Morphologie und Buchstabenbäume

Übersicht

- Vollformen
- Morphologie als Wortgrammatik
  - Simple DCG als Wortgrammatik: Stämme und Endungen, Flexionsklassen
  - Schnittstelle zwischen Syntax/Morphologie
  - Grenzen einfacher konkatenativer Morphologie
  - Überlegungen zur Effizienz
- Tries: Buchstabenbäume
  - Datenstruktur Buchstabenbäume (tries)
  - find_word/3: Wörter finden im Buchstabenbaum
  - find_morph/4: Wortteile finden im Buchstabenbaum
  - Stammbäume und Suffixbäume
- Konkatenative Morphologie mit Buchstabenbäumen

Motivation

- **Womit befasst sich die Morphologie?**

  Wortstruktur und ***Wortbildung***!
  - Flexion

    trenn+en, trenn+e, trenn+test, trenn+ten, ge+trenn+t, trenn+end,…
  - Komposition

    Fruchtbarkeit+s+gott,

    Fruchtbarkeit+s+göttinnen+verehrung+s+zeremonie+n+meister,…
  - Derivation

    Frucht, frucht+en, frucht+bar, un+frucht+bar, Un+frucht+bar+keit,…
- **Wie viele Wörter gibt es?**

  der viermillioneneinhunderttausendundzweite Schluck

  das In-der-Schlange-Stehen

Vollformen-Lexikon

Vollformen-Lexikon als Prolog-Datenbank

- für jede Wortform alle möglichen Funktionen angeben
  - lexikalisch, morphosyntaktisch, semantisch, pragmatisch, …
- **Nachteile**
  - immer unvollständig wegen produktiven Wortbildungen
  - je nach Sprache aufwändiger
    - für flexionsarmes Englisch machbar
    - für flexionsreicheres Deutsch schon anspruchsvoller (pro Substantiv <8 Formen)
    - für Finnisch problematisch: Finnische Verben haben ~12'000 Formen
  - ▸ Aber: Leistungsfähigere Computersysteme ermöglichen Dinge, die vor wenigen Jahren nicht machbar waren!

```
lex(kind, n, nom, sg, 'KIND').
lex(kindes, n, gen, sg, 'KIND').
lex(kinde, n, dat, sg, 'KIND').
lex(kind, n, dat, sg, 'KIND').
lex(kind, n, akk, sg, 'KIND').
lex(kinder, n, nom, pl, 'KIND').
lex(kinder, n, gen, pl, 'KIND').
lex(kindern, n, dat, pl, 'KIND').
lex(kinder, n, akk, pl, 'KIND').
```

Flexionsformen von »Kind«

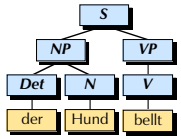
Morphologie als "Wortgrammatik"

- **Satzgrammatik =**

  Wörter +

  syntaktische Kategorien +

  Verknüpfungsregeln

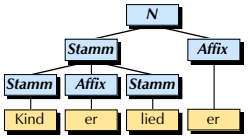


- **Wortgrammatik =**

  Morpheme +

  morphologische Kategorien +

  Verknüpfungsregeln



▸ *Beide "Grammatiken" können im DCG-Formalismus notiert werden…*

DCG: Trennen von Stamm und Endung

Das Auftrennen bzw. Zusammenfügen von Stamm und Endung als DCG-Regeln:

```
n_morph(Kas, Gen,Num) -->
  n_stamm(Gen),
  n_endung(Kas, Num).
```

Verknüpfungsregel

```
n_stamm(s) --> "kind".
n_stamm(s) --> "lied".
n_stamm(s) --> "bild".
```

Nominalstämme

```
n_endung(nom, sg) --> "".
n_endung(gen, sg) --> "es".
n_endung(dat, sg) --> "".
n_endung(dat, sg) --> "e".
n_endung(akk, sg) --> "".
n_endung(nom, pl) --> "er".
n_endung(gen, pl) --> "er".
n_endung(dat, pl) --> "ern".
n_endung(akk, pl) --> "er".
```

Nominalendungen

Flexionsklassen

Innerhalb eines Genus können unterschiedliche Flexionsklassen (-paradigmen) auftreten.

```
n_morph(Kas, Gen, Num) -->
  n_stamm(Klasse, Gen),
  n_endung(Klasse, Kas,
Num).
```

Verknüpfungsregel

```
n_stamm(1, s) -->
"kind".
n_stamm(1, s) -->
"lied".
n_stamm(1, s) -->
"bild".
n_stamm(2, s) -->
```

Nominalstämme

```
n_endung(1, nom, sg) --> "".
n_endung(1, gen, sg) --> "es".
n_endung(1, dat, sg) --> "".
n_endung(1, dat, sg) --> "e".
n_endung(1, akk, sg) --> "".
n_endung(1, nom, pl) --> "er".
n_endung(1, gen, pl) --> "er".
n_endung(1, dat, pl) --> "ern".
n_endung(1, akk, pl) --> "er".
```

Nominalendungen Klasse Nr. 1

```
n_endung(2, nom, sg) --> "".
n_endung(2, gen, sg) --> "es".
n_endung(2, dat, sg) --> "".
n_endung(2, dat, sg) --> "e".
n_endung(2, akk, sg) --> "".
n_endung(2, nom, pl) --> "e".
n_endung(2, gen, pl) --> "e".
n_endung(2, dat, pl) --> "en".
n_endung(2, akk, pl) --> "e".
```

Nominalendungen Klasse Nr. 2

Morpheme oder Buchstabenlisten?

```
n_stamm(1, s) --> "kind".

n_stamm(1, s) -->
[107,105,110,100].

n_stamm(1, s) --> [107],[105],[110],[100].
```

Morpheme + Kompositionsregeln?

- Zwischen zwei " eingeschlossene Zeichenketten stehen für die Liste der Zeichenkodes.
- Morpheme sind eigentlich Buchstabenlisten.
- ▶ Beim morphologischen Parsen müssen wir gleichzeitig noch Morphemgrenzen erkennen…

Vgl. Unterlagen zu "Ein- und Ausgabe" im WS

```
?- Kette = "kind".
Kette = [107,105,110,100]
```


DCG-Schnittstelle Syntax/Morphologie

Schnittstelle zwischen Syntax und Morphologie

- **Syntax** arbeitet mit **Atomen**
- **Morphologie** arbeitet mit **Zeichenkodelisten**

`brotes` ≠ `"brotes"`

Umwandlung mit atom_codes/2 (iso) bzw. atom_chars/2

- Überführen von Atome zu Listen aus Zeichenkodes und zurück.

```
?- atom_chars(brotes, Buchstaben).
Buchstaben = [98,114,111,116,101,115]
?- atom_chars(Wort, [98,114,111,116,101,115]).
Wort = brotes
```

DCG-Schnittstelle Syntax/Morphologie

```
np(Kas, Gen, Num) -->
   det(Kas, Gen, Num),
   n(Kas, Gen, Num).

det(nom, s, sg) --> [das].
det(nom, s, pl) --> [die].

n(Kas, Gen, Num) --> [Wort],
   { atom_chars(Wort, Buchstaben),
     phrase(n_morph(Kas, Gen, Num), Buchstaben)
   }.
```

Vgl. Unterlagen zu "DCGs" im Wintersemester

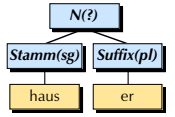
Anfragen

```
?- phrase(np(Kasus, Gen, Numerus), [die, brote]).
Kasus = nom, Gen = s, Numerus = pl
?- phrase(np(Kasus, Gen, Numerus), [die, broter]).
no
```

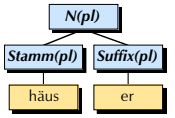

Grenzen konkatenativer Morphologie

Umlautung im Plural

- Stammänderungen stellen simple konkatenative Ansätze vor konzeptuelle Probleme!
- Ausweg: Stammmorpheme werden als lexikalische Allomorphe eines Lemmas aufgefasst, die unterschiedliche Numeri ausdrücken können.
- n_stamm(Klasse, Genus, Numerus, Lemma)



*hauser**



häuser

```
n_stamm(1, s, sg, 'HAUS') --> "haus".
n_stamm(1, s, pl, 'HAUS') --> "häus".
n_stamm(1, s, _,  'BILD') --> "bild".
n_stamm(2, s, _,  'BROT') --> "brot".
```

Nominalstämme

Erweiterte Schnittstelle

Komplexe morphologische Kategorien

- transportieren die gewünschte Information nach Aussen
- gleichen Kongruenz ab

```
n_morph(Kas, Gen, Num, Lemma) -->
  n_stamm(Klasse, Gen, Num, Lemma),
  n_endung(Klasse, Kas, Num).
```

Verknüpfungsregel

```
n(Kas, Gen, Num, Lemma) --> [Wort],
   { atom_chars(Wort, Buchstaben),
     phrase(n_morph(Kas, Gen, Num, Lemma), Buchstaben)
   }.
```

```
?- phrase(n(Kas,Gen,Num,Lem), [häuser]).
Kas = nom, Gen = s, Num = pl, Lem = 'HAUS' ;
Kas = gen, Gen = s, Num = pl, Lem = 'HAUS' ;
Kas = akk, Gen = s, Num = pl, Lem = 'HAUS' ;
no
```


Effizienz

Wie effizient ist diese morphologische Analyse?

- Buchstaben-Morphologie mit dem Standard-DCG-Parser

```
n_stamm(1, s) -->
"kind".
```

```
n_stamm(1, s, A, B) :-
        'C'(A, 107, C),
        'C'(C, 105, D),
        'C'(D, 110, E),
        'C'(E, 100, B).
```

```
?- phrase(n_stamm(K, G), "kiste").
2 2 Call: n_stamm(K,G,[k,i,s,t,e],[])
3 3 Call: 'C'([k,i,s,t,e],k,R1)
3 3 Exit:
'C'([[k,i,s,t,e],k,[i,s,t,e])
4 3 Call: 'C'([i,s,t,e],i,R2)
4 3 Exit: 'C'([i,s,t,e],i,[s,t,e])
5 3 Call: 'C'([s,t,e],n,R3)
5 3 Fail: 'C'([s,t,e],n,R3)
2 2 Fail: n_stamm(K,G,[k,i,s,t,e],[])
no
```

Geschönter Trace

Effizienz im Grossen

Verhalten bei grossen Stammlexika

- Was passiert bei der Analyse von "kiste", wenn es tausende Einträge von "aal", "kinn" bis "zinn" gibt?

Überlegungen zum Aufrufverhalten…

- Bei ausschöpfender Suche eines Stammes *S* wird jede Stammklausel (n_stamm/4) einmal aufgerufen.
- 'C'/3 wird für jede Stammklausel einmal aufgerufen.
- Für jeden Buchstaben von jedem Präfix, den Stämme mit *S* gemeinsam haben, wird 'C'/3 einmal aufgerufen.

Das Programm verhält sich einiges ineffizienter als es sein müsste!

Kompakte Lexika: *Letter Trees/Tries*

Lexikon als Buchstabenbäume

- Eine effiziente Methode, Lexikas **kompakt** zu speichern und **buchstabenweise** auf die Einträge zuzugreifen, sind Buchstabenbäume (engl. *letter tries* von *letter re**trie**val)*

Buchstabenbäume bestehen aus

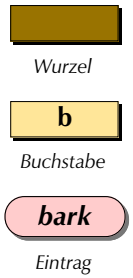
- einer **Wurzel** mit
- **Ästen**, an denen ein Buchstabe und/oder Lexikoneinträge hängen,
- wobei von den Buchstaben wieder Äste abzweigen können.

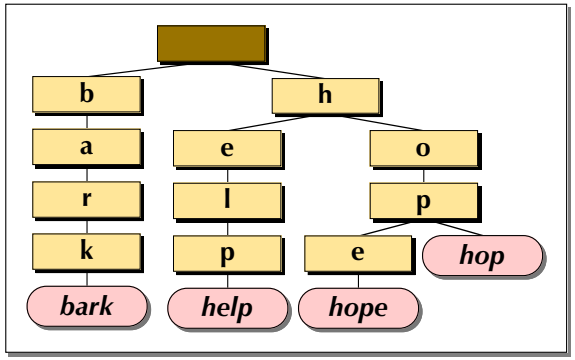
Äste

- von Buchstabenbäumen sind meist alphabetisch sortiert angeordnet.

Ein englischer Buchstabenbaum

Legende





Letter Tree als Prolog-Term

Ein Buchstabenbaum ist rekursiv aus Listen gebaut.

- Ein **Baum** ist eine Liste von **Ästen**.
- Ein **Ast** ist entweder ein Lexikoneintrag oder eine Liste der Form [Buchstabe|Baum].

```
ltree([ [b, [a, [r, [k, lex(bark)]]]],
        [h, [e, [l, [p, lex(help)]]],
            [o, [p, lex(hop),
                    [e, lex(hope)]]]]
      ]).
```

- Das Entfernen des Buchstabens aus einem Ast ergibt einen Baum!
  - *Richtige Lexikoneinträge müssten natürlich alle nötigen Informationen beinhalten…*

Einfache Suche: find_word/3

find_word/3: ein simples Zugriffsprädikat

- find_word(Buchstabenliste, Baum, Eintrag) findet den zur Buchstabenliste *L* passenden Eintrag *E* im Baum *T*

Rekursionsschritt

- Solange *L* nicht leer ist, finde einen passenden Teilbaum mit dem Anfangsbuchstaben von *L* und suche darin rekursiv nach dem Rest von *L*.

```
find_word([Char|Chars], Tree, LexEntry) :-
   member([Char|Subtree], Tree),
   find_word(Chars, Subtree, LexEntry).
```


Beispiel: Rekursionsschritte

?- ltree(T), find_word([h,o,p], T, lex(I)).

1. Suche [h,o,p] in

```
[ [b, [a, [r, [k, lex(bark)]]]],
  [h, [e, [l, [p, lex(help)]]],
      [o, [p, lex(hop),
              [e, lex(hope)]]]]]
```

2. Suche [o,p] in

```
[ [e, [l, [p, lex(help)]]],
  [o, [p, lex(hop),
          [e, lex(hope)]]]]
```

3. Suche [p] in

```
[ [p, lex(hop),
      [e, lex(hope)]]]
```

4. Suche [ ] in

```
[ lex(hop), [e, lex(hope)]]
```


Einfache Suche: find_word/3

Abbruchbedingung

- Wenn *L* leer ist, suche einen Lexikoneintrag in diesem Ast.

```
find_word([], Tree, LexEntry) :-
   member(LexEntry, Tree).
```

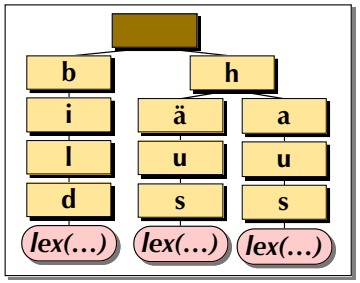
- Damit Lexikoneinträge nicht mit Ästen verwechselt werden, müssen sie einen andern Hauptfunktor haben als den Listenkonstruktor '.'/2.

```
?- find_word([], [lex(hop),[e,lex(hope)]], lex(I)).
I = hop ;
no
```


Ein deutscher Stammbaum

Ein Stammlexikon

- Die Lexikoninformation kann beliebig komplex werden…




```
n_stamm_ltree([[b,[i,[l,[d, n_stamm(1,s,_,'BILD')]]]],
               [h,[a,[u,[s, n_stamm(1,s,sg,'HAUS')]]],
                  [ä,[u,[s, n_stamm(1,s,pl,'HAUS')]]]]]).
```

Suche von Stämmen

Um Stämme in Wörtern zu finden, darf nicht die ganze Buchstabenliste konsumiert werden.

- Was übrig bleibt, wird als Restliste zurückgegeben.

```
find_morph([Char|Chars], Tree, LexEntry, Rest) :-
   member([Char|Subtree], Tree),
   find_morph(Chars, Subtree, LexEntry, Rest).
find_morph(Rest, Tree, LexEntry, Rest) :-
   member(LexEntry, Tree).
```

```
?- n_stamm_ltree(_T),
find_morph([h,ä,u,s,e,r],_T,n_stamm(K,Gen,Num,L), R).
Kla = 1, Gen = s, Num = pl, L = 'HAUS', R = [e,r] ? ;
no
```


Ein deutscher Suffixbaum

Flexionssuffixe

- können ebenfalls als Buchstabenbaum dargestellt werden.
- sind in eigenen Klassen: n_endlung_ltree(Flexionsklasse, Suffixtree)
- Leere Morpheme erscheinen direkt auf der Wurzelebene.

```
n_endung_ltree(1,
   [n_endung(nom,sg),
    n_endung(dat,sg),
    n_endung(akk,sg),
    [e,n_endung(dat,sg),
       [r, n_endung(nom, pl),
           n_endung(gen,pl),
           n_endung(akk,pl),
           [n, n_endung(dat,pl)]],
       [s, n_endung(gen,sg)]]]).
```


Morphologische Analyse mit Bäumen

Konkatenative morphologische Analyse kann effizient mit Buchstabenbäumen durchgeführt werden.

- Zusätzliche Effizienz ist möglich, wenn alphabetische Sortierung bei der Suche berücksichtigt wird.

```
n_morph(Word, Kas, Gen, Num, Lem) :-
   n_stamm_ltree(T1),
   find_morph(Word, T1, n_stamm(Kla,Gen,Num,Lem), Suffix),
   n_endung_ltree(Kla, T2),
   find_morph(Suffix, T2, n_endung(Kas, Num), []).
```

```
?- n_morph([h,ä,u,s,e,r], Gen, Kas, Num, Lem).
Gen = nom, Kas = s, Lem = 'HAUS', Num = pl ? ;
Gen = gen, Kas = s, Lem = 'HAUS', Num = pl ? ;
Gen = akk, Kas = s, Lem = 'HAUS', Num = pl ? ;
no
```


Ausblick und Literaturhinweis

Das Konstruieren von Buchstabenbäumen

- ist eine etwas mühsame Arbeit.
- überlässt man am besten einem Programm, das aus lexikalischen Fakten die entsprechenden Bäume erzeugt.

```
n_stamm(bild,1,s,_,'BILD').
n_stamm(haus,1,s,sg,'HAUS').
n_stamm(häus,1,s,pl,'HAUS').
```

```
n_stamm_ltree(
[[b,[i,[l,[d, n_stamm(1,s,_,'BILD')]]]],
 [h,[a,[u,[s, n_stamm(1,s,sg,'HAUS')]]],
    [ä,[u,[s, n_stamm(1,s,pl,'HAUS')]]]]]).
```

Literatur zu Morphologie und Buchstabenbäumen

- Covington, M. (1994): Seiten 263ff.