Platzhalter reloaded

Beginnen wir mit der Aufgabe zum Greedy-Problem aus dem vorausgehenden Kapitel. Gegeben ist eine horizontale Anordnung von Wörtern, von denen ausschließlich die Personalpronomen der zweiten Person Singular ausgewählt werden sollen:

d ich ihr dich mir ihn er uns wir sie euch du Ihnen Sie dir sie ihm andere dorthin

Der Ausdruck *d. {1,3}?_* markiert alle Vorkommen des Buchstaben *d* gefolgt von einem bis drei beliebigen Zeichen (auch Leerstellen oder Satzzeichen) - wobei aufgrund des *?* die jeweils kürzere Markierung einer längeren vorgezogen wird - wenn auf diese eine Leerstelle folgt. Markiert werden folglich *dich_*, *du_*, *dir_* und *(an)dere_*. Nun muss nur noch dere eliminiert werden, damit die Aufgabe erfüllt ist. Um die Ergebnisse zu verfeinern, ließe sich auch hier mit Positionsdefinitionen arbeiten. Alle gesuchten Pronomen zeichnen sich durch ein *d* am Wortanfang aus. Um ein Zeichen am Wortanfang zu verankern, steht uns der Positionsanker *|<* zur Verfügung (s. vorausgehendes Kapitel), mit dem wir den obigen Ausdruck zu *|<d.{1,3}?_* erweitern können. Da *dere* sich nicht am Wortanfang befindet, sondern Bestandteil des Wortes *andere* ist, wird es nicht markiert und wir erhalten die gewünschte Lösung *dich_*, *du_* und *dir_*.
Dieser Ausdruck funktioniert jedoch nur, wenn wir zum einen wissen, wie viele Buchstaben die einzelnen Wörter haben dürfen, zum anderen die Daten eine Übersichtlichkeit aufweisen, die es uns erlaubt, Störfaktoren abschätzen zu können. Ist dies nicht gegeben, macht es eventuell keinen Sinn, den Wiederholungsoperator durch einen Minimal- und einen Maximalwert wie in *{1,3}* zu definieren. Allerdings entfernen wir uns von der Lösung der vorliegenden Aufgabe, wenn wir *{1,3}* durch *** oder *+* ersetzen. Weder *+* noch *** scheinen ideal. *** bewirkt in *|<d.*?_*, dass das *d* am Zeilenanfang mit einbezogen wird, da * auch eine 0-malige Wiederholung des vorausgehenden Zeichens bedeuten kann. Ein *d* am Wortanfang gefolgt von null Zeichen vor einer Leerstelle trifft in unserem Beispiel auch auf das zeileneinleitende *d* zu. Das Ergebnis würde in diesem Fall *d_* (*d.{0}_*), *dich_* (*d.{3}_*), *du_* (*d.{1}_*) und *dir_* (*d.{2}_*) lauten.

Also "*|<d.+?_*"?
|<d.+?_ schreibtvor, dass zwischen dem *d* und der Leerstelle mindestens ein weiteres Zeichen vor der Leerstelle vorhanden sein muss. Das ist bei dem zeileneinleitenden *d* nicht der Fall, allerdings matcht der reguläre Ausdruck dennoch, da der Platzhalter . auch für Leerstellen stehen kann. Als Ergebnis erhalten wir hier *d_ ich_* (*d.{4}_*), *dich_* (*d.{3}_*), *du_* (*d.{1}_*) und *dir_* (*d.{2}_*). Dieses Problem wird von dem Wiederholungsoperator *{1,3}?* nur deshalb umgangen, da die maximale Wiederholungsanzahl auf 3 beschränkt wird. Bestünde das auf *d_* folgende Wort lediglich aus zwei Buchstaben (z.B. *du*), würde es in Form von *d_ du_* (*d.{3}_*) ebenfalls in die Ergebnisse aufgenommen werden.

Anders gesagt: Das Problem für *+* (und für *{min,max}*) darin, dass *.* auch für Leerstellen stehen kann.

Buchstaben, Ziffern, Leerräume

Das Inventar der regulären Ausdrücke enthält eine Vielzahl alternativer Platzhalter. In der Aufgabe des einleitenden Kapitels wurde darauf hingewiesen, dass sich einzelne Zeichenklassen voneinander unterscheiden. Zur Erinnerung:

	Datensatz A	**Datensatz B**	**Datensatz C**
2)	A	B	a
1)	G	A	1

Die Gemeinsamkeiten in 1 und 2 sind schnell erklärt. Während es sich in 1) bei Datensatz A (*G*) und Datensatz B (*A*) um Buchstaben handelt, besteht Datensatz C aus einer Ziffer (*1*).
Im zweiten Beispiel liegt der Unterschied zwischen den Datensätzen A (*A*) und B (*B*) auf der einen und C (*a*) auf der anderen Seite in der Groß- und Kleinschreibung der Buchstaben.

Die Syntax regulärer Ausdrücke umfasst gleich drei Möglichkeiten, Platzhalter näher zu bestimmen. Diese können entweder als eine Zeichenklasse (zum Beispiel: Großbuchstaben) gleichgesetzt oder aber auch als ein Zeichenset, welches sich entweder mit einer Zeichenklasse deckt oder aus mehreren Zeichen/ Zeichenklassen (zum Beispiel: Groß- und Kleinbuchstaben) besteht, definiert werden.
Zu beachten sei hier, dass die unterschiedlichen Methoden nicht von allen Regex-fähigen Programmen gleichermaßen unterstützt werden.

Von A bis Z, von 1 bis 9

Die erste Methode eignet sich vor allem, wenn man das mögliche Variationsspektrum der Platzhalter kennt. In eckigen Klammern wird ein Zeichenset definiert, der die möglichen Zeichen für diese Variable vorgibt. Zum Beispiel sucht *d[uieo]* nach einem *d* gefolgt von *u*, *i*, *e* oder *o*. Auch Platzhalter dieser Art lassen sich quantifizieren. So sucht *d[uieo]{2}* die Daten nach der Zeichenfolge *d[uieo][uieo]* ab und würde beispielsweise auch die Folge *die* finden, wenn sie in den Daten vorhanden wäre.
Die Zeichensets können aus Zeichen beliebiger Typen zusammengesetzt sein. *d[uieo135?]* untersucht die Daten auf ein *d*, welches einem *u*, *i*, *e*, *o*, *1*, *3*, 5 , einem Fragezeichen oder einer Leerstelle vorausgeht. Man beachte, dass das Fragezeichen in der Klammer nicht als Metazeichen (mit der Bedeutung "Wiederhole das letzte Zeichen mindestens ein Mal") interpretiert wird. Insgesamt gelten in den eckigen Klammern andere Metazeichen als außerhalb. Mehr dazu im nächsten Kapitel.
Die eckigen Klammern können auch ein Intervall enthalten, was durch ein "-" gekennzeichnet wird. Will man eine Variable für alle Kleinbuchstaben setzen, lautet der Ausdruck hierfür *[a-z]*, für alle Ziffern benutzt man *[1-9]*. Intervalle können auch kombiniert werden, indem man den Inhalt der Klammern vereint. Der Ausdruck *[a-zA-Z1-9]* matcht mit allen Kleinbuchstaben (*[a-z]*), allen Großbuchstaben (*[A-Z]*) und allen Ziffern (*[1-9]*).
Wenn nun zuvor die Rede von "alle (Klein- oder Groß-)Buchstaben" war, so stimmte das nicht ganz, denn nicht alle Buchstaben fallen in das Intervall *a-z* bzw. *A-Z*. Umlaute, mit einem Akzent versehene Buchstaben und sonstige Zeichen, die nicht im englischen Alphabet vorhanden sind, müssen in der eckigen Klammer explizit erwähnt werden. Will man ein Set für alle Großbuchstaben des Deutschen ein Set erstellen, müssen dem Intervall *A-Z* auch die Umlaute *Ä*, *Ö* und *Ü* hinzugefügt werden. Sets, die dem entsprechen, sind [A-ZÄÖÜ], [ÄÖÜA-Z], o.ä. .

Von alphas und digits

Eine alternative Möglichkeit der Zeichenklassendefinition bildet die Klammerschreibweise nach dem POSIX-Standard, der im Gegensatz zu dem oben beschriebenen Verfahren jedoch nicht von allen Regex-fähigen Programmen unterstützt wird. Auch hier wird der Platzhalter von zwei eckigen Klammern gerahmt. Innerhalb dieser können nun Zeichenklassen gelistet werden, indem sie in Form von *[[:zeichenklasse1:][:zeichenklasse2:][:zeichenklasse3:] etc.]* aneinander gereiht werden. Das Set *[a-zA-Z1-9]* ließe sich nach dieser Schreibweise als *[[:lower:][:upper:][:digit:]]* oder vereinfacht als *[[:alnum:]]* (Beinhaltet: alle Groß- und Kleinbuchstaben sowie Ziffern) umschreiben. Es sollte unbedingt darauf geachtet werden, dass eckige Klammern nicht nur jede einzelne Zeichenklasse umrahmen, sondern auch das gesamte durch Zeichenklassen definierte Set umfassen.

Und so sehen einige nützliche POSIX-Zeichenklassenkürzel aus:

[Platzhalter]	matcht...	entspricht...
[:lower:]	alle Kleinbuchstaben	[a-z]
[:upper:]	alle Großbuchstaben	[A-Z]
[:alpha:]	alle Buchstaben	[a-zA-Z]
[:digit:]	alle Ziffern	[1-9]
[:alnum:]	alle Buchstaben und Ziffern	[a-zA-Z1-9]
[:graph:]	alle sichtbaren Zeichen (d.h. keine Leerstellen)	[*Aneinanderreihung aller sichtbaren Zeichen*]
[:word:]	alle Buchstaben, Ziffern, Underscores	[a-zA-Z1-9_]
[:punct:]	alle Satzzeichen/ Symbole	[*Auflistung aller Zeichen und Symbole*]
[:blank:]	alle Leerzeichen und Tabs	[_\t]
[:space:]	alle Leerstellen und Zeilenumbrüche	[_\t\r\n\v\f]

Einige werden sich bereits gefragt haben, welche Funktion die Backslashs in der Alternativschreibweise der letzten beiden Zeilen der Tabelle erfüllen. Eine ausführliche Auseinandersetzung mit diesem Metazeichen erfolgt im nachfolgenden Kapitel. Im Zusammenhang mit Platzhaltern sei gesagt, dass es sich hierbei um sogenannte Shorthand Expressions handelt, welche anstelle bestimmter Zeichen oder Zeichenklassen treten.

Die Kurzversion

Wem die POSIX- oder die Intervallzeichensetdefinitionen zu lang und fehleranfällig sind, der kann alternativ auf einige Shorthand Expressions zurückgreifen. Die Shorthand-Methode besteht in der Kombination eines Backslashs mit einem einem begrenzten Inventar entnommenen Buchstaben, wobei auch eckige Klammern zur Rahmung gesetzt werden können. Ihr Vorteil liegt sicherlich in der Kürze des Ausdrucks. Allerdings werden auch sie nicht universell akzeptiert.

Die drei häufigsten Platzhalter, die in der Regel von allen Regex-fähigen Programmen unterstützt werden, sind:

Platzhalter	matcht...	entspricht...	[und...]
\w (oder [\w])	alle Buchstaben, Ziffern und Underscores	[a-zA-Z1-9_]	[:word:]
\d (oder [\d])	alle Ziffern (0-9)	[1-9]	[:digit:]
\s (oder [\s])	Alle Leerräume (inklusive Leerstellen, Tabs, Umbrüchen)	[_\t\r\n\v\f]	[:space:]

Weitere akzeptierte Shorthand Expressions (auf die Alternativschreibweise mit eckigen Klammern wird im Folgenden verzichtet):

Platzhalter	matcht...
\t	alle Tabs
\n	alle Zeilenumbrüche (Achtung: Letzte Zeile im Dokument endet nicht mit einem Umbruch!)

Weniger verbreitet und nicht immer akzeptiert sind dagegen:

Platzhalter	matcht...
\l	alle Kleinbuchstaben
\u	alle Großbuchstaben

Shorthand Expressions kommen ohne eckige Klammern aus, welche jedoch obligatorisch werden, will man mehrere Shorthand Expressions zu heterogenen Zeichensets kombinieren. *[\d\s]* fungiert zum Beispiel als Platzhalter für Ziffern und Leerräume und lässt sich mit *[[:digits:][:space:]]* gleichsetzen. Die Notwendigkeit der Klammersetzung lässt sich mit wenigen Worten erklären: Fehlen die Klammern, so werden *\d* und *\s* sukzessive abgearbeitet, was bedeutet, dass der Ausdruck genau dann matcht, wenn auf eine Ziffer ein Leerraum folgt.

Und genau das NICHT

Alle drei Schreibweisen bieten zudem die Möglichkeit Zeichenklassen aus dem Zeichenset "auszuklammern". In der Regel funktioniert dies mit einem "^" nach der seteröffnenden eckigen Klammer:

Platzhalter	matcht...
[^1-9]	alle Zeichen außer Ziffern (0-9)
[^[:digit:]]	alle Zeichen außer Ziffern (0-9)
[^\d]	alle Zeichen außer Ziffern (0-9)

Achtung: ^ negiert den Inhalt der gesamten Klammer. *[^135]* sucht nach allen Zeichen außer nach den Ziffern 1 3 und 5, *[^[:digit:][:punct:]]* nach allen Zeichen außer Ziffern und Interpunktionen, *[^\d\n]* nach allen Zeichen außer Ziffern und Zeilenumbrüchen. Befindet sich ^ hingegen nicht in Anfangsposition, so wird es nicht als Metazeichen, sondern als ein konkretes Zeichen gedeutet. So wird der Ausdruck *[\d\n^]* anders als *[^\d\n]* mit allen Ziffern, allen Zeilenumbrüchen und allen konkreten Zeichen ^ matchen.

Für die Shorthand Expressions existiert eine weitere Möglichkeit der Negation. Um eine solche zu erreichen kann der Kleinbuchstabe, welcher auf \ folgt, durch einen Großbuchstaben ersetzt werden:

Platzhalter	matcht...
\D	alle Zeichen außer Ziffern (0-9)

Fügt man einer negierten Shorthand Expression ein ^ hinzu, so wird das Set gleichbedeutend mit dem positiven Ausdruck:

Platzhalter	entspricht...
[^\D]	\d

Alles in einem

Die präsentierten Listen sind bei Weitem nicht komplett, was zum einen daran liegt, dass einige der ausgelassenen Platzhalter im geisteswissenschaftlichen Kontext wahrscheinlich nie zum Einsatz kommen werden, zum anderen daran, dass individuell benutzte Programme nicht gleichermaßen mit den unterschiedlichen Zeichensetdefinitionsmustern kompatibel sind. Welche Platzhalter als solche erkannt werden, lässt sich im Hilfebereich des jeweiligen Programms ermitteln.

Und was sollte das noch mal? Ausgegangen sind wir von dem Datenset

```
d ich ihr dich mir ihn er uns wir sie euch du Ihnen Sie dir sie ihm andere dorthin
```

und dem Problem, dass ein "."-Platzhalter auch für eine eine Leerstelle stehen kann, was bewirkte, dass *l<d.+?_* auch *d_ich_* (*d.{4}*) enthielt. Die hier dargelegten Konventionen der Zeichensetdefinition bieten eine Fülle an Möglichkeiten, dieses Problem zu umgehen. Im Prinzip dieses Datenset jeden Platzhalter, der auf Buchstaben, nicht jedoch auf Leerräume zutrifft. Ein paar Beispiele:

Regex	sucht...
\<d[[:word:]]+?_ \<d\w+?_	d am Wortanfang, gefolgt von mindestens einem Buchstaben/ einer Ziffer/ einem Underscore, dann von einer Leerstelle
\<d[a-z]+?_	d am Wortanfang, gefolgt von mindestens einem Kleinbuchstaben, dann von einer Leerstelle
\<d[^ _]+?_	d am Wortanfang, gefolgt von mindestens einem Zeichen, das mit Ausnahme einer Leerstelle alles sein kann, dann von einer Leerstelle
\<d[^\s]+?_ \<d[\S]+?_	d am Wortanfang, gefolgt von mindestens einem Zeichen, das mit Ausnahme eines Leerraums (auch nicht Tabs/ Umbrüche) alles sein kann, dann von einer Leerstelle

Vielleicht werden sich einige bereits folgende Frage gestellt haben: Aber was, wenn ich einen Platzhalter brauche, der tatsächlich gleichbedeutend sein soll mit einem Set, welches aus einem Backslash und einem Buchstaben besteht? Mehr dazu im nächsten Kapitel.

Zeichensets und Zeichenklassen

- Zeichenklassen (z.B. *\d*) kodieren unterschiedliche Typen von Zeichen und können zu Zeichensets (z.B. *[\d\s]*) kombiniert werden.
- Zeichensets können auch aus einzelnen Zeichen zusammengesetzt sein (z.B. *[A1ab]*)
- Zeichensets werden mithilfe eckiger Klammern markiert.
- Mit Ausnahme der Shorthand Expressions ist jede Zeichenklasse formal in ein Zeichenset eingebettet und wird somit von eckigen Klammern gerahmt.
- Auch Shorthand Expressions müssen in eckige Klammern gesetzt werden, sobald sie mehrere Zeichenklassen zu einem Set vereinen.

Previous Von null bis unendlich Up Reguläre Ausdrücke in der Arbeit mit Textdaten Escape Next