

Programmiersprachen II – Wintersemester 2011/2012
Übungsblatt 1
Besprechung der Aufgaben in der nächsten Übung am 22.12.2011.

1 Abgabe

Das Bearbeiten der Aufgaben ist für die Vorlesung Programmiersprachen 2 *nicht* verpflichtend. Eine freiwillige Abgabe ist möglich unter: John.Witulski@uni-duesseldorf.de. Termin für die Besprechung ist Donnerstag der 22.12.2011 14:30 in 25.12.02.55.

2 Aufgaben

Aufgabe 1 (Parsing mit DCGs)

Wiederholen Sie kurz, was Sie zu DCGs gelernt haben. Gegeben sei folgende Grammatik:

```
BExpr -> Basic
BExpr -> Basic or BExpr
BExpr -> Basic and BExpr
BExpr -> not BExpr
Basic -> true | false
```

- a) Schreiben Sie mit Hilfe von DCGs einen Parser `parse/1` für diese Grammatik. Bauen Sie noch keinen Syntaxbaum auf. Überprüfen Sie lediglich, ob die Grammatik eingehalten wird.

Beispiel:

```
?- parse("true and false").
yes

?- parse("false or true and false").
yes
```

- b) Sie haben in der Vorlesung gesehen, wie man mit Hilfe einer DCG einen Syntaxbaum aufbauen kann. Benutzen Sie die Strukturen `const(true)` bzw. `const(false)`, `or(A,B)`, `and(A,B)` und `not(A)` für den Syntaxbaum. Erweitern Sie nun den Parser aus a) so, dass dieser einen zweiten Parameter annimmt, der den Syntaxbaum zurückliefert.

Beispiel:

```
?- parse("true and false", Tree).
Tree = and(const(true), const(false)) ?
yes
```

- c) Benutzen Sie nun den Interpreter für boolesche Ausdrücke¹, um diesen Parser so zu erweitern, dass er den booleschen Ausdruck auflöst. Schreiben Sie dazu ein neues Prädikat `solve/1`.

Beispiel:

¹http://www.stups.uni-duesseldorf.de/sites/files/143/5_prop_int_neg.pl

```

?- solve("true and false").
no
?- solve("true or false and true").
yes

```

d) Erweitern Sie Ihr Programm um die logische Funktion `xor/2`.

Aufgabe 2 (Differenzlisten)

Wie Sie in der Vorlesung gesehen haben, können Differenzlisten die Effizienz von Listenoperationen enorm erhöhen.

a) Welche der folgenden Differenzlisten repräsentieren die gleiche Liste?

- X = [a,b|[]]-[]
- X = [a,b,c]-[c]
- X = [a,b,c]-[c,d]
- X = [a,b|R1]-[R1]
- X = [a,b,c,d,e,f|R]-[d,e,f|R]

b) Schreiben Sie ein Prädikat `toDL/3`, das aus einer normalen Liste eine Differenzliste erzeugt. Das erste Argument ist die normale Liste, die beiden weiteren Argumente stellen die Differenzliste dar.

Beispiel:

```

?- toDL([a,b,c], X, R).
X = [a,b,c|R]
yes
?- toDL([a,b],[a,b,c],[c]).
yes
?- toDL([a,b],X,[c,d]).
X = [a,b,c,d] ?
yes

```

c) Schreiben Sie `dlconcat/3`, das zwei Differenzlisten, die durch `X-Y` repräsentiert werden, konkateniert. Beispiel:

```

?- dlconcat([a,b,c|A]-A,[d,e,f|B]-B,List).
A = [d,e,f|B],
List = [a,b,c,d,e,f|B]-B ?
yes

```

Aufgabe 3 (Operatoren)

Beim Skat werden die Karten zuerst nach der Farbe sortiert: `Karo`, `Herz`, `Pik`, `Kreuz`. Danach werden die Bilder der Karten sortiert: `7`, `8`, `9`, `Bube`, `Dame`, `König`, `10`, `Ass`. Repräsentieren Sie die Karten in Prolog mit den Fakt `card(Farbe, Bild)`, z.B.: `card(kreuz, bube)`.

a) Definieren Sie diese Reihenfolgen in Prolog mit Hilfe von zwei Operatoren (Für Bild und Farbe getrennt). Z.B.: `7 >>> 8`. Benutzen Sie das Prolog-BuiltIn `op/3`, um die Operatoren zu definieren. Sie finden hier² eine Anleitung dazu.

²<http://www.amzi.com/AdventureInProlog/>

- b) Schreiben Sie nun ein Prädikat `cardsort/2`, welches eine Liste von Karten erhält und diese Liste nach den oben definierten Kriterien sortiert und als zweites Argument zurückgibt.
Beispiel:

```
?- cardsort([card(herz,dame),card(pik,10),card(pik,7),card(karo,ass)], S).  
S = [card(karo,ass),card(herz,dame),card(pik,7),card(pik,10)] ?  
yes
```

Tipp: Sortieren Sie erst nach den Bild und dann nach der Farbe oder umgekehrt. Es kann auch sehr hilfreich sein, zuerst ein Prädikat zu schreiben, welches das Minimum einer Liste findet.

Aufgabe 4 (Reguläre Ausdrücke)

Reguläre Ausdrücke finden sich in vielen Sprachen wieder. Leider gibt es daher viele verschiedene Ansätze. Die in dieser Aufgabe verwendete Syntax lehnt sich an die PCRE³ an. Schreiben Sie einen Parser für Suchmuster regulärer Ausdrücke. Betrachten Sie dabei folgende Teilmenge der PCRE-Suchmuster:

1. Ein Suchmuster beginnt und endet immer mit den Zeichen `"/"`
2. Ein Suchmuster darf folgende Zeichen verwenden: Gross- und Kleinbuchstaben, Zahlen, `" "`, `"."`, `"*"`, `"+"`, `"?"`, `"("`, `)"`, `"["` und `"]"`
3. Innerhalb der runden Klammern darf ebenfalls der Alternativ-Operator verwendet werden
`"|"`
4. Innerhalb der eckigen Klammern darf der Intervall-Operator `"-"` zwischen zwei Zeichen und das Komplement `"^"` (nur direkt hinter `"["`) verwendet werden
5. Der `"."` ist ein Sonderzeichen und kann durch jedes beliebige andere Zeichen ersetzt werden
6. Es steht Ihnen frei jede weitere Anpassung an den Parser vorzunehmen, solange Sie sich dabei an die PCRE Syntax halten

Beispiele:

```
?- parse("/[a-zA-Z]+strasse nr.[0-9]+/").  
yes  
?- parse("/[-a]/").  
no  
?- parse("/(19|20)[0-9][0-9]/").  
yes
```

Tipp: Überlegen Sie sich zunächst eine Grammatik.
Auf folgender Seite finden Sie eine umfangreiche Übersicht aller PCRE Regeln:
<http://www.php.net/manual/de/reference.pcre.pattern.syntax.php>

³Siehe auch <http://de.wikipedia.org/wiki/PCRE>