Logische Programmierung

Programm	Menge von Formeln (Klauseln)
Berechnung	Beweis einer Formel (eines Ziels/Goals) mit Hilfe des Programms

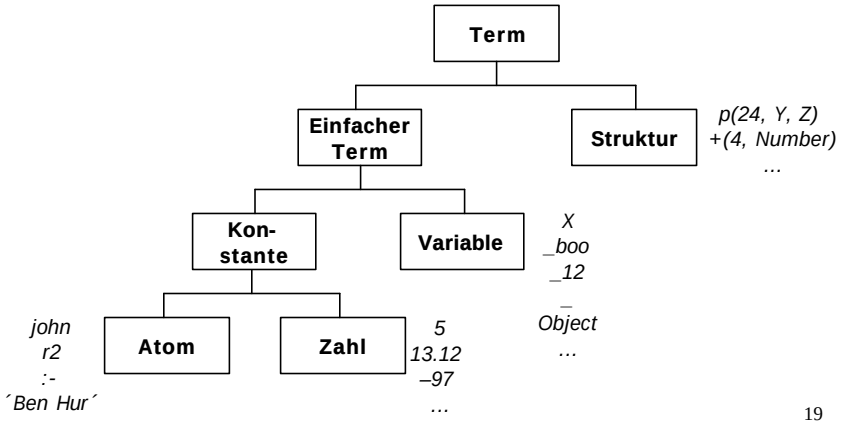
Anwendungen

- Expertensysteme
- Sprachverarbeitung
- Symbolische Informationsverarbeitung
- Graphentheoretische Probleme
- Planungsprobleme
- Rapid Prototyping
- ...

University of Bielefeld

Syntax
Terme

Prolog-Programme sind aus Termen gebildet.



University of Bielefeld

Syntax
Konstanten: Atome

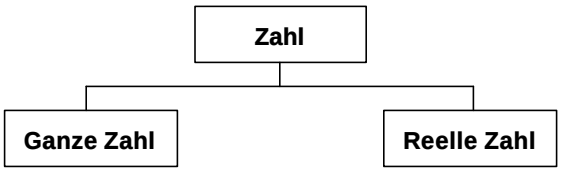
Atome werden benötigt, um bestimmte Objekte oder bestimmte Beziehungen zu benennen.

1. **Strings aus speziellen Characters**
   Beispiele: ?- :- + .:..
2. **Strings aus Buchstaben, Ziffern und dem *Underscore* Character, beginnend mit einem Kleinbuchstaben**
   Beispiele: susie is_person r2
3. **Strings aus beliebigen Characters, eingeschlossen in *Single Quotes***
   Beispiele: ´Bob Dole´ ´bill-1´ ´Calvin´

Characters	
Grossbuchstaben:	**A, B, ..., Z**
Kleinbuchstaben:	**a, b, ... , z**
Ziffern:	**0, 1, ... 9**
Spezielle Character:	**+ - * / \ ~ ^ < > : . ? @ # $ &**

University of Bielefeld

Syntax
Konstanten: Zahlen



1
1312
–97
…

3.14
–0.0034
123.22
…

Reelle Zahlen werden in der Prolog-Programmierung selten verwendet, da Prolog eher für symbolische, nicht-numerische Berechnungen verwendet wird

University of Bielefeld

Syntax
Variablen

Variablen sind Strings aus Buchstaben, Ziffern und dem *Underscore* Character, beginnend mit einem Grossbuchstaben oder mit einem *Underscore* Character. Variablen gelten innerhalb einer Klausel.

Beispiele: Answer_1 X Object _23

Anonyme Variable

Wenn eine Variable in einer Klausel nur einmal erscheint, muss sie nicht gebunden werden. Ihr Name ist daher irrelevant.

Kommen mehrere anonyme Variablen in derselben Klausel vor, müssen daher auch keine konsistente Interpretationen für die Variablen gefunden werden.

is_mother(X) :- mother(X, Y).
äquivalent zu
is_mother(X) :- mother(X, _).

is_parent(X) :- mother(X, Y), father(X, Z).
äquivalent zu
is_parent(X) :- mother(X, _), father(X, _).

University of Bielefeld

Syntax
Strukturen

Strukturen sind Objekte, die aus Komponenten bestehen.

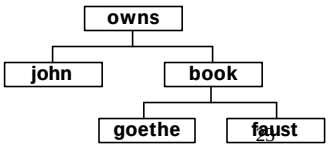
date(Day, Month, Year) **owns(john, book)**

Funktor *Argumente* *Funktor* *Argumente*

Die Komponenten können selber wieder Strukturen sein.

owns(john, book(goethe, faust))

Strukturen können als Bäume dargestellt werden.



University of Bielefeld

Syntax
Matching

Die wichtigste Operation auf Prolog-Termen ist das *matching*.

Das *matching* zweier Terme ist erfolgreich, falls

1. die Terme identisch sind
2. die Variablen in beiden Termen durch Objekte instantiiert werden können, so dass die Terme nach der Substitution der Variablen durch diese Objekte identisch sind

Term1	Term2	Matching ...
date(Day, Month, 2001)	date(24, M, Y)	... succeeds
point(X, Y)	point(3, 5)	... succeeds
plus(2, 2)	4	... fails
X	human(socrates)	... succeeds

University of Bielefeld

Syntax: Matching
Übung (Lösungen)

Ist das *matching* der folgenden Terme erfolgreich?

drinks(anna, wine)	drinks(Whom, What)	*ja: Whom = anna, What = wine*
drinks(anna, Beverage)	drinks(Person, water)	*ja: Person = anna, Beverage = water*
book(14, goethe, X)	book(A, B, Y)	*ja: A = 14, B = goethe, X = Y*
f(a, b, X)	f(a, c, X)	*nein: b ¹ c*
f(X, Y)	f(P, P)	*ja: X = P, Y = P (→ X = Y)*
cd(12, k(1685, 1750), bach)	cd(X, Y, _)	*ja: X = 12, Y = k(1685, 1750)*
f(X, Y)	g(X, Y)	*nein: f ¹ g*

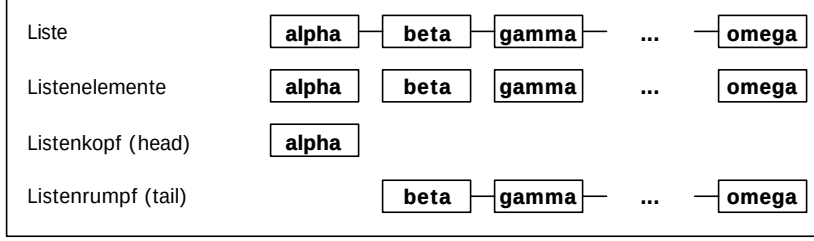
University of Bielefeld

Listen

Liste: - beliebig lange Sequenz von Listenelementen
- syntaktisch sind Listen Strukturen

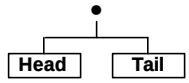
Listenelemente: Terme (→ können auch selbst wieder Listen sein)

<u>Begriffe</u>

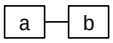


University of Bielefeld

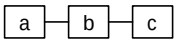
Repräsentation von Listen
Funktor-Notation

	Notation	Graphische Darstellung
Liste mit Kopf „Head“ und Rumpf „Tail“	**(Head, Tail)** *Funktor Argumente*	
Leere Liste (hat weder Kopf noch Rumpf)	**[]**	**[]**

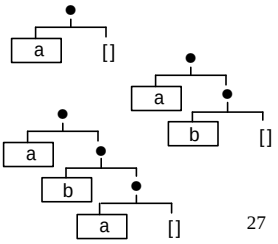
• (a, [])



• (a, • (b, []))

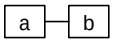
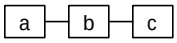


• (a, • (b, • (c, [])))



University of Bielefeld

Repräsentation von Listen
Listen-Notation

	Notation 1	Notation 2
	**[a]**	**[a|[]]**
	**[a, b]**	**[a|[b]]**
	**[a, b, c]**	**[a|[b, c]]**

Beispiele

[[the, man], walks]	=	[[the, man]|[walks]]
[the, man, [walks, fast]]	=	[the|[man, [walks, fast]]]
[the, [man, walks, fast]]	=	[the|[[man, walks, fast]]]
[]	=	[]

University of Bielefeld

Repräsentation von Listen
Übung

1. Ist das *matching* der folgenden Listen erfolgreich? Falls ja, wie werden die Variablen instanziiert?

[man]	[X|Y]	*ja: X = man, Y = []*
[[X, cat], [is, cute]]	[[the, Y]|Z]	*ja: X = the, Y = cat, Z = [[is, cute]]*
[united, kingdoms]	[united|Tail]	*ja: Tail = [kingdoms]*

2. Gegeben ist folgendes Fakt: p([H|T], H, T).
Welche Antworten erfolgen auf die folgenden Fragen?

?- p([2, 4, 8], X, Y).	*X = 2, Y = [4, 8]*
?- p([sunny], X, Y).	*X = sunny, Y = []*
?- p([[a, b, c], [A, B, C]], X, Y).	*X = [a, b c], Y = [[A, B, C]]*

University of Bielefeld

Operationen auf Listen
Operation *member*

```
% member(X, L) : X ist ein Element der Liste L
member(X, [X|Tail]).
member(X, [Head|Tail] :- member(X, Tail).
```

?- member(beta, [alpha, beta, gamma, delta]).

yes

Operationen auf Listen
Operation *append*

```
% append(X, Y, Z) :   Z ist die Liste, die entsteht, wenn man die
%                     Elemente der Liste Y der Reihe nach hinten
%                     an die Liste X anhängt (Konkatenation)
append([], L, L).
append([X|L1], L2, [X|L3] :- append(L1, L2, L3).
```

?- append([a, b], [c, d], U).
U = [a, b, c, d].

Operationen auf Listen
Übung

Definieren Sie die folgenden Prädikate:

% prefix(P, L) : P ist ein Präfix von L
Beispiel: prefix([a, b], [a, b, c, d]).

% last(X, L) : X ist das letzte Element von L
Beispiel: last(c, [a, b, c]).

% reverse(L1, L2) : L2 ist die invertierte Liste von L1
Beispiel: reverse([a, b, c], [c, b, a]).
Tip: Verwenden Sie die Prozedur append

Operationen auf Listen
Übung (Lösungen)

```
% prefix(P, L) : P ist ein Präfix von L
prefix([], _).
prefix([X|R], [X|S]) :- prefix(R, S).

% last(X, L) : X ist das letzte Element von L
last(X, [X]).
last(X, [_|Y]) :- last(X, Y).

% reverse(L1, L2) : L2 ist die invertierte Liste von L1
reverse([], []).
reverse([X|L], M) :- reverse(L, N), append(N,[X], M).
```


Layout von Prolog-Programmen

- Alle Klauseln einer Prozedur unmittelbar hintereinander schreiben
- Eine Leerzeile zwischen zwei Prozeduren einfügen
- Jede Klausel auf einer neuen Zeile beginnen
- *Falls genügend Platz vorhanden ist:* Jede Klausel auf eine eigene Zeile schreiben
  Sonst: Den Kopf und das Zeichen :- auf die erste Zeile und jedes zu beweisende Ziel (eingerückt) auf eine neue Zeile schreiben
- Alle Prozeduren kurz dokumentieren

```
% Dies ist ein einzeiliger Kommentar
/* Dies könnte auch ein mehrzeiliger Kommentar sein */
```

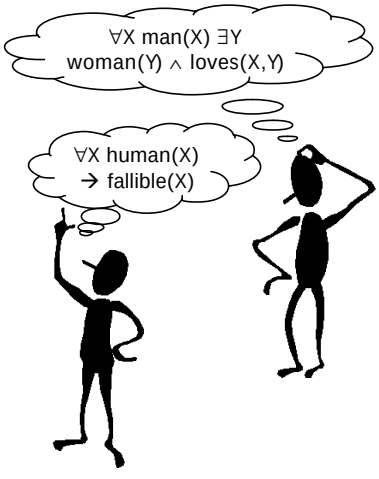
University of Bielefeld

Logische Basis von Prolog
Logik

- Logik ist die Lehre vom folgerichtigen Denken.
- Die Sprache der Logik kann verwendet werden, um gewissen Aussagen Wahrheitswerte zuzuordnen.
- Aussagen können miteinander verknüpft werden.
- Neue Aussagen können aus vorhandenen Aussagen hergeleitet werden.

University of Bielefeld

Logische Basis von Prolog
Prädikatenlogik



- Die gebräuchlichste Form der Logik ist die sogenannte Prädikatenlogik.
- In der Prädikatenlogik werden Aussagen mit Hilfe von *und* ($\wedge$), *oder* ($\vee$), *nicht* ($\neg$), *wenn* ($\rightarrow$), und *genau dann wenn* ($\leftrightarrow$), verknüpft.
- Aussagen können Variablen verwenden. Diese müssen durch einen Quantor eingeführt werden ($\forall$: Allquantor, $\exists$: Existenzquantor).

University of Bielefeld

Logische Basis von Prolog
Konjunktive Normalform und Klauseln

KNF P1 ∧ P2 ∧ ... ∧ Pn

Klausel Q1 ∨ Q2 ∨ ... ∨ Qm

Literal
= Atom oder negiertes Atom

Beispiel (einer Klausel):

¬human(X) ∨ fallible(X)

negatives Literal *positives Literal*

Formeln der Prädikatenlogik können vereinfacht werden:

- Die Quantoren werden entfernt (alle Variablen werden als implizit allquantifiziert betrachtet).
- Nur noch die elementaren Verknüpfungen *und*, *oder* und *nicht* kommen vor.
- Die Verknüpfungen *und*, *oder* und *nicht* sind von aussen nach innen sortiert.

Diese Form heisst *konjunktive Normalform (KNF).* Die durch *und* verknüpften Elemente heissen *Klauseln*.

University of Bielefeld

Logische Basis von Prolog
Hornklauseln

Hormklausel mit *einem* positiven Literal:

¬human(X) ∨ fallible(X)

↔

fallible(X) ← human(X)

Hormklausel mit *keinem* negativen Literal:

human(socrates)

- Im Programmieren in Logik beschränkt man sich auf eine eingeschränkte Klauselform: auf *Hornklauseln*.
- Hornklauseln sind Klauseln mit höchstens einem positiven Literal.

Logische Basis von Prolog
Inferenzregeln

Beispiele von Inferenzregeln

$$\frac{a,\ a \rightarrow b}{b} \qquad \textit{Modus Ponens}$$

$$\frac{a,\ b}{a \wedge b} \qquad \textit{Und-Einführung}$$

Enthält eine Formelmenge die beiden Formeln über dem Strich, so darf auch die Formel unter dem Strich hinzugefügt werden.

- Eine Inferenzregel gibt an, wie aus einer Menge von Formeln eine neue Formel abgeleitet werden kann.
- Eine Menge von Inferenzregeln ist *korrekt*, wenn jede hergeleitete Formel inhaltlich korrekt ist.
- Eine Menge von Inferenzregeln ist *vollständig*, wenn jede inhaltlich korrekte Formel hergeleitet werden kann.