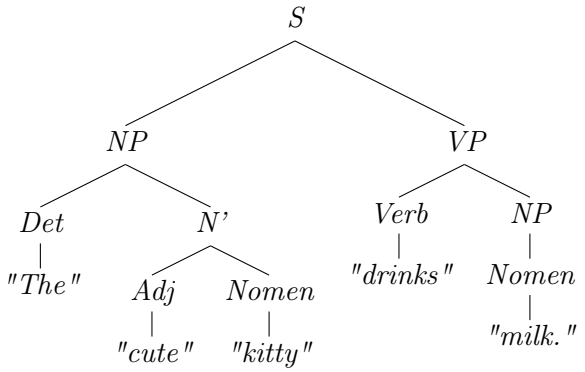
1 Formale Grammatiken in Natürlichen Sprachen

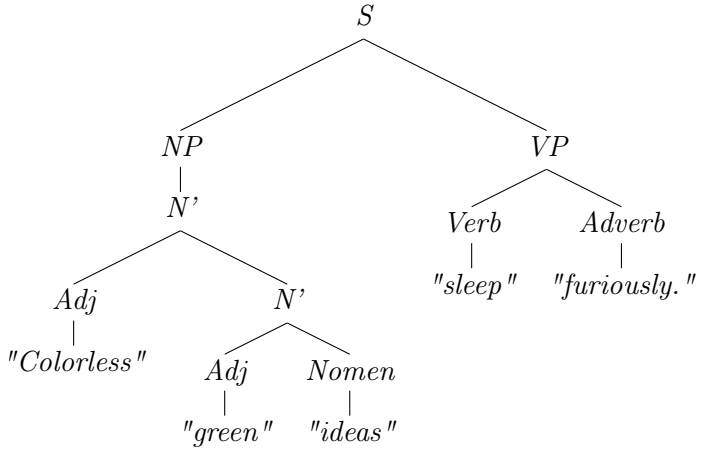
Bevor es zu Alignments mit Grammatiken geht, eine kleine Motivation zu *formalen Sprachen* und *formalen Grammatiken*, damit die Begriffsherkunft deutlich wird.

In den 50er Jahren veröffentlichte Noam Chomsky seine Ideen zu Syntaxbäumen in natürlichen Sprachen und ihr unterliegenden (verdeckten) Regeln, welche hinter dem Aufbau von Sätzen "versteckt" sind. (Hier nur eine Vereinfachung der Regeln.)

Beispiel 1.1 *"The cute kitty drinks milk."*



Beispiel 1.2 *"Colorless green ideas sleep furiously."*



In diesen Beispielen kann man bereits erkennen, dass jeder Satz aus einer Nominal- (NP) sowie einer Verbalphrase (VP) besteht. Eine NP enthält ein Nomen, welches einem Artikel und/oder Adjektive(n) folgt. N' ist lediglich eine Hilfskonstruktion für die NP. Diese und weitere Regeln beschreiben dann eine Sprache – in den obigen Beispielen ist dies Englisch.

Formal kann eine Sprache $\mathcal{L}$ durch eine Grammatik G beschrieben werden, welche sich wiederum aus *Non-Terminalen* N (Variablen, welche Zwischenschritte zum Erzeugen eines Satzes sind) und *Terminalen* T (hier Wörtern) zusammen setzt. Desweiteren wird eine Menge an *Produktionsregeln* P benötigt, welche dann erlaubte Ableitungsschritte enthält.

In der Grammatik für englisch findet man dann u.A. folgende Produktionsregeln:

$$
\begin{aligned}
P = \{ & S \rightarrow NP\ VP, \\
& NP \rightarrow Det\ N' \mid N', \\
& N' \rightarrow Adj\ N' \mid Adj\ Nomen, \\
& \vdots \\
& Det \rightarrow\ \text{"}The\text{"}, \\
& Nomen \rightarrow \text{"}kitty\text{"} \mid \text{"}ideas\text{"}, \\
& \vdots \}
\end{aligned}
$$

Somit haben wir alles gegeben, um eine Grammatik G eindeutig zu definieren:

$$G = (N, T, P, S)$$

Sie ist also ein 4-Tupel (ein formales Konstrukt, welches aus vier Teilen besteht) mit den Non-Terminalen N, den Terminalen T, den Produktionsregeln P und dem Startsymbol S, wobei dieses aus der Menge der Non-Terminale ist, also $S \in N$. (Die Ausnahme bilden die backward/outside Varianten der Alignment-Algorithmen—hierzu später mehr.)

Wie man dem zweiten Beispiel dieses Kapitels entnehmen kann, sorgt eine Grammatik lediglich für die Korrektheit der Syntax, nicht jedoch für einen sinnvollen Satz. Eine formale Grammatik spannt lediglich einen Suchraum auf (den aller zulässigen Sätze in einer Sprache) – die formale Sprache, die es beschreibt. Die Auswertung dieses Suchraumes bleibt einer *Algebra* überlassen.

2 Formale Grammatiken auf Strings

Im vorangegangenen Kapitel haben wir Formale Grammatiken als Schritt zum Formalisieren natürlicher Sprachen kennen gelernt. In diesem Kapitel abstrahieren wir dieses formale Konzept auf Sprachen, welche nicht mehr Sätze enthalten, sondern Wörter oder Strings (wie z.B. Gensequenzen; siehe Definition String). Eine formale Grammatik beschreibt also nun, welche Schritte unternommen werden müssen, um gültige Strings einer formalen Sprache zu erzeugen.

Da wir nun mit einzelnen Buchstaben als kleinste Einheit (und nicht mehr Wörtern) arbeiten, wird die Menge der Terminale häufig als *Alphabet* mit dem Symbol Σ bezeichnet.

Beispiel 2.1 *Gesucht ist die Grammatik zu einer Sprache $\mathcal{L}_a$, welche beliebig lange Strings aus a's enthält.*

$G_a = (N_a, \Sigma_a, P_a, S_a)$ mit $\Sigma = \{a\}$, $P_a = \{S_a \rightarrow aS|\epsilon\}$ und $N_a = \{S_a\}$. Hier steht ϵ für das leere Wort *und führt zum Abbruch der sich entwickelnden Zeichenkette.*

Für die, die sich etwas mit Berechenbarkeit auskennen: Es gibt auch andere Möglichkeiten diese Sprache als Grammatik hinzuschreiben. Wir bevorzugen jedoch diese, da wir mit ihr sicherstellen können, dass alle Probleme, die auf diese Sprache angewandt werden, in polynomieller Zeit lösbar sind. (Zum Weiterlesen nach Chomsky-Hierarchie, regulären Grammatiken, Typ III Grammatiken suchen.)

Um ein besseres Gefühl für Grammatiken auf Strings zu bekommen, folgt noch ein weiteres Beispiel.

Beispiel 2.2 *Gesucht ist die Grammatik zu einer Sprache $\mathcal{L}_{ab}$, welche eine beliebig lange Strings aus der alternierenden Folge von a's und b's enthält.*

$G_{ab} = (N_{ab}, \Sigma_{ab}, P_{ab}, S_{ab})$ mit $\Sigma_{ab} = \{a, b\}$, $P_{ab} = \{S_{ab} \to aB \mid \epsilon, B \to bS_{ab} \mid \epsilon\}$ und $N = \{S_{ab}, B\}$. Mit dem zweiten Non-Terminal B stellen wir sicher, dass immer ein b eingefügt werden, sobald ein a erzeugt wurde. Es kann aber auch an jeder Stelle terminiert werden (durch Benutzen der ϵ).

Wir können nun also eine Menge abstrakter Zeichenketten erzeugen. Wenn man sich als nächstes Beispiel die Sprache anschaut, welches eine beliebige DNA Sequenz liest, wird auffallen, dass auch dies sehr unkompliziert ist.

Beispiel 2.3 *Gesucht ist eine Grammatik zu der Sprache $\mathcal{L}_{DNA}$, welche beliebige DNA-Sequenzen erzeugt.*

$G_{DNA} = (N_{DNA}, \Sigma_{DNA}, P_{DNA}, S_{DNA})$ mit $\Sigma_{DNA} = \{a, t, g, c\}$ (der Konvention nach sind Terminale immer als Kleinbuchstaben gegeben, Non-Terminale werden durch Großbuchstaben erkennbar). Die Menge der Produktionsregeln ist nun etwas größer als zuvor: $P_{DNA} = \{S_{DNA} \to aS \mid tS \mid gS \mid cS \mid \epsilon\}$. Somit ergibt sich $N_{DNA} = \{S_{DNA}\}$ als Menge der Non-Terminale und G_{DNA} ist vollständig beschrieben.

Als letztes Beispiel für Grammatiken auf Strings wollen wir beliebige Proteinsequenzen erzeugen. Wenn man das vorherige Beispiel mit DNA-Sequenzen beachtet, kann man schnell sehen, dass das Hinschreiben jeder einzelnen Produktionsregel für jede einzelne Aminosüre eher ermüdend ist. Deshalb führen wir eine Variable α ein, welche als Platzhalter für Buchstaben aus Σ fungiert.

Beispiel 2.4 *Gesucht ist eine Grammatik zu der Sprache $\mathcal{L}_{Prot}$, welche beliebige Sequenzen aus Aminosäuren erzeugt.*

$G_{Prot} = \{N_{Prot}, \Sigma_{Prot}, P_{Prot}, S_{Prot}\}$ mit $\Sigma_{Prot} = \{a, r, n, d, c, q, e, g, h, i, l, k, m, f, p, s, t, w, y, v\}$. Für $\alpha \in \Sigma_{Prot}$ lassen sich die Produktionsregeln verkürzen auf $P_{Prot} = \{S_{Prot} \to \alpha S \mid \epsilon\}$. Daraus ergibt sich $N_{Prot} = \{S_{Prot}\}$.

Im Verlauf der Vorlesung werden wir weiterhin mit Platzhaltern arbeiten. Zum einen bleibt die Menge der Produktionregeln minimal, zum anderen ist es aber auch möglich Σ beliebig auszutauschen. (In unserer Gruppe wurden bereits Alignments von natülich DNA- und Aminosäure-Sequenzen gemacht, aber auch von Wörtern aus natürlichen Sprachen— alles mit **derselben** Grammatik!)

3 Grammatiken für Sequenzalignment

In diesem Kapitel kommen wir zu den eigentlichen Alignments von Sequenzen. Σ wird nicht weiter spezifiziert, da es beliebig austauschbar ist. (Wer möchte, kann sich von nun an gern vorstellen, dass wir mit DNA-Sequenzen arbeiten, also $\Sigma = \{a, c, g, t\}$.)

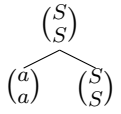
Das Ziel eines Sequenzalignments ist: alle Sequenzen optimal aneinander zu legen. Optimal bezieht sich hier auf den bestmöglichen Alignment Score. Wir werden jedoch erst im folgenden Kapitel auf Scores eingehen. Hier interessiert uns lediglich, wie ein Alignment unter der Verwendung von Grammatiken aussieht.

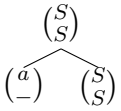
Wie gerade eben schon erwähnt, haben wir jetzt nicht nur eine Sequenz sondern mindestens zwei Sequenzen, die unsere Grammatik abarbeiten muss. Für den Moment ist es

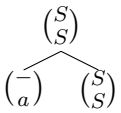
auch ausreichend, den 2-dimensionalen Fall zu behandeln. Erst im Ausblick soll auf *Multiple Sequence Alignments* (von nun an mit *MSA* abgekürzt) eingegangen werden.

Grammatik für paarweises Needleman-Wunsch-Sequenzalignment: $G_{NWA} = \left(N_{NWA}, \Sigma, P_{NWA}, \binom{S}{S}\right)$, mit $P_{NWA} = \left\{\binom{S}{S} \to \binom{u}{v}\binom{S}{S} \middle| \binom{u}{-}\binom{S}{S} \middle| \binom{-}{v}\binom{S}{S} \middle| \binom{\epsilon}{\epsilon}\right\}$ für $u, v \in \Sigma$ und $N_{NWA} = \left\{\binom{S}{S}\right\}$. Hier kann man wie im Standard-Needleman-Wunsch alle drei Fälle wieder finden. $\binom{u}{v}$ bildet den *Match*-Fall ab, $\binom{u}{-}$ steht für eine *Deletion* und $\binom{-}{v}$ für eine *Insertion*. Um den Aufbau des Syntaxbaumes abzubrechen, kann $\binom{\epsilon}{\epsilon}$ eingefügt werden.

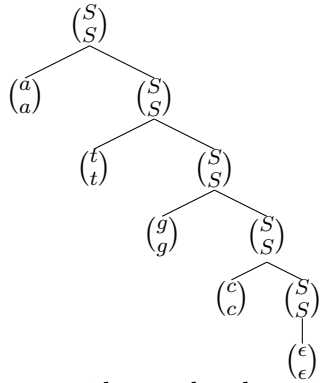
Beispiel 3.1 *In diesem Beispiel schauen wir uns an, wie ein Alignment aufgebaut wird. Dabei nehmen wir zwei identische Sequenzen S_1 und S_2, damit das Thema der Auswertung hier noch keine Rolle spielt. $S_1 = atgc = S_2$, $\Sigma = \{a, t, g, c\}$; G definiert wie oben. Für jeden Schritt, der beide Sequenzen abarbeitet, gibt es jeweils 3 Möglichkeiten (für jeden möglichen Ableitungsschritt genau einen). Daraus resultieren bereits 3 verschiedene Syntaxbäume für nur den ersten Schritt:*

1. 

2. 

3. 

Offensichtlich muss der erste Fall der beste sein, da wir in beiden Sequenzen (ATGC) as lesen (perfekter Match). Dies gilt auch für alle folgenden Schritte. Somit besitzt das perfekte Alignment hat dann folgenden Syntaxbaum.



Das komplette Alignment ist dann unüberraschenderweise:

$$\begin{array}{cccc} a & t & g & c \\ | & | & | & | \\ a & t & g & c \end{array}$$

Dabei stehen die senkrechten Striche in der Regel für matching pairs.

Im nächsten Kapitel wird darauf eingegagen, wie man zum optimalen Alignment genau kommt.