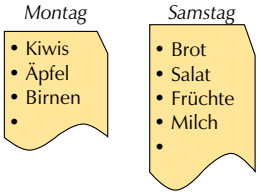
Listen

Übersicht

Listen – Die wichtigste nicht-nummerische Datenstruktur

- beliebige Länge und fixe Reihenfolge
  - Listen vs. n-stellige Terme
- Spezialnotation
  - Klammerschreibweise
  - Listenrest-Strich
- Listen-Unifikation
- Der rekursive Aufbau von Listen
  - Rekursive Datenstruktur – Geschachtelte Termstruktur
  - Die interne Punktdarstellung für Listen
- Listen als Elemente

Einkaufslisten und Wortlisten



Einkaufslisten

Yes.
I know.
Peter saw Mary.

Sätze als Wortlisten

Listen werden geführt für…

- Dinge zum Einkaufen
- Buchstaben eines Wortes
- Wörter in einem Satz
- …

☑ **Beliebige Länge**
▸ Wieviele Elemente?
☑ **Strikte Reihenfolge**
▸ Wie geordnet?

Problem: Erstes Wort von Sätzen

```
satz(yes)
satz(i,know)
satz(peter,saw,mary)
…
```

Datenstruktur für Sätze

```
anfang(satz(W1), W1).
anfang(satz(W1,_), W1).
anfang(satz(W1,_,_), W1).
…
```

Prädikat für Satzanfang

Ansatz: Listenelemente = Argumente komplexer Terme

- Sätze: Term der Stelligkeit *L* repräsentiert Satz der Länge *L*
- Prädikate: Satzverarbeitende Prädikate müssen soviele Fälle berücksichtigen, wie es unterschiedlich lange Sätze gibt.
  - Ein einfaches Prädikat zum Bestimmen des Satzanfangs ist mühsam zum Definieren, da jede Satzlänge eine eigene Klausel braucht!

▸ Schwierigkeit: **Fixe Stelligkeit komplexer Terme**

Gewünschte Eigenschaften

Wunschliste für Listen

- Listen nehmen in Prädikaten und Termen nur ***eine*** Argumentstelle ein!
- Listen können beliebig viele Elemente enthalten.
- Listenelemente sind geordnet.
- Listenelemente können mehrfach vorkommen.

  Beispielliste: Wörter im Satz "Wenn Fliegen hinter Fliegen fliegen, fliegen Fliegen Fliegen nach."
- Listen können auch keine Elemente enthalten.
  - Sie können leer sein (sog. *leere Liste*)

Trick I: Klammerschreibweise

Für Listen gibt es eine eigene Schreibweise

- Elemente sind zwischen eckigen Klammern eingeschlossen
- Elemente sind durch Kommata getrennt
- Elemente sind beliebige Terme

```
[kiwi]
```
Ein Element

```
[]
```
Kein Element
(leere Liste)

```
[apfel,Frucht]
```
Zwei Elemente
(ein Atom und eine Variable)

Die halbe Lösung des Problems

```
satz([yes])
satz([i,know])
satz([peter,saw,mary])
…
```
Datenstruktur für Sätze

```
anfang(satz([W1]), W1).
anfang(satz([W1,_]), W1).
anfang(satz([W1,_,_]), W1).
…
```
Prädikat für Satzanfang

Ansatz: Sätze als Wortlisten

- Sätze: Liste mit *L* Elementen repräsentiert Satz der Länge *L*.
  - *Dank Klammerschreibweise brauchen die Wörter von Sätzen nur noch eine Argumentstelle. Der Funktor satz/1 reicht aus!*
- Prädikate: Satzverarbeitende Prädikate müssen weiterhin Sätze unterschiedlicher Länge als Fälle unterscheiden.
  - *Definition für Satzanfangsprädikat bleibt mühsam, da jede Satzlänge weiterhin eine eigene Klausel braucht!*
- ▶ Schwierigkeit: **Fixe Länge von Listen**

Trick II: Der Listenrest-Strich

Der Listenrest-Strich dient dazu, Listen *beliebiger* Länge zu bezeichnen.

- Vor dem Strich steht mindestens ein Anfangselement.
- Nach dem Strich steht der Listenrest.

Der Listenrest

- ist normalerweise eine Variable, die instantiiert werden kann!
- ist selbst eine Liste, die Restliste!

```
[kiwi,apfel|Fruechte]
```
Mindestens zwei Elemente
(zwei Atome und eine Variable als Listenrest)

Die ganze Lösung des Problems

```
satz([yes])
satz([i,know])
satz([peter,saw,mary])
…
```
Datenstruktur für Sätze

```
anfang(satz([W1|_]), W1).
```
Prädikat für Satzanfang

Listenrest-Strich und Klammerschreibweise lösen das Problem des Satzanfangs mit *einer* einzigen Klausel!

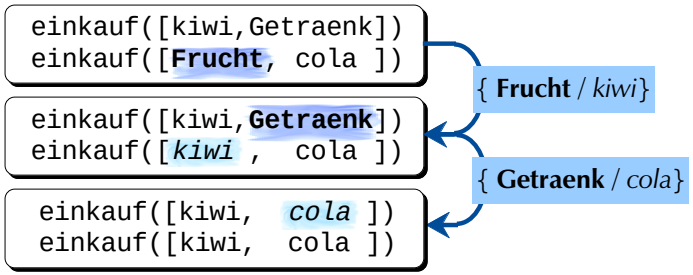
- *Pattern Matching* mit Listenrest-Strich extrahiert aus Sätzen verschiedenster Länge das erste Element!

```
?- anfang(satz([yes]), Anfang).
Anfang = yes
?- anfang(satz([mary,saw,peter]), Anfang).
Anfang = mary
```

Unifikation von Listen

Zwei Listen sind unifizierbar, falls

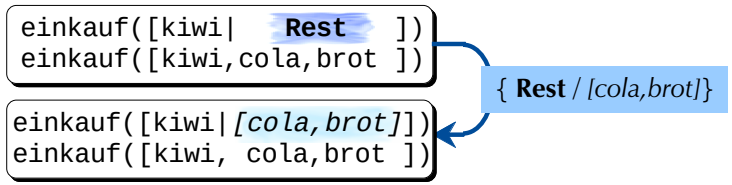
- die einzelnen Elemente paarweise unifizierbar sind
- die Länge beider Listen übereinstimmt



Unifikation von Listen

Eine Liste mit variablem Listenrest und eine Liste ohne eine Listenrest sind unifizierbar, falls

- die Anfangselemente paarweise unifizierbar sind
- der Listenrest mit der Liste der restlichen Elemente unifiziert wird



Listenrest-Strich und Unifikation

Mit | ist der Rest einer Liste erreichbar.

- Extrem wichtig für *Pattern Matching*!

```
?- [a, b, c, d] = [a, b | Rest].
Rest = [c, d]
```

```
?- [a, b, c, d] = [a, b, c, d | Rest].
Rest = []
```

```
?- [Anfang | Rest] = [a, b | [c, d]].
Anfang = a, Rest = [b, c, d]
```


Erklärung

Wir wissen:

- Prolog-Programme bestehen aus Termen.
- Listen können in Prolog wie Terme verwendet werden.

Aber:

- Die Klammerschreibweise mit [und] entspricht **keiner** Term-Notation!

Intern sind Listen durch rekursiv geschachtelte Terme aufgebaut.

- Die Klammerschreibweise ist eine Kurznotation.

Listen als rekursive Datenstruktur

Rekursive Definition von Listen

Rekursionsfundament

- Die leere Liste bildet eine Liste.

Rekursionsschritt

- Die Verknüpfung eines Elements mit einer Liste bildet wiederum eine Liste.

Am einfachsten konstruktiv betrachtet…

Um eine neue Liste zu bauen, nimm ein Element und hänge eine beliebige vorhandene Liste (leer oder nicht leer) daran.

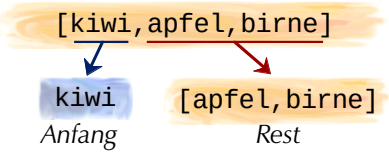
Listen in Prolog

Rekursionsfundament

- Die leere Liste ist das Atom `[]`.

Rekursionsschritt

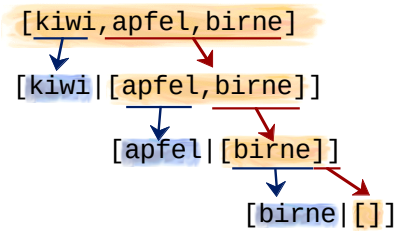
- Jede nicht-leere Liste besteht aus der Verknüpfung
  - eines beliebigen Terms als Anfang (Kopf, *Head)*
  - und einer Liste als Rest (Schwanz, *Tail)*.
    - Die Liste kann leer oder nicht leer sein!



Rekursiv geschachtelte Struktur

Listenreststrich als Verknüpfung

- Der rekursive Aufbau wird durch die Notation mit Listenreststrich sichtbar.
- Klammerschreibweise ist eine verflachte Notation.

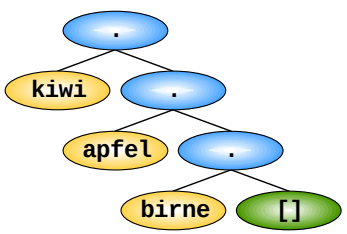



```
?- [kiwi,apfel,birne] = [kiwi|[apfel|[birne|[]]]].
yes
```

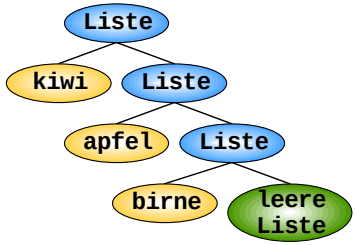

Interne Repräsentation

Rekursive Struktur heisst geschachtelte Terme

- Intern repräsentiert Prolog die Verknüpfung von Kopf und Restliste durch den Funktor '.'/2



Interne Term-Struktur



Logische Struktur

Listen als normale Terme

Listen sind intern ganz normale Terme

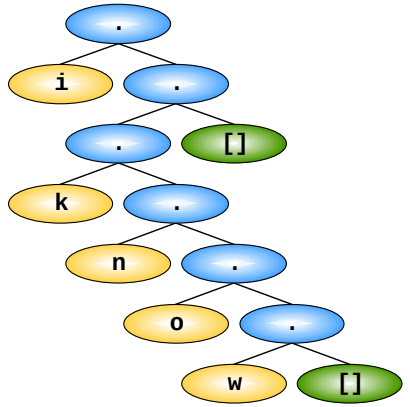
```
?- write_canonical([kiwi,apfel,birne]).
'.'(kiwi,'.'(apfel,'.'(birne,[])))
```

- Diese Punkt-Notation (*Dot-Notation*) mit dem Funktor ./2 wäre aber zu unübersichtlich, um brauchbar zu sein!
- Das 2. Argument des Punkt-Funktors ist die Liste hinter dem Listenrest-Strich.

```
?- '.'(kiwi,Fruechte) = L.
L = [kiwi|Fruechte]
```


Listen als Elemente

Wenn Listen normale Terme sind, können sie auch Elemente von Listen sein…




```
[i,[k,n,o,w]]
```


Das Listenprädikat

Datenstruktur Liste als Prädikat?

- Die Datenstruktur Liste tritt – wie alle Datenstrukturen – nur als Argument von Prädikaten auf!
- Als Prädikat verwendet bedeutet die Listenschreibweise dasselbe wie consult/1.
  - Die listenförmige consult/1 wird meist in grösseren Programmen verwendet, um Programmteile hineinzuladen, die in anderen Dateien vorhanden sind,

```
:- [datei1].
:- [datei1,datei2].
```

```
:- consult(datei1).
:- consult([datei1,datei2]).
```