# Syntax und Datenstrukturen

## Übersicht

- Woraus besteht ein Prolog-Programm?
- Syntaxdiagramme für Bildungsregeln
  - Programm, Klausel, Anfrage, Term
- 3 Sorten von Termen
  - atomare Terme, Variablen, komplexe Terme
- Termklassifikation in Prolog
  - atomic/1, atom/1, number/1, integer/1, float/1, var/1, compound/1
- Prädikate vs. komplexe Objektbezeichnungen
- Kommentare



# Bestandteile von Prolog-Programmen

## Ein Programm besteht aus

- **Fakten**

  Fido ist ein Hund.
  Hans ist Gabis Vater.

```
hund(fido).
vater(hans, gabi).
```

- **Regeln**

  Hunde sind bissig.

```
bissig(X) :- hund(X).
```

## An ein solches Programm werden Anfragen gestellt

- **Anfragen**

  Ist Fido bissig?
  Welche Kinder hat Hans?

```
?- bissig(fido).
?- vater(hans, Kinder).
```



# Legende Syntaxdiagramme


**Nicht-Terminales Symbol**

- Klasse von Ausdrücken der Sprache


**Terminales Symbol**

- Wörtlicher Text der definierten Sprache

*NT*

**Definiertes Nicht-Terminales Symbol**

- durch nachfolgenden Diagramm definiertes Symbol



**Übergang (mit beliebigem Leertext)**

- Leerzeichen, Zeilenenden, Kommentare

**Übergang (kein Leertext erlaubt)**

- verbundene Elemente müssen direkt folgen

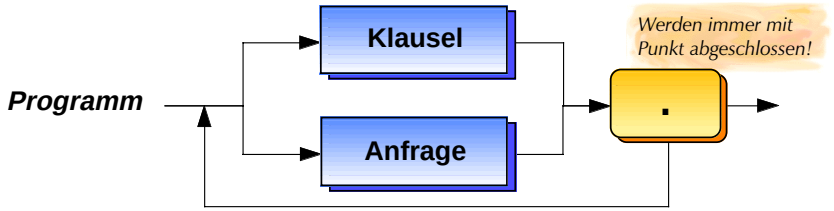


# Lesen von Syntaxdiagrammen

## Terminale und Nicht-Terminale Symbole sind durch Einbahnstrassen verbunden

- **Start**: rechts neben dem kursiv geschriebenen Nicht-Terminal ohne Kästchen
- **Weiterfahren**: immer in Pfeilrichtung
- **Schluss**: wo ein Pfeil ins Weisse zeigt
- **Rundes Kästchen**: die Zeichenkette im Kästchen muss wörtlich angetroffen werden
- **Eckiges Kästchen**: das Syntaxdiagramm des im Kästchen erwähnten Nicht-Terminals muss durchlaufen werden
- **Ziel**: vom Start bis zum Schluss durchkommen

# Syntax von Programmen




**Klauseln (*clauses*)**

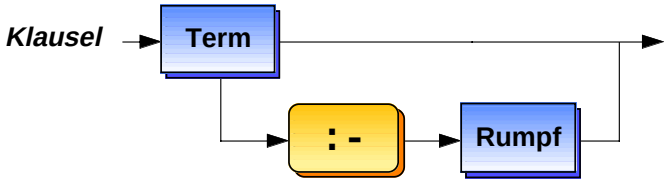
- Definieren **Prädikate** *(predicates)* durch Fakten und Regeln

**Anfragen (*queries*)**

- Spezifizieren **Beweisziele** *(goals)*



# Syntax von Klauseln




**Klausel (*clause*) ist Überbegriff für Fakten und Regeln.**

**Fakten (*facts*)**

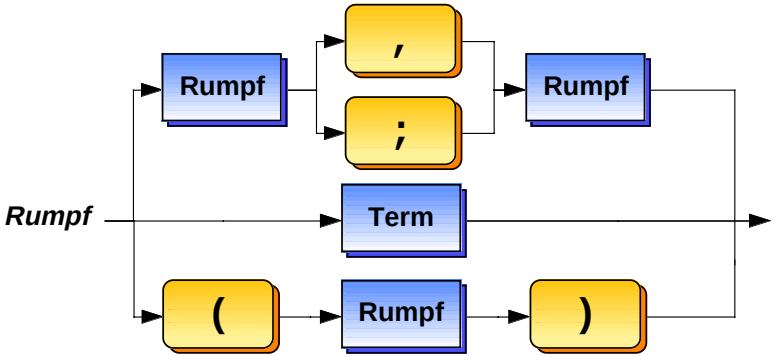
- bestehen aus einem Term

**Regeln (*rules*)**

- bestehen aus einem Term – genannt Kopf (*head*) –, dem Terminal-Symbol ":–" und einem Rumpf (*body*)



# Syntax von Rümpfen

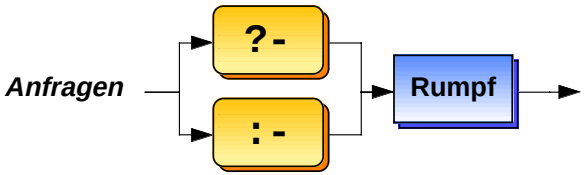



**Rümpfe**

- Beliebig komplexe Verschachtelung ist möglich!



# Syntax von Anfragen

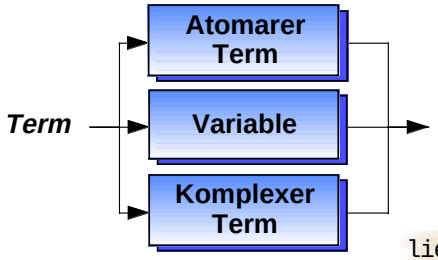



**Anfragen (*queries*) sind sofort zu beweisen!**

- "?-" für **interaktive** Anfragen: Prolog-Interpreter kann mehrere Antworten (yes/no mit ev. Variablenbelegungen) ausgeben.
- ":-" für **Anweisungen** (**Direktiven**) in Dateien. Höchstens *eine Lösung* wird berechnet und keine Antwort erzeugt!
  - Per Direktiven werden z.B. andere Prolog-Dateien automatisch aus Prolog-Dateien konsultiert.

```
:- consult(datei2).
```

# Syntax von Termen



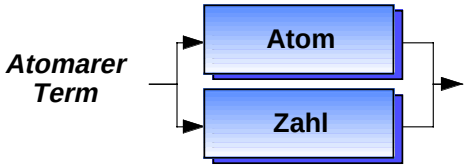

```
klara
X
liebt(paul, mutter(gabi))
```

**Terme (*terms*) sind**

- atomare Terme (*atomic terms*)
- Variablen (*variables*)
- komplexe Terme (*compound terms*)



# Atomare Terme




**Atomare Terme (*atomic terms*) sind**

- Atome (*atoms*)
- Zahlen (*numbers*)

Das eingebaute Prädikat **atomic**(*T*) ist wahr, falls *T* ein atomarer Term ist.

```
?- atomic(fido).
yes
?- atomic(23).
yes
?- atomic(X).
no
```



# Atome

**Lexikalische Bildungsregeln**

- **Normale Atome**: Kleinbuchstabe, gefolgt von beliebig vielen Klein-, Grossbuchstaben, Ziffern und "_" `klara` `a4711_b23_`
- **Zitierte Atome**: beliebige Zeichen zwischen zwei Hochkommata `'Ich bin ein Atom.'`
  - Um Hochkommata in solchen Atomen zu schreiben, muss man sie verdoppeln! `'Atom mit ''nem Hochkomma.'`
- **Symbolatome**: beliebige Folge aus `+-*/\^<>=`~:.?@#$&` `+` `>=`
- **Sonderatome**: `!`, `;`, `[]`, `{}`



# X ist Atom – X ist Zahl

**Klassifikationsprädikat für Atome**

Das eingebaute Prädikat **atom**(*T*) ist wahr, falls *T* ein Atom ist.

```
?- atom(fido).
yes
?- atom('23').
yes
?- atom(hund(fido)).
no
?- atom(23).
no
?- atomic(X).
no
```

**Klassifikationsprädikat für Zahlen**

Das eingebaute Prädikat **number**(*T*) ist wahr, falls *T* eine Zahl ist.

```
?- number(23).
yes
?- number(-4.5).
yes
```

# Zahlen

**Entsprechend der internen Repräsentation werden auf Rechnern meist 2 Zahlentypen unterschieden.**

- Ganzzahlen (*integers*) `12` `-20`
- Gleitpunkt- oder Gleitkommazahlen (*floats*) `123.45` `-2.0e4`

Das eingebaute Prädikat **integer**(*T*) ist wahr, falls *T* eine Ganzzahl ist.

```
?- integer(3).
yes
```

```
?- integer(2.0).
no
```

Das eingebaute Prädikat **float**(*T*) ist wahr, falls *T* eine Gleitpunktzahl ist.

```
?- float(-23.e4).
yes
```

```
?- float(2).
no
```



# Variablen

**Lexikalische Bildungsregeln**

- **Normale Variablen**: Grossbuchstabe oder "_" (Unterstrich), gefolgt von beliebig vielen Gross-, Kleinbuchstaben, Ziffern und "_".

  `Futter` `_futter4FIDO`

- **Spezielle anonyme Variablen**: "_"
  - Jeder einzelne Unterstrich in einer Klausel bezieht sich auf eine andere anonyme Variable.
  - Mit der anonymen Variable drücken wir aus, dass uns das betreffende Objekt nicht interessiert.

```
vater(Vater) :-
   kind(_, Vater),
   maennlich(Vater).
```



# Variablen als Platzhalter

**Variablen sind Platzhalter.**

- Wem alles die Eigenschaft Frau zu sein gesprochen wird, hängt davon ab, wer eine Person und weiblich ist.

```
frau(Jemand) :-
  person(Jemand),
  weiblich(Jemand).
```

**Innerhalb einer Klausel oder einer Anfrage stehen gleiche Variablen für das Gleiche.**

- Es wäre nicht gut, wenn das Frausein aus der Personenhaftigkeit und Weiblichkeit unterschiedlicher Wesen bestehen könnte.

**Allerdings: Variablen mit unterschiedlichem Namen können manchmal für das Gleiche stehen.**



# Variable oder nicht Variable?

Das eingebaute Prädikat **var**(*T*) ist wahr, falls *T* eine Variable ist.

```
?- var(X).
yes
```

```
?- var(hund).
no
```

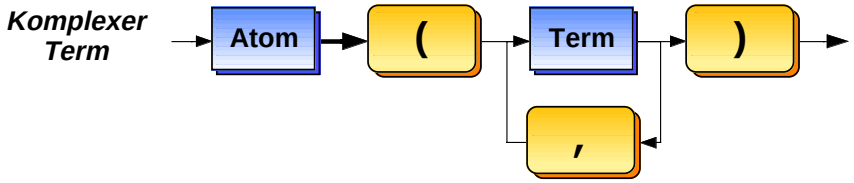
- Ob ein Term eine Variable ist, kann nicht textuell entschieden werden, da Variablen während dem Beweis zu anderen Termen instantiiert werden können.

```
liebt(katrin, Alles).
liebt(mauz, whiskas).
```

```
?- liebt(katrin, Was), var(Was).
true ?
yes
```

```
?- liebt(mauz, Was), var(Was).
no
```

# Syntax von komplexen Termen




**Ein komplexer Term (*compound term*) besteht aus einem Atom, direkt gefolgt von einem oder mehreren durch Kommata getrennte Terme in Klammern.**

- Komplexe Terme enthalten wiederum Terme. D.H. beliebige Schachtelung ist möglich!



# X ist komplexer Term

Das eingebaute Prädikat **compound**(*T*) ist wahr, falls *T* ein komplexer Term ist.

```
?- compound(hund(fido)).
yes
```

```
?- compound(hund(X)).
yes
```

```
?- compound(1).
no
```

```
?- compound(X).
no
```



# Funktor und Argumente

**Termanalyse**

- **Funktoren** stehen vor den Klammern.
- **Argumente des Funktors** stehen zwischen den Klammern.
- Argumente einer Ebene werden durchnummeriert.

`vater(hans, schwester(kevin))`

- `hans` ist 1. Argument des Funktors `vater`
- `schwester(kevin)` ist 2. Argument von `vater`
- `kevin` ist 1. Argument des Funktors `schwester`



# Stelligkeit von Termen

**Argumentanzahl eines Terms heisst Stelligkeit (*arity*)**

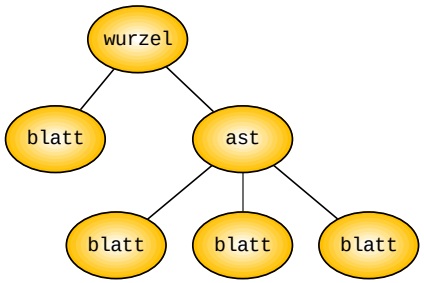
- Kurznotation im Prolog-Slang
  - Funktor und Stelligkeit durch Schrägstrich getrennt
  - ohne die Argumente zu nennen

| *n* | *n-stelliger Term* | *Funktor/Stelligkeit* |
|---|---|---|
| 0 | `fido` | `fido/0` |
| 1 | `hund(fido)` | `hund/1` |
| 2 | `frisst(fido, X)` | `frisst/2` |
| 3 | `gibt(hans, fido, fleisch)` | `gibt/3` |
| … | … | … |
| n | `funktor(a_1, a_2, …, a_n)` | `funktor/n` |

# Terme als Bäume

**Terme können graphisch als Bäume dargestellt werden.**

- Funktor: Verzweigung
- Argumente: Äste
- Stelligkeit: Anzahl Äste
- Atomare Terme: Blätter




```
wurzel(blatt,ast(blatt,blatt,blatt))
```



# Verwendung von Termen

**Atome werden verwendet als Name von**

- Objekten
- Relationen und Eigenschaften

**Zahlen werden verwendet als Name für**

- numerische Grössen

**Variablen werden verwendet als Platzhalter für**

- Terme

**Komplexe Terme werden verwendet für**

- Prädikatsausdrücke
- komplexe Objektbezeichnungen



# Komplexe Objektbezeichnungen

**Nicht alle Objekte, die wir bezeichnen, müssen einen Namen tragen.**

"Klara liebt den Vater von Kevin."

```
liebt(klara, vater(kevin)).
```

- `vater` ist hier kein Prädikatsname, sondern Teil einer komplexen Objektsbezeichnung – ein Funktionsausdruck.

**Prädikatsausdrücke und komplexe Namen sind gleich gebaut, bedeuten aber verschiedenes!**

"der Vater von Kevin"

```
vater(kevin)
```

*Komplexer Name*

"Kevin ist Vater."

```
vater(kevin).
```

*Prädikatsausdruck*



# Kommentare

```
/* Bla bla.
   Bla bla bla. */
```

*Zwischen /* und */*

```
% Bla bla.
% Bla bla bla.
```

*Ab % bis Zeilenende*

**Kommentare erhöhen die Verständlichkeit für Menschen**

- Prolog-Interpreter ignorieren Kommentare
  - Sie werden wie ein Leerzeichen aufgefasst!
- ein Programm ohne Kommentare ist nur sehr schwer verständlich
  - Sogar für AutorIn des Programms, wenn etwas Zeit verstrichen ist!
- besser zu viel als zu wenig kommentieren