichtlösbarkeit schließen, dass Programme schwer zu verstehen sind, und ebenso, dass es kein Vereinfachungsprogramm geben kann, das unnötig komplizierte Programme in eine lesbarere oder wenigstens effizientere Form bringt.

Um diese Schwierigkeiten zu umgehen haben sich beim Programmieren vor allem zwei Ansätze durchgesetzt: Erstens ermutigt man Programmierer, nicht nur auf die Korrektheit ihrer Programme zu achten, sondern auch auf die Lesbarkeit. Zweitens schränkt man die Anforderungen an Hilfsprogramme zum Optimieren und Vereinfachen anderer Programme ein. Da exakte und optimale Optimierungen nicht garantiert werden können, verwenden diese Hilfsprogramme Faustregeln

("Heuristiken"), die in den meisten Fällen wie gewünscht arbeiten, in anderen Fällen aber zu schlechteren Programmen führen können.

Auch wenn sich reguläre Ausdrücke oft einfacher schreiben lassen als Programme, genügt eine einzige Variable, um mit genau den gleichen Schwierigkeiten und Problemen kämpfen zu müssen. Das ist aber kein Grund auf die Verwendung von Variablen zu verzichten; man sollte nur wissen, worauf man sich einlässt.