### 6.1.5. Der Pattern-Matchern als Ersetzungsgrammatik-Akzeptor (d'oh)

Auf S. 60 wurde bereits ausgesagt, dass das zentrale Problem des Pattern-Matchers darin liegt, dass sich die Muster exklusiv auf die lineare Anordnung von (Elementen von) Lexemklassen beziehen und diese durchgehend handgestrickt sind. Was daran verkehrt ist? Nun, die Verwendung von Lexemklassen statt konkreter Wörter bzw. Wortformen stellt wohl eine gewisse Generalisierung dar, besonders weit aber kommt man damit nicht. So kann beispielsweise das rekursive Potential von Sprache (wie es sich ua. in der folgenden Konstruktion manifestiert: *a tree in a garden near a house beside a lake*) in der Grammatik nicht abgebildet werden. Außerdem wird im Pattern-Matcher überhaupt nicht deutlich, dass z.B. die folgenden Konstruktionen strukturell betrachtet große Analogien aufweisen:

*I know that* { *the cat is hungry* / *the black cat is hungry* / *the black cat is very hungry* }

Hier haben wir es jeweils mit einem eingebetteten Satz zu tun, da uns aber im Pattern-Matcher das Konstrukt 'Satz' genau so wenig zur Verfügung steht wie die Konstrukte NP, VP usw., hätte jede der oa. Formen ein eigenes Muster. Das zentrale Anliegen dieses Abschnittes ist, zu zeigen, wie der Pattern-Matcher um syntaktische Kategorien, sprich Konstituentenklassen, erweitert werden kann dahingehend, dass sich die Muster aus den Regeln für diese syntaktischen Kategorien ergeben.

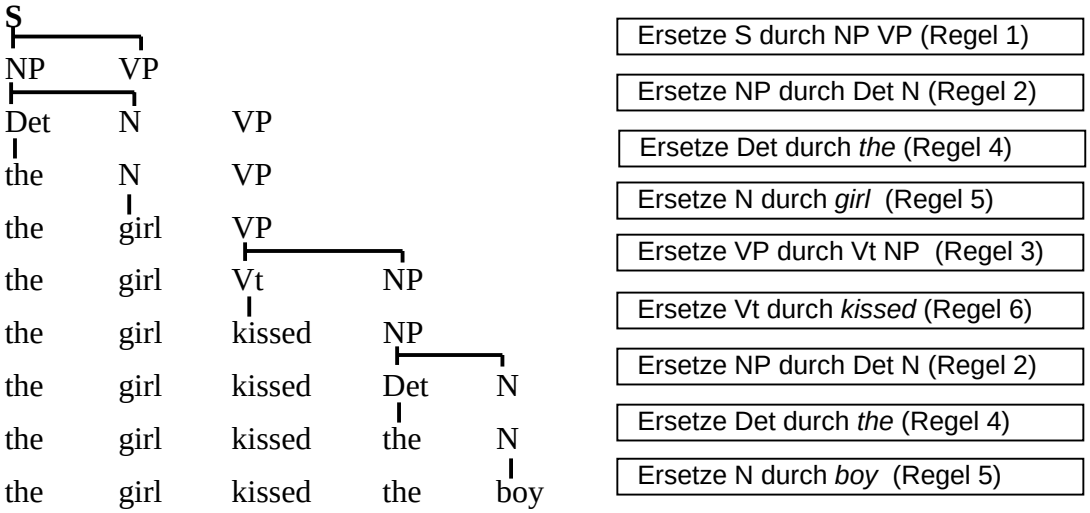
#### *6.1.5.1. Phrasenstrukturregeln als Ersetzungsregeln*

Eine der interessanten Fragestellungen bei der Beurteilung einer (modernen, formalen) Grammatik ist die Frage danach, wie man eigentlich von der Grammatik (die sich z.B. aus einem Lexikon und einer Regelkomponente konstituiert) **genau** zu der Aussage kommt, ein Satz sei – mit Bezug auf die Grammatik – wohlgeformt oder nicht.[28] Eine Möglichkeit dafür, diese Ableitung transparent zu machen, besteht darin, Phrasenstruktregeln als Ersetzungsregeln zu interpretieren. Eine Regel wie NP → Det N ist dann eine Aussage, die besagt, dass in einer Symbolkette überall dort, wo das Symbol NP auftritt, dieses durch die Symbolfolge Det–N substituiert werden kann. Betrachten wir dazu die folgende Grammatik:

| **PS-Regeln** | **Lexikon** |
|---|---|
| 1. S → NP VP | 4. Det → *the* |
| 2. NP → Det N | 5. N → *boy, girl* |
| 3. VP → Vt NP | 6. Vt → *kissed* |

Ausgehend von dem Startsymbol 'S' kann die Kette *the girl kissed the boy* über die folgende Ableitung als wohlgeformt (mit Bezug auf die Grammatik) identifiziert werden. Dabei ist zu berücksichtigen, dass

1. die Kette von links nach rechts abgearbeitet und
2. pro Ableitungsschritt nur ein Symbol ersetzt wird:




[28] Auf diesen interessanten Punkt werden später im Zusammenhang mit der Frage 'wo kommen die Strukturbeschreibungen eigentlich **genau** her' noch detaillierter eingehen.

### *6.1.5.2. Die Umsetzung in Prolog*

Das Ganze wird hier deshalb so explizit aufgeführt, weil sich für uns natürlich die Frage stellt, wie man so etwas in Prolog umsetzen kann. Entgegen der gängigen Praxis in diesem Skript wird dieser Punkt in den nächsten Abschnitten relativ straight-forward angegangen, denn eine umgangsprachliche Formulierung der Aufgabenstellung würde wahrscheinlich mehr verwirren als erklären.

Wir schreiben ein Prädikat match/2, welches eine Wortliste auf eine 'Ableitungsliste' abbildet und gehen von der folgenden Anfrage aus:

**?- match([the,boy,sleeps],[s]).**

Umgangssprachlich: kann die Liste [the,boy,sleeps] auf die Liste [s] abgebildet werden; noch umgangssprachlicher: ist die Kette *the boy sleeps* ein Satz? Bei dieser Anfrage passiert folgendes:

| Wortliste | Ableitungsliste | Aktion | Wissensbasis |
|---|---|---|---|
| **[the,boy,sleeps]** | **[s]** | | |
| | → | Prüfe, ob das erste Element der Wortkette mit dem Kopf der Ableitungliste einen Lexikoneintrag bildet: | FAIL! (Keine Regel s→ the) |
| | | Wenn nicht: suche nach einer PS-Regel, in der das erste Element der Ableitungsliste auf der linken Seite auftritt: | OK! (Regel s → np vp) |
| | | Ersetze das Symbol in der Ableitungskette durch das/die Symbol/e auf der rechten Seite der PS-Regel. Das Ergebnis ist ein neues Listenpaar. | |
| **[the,boy,sleeps]** | **[np,vp]** | ← | |
| | → | Prüfe, ob das erste Element der Wortkette mit dem Kopf der Ableitungsliste einen Lexikoneintrag bildet: | FAIL! (Keine Regel np→ the) |
| | | Wenn nicht: suche nach einer PS-Regel, in der das erste Element der Ableitungsliste auf der linken Seite auftritt: | OK! (np → det noun) |
| | | Ersetze das Symbol in der Ableitungsliste durch das/die Symbol/e auf der rechten Seite der PS-Regel. Das Ergebnis ist ein neues Listenpaar. | |
| **[the,boy,sleeps]** | **[det,noun,vp]** | ← | |
| | → | Prüfe, ob das erste Element der Wortkette mit dem Kopf der Ableitungsliste einen Lexikoneintrag bildet: | OK! (det → the) |
| | | Schneide in beiden Listen den Kopf ab und mache mit den Listenresten weiter. Das Ergebnis ist ein neues Listenpaar. | |
| **[boy,sleeps]** | **[noun,vp]** | ← | |
| | → | Prüfe, ob das erste Element der Wortkette mit dem Kopf der Ableitungliste einen Lexikoneintrag bildet: | OK! (noun → boy) |
| | | Schneide in beiden Listen den Kopf ab und mache mit den Listenresten weiter. Das Ergebnis ist ein neues Listenpaar. | |
| **[sleeps]** | **[vp]** | ← | |
| | → | Prüfe, ob das erste Element der Wortkette mit dem Kopf der Ableitungliste einen Lexikoneintrag bildet: | FAIL! (Keine Regel vp → sleeps) |
| | | Wenn nicht: suche nach einer PS-Regel, in der das erste Element der Ableitungsliste auf der linken Seite auftritt: | OK! (vp → vi) |
| | | Ersetze das Symbol in der Ableitungsliste durch das/die Symbol/e auf der rechten Seite der PS-Regel. Das Ergebnis ist ein neues Listenpaar. | |
| **[sleeps]** | **[vi]** | ← | |
| | → | Prüfe, ob das erste Element der Wortkette mit dem Kopf der Ableitungliste einen Lexikoneintrag bildet. | OK! (vi → sleeps) |
| | | Schneide in beiden Listen den Kopf ab und mache mit den Listenresten weiter. Das Ergebnis ist ein neues Listenpaar: | |
| **[ ]** | **[ ]** | ← | |
| | → | Wenn die Listen leer sind: **TRUE** (Beende die Prozedur) | |
| Ergebnis: **YES** | | | |

Wie die Aktionsspalte zeigt, müssen bei der Abarbeitung von Wort- und Ableitungsliste zwei Fälle klar voneinander unterschieden werden:

- der Kopf der Wortliste und der Kopf der Ableitungsliste sind über einen Lexikoneintrag zueinander in Beziehung gesetzt – dieser Fall tritt logischerweise erst dann ein, wenn der Kopf der Ableitungsliste eine lexikalische, keine syntaktische Kategorie ist
- der Kopf der Wortliste und der Kopf der Ableitungsliste sind nicht über einen Lexikoneintrag zueinander in Beziehung gesetzt – dieser Fall ist immer dann gegeben, wenn es sich bei dem Kopf der Ableitungsliste um eine syntaktische Kategorie handelt.

Es geht letzendlich um nichts anderes, als – ausgehend von einem Startsymbol 'S' – in der Ableitungsliste schrittweise die Ersetzungen vorzunehmen, die in den PS-Regeln vorgegeben sind, und zwar solange, bis die Ebene der lexikalischen Kategorien erreicht ist. Wenn wir uns die Spalte der Ableitungsliste ansehen, stellen wir - beispielsweise beim Übergang der Liste [np, vp] zur Liste [det, noun, vp] einen wichtigen Aspekt dieser Ersetzungen fest, nämlich die Tatsache, dass ja immer nur ein Symbol zur Zeit substituiert wird, nämlich der **Kopf** der Ableitungsliste (remember: die Abarbeitung erfolgt von links nach rechts), hier also das Symbol 'np' durch die Symbolfolge 'det noun', während der **Rest** der Ableitungsliste (hier: das Symbol 'vp') in die neue Ableitungsliste übernommen werden muss. Die neue Ableitungsliste ist mithin die Kombination aus

1. dem Symbol / der Symbolfolge, durch die der Kopf der alten Ableitungsliste laut einer PS-Regel substituiert wird
2. dem Rest der alten Ableitungsliste

Das müssen wir für die Umsetzung unbedingt im Hinterkopf behalten!

Kommen wir jetzt zur Realisierung in Prolog. Unser Programm wird erneut drei Komponenten umfassen, nämlich ein Lexikon, eine Reihe von PS-Regeln (anstatt der handgestrickten Muster) und die Steuerdatei, die die Abarbeitung einer vom Benutzer eingegebenen Wortliste regelt.

#### *6.1.5.3. Das Lexikon*

Das Lexikon wird in Form der altbekannten lex/2-Fakten notiert. Wir verzichten zunächst auf Fisimatenten wie Flexionsmerkmale, kommen darauf aber wieder zurück. (Für diejenigen, die das 'alte' Lexikon ohne diese Merkmale nicht in irgendeiner Weise beibehalten haben, bedeutet das leider, das Lexikon nochmal anzulegen).

#### *6.1.5.4. Die PS-Regeln*

Wir übersetzen die PS-Regeln in Form von regel/2-Fakten in Prolog. Das erste Argument von regel/2 ist das Symbol zur linken des Pfeils, das als Atom wiedergegeben ist, das zweite Argument ist eine Liste der auf der rechten Seite des Pfeils aufgeführten Konstituenten. Wichtig: auch in denjenigen Fällen, in denen auf der rechten Seite des Pfeiles nur ein einziges Symbol auftritt (z.B. bei VP $\rightarrow V_i$) muss dieses in Listenform notiert werden. Die weiter oben aufgeführten PS-Regeln hätten mithin die folgende Form:

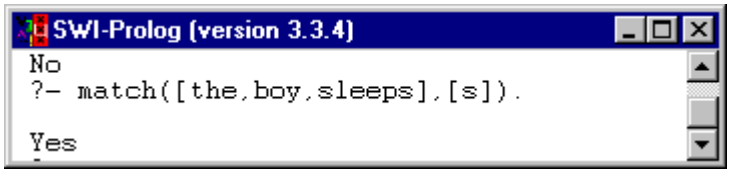
```
regel(s, [np, vp]).
regel(np, [det, noun]).
regel(vp, [vi]).
```

#### *6.1.5.5. Die Steuerdatei*

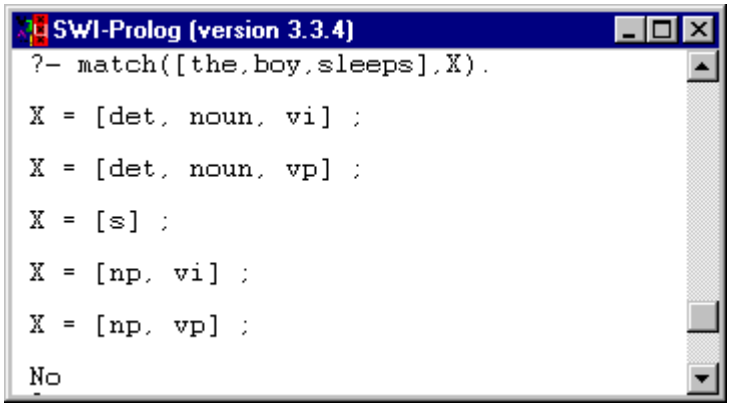
Hier liegt der Hase im Pfeffer. Wir wollen ein Prädikat match/2 schreiben. Das erste Argument ist die zu analysierende Wortkette; das zweite Argument ist die Ableitungsliste:

**match(Wortliste,Ableitungsliste).**

Eine Anfrage mit instantiiertem zweiten Argument (dem Startsymbol 'S') soll als Ergebnis die Meldung **Yes** bekommen:



In den Fällen, in denen das zweite Argument ein Variable ist, können die einzelnen Ableitungsschritte angezeigt werden (die Reihenfolge ist hier irrelevant):




Das Prädikat match/2 ist in drei Regeln notiert:

1. die Abbruchbedingung
2. der Fall, in denen match/2 auf das Lexikon rekurriert (rekursive Regel)
3. der Fall, in denen match/2 auf die PS-Regeln rekurriert (ebenfalls rekursive Regel)

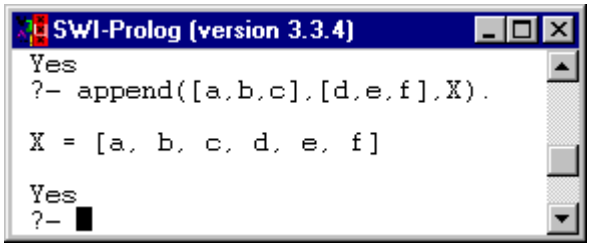
**Regel 1:**
Die Abbruchbdingung ist klar:
match([ ], [ ]).

**Regel 2 (Bezug zu lex/2):**
In der zweiten Regel wird auf das Lexikon Bezug genommen:

```
match([Wort|Restwortliste], [Kategorie|Restkategorienliste]):-
        lex(Wort,Kategorie),
        match(Restwortliste,Restkategorienliste).
```

Sobald es der Fall ist, dass eine Abbildung zwischen einem Wort und einer lexikalischen Kategorie erfolgt, werden im rekursiven Aufruf der Regel die Köpfe der Listen abgeschnitten und das Prädikat mit den Listenresten erneut aufgerufen.

**Regel 3 (Bezug zu regel/2):**
Durch die Formulierung der zweiten Regel mit Bezug auf die lex/2-Fakten ist für die dritte Regel von vorneherein klar, dass sie nur zur Anwendung kommt, wenn das erste Symbol der Ableitungsliste (sprich der Kopf der Ableitungsliste) eine syntaktische Kategorie ist. Dabei ist zu berücksichtigen, dass sich durch die Anwendung der dritten Regel an der Wortliste selber nichts ändert. Dazu braucht man sich bloß die Tabelle auf Seite 67 anzusehen – wann immer eine Ersetzung über eine PS-Regel, also nicht über eine Lexikonregel erfolgt, bleibt die Wortliste in voller Form erhalten. Prolog-technisch betrachtet hat das die Konsequenz, dass das erste Argument von match/2 in der dritten Regel beim rekursiven Aufruf nicht angetastet wird, ergo muss hier auch keine Zerlegung in einen Kopf und einen Rest erfolgen. Das zweite Argument hingegen muss in einen Kopf und einen Rest zerlegt werden. Der Kopf – eine syntaktische Kategorie – wird dann in den regel/2-Fakten 'gesucht'. Wird dort eine Regel gefunden, muss folgendes passieren: das zweite Argument der Regel (welches ja in Form einer Liste notiert ist) wird mit dem Rest der Ableitungsliste zu der 'neuen' Ableitungsliste verbunden. Dafür benutzen wir ein eingebautes Prolog-Prädikat namens append/3, welches die Funktion hat, zwei Listen zu einer dritten Liste zusammen-zuhängen und wie folgt aussieht:

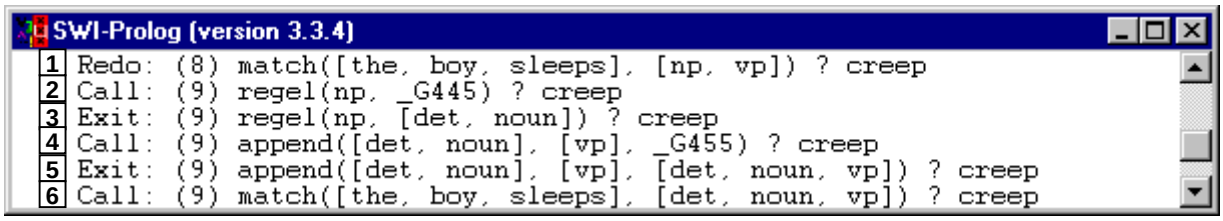



Das dritte Argument von append/3 ist eine neue Liste, die aus der Verkettung des ersten und des zweiten Argumentes (jeweils Listen) besteht, und genau diese Funktion benötigen wir.

Also kann die dritte Regel wie folgt notiert werden:

```
match(Wortliste, [Kategorie|Restkategorienliste]):-
        regel(Kategorie,Konstituenten),
        append(Konstituenten,Restkategorienliste,Neue_Ableitungsliste),
        match(Wortliste,Neue_Ableitungsliste).
```

Der nachfolgende kommentierte Screen-Shot gibt einen Teil der getracten Abarbeitung wieder, der genau zeigt, wie das Ganze funktioniert. Die Zeilennumerierung ist nachträglich hinzugefügt:




In Zeile 1 wird match/2 mit der Wortliste [the, boy, sleeps] und der Ableitungsliste [np,vp] aufgerufen. In Zeile 2 wird überprüft, ob es ein regel/2 Fakt gibt, in dem der Kopf der Ableitungsliste, also np, das erste Argument ist. Zeile 3 zeigt das Ergebnis an - es gibt eine entsprechende Regel, und die Konstituenten, durch die np substituiert werden soll, sind in Form der Liste [det, n] aufgeführt. In Zeile 4 nun wird diese Liste per append/3 mit dem Rest der Ableitungsliste – sprich der Liste [vp] zu einer neuen Liste verkettet, das Ergebnis ist in Zeile 5 zu sehen: die Liste [det, noun, vp]. Mit dieser neuen Ableitungsliste wird match/2 (mit unverändertem ersten Argument!) in Zeile 6 rekursiv aufgerufen.

Wie man sieht, ist match/2 eigentlich furchtbar einfach und umfasst insgesamt nur 8 Programmzeilen – deren Erklärung aber erfordert ein ziemlich langes und umständliches Geschreibe. Wie bereits häufiger in diesem Skript wird deshalb einerseits empfohlen, die Sache (der nachstehenden Aufgabe entsprechend) nachzuprogrammieren und dann üppigen Gebrauch von der trace-Funktion zu machen, die häufig mehr sagt als tausend Worte, und andererseits auf die animierte Darstellung in der Sitzung verwiesen, die möglicherweise auch zum besseren Verständnis beiträgt.

## Aufgabe 4:

4. Legt eine Datei psg_lex.pl an, in der Lexikon von Seite 58 in der Form von lex/2–Fakten realisiert ist (also ohne Flexionsmerkmale)
5. Legt eine Datei psg_reg.pl an, in der die den Sätzen auf Seite 58 zugrundeliegenden PS-Regeln in der Form von regel/2–Fakten realisiert sind.
6. Legt eine Datei namens psg_pat.pl an
   **a)** Lasst darin die Dateien psglex.pl und psg_reg.pl per consult/1 aufrufen.
   **b)** Definiert darin das Prädikat match/2 wie in diesem Kapitel vorgestellt. Macht Euch innerlich dafür bereit, zu diesem Prädikat noch eine Reihe unangenehmer Fragen zu beantworten (schreibt es also nicht einfach ab, sondern versucht, es wirklich zu verstehen).

Im nächsten Kapitel werden wir einen anderen Zugang zur Implementierung einer Phrasenstrukturgrammatik wählen. Von seiner inneren Ausstattung her ist das hier vorgestellte Programm bereits schon jetzt in der Lage, zahlreiche potentielle Modifikationen (von Erweiterungen in Richtung Kongruenz- und Rektionsphänomenen über Ambiguitäten und Verschachtelungen bis zur X-bar Syntax) zu durchlaufen. Das Problem liegt in der Erzeugung einer Strukturbeschreibung. Das soll nicht heißen, dass das nicht ginge - es ist aber ein wenig komplizierter, als es zum jetzigen Zeitpunkt sinnvoll wäre.