Aufgabe 1.

Legt eine Datei namens psg1.pl an.
Lasst von dort aus lexikon.pl aufrufen.
Setzt die Grammatik der vorherigen Seite psg1.pl um

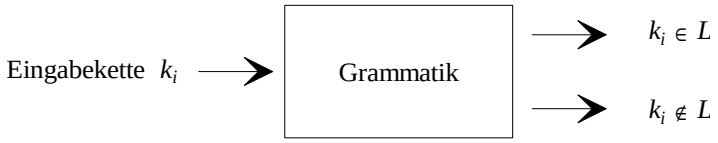
Aufgabe 2.

Ergänzt die Datei psg1.pl um Phrasenstrukturregeln, die auch die Analyse der folgenden Sätze ermöglichen und überprüft anschließend, ob die Sätze akzeptiert werden.:
Bart slept
The dog bit Lisa
Barney sold the beer to Homer
Marge read a rather silly cartoon.
Tip: Überlegt Euch zunächst, welche PS-Regeln für diese Sätze nötig sind, und übersetzt diese dann anschließend in Prolog. In den Fällen, in welchen neue Wortarten auftreten (Eigennamen, Präpositionen usw.), müssen natürlich auch entsprechende Regeln für die lexikalischen Kategorien formuliert werden! Wenn die Analyse nicht klappt, kann es auch immer am Lexikon liegen

Teil 2: Die Ausgabe der Strukturbeschreibung

Die PSG-Grammatik hat bis jetzt den Status eines Akzeptors – sie erkennt, ob eine gegebene Wortkette mit Bezug auf die Grammatik ein Satz (bzw. eine NP oder eine VP usw.) ist, oder nicht, und sie gibt dafür die Positionen an.

Etwas abstrakter betrachet: wenn wir eine Grammatik einer Sprache *L* als eine Ein-Ausgabe-Vorrichtung auffassen, die Ketten aus einem gegebenen Vokabular als Eingabe erhält und als Ausgabe die Entscheidung liefert, ob die Eingabe ein Satz von *L* ist oder nicht (Werte: {ja, nein}, oder {wahr, falsch}), haben wir es mit einer Erkennungsgrammatik zu tun.



Erkennungsgrammatik

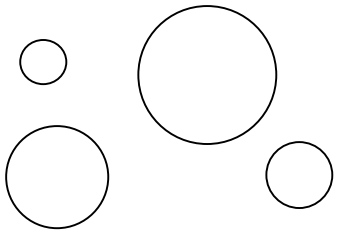
Interessanter als Erkennungsgrammatiken sind jedoch Grammatiken, die darüber hinaus jedem erkannten Satz eine bzw., bei Ambiguität, mehrere Strukturbeschreibungen zuordnet:

Generative Grammatik

Eine Grammatik, die über die bloße Entscheidung über die Wohlgeformtheit hinaus jedem erkannten Satz eine Menge von Strukturbeschreibungen zuordnet, ist eine generative Grammatik. An dieser Stelle kommt einem vielleicht der folgende Gedanke: Moment mal – generative Grammatik – das ist doch Chomsky-Grammatik? Generative Transformationsgrammatik, Government & Binding, Minimalist Program und so?!? Es ist in der Tat so, dass die Bezeichnung 'generative Grammatik' oft synonym mit einer der Modellvarianten bzw. dem gesamten Ansatz aus dem chomskyschen Dunstkreis verwendet wird. Das Attribut 'generativ' soll hier aber einerseits allgemeiner, andererseits auch eingeschränkter verstanden werden, als es häufig der Fall ist. Allgemeiner deshalb, weil es auf eine ganze Reihe von Grammatiktypen anwendbar ist, denen allesamt gemein ist, dass sie Verfahren für die Erzeugung der

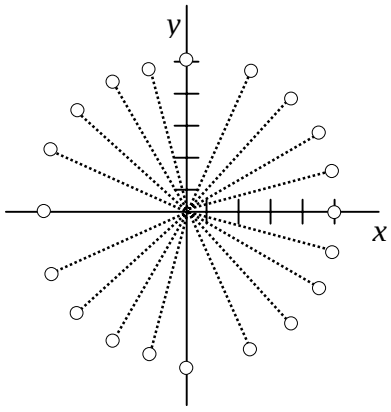
Strukturbeschreibung von Sätzen zur Verfügung stellen. Dazu gehören – neben den oa. Grammatikmodellen – auch Ansätze aus dem Bereich der Lexikalisch-Funktionalen-Grammatik (LFG); der Headdriven-Phrase-Structure-Grammar (HPSG); der Kategorialgrammatik usw. 'Generativ' ist andererseits eingeschränkt, weil es hier nur **mittelbar** um die wie auch immer geartete Produktion konkreter Sätze geht (und darum ging es auch Chomsky nie, was in der Vergangenheit aber häufig fehlinterpretiert wurde). Eine generative Grammatik kann durchaus eingesetzt werden, um über die Erzeugung von Strukturbeschreibungen konkrete Sätze zu generieren, was man mit Prolog auch einfach nachprüfen kann. Dieser Punkt ist aber im Grunde ein reines Nebenprodukt der Art und Weise, in der Aussagen über das Konstrukt 'Satz' formuliert sind. Während eine Erkennungsgrammatik einer eingegebenen Wortkette Grammatikalität in einer Sprache *L* attestiert (oder nicht), muss eine generative Grammatik qua ihres Aufbaus in der Lage sein, über das System der Regeln alle potentiellen Sätze der Sprache *L* zu generieren. Diese Menge potentieller Sätze ist natürlich, wie wir wissen, unendlich. Ergo kann es gar nicht darum gehen, tatsächlich alle Sätze zu generieren (im Sinne von 'produzieren'); sondern nur darum, ein System zu erstellen, über welches diese Menge in ihren Eigenschaften vollständig beschrieben ist. 'Generativ' darf hier also nicht im Sinne von 'produzieren', sondern sollte im Sinne von 'beschreiben' verstanden werden.

Nehmen wir dazu ein Beispiel aus einem ganz anderen Bereich. Die folgenden Objekte gehören allesamt zur gleichen Klasse, nämlich zu den Kreisen:



Kreise

Um diese Objekte einigermaßen explizit zu beschreiben und sie von Objekten, die keine Kreise sind (wie z.B. Rechtecken) abzugrenzen, reichen Aussagen wie 'Kreise sind rund' oder gar 'Kreise sind kreisförmig' natürlich nicht aus. Eine Möglichkeit, das Konstrukt 'Kreis' präzise zu beschreiben, besteht darin, die in der Geometrie verwendete Definition für Kreise heranzuziehen. Demnach ist der Kreis der geometrische Ort aller Punkte, für die gilt, dass sie in derselben Entfernung (= Radius) zu einem gegebenen Punkt (= Mittelpunkt) in einer Ebene liegen. Mit Bezug auf ein zweiachsiges Koordinatensystem und Kreise, die um den absoluten Nullpunkt liegen, kann diese Aussage wie folgt formalisiert werden: $x^2 + y^2 = r^2$. Der Radius *r* des Kreises entspricht mithin der Wurzel aus der Summe von x^2 und y^2: $r = \sqrt{x^2 + y^2}$. Wenn wir diese Formel anhand spezifischer Zahlenwerte konkretisieren, beispielsweise $x = 3$ und $y = 4$, liefert sie uns sie uns für *r* den Wert 5 (die Wurzel aus der Summe von 9 und 16, also 25, ist 5). D.h. dass die Gesamtheit aller Punkte, die im Radius 5 um den Nullpunkt liegen, ein geometrisches Objekt vom Typ 'Kreis' beschreiben. Exemplarisch soll das in der nachfolgenden Graphik dargestellt werden: [30]



[30] Das Koordinatensystem ist handgemalt und deshalb unpräzise. Für die Mathe-Nachhilfe vielen Dank an Prof. Wagner

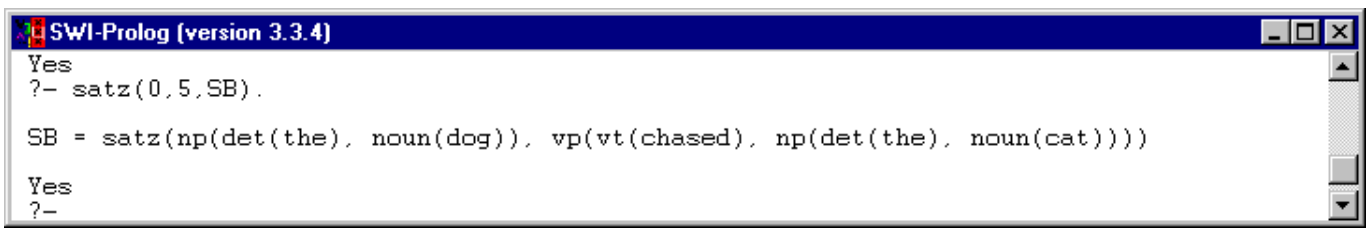
Über diese Definition ist das Konstrukt 'Kreis' (mit Bezug auf andere, unbhängig definierte theoretische Konstrukte wie 'Punkt' einerseits und bestimmte Operationen wie z.B. 'Addition' andererseits) präzise definiert. Mit entsprechendem Werkzeug, z.B. Lineal und Zirkel, könnte man die Definition dazu benutzen, alle potentiell möglichen Kreise zu erzeugen: dadurch, dass man jeweils unterschiedliche Werte für die Variablen *x* und *y* einsetzt, lassen sich Kreise in beliebiger Größe generieren. In der Tat kann eine Formel wie die oben angegebene – beispielsweise in *C*(omputer)-*A*(ssisted)-*D*(esign)-Programmen – genau für diesen Zweck eingesetzt werden. Sie ist aber dennoch keine 'Kreisgenerierungsformel' in dem Sinn, dass es ihre exklusive Aufgabe wäre, die Menge aller Kreise zu produzieren, denn auch hier gilt - genau wie bei den Sätzen einer Sprache L – dass diese Menge undendlich ist. Die oa. Defintion liefert stattdessen eine explizite Beschreibung des (geometrischen) Konstruktes 'Kreis'.

Die Analogie zu den Ausführungen über generative Grammatik sollte klar sein – hier geht und ging es auch früher nicht darum, eine Grammatik im Sinne einer 'Satzerzeugungsmaschine' zu erstellen, die mehr oder weniger undifferenziert Wortketten ausspuckt, die in einer Sprache L den Status von Sätzen haben, sondern stattdessen um eine explizite Beschreibung des Konstruktes 'Satz'. Diese explizite Beschreibung erfolgt in einer PSG als generativer Grammatik mit Bezug auf andere, unabhängig definierte theoretische Konstrukte wie z.B. 'NP' oder 'Präposition' einerseits und bestimmte Operationen, wie z.B. Phrasenstrukturregeln andererseits.

Diese ganze Herangehensweise an den Umgang mit Sprache ist natürlich entscheidend von dem Bestreben beeinflußt, diejenigen Aussagen, die man über den Gegenstandsbereich (im vorliegenden Fall eben die Syntax einer Sprache *L*) macht, so zu halten, dass sie den Kriterien für Wissenschaftlichkeit und Präzision genügen, wie z.B. intersubjektive Überprüfbarkeit, Eindeutigkeit, Widerspruchsfreiheit, Transparenz der Argumentation usw. Dieses Bestreben verdient natürlich, gewürdigt zu werden, die konkrete Umsetzung kann aber durchaus auch hinterfragt werden. Um das Konstrukt 'Satz' in einer Sprache *L* nach dem Muster wie in der hier verwendeten PSG zu definieren, werden etliche Abstraktionen getroffen. Man bezieht sich in der Grammatik auf lexikalische und syntaktische Kategorien und ggf. auf die Flexionsmorphologie, aber bestimmte andere Aspekte werden dabei völlig ausgeblendet – z.B. Fragen der Semantik oder Pragmatik. Satzübergreifende syntaktische Phänomene können mit der von uns verwendeten PSG ebenfalls nicht erfasst werden. Genau diese Arten von Information sind in anderen Ansätzen (und auch in moderneren Formen der generativen Grammatik chomsyischer Prägung) von vorneherein viel systematischer in die Grammatik integriert. Soviel erstmal zu einer Einschränkung des Begriffes 'generativ Grammatik' – damit könnte man allerdings Jahrzehnte zubringen.

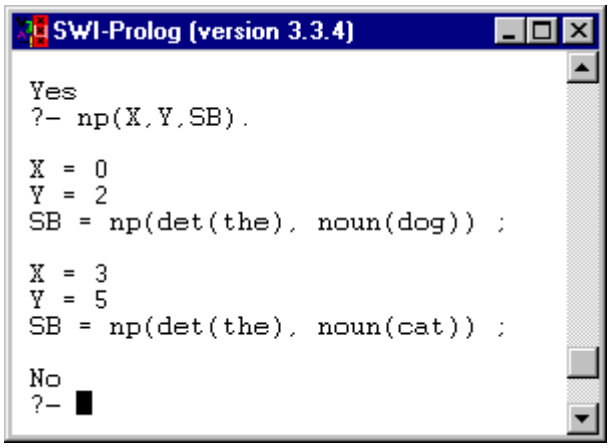
Was die Implementierung einer generativen Grammatik in Prolog betrifft (hier auf Grundlage einer PSG, also eines Ansatzes, der in der Tat mit dem Hause Chomsky assoziiert ist), zuerst einmal eine gute Nachricht: im Kern, d.h. vom gesamten Programmaufbau her, haben wir die dafür notwendigen Elemente bereits in Teil 1 dieses Kapitels, also in dem PSG-Akzeptor angelegt. In der Tat wird eine Wortkette ja mit Bezug auf Phrasenstrukturregeln – und nicht etwa auf so etwas wie die Satzmuster im ersten Teil des 6. Kapitels – analysiert. Das Ergebnis dieser Analyse, sprich die Aussagen 'X ist ein Satz / X ist kein Satz' rekurriert auf die Grammatik, das heißt der Satz entspricht einer der in der Grammatik explizit beschriebenen syntaktischen Strukturen. Diese aber werden bei der Analyse nicht angezeigt. Um dieses zu bewerkstelligen, werden alle Regeln für die syntaktischen und die lexikalischen Kategorien renotiert. Diese Modifikation läuft daraufhinaus, dass wir diese Regeln, genauer gesagt die Prädikate, um die es geht, also satz/2, np/2, vt/2, det/2 usw. usf., um ein drittes Argument erweitern, und genau dieses Argument wird auf die Strukturbeschreibung der Wortketten in question abgebildet.

Eine Eingabe wie satz(0,5,SB), wobei das dritte Argument, die Variable 'SB', für 'Strukturbeschreibung' steht, soll die folgende Ausgabe erhalten:



Auf die Tatsache, dass eine Funktor-Argumentstruktur bzw. ein Klammerausdruck wie in dem obigen Screen-Shot 1-zu-1 auf einen Baumgraphen abgebildet werden kann, wird hier mit keinem Wort eingegangen, denn das ist Syntax-Basiswissen.

Auch andere Anfragen sind nun möglich, beispielsweise nach den Strukturbeschreibungen der Nominalphrasen, hier mit Variablen für die Positionen:



Wie setzt man das konkret um? Am Beispiel der Regel für den Satz soll dieser Vorgang erläutert werden. Die bisherige Regel für einen Satz lautete:

```
satz(P0,Pn):-
        np(P0,P1),
        vp(P1,Pn).
```

Der Regelkopf satz(P0,Pn) soll nun um ein Argument dahingehend erweitert werden, daß auch die Strukturbeschreibung des Satzes enthalten ist. Das Problem dabei ist: wir wissen ja beim Aufruf des Prädikates überhaupt nicht, wie diese Strukturbeschreibung auszusehen hat – diese muss bei der Abarbeitung der eingegebenen Wortkette ja erst generiert werden. Welche Form muss also das dritte Argument von satz/3 haben?

Nun, alles, was wir hier sagen können, ist die Tatsache, dass es sich bei der Strukturbeschreibung des Satzes um eine Funktor-Argument-Struktur handeln wird, deren Funktor 'satz' ist und deren Argumente

a) die Strukturbeschreibung der (Subjekts-) NP, die erst ermittelt werden muss und

b) die Strukturbeschreibung der VP, die ebenfalls erst ermittelt werden muss, sind

Wie diese Strukturbeschreibungen intern genau aussehen ist nicht bekannt, sondern muss erst generiert werden. Die folgenden Beispiele zeigen, dass die NP und VP jeweils ganz unterschiedlich aufgebaut sein können:

{ *John* / *The boy* / *A student with glasses* } { *slept* / *read a book* / *put the money on the counter* }

Um diesen Sachverhalt in Prolog umzusetzen, wird das dritte Argument von satz/3 in einer Funktor-Argumentstruktur notiert, in der der Funktor 'satz' lautet (soviel ist ja klar) und dessen Argumente als Variablen auf die jeweiligen Strukturbeschreibungen von Subjekts-NP und VP verweisen.

```
satz(P0,Pn, satz(NP,VP)):- np(P0,P1,NP), vp(P1,Pn,VP).
```

Strukturbeschreibung des Satzes in Form einer Struktur mit dem Funktor *satz* und den Strukturbeschreibungen der NP und der VP als Argumenten.

Struktur-beschreibung der NP

Struktur-beschreibung der VP

Natürlich müssen wir die Regeln für die NP und die VP ebenfalls um ein drittes Argument anreichern. Im Einzelnen sieht das so aus:

- die Strukturbeschreibung der Kategorie NP besteht aus den jeweiligen Strukturbeschreibungen des Determinators und des Nomens:

```
np(P0,Pn,np(Det,Noun)):-
        det(P0,P1,Det),
        n(P1,Pn,Noun).
```

- die Strukturbeschreibung der Kategorie VP besteht aus den jeweiligen Strukturbeschreibungen des Verbes und der NP:

```
vp(P0,Pn,vp(VT,NP)):-
        vt(P0,P1,VT),
        np(P1,Pn,NP).
```

Soviel zu den Regeln für die syntaktischen Kategorien.

Die Regeln für die lexikalischen Kategorien det, noun und vt sahen ja etwas anders aus als die für die phrasalen Kategorien:

```
det(P0,P1):- link(P0,Wort,P1), lex(Wort,det).
noun(P0,P1):- link(P0,Wort,P1), lex(Wort,noun).
vt(P0,P1):- link(P0,Wort,P1), lex(Wort,vt).
```

Auch hier müssen die Regelköpfe um ein Argument erweitert werden. Der Unterschied zu den phrasalen Kategorien und dem Satz besteht aber darin, daß sich die Strukturbeschreibung hier nicht aus zwei anderen Strukturbeschreibungen zusammensetzt – mit den lexikalischen Kategorien ist sozusagen das Ende der Verzweigungen erreicht.

Anders ausgedrückt: die Strukturbeschreibung einer lexikalischen Kategorie besteht nur in der Angabe der lexikalischen Kategorie und dem Lexem, durch welches diese repräsentiert ist. Die Regeln für die lexikalischen Kategorien müssen also wie folgt erweitert werden:

```
det(P0,P1,det(Wort)):-
        link(P0,Wort,P1),
        lex(Wort,det).
```

Hier gibt der Funktor Aufschluß über die lexikalische Kategorie, das Argument ist das Wort selbst.

```
noun(P0,P1,noun(Wort)):-
        link(P0,Wort,P1),
        lex(Wort,noun).
vt(P0,P1,vt(Wort)):-
        link(P0,Wort,P1),
        lex(Wort,vt).
```

Wenn die Grammatik auf diese Art geändert wird, gibt Prolog für die jeweiligen syntaktischen Kategorien eine Strukturbeschreibung in Form eines Klammerausdruckes aus. Diese Strukturbeschreibungen werden bei Abarbeitung der entsprechenden Anfragen generiert, und es sollte mithin klar werden, wie genau das Verhältnis zwischen der Grammatik auf der einen und den Strukturbeschreibungen auf der anderen Seite ist – hier sieht man ganz genau, wie diese aus der Grammatik abgeleitet werden. Diese schöne Transparenz ist bei manchen der modernen Grammatikmodellen nicht mehr in dieser Form gegeben, insofern sich diese von klar regelbasierten Systemen weg- und zu eher restriktionsbasierten Systemen hinbewegt haben (zumindest auf dem Papier, wenn man versucht, sie in einem Programm konkret umzusetzen, kann sich das das Bild wieder ändern). 'Restriktionsbasiert' (die englische Bezeichnung heißt 'constraint based') kann hier so verstanden werden, dass die Grammatik eine Reihe von Strukturprinzipien kodiert, die von einer Wortkette erfüllt sein müssen, damit diese als wohlgeformt 'durchgeht'. Diese Strukturprinzipien (die beispielsweise auch Bezug nehmen auf bestimmte semantische Aspekte) sind aber ganz anderer Natur und sehr viel abstrakter als beispielsweise PS-Regeln. Mit Bezug auf die Rektions- und Bindungstheorie kann dieser Aspekt wie folgt beschreiben werden:

> Thus rather than building conditions into the rules themselves (as in the standard theory), or even imposing general conditions on the operation and interaction of simplified rules within the system (as in the early stages of the extended standard theory), Chomsky has now developed a theory of conditions on *the*

> *representations themselves*. The rules [...] do no more than specify those aspects of representations that do not follow from general principle. (GEOFFREY HORROCKS, *Generative Grammar*, Longman 1987:98)

Diese interessante Entwicklung im Bereich moderner Grammatikmodelle kann an dieser Stelle weder genauer erklärt noch weiter nachgezeichnet werden. Interessierte Studierende werden an entsprechende Lehrveranstaltungen verwiesen.

Nun ist die Rohform unseres Programms erstellt – es akzeptiert eine grammatische Wortfolge als Satz und gibt die Strukturbeschreibung an. So richtig prall ist das Programm aber nicht – weder die umständliche Eingabe von Sätzen als link/3 Fakten noch die Form, in der die Strukturbeschreibungen angezeigt werden (als Klammerausdruck) erfreuen die Benutzer. Um die Revision dieser Mängel geht es in den nächsten Abschnitten.

Aufgabe 3

Lasst die Sätze
Lisa slept
John cried
The girl loved the dog
Some boy kicked the table
John tied the man with the rope
Mary gave the dog to the girls
Bart put the book under the table
A terribly ugly boy kicked the dog

derart analysieren, dass jeweils eine Strukturbeschreibung erzeugt wird. Bedenkt dabei, dass Ihr vor der Eingabe des jeweiligen Prädikats link/3 das "alte" Prädikat link/3 mit abolish/1 aus der Wissensbasis entfernen müsst. Bitte kein Gemaunze wg. der umständlichen Eingabe – die Änderung der Bequemlichkeit steht als nächtest auf der Agenda.