Lektion 6

Im nun beginnenden Kapitel 4 wird ein sehr allgemeines Konzept für Sprachen (auf rein syntaktischer Ebene) eingeführt. Dieses Konzept der formalen Sprache bildet die Grundlage für die folgenden Kapitel, in denen verschiedene Methoden betrachtet werden, wie Sprachen formal definiert werden können und wie entsprechend dann eine "Syntaxanalyse" durchgeführt werden kann.

Mit XML lernen Sie in dieser Lektion auch eine spezielle Sprache zur Beschreibung hierarchisch strukturierter Daten kennen, die in der Informatik große praktische Bedeutung hat.

Kap. 4 Formale Sprachen

Inhalt

- Alphabete und Wörter
- Sprachen
- XML

Motivation: Anwendung formaler Sprachen

Bei der Theorie der formalen Sprachen geht es um textuelle, d.h. geschriebene Sprachen, also nicht um gesprochene Sprachen oder bildliche Sprachen (wie z.B. UML-Diagramme). Solche Sprachen werde in der Informatik an vielen Stellen verwendet. Die prominentesten Beispiele dafür sind Programmiersprachen, wie Java oder C++, Dokumentenbeschreibungssprachen wie HTML und Datenbeschreibungs-sprachen, wie XML (eXtended Markup Language) und JSON (Java Script Object Notation).

Aber auch an vielen andere Stellen werden in der Informatik spezielle anwendungs-spezifische Sprachen (sog. *Domain Specific Languages*, *DSL*) eingesetzt. Beispielsweise hat die Spiel-Engine Godot eine eigene Skriptsprache, die spezifisch an den Bedarf der Spieleentwicklung angepasst ist. Auch SQL als Sprache zum Aufbau und zur Abfrage relationaler Datenbanken kann als eine DSL betrachtet werden. Und im Textverarbeitungssystem LibreOffice/OpenOffice können mathematische Formeln in einer eigenen Beschreibungssprache eingegeben werden.

Zu den Aufgaben eines Informatikers kann es also auch gehören, solche Sprachen

eindeutig zu definieren und zu implementieren. Die Theorie der formalen Sprachen bildet dafür die Grundlage.

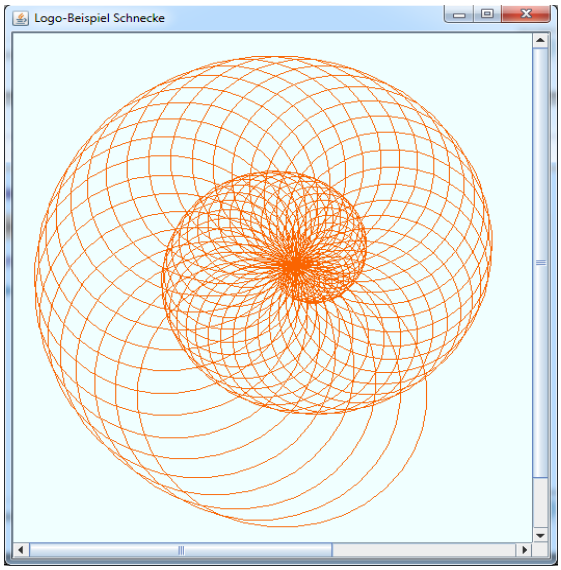
Beispiel: Programmiersprache HSULogo

- Vor einigen Semester habe ich einen Interpreter für eine Variante der Programmiersprache Logo implementiert. Logo wurde ursprünglich Ende der 1960er Jahre als Ausbildungssprache entwickelt, um auf anschauliche Weise Programmierkonzepte zu vermitteln. Die Basis ist dabei sog. Turtle-Grafik, eine Form der 2D-Grafik, bei der Turtles (Schildkröten = Zeichenstift) durch Befehle über den Bildschirm bewegt werden und dabei eine Zeichenspur hinter sich herziehen, so wie Schildkröten eine Spur auf dem Sandstrand hinterlassen.
- Hier sehen Sie ein Beispielprogramm meiner Logo-Variante und das Ergebnis der Programmausführung:

```
program "Logo-Beispiel Schnecke"
; Ein erstes Logo-Beispiel

to circle (:len)
  repeat 36 [
      forward :len
      turnright 10
  ]
end

setbackcolor 240 255 255
setpencolor 250 100 0
penup
moveto 300 300
pendown
make :len 3
repeat 72 [
      circle(:len)
      turnleft 10
      make :len :len+0.3
]
```



Wie kann eine solche Sprache definiert werden? Und wie können später Programme verarbeitet werden? Einen ersten Überblick haben Sie schon im Einführungskapitel bekommen. In den folgenden Kapiteln wird dies nun genauer behandelt.

Die erste Frage ist die: Was ist überhaupt eine Sprache? Im Bereich der formalen Sprachen - und in den folgenden Kapiteln - wird nur der Aspekt der *Syntax* betrachtet, nicht aber die *Semantik* (d.h. Bedeutung) von sprachlichen Äußerungen.

Wir beschränken uns hier auf textuelle Sprachen, wie z.B. geschriebene natürliche Sprachen oder auch Programmiersprachen. Ein Text ist nichts anderes als eine Folge von Zeichen (auch Symbole genannt). Es geht also letztendlich um Zeichenfolgen, die aus Zeichen eines bestimmten Zeichenvorrats gebildet werden können.

4.1 Alphabete und Wörter

Definition 4.1 - Alphabet

Ein **Alphabet** Σ ist eine endliche, nicht leere Menge von *Zeichen* (auch *Symbole* genannt).

Anmerkung

- Die Bezeichnung von Alphabeten Σ mit großem Sigma kommt von "Symbole".
- Ein "Zeichen" oder "Symbol" kann alles mögliche sein, z.B.
  - Ein Zeichen, wie auf der Tastatur oder in einem ASCII- oder Unicode-Zeichensatz gegeben, etwas ein Buchstabe, eine Ziffer oder ein Sonderzeichen.
  - Auf einer höheren Ebene kann aber auch ein Folgen von elementaren Zeichen selbst als ein Symbol betrachtet werden, z.B. eine Zahl, ein Schlüsselwort oder ein Bezeichner einer Methode oder Klasse in einem Programm. Bei der Implementierung von Programmiersprachen werden solche Zeichenfolgen, die eine lexikalische Einheit bilden, als "Token" bezeichnet.

4.1.1 Wörter

Wenn man ein Alphabet als Zeichenvorrat hat, kann man damit Zeichenfolgen bilden. Wir betrachten für die formalen Sprachen nur *endlich lange Zeichenfolgen* (es gibt auch eine erweiterte Theorie der formalen Sprachen für unendlich lange Folgen).

Definition 4.2 - Wörter

- Ein **Wort** (engl. *string, word*) über einem Alphabet Σ ist eine **endlich lange Folge von Zeichen** aus Σ.
- $|w|$ bezeichnet die **Länge des Worts** *w.*
- ε ("epsilon") bezeichnet das **leere Wort** mit Länge 0, d.h. $|\varepsilon| = 0$.
- $|w|_a$ bezeichnet die **Anzahl der Vorkommen** des Zeichens *a* in *w*.
- Σ^* ("Sigma Stern") bezeichnet die **Menge aller Wörter**, die mit Zeichen des Alphabets Σ gebildet werden können.

Anmerkung

- Σ^* enthält immer das leere Wort ε, unabhängig vom Alphabet Σ.

- Σ^* ist immer eine (abzählbar) unendliche Menge, das Wörter beliebiger Länge gebildet werden können.

Beispiel 4.3 - Alphabet und Wörter

Gegeben sei Alphabet $\Sigma_1 = \{a, b, c\}$.

- Einige Beispiele für Wörter über Σ_1:

  abba, a, abbccc, ε

- Länge der Wörter

  $|\ abba\ | = 4$

  $|\ a\ | = 1$

- Anzahl der Zeichenvorkommen

  $|\ abba\ |_a = 2$

  $|\ abba\ |_b = 2$

  $|\ abba\ |_c = 0$

Alphabet $\Sigma_2 = \{0, 1\}$:

- Wörter über Σ_2: z.B. 0, 1001

- Menge aller Wörter, die mit Σ_2 gebildet werden können:

  $\Sigma_2^* = \{ \varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, ...\}$

  d.h. alle *endlich langen* Folgen aus Nullen und Einsen, einschließlich der leeren Folge ε.

4.1.2 Konkatenation

Welche Operationen kann man auf Wörtern ausführen? Neben der Länge gibt es eine weitere Grundoperation, man kann Wörter aneinanderhängen. Das nennt man *Konkatenation*.

Definition 4.4 - Konkatenation

Die **Konkatenation** (das Aneinanderhängen) zweier Wörter v und w wird als

$v \cdot w$

notiert. Statt $v \cdot w$ schreibt man üblicherweise nur

vw

d.h. der Operator wird weggelassen.

Aufgabe 4.5 - Konkatenation

Gegeben ist Alphabet $\Sigma_1 = \{a,b,c\}$. Wort u_1 = ab, Wort u_2 = bab.
Was ergibt sich in folgenden Fällen durch die Konkatenation:

(1) u_1u_2

(2) u_2u_1

(3) u_1u_1

(4) u_2u_2

Notationskonvention

- Für einzelne Zeichen werden in den folgenden Beispielen meist Buchstaben vom Anfang des Alphabets genommen (z.B. a, b, c), oder auch Ziffern (z.B. 0, 1)
- Für die Bezeichnung von Wörtern werden meist Buchstaben vom Ende des Alphabet verwendet (z.B: *u*, *v*, *w*)

Eigenschaft 4.6 - Konkatenation

Die Konkatenation von Wörtern hat folgende Eigenschaften:

- **Assoziativität**: Für beliebige Wörter $u,v,w \in \Sigma^*$ gilt:
  $u \cdot (v \cdot w) = (u \cdot v) \cdot w$
- ε ist **neutrales Element**: Für beliebige Wörter $w \in \Sigma^*$ gilt:
  $\varepsilon \cdot w = w \cdot \varepsilon = w$
- Addition der Längen: für beliebige Wörter $v, w \in \Sigma^*$:
  $|v \cdot w| = |v| + |w|$

Anmerkungen

- Assoziativität bedeutet: Wann man drei Wörter *u*, *v* und *w* aneinanderhängt, dann ist es egal, ob man erst *u* und *v* zusammenhängt und an das Ergebnis das dritte Wort *w* anhängt, oder ob man *u* nimmt und das Ergebnis der Konkatenation von *v* und *w* anhängt. In beiden Fällen entsteht die gleiche Zeichenfolge.

  Man kann sich also bei $u \cdot (v \cdot w)$ oder $(u \cdot v) \cdot w$ die Klammern sparen und schreibt deshalb üblicherweise nur *uvw.*
- Neutrales Element: Wenn man das leere Wort vorne oder hinten an ein anderes Wort hängt, ändert sich nichts.
- Hängt man zwei Wörter zusammen, dann addiert sich die Länge der beiden Teilwörter.

Aus der Arithmetik sind Sie die Notation x^n für $x \cdot x \cdot \ldots \cdot x$ (d.h. *n* mal *x* miteinander multipliziert) gewohnt. Bei der Konkatenation wird in analoger Weise ein "Potenzoperator" w^n verwendet für $w \cdot w \cdot \ldots \cdot w$ (*n* mal *w* aneinander gehängt). Da mathematische Definitionen mit "…" ungenau sind, wird der Potenzoperator für Wörter hier rekursiv definiert.

Definition 4.7 - "w hoch n"

Die ***n*-fache Konkatenation** eines Worts *w* wird mit w^n bezeichnet und ist folgendermaßen rekursiv definiert:

$$w^0 = \varepsilon$$
$$w^n = w \cdot w^{n-1}, \quad \text{für } n > 0$$

Wenn man ein Wort *w* null Mal nimmt, d.h. w^0 bildet, dann ergibt sich nicht "Nichts", sondern das leere Wort ε.

Folgend Eigenschaft von w^n sollte anschaulich klar sein:

Eigenschaft 4.8

$$|w^n| = n \cdot |w|$$

Wird *n* mal das Wort *w* aneinander gehängt, dann hat das Ergebnis *n* mal die Länge von *w*.

Aufgabe 4.9 - "w hoch n"

- gegeben: $\Sigma_1 = \{a,b\}$, Wort $w_1 = a$, Wort $w_2 = aab$
- Was ist

  w_1^0, w_1^1, w_1^2, w_1^3,
  w_2^0, w_2^1, w_2^2, w_2^3 ?

4.2 Sprachen

Nun zurück zur Frage: Was ist eine Sprache? Nachfolgend ist eine sehr universelle und auch sehr abstrakte Definition des Konzepts "(formale) Sprache" angegeben. Sie werden im weiteren Verlauf dann sehen, dass dieser Begriff, gerade weil er so simple und allgemein ist, für sehr vieles anwendbar ist.

Es muss an dieser Stelle aber auch darauf hingewiesen werden, dass wir uns im Bereich der formalen Sprachen nur mit einem Teilaspekt von Sprache beschäftigen - der *Syntax*.

Aus der Einführung in das Programmieren kennen Sie vielleicht die Begriffe *Syntax*, *Semantik* und *Pragmatik* im Zusammenhang mit Sprachen - nicht nur bei Programmiersprachen, sondern auch bei natürlichen Sprachen, wie z.B. Deutsch oder Englisch.

- Die **Syntax** einer Sprache beschreibt die Regeln, wie strukturell aus den Grundbestandteilen die "Äußerungen" der Sprache gebildet werden können.
- Die **Semantik** einer Sprache beschreibt die Bedeutung der in der Sprache formulierbaren Äußerungen.

- Die **Pragmatik** beschäftigt sich mit der Nutzung der Sprache - damit, wie die Möglichkeiten der Sprache situationsabhängig verwendet werden.

Definition 4.10 - Sprache

Eine (formale) **Sprache** *L* über einem Alphabet Σ ist eine **Menge von Wörtern** über Σ, d.h.

$$L \subseteq \Sigma^* .$$

Das üblicherweise verwendete Kürzel *L* für Sprachen kommt von engl. *language*.

Viele der kommenden Beispiele für Sprachen werden abstrakt und völlig sinnfreie (aber einfach) sein, d.h. ohne jegliche Semantik. Bei "echten" Sprachen ist mit der Syntax natürlich immer auch eine Semantik verbunden. Typischerweise hängt die Semantik dann vom strukturellen Aufbau der Äußerungen, d.h. vom syntaktischen Aufbau, ab.

Beispiel 4.11 - Sprachen

Einige Sprachen über Alphabet Σ = {a, b}:

- L_1 = {ε, ab, ba} endliche Sprache, besteht nur aus drei Wörtern.
- L_2 = {b, bb, bbb, bbbb, …} Sprache beliebig langer Folgen von b, unendliche Sprache
- L_3 = {} leere Sprache, enthält gar kein Wort
- L_4 = {ε} Sprache, die nur das leere Wort enthält
- L_5 = {ε, a, b, aa, bb, aaa, aba, bab, bbb, aaaa, abba, baab, bbbb, ...}

Natürliche Sprachen und Programmiersprachen als formale Sprachen

Eine Sprache ist also einfach nur eine Menge von Wörtern, die mit den Zeichen eines vorgegebenen Alphabets gebildet werden können. Diese Definition erscheint Ihnen vielleicht im ersten Moment eigenartig.

Bei natürlichen Sprachen gibt es den Begriff "Satz", bei Programmiersprachen spricht man von "Programmen" als grundlegende "Äußerungen", die in der Sprache formuliert werden. Wie passt das zur oben angegebenen Definition von Sprache?

Betrachten wir die Äußerung

Der Hund bellt laut.

dann stimmen Sie sicher zu, dass das Deutsch ist. Bei den folgenden Äußerungen

laut Hund Der. bellt

laaaudt Huxlakjsd der .&56% bxx§§xxx

ist klar, dass es sich nicht um korrektes Deutsch handelt.

Die Grundeinheit für Äußerungen in einer natürlichen Sprache sind Sätze. Manche Zeichenfolgen (= Sätze) sind also korrektes Deutsch, gehören somit zur Menge *Deutsch* dazu, andere Zeichenfolgen (= Sätze) sind nicht richtiges Deutsch und gehören zur Menge *Deutsch* nicht dazu. Ein *Satz* in *Deutsch* entspricht also einem *Wort* gemäß Terminologie der formalen Sprachen.

Wie sieht es mit Programmiersprachen, wie z.B. Java, aus? Die grundlegende "Äußerung" bei Programmiersprachen ist ein Programmtext, der in einer Quelltextdatei steht. Eine Quelltextdatei ist nichts anderes als eine Folge von Zeichen, also ein Wort im Sinn der Theorie der formalen Sprachen. Manche Quelltextdateien (= Zeichenfolgen = Wörter) sind korrektes Java, d.h. in der Menge der Sprache *Java* enthalten, andere Quelltextdateien sind kein korrektes Java und gehören zur Menge *Java* nicht dazu.

Beispiel 4.12 - Weitere Beispiele für Sprachen

- Menge der Zeit- oder Datumsangaben in einem bestimmten Format.
- Menge der wohlgeformten XML-Dateien (siehe später)

Definition und Analyse von Sprachen

Sie sehen also, dass diese sehr abstrakte Definition für Sprachen als Menge von Wörtern sehr universell ist. Sie ist in der Tat sowohl die Basis für die Formalisierung von natürlichen Sprachen in der Linguistik als auch für die Definition und Implementierung von Programmiersprachen in der Informatik.

Diese Definition alleine ist allerdings noch nicht allzu nützlich. Will man (künstliche) Sprachen verwenden, hat man zwei Herausforderungen zu bewältigen:

(1) **Definitionsproblem**: Wie kann man exakt und unmissverständlich definieren, was zur Sprache dazugehört und was nicht? Welche Sprache ist beispielsweise mit L_5 aus *Beispiel 4.11 - Sprachen* gemeint? Haben Sie erkannt, wie es bei "…" weitergehen soll?
(Ich hatte die Sprache der Palindrome im Kopf, d.h. aller Wörter, die von hinten nach vorne gleich sind wie von vorne nach hinten).

Mit Umgangssprache oder mit den Mitteln der Mengenlehre ist es schwierig, reale Sprachen exakt zu definieren. Sie werden in den folgenden Kapitel mit *regulären Ausdrücken* und *kontextfreien Grammatiken* zwei Methoden kennenlernen, um komplexe Sprachen exakt spezifizieren zu können.

(2) **Analyseproblem**: Wie kann geprüft werden, ob eine Äußerung/ein Wort (z.B. ein Satz in einer natürlichen Sprache oder ein Programm in einer Programmiersprache) korrekt ist, d.h. zur Sprache dazugehört?

Wenn die Syntax formal definiert wurde, dann können darauf aufbauen auch formale Analyseverfahren entwickelt werden. Auch das werden wir in späteren

Kapiteln behandeln.

Doch bevor Sie diese Methoden lernen, hier noch eine Aufgabe, eine Sprache auf Ebene der Mengenlehre zu beschreiben.

Aufgabe 4.13 - Sprache der Zeitangaben

Zeitangaben in der Form "12:49 Uhr" zwischen 00:00 Uhr und 23:59 Uhr stellen eine Sprache dar. Wie könnte diese Sprache als Menge von Wörtern definiert werden?

- Wählen Sie zunächst ein passendes Alphabet.
- Geben Sie dann die Menge der Wörter an, die zur Sprache gehören.

Je nach Wahl des Alphabets kann es hier durchaus unterschiedliche Lösungen geben

4.2.1 Operationen für Sprachen

Auch auf Sprachen (= Mengen von Wörtern) können Operationen definiert werden. Diese Operationen benötigen wir im folgenden Kapitel, um die Bedeutung sog. *regulärer Ausdrücke* exakt definieren zu können.

Definition 4.14 - Operationen auf Sprachen

- Die **Konkatenation** $L_1 \cdot L_2$ zweier **Sprachen** L_1 und L_2 ist

  $$L_1 \cdot L_2 = \{ vw \mid v \in L_1, w \in L_2 \}.$$

- Für eine Sprache L ist L^n ("L hoch n") wie folgt definiert:

  $$L^0 = \{ \varepsilon \}$$
  $$L^n = L \cdot L^{n-1}, \text{ für } n > 0$$

- Die **Kleene'sche Hülle** L^* (gesprochen "L-Stern") einer Sprache L ist definiert durch

  $$L^* = \bigcup_{n \geq 0} L^n = L^0 \cup L^1 \cup L^2 \cup \ldots$$

Anmerkung

- $L_1 \cdot L_2$ sind alle Wörter uv, die durch Konkatenation von einem Teilwort u aus L_1 am Anfang und einem Rest v aus Sprache L_2 gebildet werden können.
- L^n ist die Menge aller Wörter, die man durch Konkatenation von n beliebigen Wörtern aus L bilden kann. Das gleiche Wort kann dabei auch mehrfach verwendet werden. Die Kombination von 0 Wörtern ergibt nur das leere Wort ε.
- **L*** (*Kleene'scher Stern-Operator*) ist die Menge aller Wörter, die man bilden

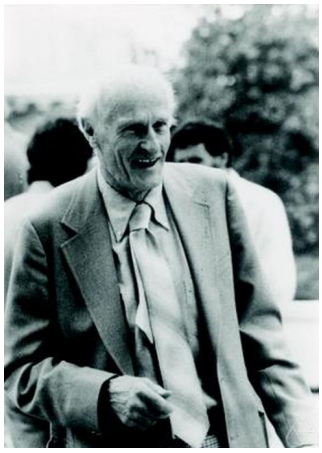
kann, indem man beliebig viele Wörter aus L aneinanderhängt (es können dabei auch mehrfach die gleichen Wörter gewählt werden).

$L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup ...$

L^0 sind alle Wörter, die man aus 0 Wörtern von L bilden kann (das ist nur ε)
L^1 sind alle Wörter, die man aus 1 Wort von L bilden kann (d.h. das ist L)
L^2 sind alle Wörter, die man aus 2 beliebig gewählten Wörtern von L bilden kann
L^3 sind alle Wörter, die man aus 3 beliebig gewählten Wörtern von L bilden kann
...

Alles, was man mit endlich vielen Wörtern aus L bilden kann, ist also bei L^* dabei.
Zu beachten: das leere Wort ε ist für jede Sprache L bei L^* immer mit dabei!

Anmerkung - Stephen Cole Kleene (1909 – 1994)



Der Stern-Operator ist nach dem amerikanischer Mathematiker und Logiker *Stephen Cole Kleene* benannt. Er ist einer der Begründer der theoretischen Informatik und hat wichtige Beiträge zur Theorie der formalen Sprachen geleistet (u.a. auch diesen Stern-Operator eingeführt)

Aufgabe 4.15 - Operationen auf Sprachen

Gegeben: Alphabet $\Sigma = \{0, 1\}$

$L_1 = \{00, 11\}$

$L_2 = \{1, 11, 111, 1111\}$

bestimmen Sie folgende Sprachen:

(1) $L_1 \cdot L_2$

(2) $L_1^2 = L_1 \cdot L_1$

(3) L_1^3

(4) L_1^*

4.3 Exkurs: XML

Nach diesen zunächst sehr abstrakt-theoretischen Vorarbeiten wird hier noch ein Ausblick auf etwas sehr Praktisches gegeben, das in fast jedem Software-Projekt oder in jeder Abschlussarbeit irgendwie verwendet wird: Die Datenbeschreibungssprache XML.

XML (E**x**tensible **M**arkup **L**anguage) ist eine Standard dafür, wie hierarchisch strukturierte Daten für den Datenaustausch zwischen Systemen oder auch zur Speicherung in Textform beschrieben werden können.

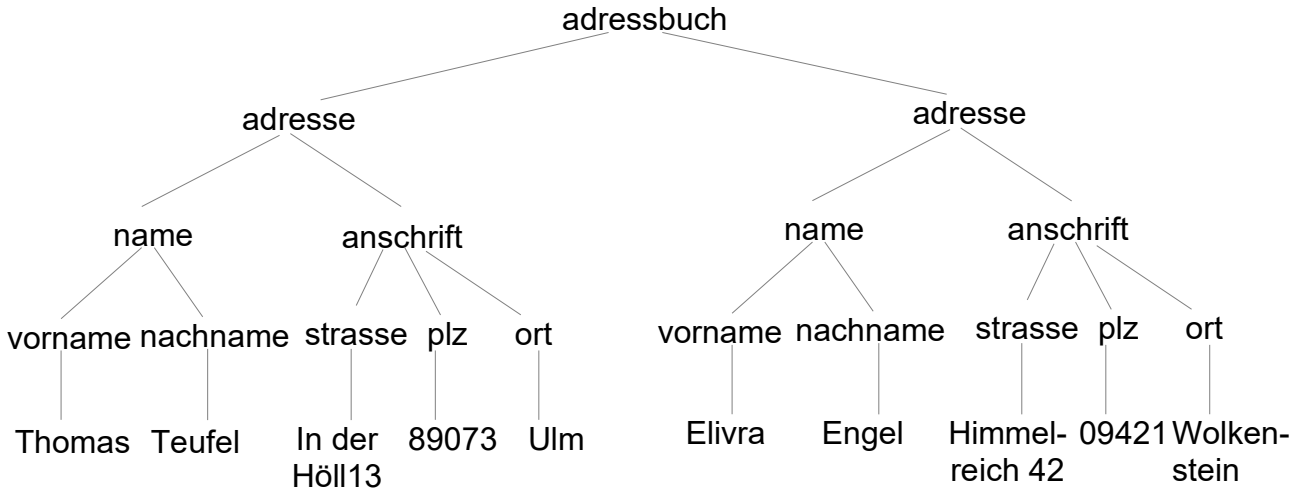
*XML = E**x**tensible **M**arkup **L**anguage*

- 1996 definiert durch das World Wide Web Consortium (W3C)
- Weit verbreitete, plattformunabhängige Beschreibungssprache für strukturierte Daten. War ursprünglich vor allem für den Bereich der Web-Anwendungen vorgesehen, wird heutzutage so gut wie in jedem Anwendungsumfeld verwendet.
- XML war ursprünglich orientiert an der Dokumenten-Layout-Beschreibungssprache SGML (Standardized Generalized Markup Language) (aus der auch HTML zur Beschreibung von Webseiten abgeleitet wurde).

4.3.1 Serialisierung strukturierter Daten

Serialisierung = Abbildung strukturierter Daten in eine sequentielle Darstellungsform (Folge von Zeichen oder Bytes)

Hierarchisch strukturierte Daten kann man als Bäume betrachten, wie z.B. in folgendem Beispiel eines Adressbuchs, das mehrere Adressen enthält:



Beispiel 4.16 - XML

Der oben als Baum dargestellt Adressbuchinhalt würde folgendermaßen in XML repräsentiert:

```
<adressBuch>
    <adresse>
        <name>
            <vorname>Thomas</vorname>
            <nachname>Teufel</nachname>
        </name>
        <anschrift>
            <strasse>In der Höll 13</strasse>
            <plz>89073</plz>
            <ort>Ulm</ort>
        </anschrift>
    </adresse>
    <adresse>
        <name>
            <vorname>Elvira</vorname>
            <nachname>Engel</nachname>
        </name>
        <anschrift>
            <strasse>Im Himmelreich 42</strasse>
            <plz>09421</plz>
            <ort>Wolkenstein</ort>
        </anschrift>
    </adresse>
</adressBuch>
```

Die wesentlichen Beschreibungselemente sind sog. *XML-Elemente*, die jeweils durch einen Anfangs-*Tag* und einen End-Tag (engl. *tag* =Etikett, Auszeichner, Schildchen) geklammert werden, z.B.

`<adresse>` ... `</adresse>`

Eine Baumstruktur ergibt sich, da solche Elemente aufeinander folgen oder ineinander geschachtelt sein können.

Ausblick: Anwendungsbeispiele für XML

XML wird in sehr vielen Anwendungsbereichen verwendet. Hier sind nur einige Beispiele aufgeführt [zur Veranschaulichung, muss nicht für die Klausur gelernt werden].

- **XHTML**: Beschreibung von Web-Seiten
- **SVG** (Scalable Vector Graphics): zweidimensionale Vektorgraphik
- **ODF** (Open Document Format): genormtes Austauschformat für Bürodokumente
- **MathML** (Mathematical Markup Language): Dokumentenformat zur

Darstellung mathematischer Formeln.

- **MusicXML**: offenes Dateiformat zum Austausch von Musiknoten.
- Beschreibung der strukturellen Aufbaus von Benutzeroberflächen z.B. in JavaFX, Android, ...
- **WSDL**: (Web Services Description Language): Schnittstellen-Beschreibung für Web-Services
- Nachrichtenformat bei Web-Services oder in Microservice-Architekturen.
- Geodaten im **OpenStreetMap**-Projekt
- Austauschformat für **SEPA**-Lastschriften und -Überweisungen im bargeldlosen Zahlungsverkehr

4.3.2 Wohlgeformtheit

Die folgende Definition gibt die Grundregeln an, die jedes XML-Dokument erfüllen muss. Durch diese Regeln wird sichergestellt, dass ein XML-Dokument immer eine Baumstruktur repräsentiert.

Definition 4.17 - Wohlgeformtheit für XML

Ein Text ist ein **wohlgeformter XML-Text**, wenn er folgende Regeln erfüllt:

- Ein **XML-Text** besteht aus genau einem XML-Element (dem sog. *Wurzelelement*)
- Ein ***XML-Element*** beginnt mit einem Anfangstag <*tag*> und endet mit einem gleichnamigen Endtag </ *tag*>.
  Dazwischen steht eine (eventuell leere) Folge von XML-Elementen und elementaren Texten.
- **Elementare Texte** können beliebige Zeichenfolgen sein, die aber keine Tags enthalten.

Der erste Punkt der Definition besagt, dass es *ein* Baum mit *einer* Wurzel ist, so dass alles dieser Wurzel untergeordnet ist (d.h. in diesem XML-Element enthalten ist).

Der zweite Punkt besagt, dass eine XML-Element jeweils mit einem Anfangstag beginnt und dem entsprechend benannten Endtag </...> endet. Dazwischen können beliebig viele weiter XML-Elemente oder elementare Texte stehen, auch Texte und Elemente beliebig gemischt.

- Ein XML-Element entspricht somit einem Teilbaum.
- Ein elementarer Text entspricht dem Blatt des Baums.

Aufgabe 4.18 - XML-Dokumente

- Welche der folgenden Texte sind wohlgeformt? (ggf. Fehler korrigieren)
- Welche Baumstruktur wird repräsentiert?

(1)

```
<a>
   <b>
      <c> 1 </c>
      <d> 2 </d>
   </b>
   <e> 3 </e>
<a>
```

(2)

```
<a>
      <b> 1 </b>
      <b> 2 </b>
</a>
<a>
      <c> 3 </c>
</a>
```

(3)

```
<a>
   <b>
      1
      <b> 2 </b>
   </b>
   <a> 3 </a>
</a>
```

(4)

```
<a>
   <b>
      <c> 1 </b>
      <d> 2 </c>
   </b>
</a>
```

4.3.3 Ausblick: Weitere XML-Details

Aus Sicht der Theorie ist vor allem der Zusammenhang zwischen XML-Dokumenten und Bäumen interessant. XML bietet aber noch einige weiter Möglichkeiten, die bei praktischen Anwendungen wichtig sind, hier aber nur kurz erwähnt werden sollen:

- Anfangstags können mit Attributen versehen werden, um zusätzliche Informationen mit einem Knoten des Baums zu verbinden.

  ```
  <person alter="24">  … </person>
  ```

- Steht zwischen Anfangstag und Endtag kein Inhalt, kann das Element zu einem Tag mit < ... /> zusammengefasst werden

  ```
  <image name="hochschule.jpg" />
  ```

  entspricht

  ```
  <image name="hochschule.jpg"></image>
  ```

- Eine XML-Datei beginnt immer mit Angaben zu XML-Version, Zeichencode und ggf. weiteren Metainformationen, z.B. Strukturdefinitionen.

  ```
  <?xml version=1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE adressBuch SYSTEM "adressBuch.dtd">
    …
  ```

  Danach folgt dann das Wurzelelement als eigentlicher Inhalt.

Fazit zu Lektion 6

Diese Fragen sollten Sie nun beantworten können

- Was versteht man unter einem Wort und einer Sprache?
- Was ist die Konkatenation von Wörtern?
- Was ist eine formale Sprache?
- Wie ist die Komposition von Sprachen definiert?
- Was ist die Kleene'sche Hülle einer Sprache?
- Wie wird XML zur Beschreibung von hierarchisch strukturierten (baumartigen) Daten verwendet?
- Welche Regeln muss ein wohlgeformtes XML-Dokument erfüllen?