Computerlinguistische Algorithmen SS 00 4. Übung (11. Mai 2000)

NP-Automaten

In dieser Aufgabe geht es um die Anpassung vorhandener Funktionen und deren Anwendung auf die Phrasenerkennung. Die Verarbeitung erfolgt in zwei Schritten:

- Aus einer Datei wird zeilenweise Text gelesen (mit `read-line`), mittels eines spezialisierten *deterministischen* Automaten in einzelne Wörter zerlegt (*tokenization*) und gleichzeitig (anhand eines Lexikons) bezüglich syntaktischer Kategorien analysiert (*tagging*); das Ergebnis der ersten Verarbeitungsebene ist eine Folge (d.h. Liste) von `(`*`wort`*` . `*`tag`*`)` Paaren; der Einfachheit halber nehmen wir an, daß jedem Wort eine eindeutige Kategorie zugewiesen wird;
- die zweite Verarbeitungsebene verwendet einen Automaten, der anhand einer Menge von Mustern der Form

  `(NP PIDAT NP)` (für eine NP bestehend aus PIDAT und NN)

  Phrasen erkennt und alle erkannten Muster für jede Position der Eingabe auflistet; für die Phrasenerkennung werden dabei nur noch die syntaktischen Kategorien, nicht die eigentlichen Wörter, verwendet.

Für den Satz

`Moegen Puristen aller Musikbereiche auch die Nase rumpfen` [...]

ist das Ergebnis der ersten Verarbeitungsebene etwa

```
(("Moegen" . VMFIN)
 ("Puristen" . NN)
 ("aller" . PIDAT)
 ("Musikbereiche" . NN)
 ("auch" . ADV)
 ("die" . ART)
 ("Nase" . NN)
 ("ruempfen" . VVINF))
```

und das Ergebnis der Phrasenerkennung (der Kommentar hinter dem '—' gehört nicht zum Ergebnis):

```
(
 nil                             — Position 0 ('Mögen'): keine Phrase gefunden;
 nil                             — Position 1 ('Puristen): keine Phrase gefunden;
 ((NP PIDAT NN))                 — Position 2 ('aller): eine NP gefunden;
 nil                             — Position 3 ('Musikbereiche'): keine Phrase gefunden;
 ((NP ADV ART) (NP ADV ART NN))  — Position 4 ('auch'): zwei NPs gefunden;
 ((NP ART NN))                   — Position 5 ('die'): eine NP gefunden;
 nil                             — Position 6 ('Nase'): keine Phrase gefunden;
 nil                             — Position 7 ('ruempfen'): keine Phrase gefunden.
)
```

Natürlich ist es möglich, daß an einer Position der Eingabe mehrere Phrasen(anfänge) gefunden werden. Im Beispiel feuert an Position 4 sowohl das Muster `(NP ADV ART)` ('auch die') als auch `(NP ADV ART NN)` ('auch die Nase'); daher ist das Ergebnis eine Liste von Listen (von erkannten Phrasen).

Die verwendeten Kategorien sind dem STUTTGART-TÜBINGEN-TAGSET entnommen:

- `http://www.dfki.de/~becker/lispkurs/vorlesungen.ss00/STTS`

Lexikon und Phrasenmuster liegen vor in

- `http://www.dfki.de/~becker/lispkurs/vorlesungen.ss00/lexicon.lisp`
- `http://www.dfki.de/~becker/lispkurs/vorlesungen.ss00/patterns.lisp`

1 Zerlegung der Eingabe (2 + 8 + 2 Punkte)

a) Falls nötig, modifiziere die Funktion `stringlist2table` (vom letzten Übungsblatt) so, daß der erzeugte Automat *deterministisch* ist; dazu ist es nötig, bei der Konstruktion der Übergangstabelle zu überprüfen, ob ein Präfix des aktuellen Wortes bereits eingetragen ist, und neue Übergänge nur wo nötig zu erzeugen;

b) Schreibe eine zweistellige Funktion `tokenize`, die ebenso wie `traverse` einen Automaten (das Lexikon) und eine Zeichenkette als Argumente nimmt; `tokenize` unterscheidet sich von der deterministischen `traverse` Variante wie folgt:

   – für Zeichen, die nicht zum Alphabet gehören (`alphanumericp` = `token2code` = `nil`), bricht `tokenize` nicht sofort ab, sondern geht stattdessen in den Eingangszustand zurück;
   – so werden Zeichen, die nicht in Wörtern vorkommen können (z.B. Satzzeichen und Wortzwischenräume), überlesen; folgen mehrere nicht alphanumerische Zeichen hintereinander, bleibt der Automat im Eingangszustand;
   – zusätzlich prüft `tokenize` am Wortende, ob ein Endzustand erreicht wurde, und sammelt alle Endzustände auf, die an einem Wortende liegen;
   – `tokenize` arbeitet die Eingabe *immer* vollständig (d.h. bis zum Ende der Zeichenkette) ab.

   Das Verhalten von `tokenize` entspricht einer Erweiterung des Lexikonautomaten um *default* Übergange für unbekannte Eingabesymbole; diese *default* Übergange führen aus jedem Zustand (inklusive des Eingangszustands) zurück in den Eingangszustand, komsumieren ein Zeichen und sammeln gegebenenfalls den Wert eines erreichten Endzustands auf.

c) Lies die Spezifikationen von `with-open-file` und `read-line` in

   `http://www.dfki.de/~becker/lispkurs/clm/new-index.html`

   und integriere Dein `tokenize` mit dem Funktionsrumpf in

   `http://www.dfki.de/~becker/lispkurs/vorlesungen.ss00/io.lisp`

   so daß das oben gezeigte Ergebnis der ersten Verarbeitungsebene (als Liste von tokenisierten Sätzen) zurückgegeben wird.

2 Matching (5 + 3 Punkte)

a) Schreibe eine (nicht deterministische und exhaustive) `traverse` Variante `match-and-collect`, die

   – die Ausgabe der Tokenisierung (d.h. eine Liste von (*wort* . *tag*) Paaren) als Eingabe akzeptiert; *wort* wird bei der Phrasenerkennung nunmehr ignoriert;
   – einen mit `patterns2table` (letztes Übungsblatt) erzeugten Automaten verwendet, um *tag* Sequenzen zu erkennen, die einem bekannten Phrasenmuster entsprechen;
   – alle beim Erkennen eines Musters passierten Endzustände aufsammelt, also im obigen Beispiel für die *tag* Folge ADV ART NN beide anwendbaren Muster.

b) Benutze die Hilfsfunktionen, um die Funktion `match` zu schreiben, die als Eingabe ebenfalls einen Muster-Automaten und einen tokenisierten Satz erhält und als Ausgabe alle in dem Satz erkannten Muster zurückgibt.

   Ein naheliegender Ansatz ist beispielsweise, `match-and-collect` für jede Position eines Eingabesatzes aufzurufen (d.h. für jedes Eingabeelement neu zu starten) und aus den Einzelergebnissen das Gesamtergebnis zusammenzusetzen.

   Erprobe `match` mit einem kleinen Auszug aus der FRANKFURTER RUNDSCHAU:

   `http://www.dfki.de/~becker/lispkurs/vorlesungen.ss00/text`

Abgabetermin ist Freitag, der 19. Mai 2000