Logische und funktionale Programmierung

Labor 1: Einführung in Prolog

Babeş-Bolyai Universität, Department für Informatik, Cluj-Napoca
csacarea@cs.ubbcluj.ro

Organisatorisches

Literatur

- W. Clocksin, Ch. Mellish, Programming in Prolog using the ISO Standard, Springer Verlag
- I. Bratko, PROLOG - Programming for Artificial Intelligence, Addison-Wesley Publishing Company
- werden über Dropbox bereitgestellt.
- Prolog Lexikon im Web http://www.cse.unsw.edu.au/~billw/prologdict.html

Sprechstunden

Nach Vereinbarung

Ziel der Veranstaltung

- Erlernen der Grundlagen der Programmiersprache Prolog
- Selbstständiges Schreiben von Programmen
- Fähigkeit ein grösseres Prologsystem zu verstehen, insbesondere als Voraussetzung des Kurses im 4. Semester, Künstliche Intelligenz.

Prolog Charakteristiken

Prolog steht für Programming in Logic.

- Entwickelt in den 70er Jahren von Alain Colmerauer und Robert Kowalski.
- Ein Programm in einer logischen Programmiersprache ist eine Sammlung von Formeln.
- Die Auswertungsstrategie gehört nicht zum Programm sondern ist in der Sprache selbst implementiert.
- Schleifenkonstrukte und Variablenzuweisungen sind nur sehr eingeschränkt möglich.
- Einmal gebundenen Variablen kann nur über Backttracking ein neuer Wert zugewiesen werden.

PROLOG IST...

- eine deklarative Programmiersprache
- Der Programmierer stellt das Problem in Form von Fragen auf;
- Das Prolog System findet selbst die Lösung.
- Ist nur mit Hilfe von Logik möglich.

Deklarative vs. prozedurale Programmiersprachen

- In einer prozeduralen Programmiersprache muss der Programmierer alle Schritte detailliert angeben.
- In einer deklarativen Programmiersprache versucht das System die Lösung zu finden.

Einführung in Prolog

Prolog

ist eine Sprache mit der man Probleme von Objekten und deren Beziehungen beschreiben kann.

- Die Programme definieren Eingaben für den Ablaufalgorithmus.
- Probleme werden deklarativ dargestellt.
- Rekursion spielt eine grosse Rolle.
- Klarheit über den Abarbeitungsmechanismus ist unverzichtbar.

Einführung in Prolog

Pro's

Auch komplexe Programme u.a. im Bereich der Sprachmodellierung, der Beweismodellierung und der strategischen Planung können implementiert werden.

Con's

Man kann die Programme kaum übersehen (unvorhersehbare Seiteneffekte von Änderungen) und kooperatives Programmieren ist sehr schwer.

Anwendungen

- intelligent data base retrieval
- natural language understanding
- Expert Systeme
- Spezifikationen
- machine learning
- robot planning
- automated reasoning
- problem solving

Einführung in Prolog

Prolog ist eine Programmiersprache um z.T. für andere Sprachen sehr schwierige Probleme zu lösen.

Hauptmerkmal:

Backtracking Algorithmus schon implementiert.

- Deklariere Fakten über Objekte und ihre Beziehungen;
- Definiere Regeln über Objekte und ihre Beziehungen;
- Stelle Fragen über Objekte und ihre Beziehungen.

Einführung in Prolog

- Prolog kümmert sich weniger um Variablen und Prozeduren, als um Relationen zwischen Objekten die in einem Programm bearbeitet werden sollen.
- Ein Prolog Programm beschreibt die Fakten und Relationen eines Problems und nicht die sequentielle Abfolge von Anweisungen die die Werte von Variablen verändern.
- In einem Prolog Programm ergibt sich die eigentliche Abfolge der Anweisungen nur durch die logische deklarative Semantik der Fakten und Regeln des Programms.
- Soll ein Problem mit Hilfe von Prolog gelöst werden, muss zuerst das Prolog Programm mit den Fakten und Regeln des Problems *gefüttert* werden.

Einführung in Prolog

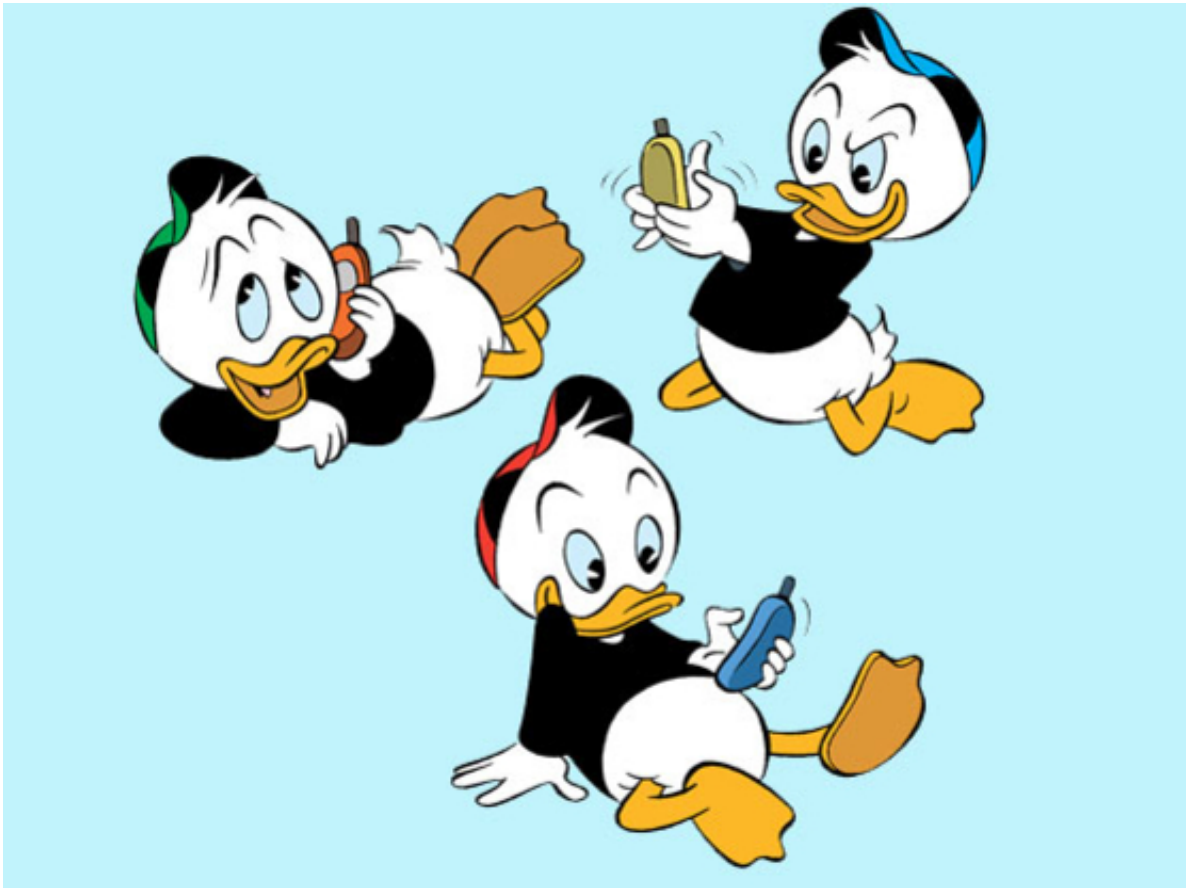
- Prolog hält eine Datenbank von Fakten und Regeln. Daraufhin geht Prolog in einen Fragemodus und wartet auf Anfragen des Benutzers, die Prolog mit Anfrage ist wahr oder Anfrage ist falsch beantwortet.
- Um zu dieser Antwort zu kommen testet Prolog alle Fakten und Regeln die etwas mit dieser Anfrage zu tun haben.
- Regeln können dazu führen, dass es bei bestimmten Tests Alternativen gibt. Prolog testet alle Alternativen und versucht immer zu einem Entschluss zu kommen. Ob die Anfrage, in Bezug auf die Fakten und Regeln logisch wahr oder falsch ist.
- Stellt man Prolog eine Anfrage bei der es dem Programm überlassen wird Antworten zu finden, dann listet Prolog alle Antworten auf.

Zusammenfassung

1. Ein Prolog Programm besteht logischen Zusammenhängen:
   - Fakten über Objekte und ihre Beziehungen;
   - Regeln über Objekte und ihre Beziehungen.
2. der Anwender stellt Anfragen an die logischen Zusammenhänge: Ein Benutzer kann an dieses Programm Fragen stellen und der Abarbeitungsmechanismus von PROLOG sorgt dafür, dass alle logisch zulässigen Kombinationen von Fakten und Regeln die etwas mit dieser Frage zu tun ausprobiert werden um zu einer Antwort zu kommen, die sagt, ob die Anfrage wahr oder falsch ist.

Beispiel

Tick, Trick und Track sind Brüder. D.h. sie sind männlich und haben dieselbe Eltern. Hier handelt es sich um eine Definition und nicht darum alle Aspekte abzubilden die mit dem Begriff Brüder verbunden sind.

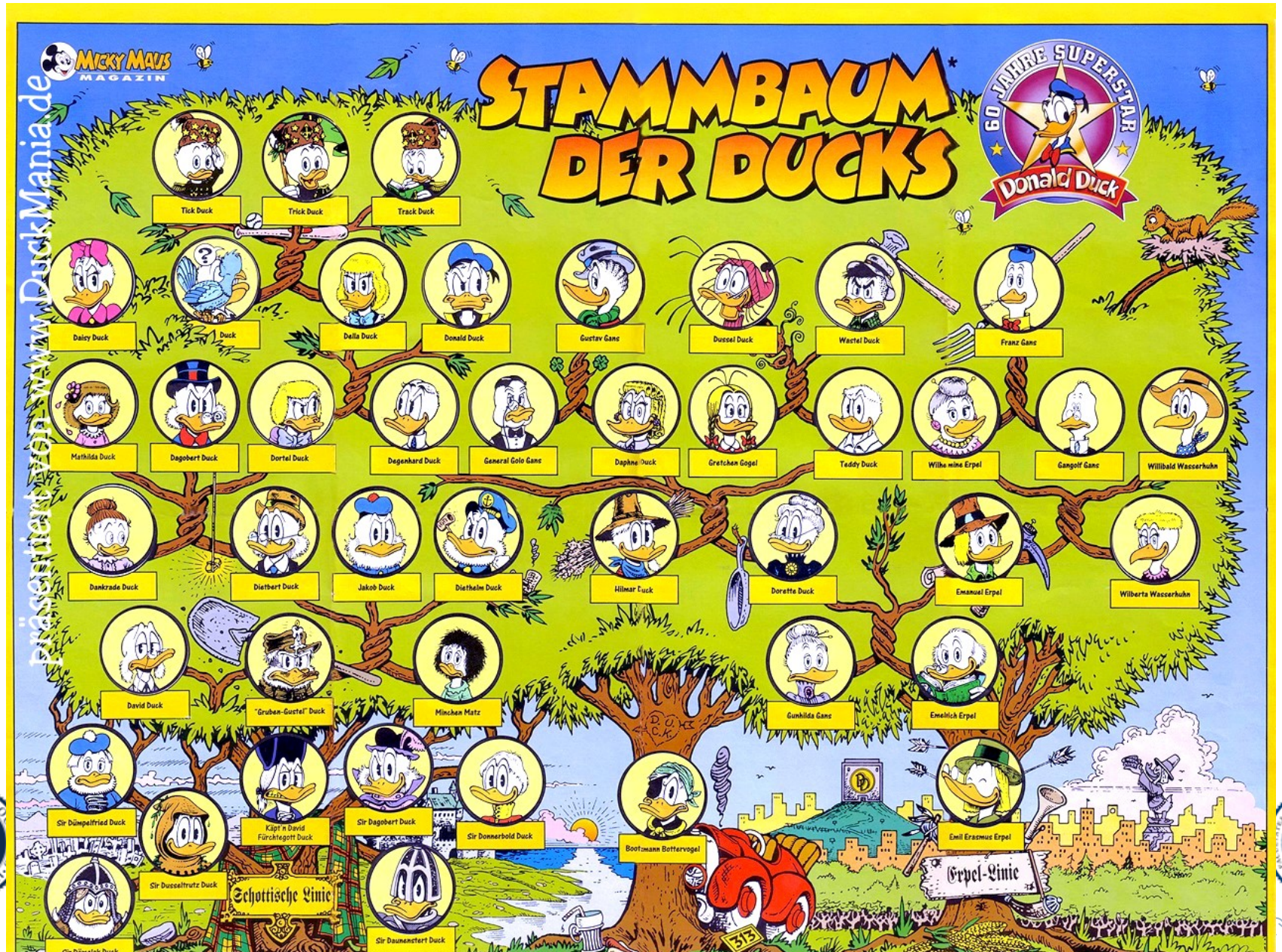


Einführung in Prolog

PROLOG spezifiziert Relationen zwischen Gegenstände und Merkmale.

- Objekte: Tick, Trick, Track
- Fakten: Sind angehörige der Ducks Familie.
- Tick = kindvon(DellaDuck,XDuck), Z = kindvon(X,Y) (Relation 2-stellig).
- Regeln: Das was man aus den Fakten oder aus anderen Regeln schlussfolgern kann.

Beispiel


Micky Maus Magazin
STAMMBAUM DER DUCKS
60 Jahre Superstar
Donald Duck
Tick Duck
Trick Duck
Track Duck
Daisy Duck
Duck
Della Duck
Donald Duck
Gustav Gans
Dussel Duck
Wastel Duck
Franz Gans
Mathilda Duck
Dagobert Duck
Dortel Duck
Degenhard Duck
General Golo Gans
Daphne Duck
Gretchen Gogel
Teddy Duck
Wilhelmine Erpel
Gangolf Gans
Willibald Wasserhuhn
Dankrade Duck
Dietbert Duck
Jakob Duck
Diethelm Duck
Hilmar Duck
Dorette Duck
Emanuel Erpel
Wilberta Wasserhuhn
David Duck
"Gruben-Gustel" Duck
Minchen Matz
Gunhilda Gans
Emeltich Erpel
Sir Dümpelfried Duck
Käpt'n David Fürchtegott Duck
Sir Dagobert Duck
Sir Donnerbold Duck
Bootsmann Bottervogel
Emil Erasmus Erpel
Sir Dusseltrutz Duck
Schottische Linie
Erpel-Linie
Sir Daunenstert Duck
313
Präsentiert von www.DuckMania.de

Gross- und Kleinschreibung in PROLOG

Bei der Programmiersprache PROLOG werden Regeln, Fakten und Objekte immer klein geschrieben. Großgeschriebenes wird für Platzhalter (Variablen) verwendet, denen Prolog beim Abarbeiten Werte zuweisen kann. Kommentarzeilen in PROLOG Scripten beginnen mit dem Zeichen %.

EINFÜHRUNG IN PROLOG

- Objekt 1: Tick Duck
- Objekt 2: Della Duck
- Gerichtete Relation zwischen Objekte: kindvon

Beispiel

Tick, Trick und Track sind Brüder wenn sie männlich sind und dieselben Eltern haben.

X = männlich, eltern(X,E1,E2), Y = männlich, eltern(Y,E1,E2), Z = männlich, eltern(Z,E1,E2).

',' wird in Prolog für die Konjunktion $\wedge$ verwendet; das ist später sehr wichtig für die Variablenbindung, d.h. nur die Konjunktion stellt hier sicher, dass die Variablen E1 und E2 in beiden Teilklauseln für dasselbe stehen.

Elemente eines PROLOG Programmes

- Deklarieren von Fakten - Objekte, Beziehungen;
- Definieren von Regeln - Objekte, Beziehungen; Stellen von Fragen - Objekte Beziehungen.

Programmieren:

Stelle Fakten und Regeln in einem oder mehreren Programmen bereit.

Programmablauf:

Eine Frage setzt Inferenzen von einem Fakt zum anderen von einer Regel zur anderen in Gang.

Programmieren in PROLOG

- SWI-PROLOG;
- Namen aller Beziehungen und Objekte beginnen mit Kleinbuchstaben;
- Relation wird nach vorne geschrieben, Präfixnotation
- Objekte der Relation werden kommasepariert in runden Klammern angeordnet;
- Punkt am Ende eines Fakts;
- Namen zwischen Klammern, die Objekte bezeichnen heissen Argumente;
- Name der Relation vor der Klammer heisst Prädikat.

Fakten, Prädikate

- LFP Vorlesung findet im Raum X statt: `findetstatt(LFP, x).` Achtung: Raum X wird hier kleingeschrieben, da es keine Variable ist!
- `likes(Romeo, Julia)`: likes Prädikat mit zwei Argumenten nämlich Romeo und Julia.
- Namen für Prolog beliebig, Zahl der Argumente beliebig.
- Kollektion von Fakten und Regeln wird auch Datenbank (database) genannt.
- Achtung: Romeo und romeo sind verschieden, Julia und Jullia auch!

PROLOG FRAGEN

Eine Frage sieht in Prolog aus wie ein Fakt mit einem speziellen führenden Symbol: `?:- owns(john, house).`
Prolog nimmt das Fragesymbol zum Anlass seine Datenbank zu durchsuchen. Es sucht Fakten, die mit dem Fakt der Frage matchen.

```
?:- likes(Romeo,Julia).
?:- likes(Julia,Romeo).
```

PROLOG MATCH

- Suche einen Fakt in der Datenbank der mit der Frage matcht.
- Wann gibt es einen Match?
- Prädikate müssen gleich sein. Achtung Spelling!
- Argumente müssen gleich sein. Achtung Ordnung! Achtung Spelling!
- Findet Prolog einen Fakt der mit der Frage matcht: yes sonst no.

BEISPIELDATENBANK

Datenbank:

```
likes (joe, fish).
likes (joe, mary).
likes (mary,book).
likes (john,book).
```

Anfragen:

```
?- likes(joe,money).
?- likes(mary, joe).
?- likes(mary,book).
?- likes(marry,book).
?- king(john,france).
```

BEISPIELDATENBANK

Datenbank:

```
likes (joe, fish).
likes (joe, mary).
likes (mary,book).
likes (john,book).
```

Anfragen:

`?- likes(joe,money).` no. /*Prädikat matcht, Anzahl Argumente, aber zweites Argument nicht */

`?- likes(mary, joe).` no. /*Prädikat matcht, Anzahl Argumente, aber Ordnung der Argumente stimmt nicht. Scheitert schon am ersten Argument*/

`?- likes(mary,book).` yes. /*Prädikat, Argumentzahl und Argumente */

`?- likes(marry,book).` no. /* mary = mary ? no */

`?- king(john,france).` no /* scheitert schon beim Prädikat */

WAS BEDEUTET 'NO'?

- 'no' heisst, kein match mit der Frage. Also, 'no' $\neq$ falsch.
- 'no' heisst nicht beweisbar gegeben die Datenbank.

```
human(socrates).
human(aristoteles).
athenian(socrates).
?- athenian(socrates). yes.
?- athenian(aristoteles). no.
?- greek(socrates). no.
```

Aus unserer unvollständigen Datenbank können wir nicht die reale Geschichte beweisen!

Beispiel: Stundenplan

Im diesem Beispiel wollen wir Fakten und Regeln über Objekte definieren und zeigen, wie sie mit PROLOG verarbeitet werden. Fakten und Regeln werden kleingeschrieben und mit einem Punkt beendet. Es gibt einfache Fakten und Fakten, die die Relation von Objekten definieren. Der Relationsname wird zuerst geschrieben, die Objekte werden in runden Klammern, mit Komma getrennt angefügt.

BEISPIEL: ERSTES PROLOG PROGRAMM

```
%%
%% Studienplan Bachelor im 1.Semester
%% Autor: Max
%% CIS
%% Filename: studium.pl
%%

einfprog.
einfcl.
sprachkurs.
semester(1,einfprog).
semester(1,einfcl).
semester(1,sprachkurs).
```

Die Relation semester stellt eine Relation zwischen 1, einfprog, einfcl und sprachkurs dar.

semester		
	1	einfprog
	1	einfcl
	1	sprachkurs

Abfrage des PROLOG Programms studium.pl

Damit PROLOG die Fakten kennt und der Benutzer sie abfragen kann, muss Prolog gestartet werden, die Datei studium.pl geladen werden und über die Konsole Anfragen an das System gestellt werden.

Starten von PROLOG

```
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 5.6.64)
Copyright (c) 1990-2008 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?-
```

LADEN DES PROLOG PROGRAMMS UNTER LINUX

„füttern“ von Prolog mit den Fakten in der Datei studium.pl:

```
Menü:
 File
Consult …
 Öffnen über einen Filebrowser
```

oder

```
?- consult('aktuell/studium.pl').
```

Der Prolog Sourcefile wird geladen.

(Achtung: Der Pfad muss exakt stimmen, dabei hilft das Prologprädikat ‚pwd’)

```
?- pwd.
```

oder Aufrufen des emacs Editor

```
?- emacs.
```

(Achtung: Das Prolog Programm studium.pl wird in einem emacs Buffer geladen und dieser Buffer in PROLOG hineinkompiliert.)

Abfragen von Fakten und Regeln

Der Anwender fragt das PROLOG System, das sein *Weltwissen* geladen hat, ob Fakten oder Regeln wahr oder falsch sind und testet logisch korrekte Zusammenhänge die zu einer Lösung führen.

Anfrage von einem Fakt

```
1 ?- einfprog.
Yes
```

Erweiterung des Beispiels

Wie gelernt, entspricht einem Prolog Programm eine Auflistung von Klauseln oder auch eine Menge von Axiomen. Klauseln sind Fakten oder Regeln.
Fakten definieren Relationen der Objekte eines Problems und somit definieren sie die Basiseigenschaften eines programmierten Problems. Fakten werden mit einem Punkt abgeschlossen.
- es gibt einfache Fakten:

```
einfprog.
sprachkurs.
einfcl.
franz.
franz-ist-im-ersten-semester.
franz-ist-im-ersten-semester-und-hoert-deshalb-einf
```

Es wäre sehr mühsam das gesamtes Wissen eines PROLOG Programms mit reinen Fakten aufzuzählen. Deshalb definiert man Fakten, die als Argument Objekte haben.

Erweiterung des Beispiels

- es gibt Fakten mit Objekten als Argument:
Im Beispiel definiert man sich das Faktum Veranstaltung mit den Objekten `einfprog` und `einfcl`.

```
veranstaltung(einfprog).
veranstaltung(einfcl).
```

Die Anzahl der Objekte einer Relation nennt man die Stelligkeit einer Relation. Die Stelligkeit notieren wir mit einem /n am Ende einer Relation:

```
veranstaltung/1
```

Erweiterung des Beispiels

Man definiert Veranstaltung das beschreibt, wer welche Veranstaltung hört. Außerdem das Faktum Semester legt fest, welche Veranstaltung in welchem Semester gehört werden.

```
veranstaltung(einfprog,franz).
veranstaltung(einfprog,moni).
veranstaltung(einfprog,ali).
veranstaltung(morphologie,susi).
veranstaltung(hoehereprog,heini).
semester(1,einfprog).
semester(1,einfcl).
semester(1,sprachkurs).
semester(2,morphologie).
semester(2,hoehereprog).
semester(2,mathe).
```

Man notiert : `veranstaltung/2, semester/2`

Regeln

Auch mit Fakten, die Objekte als Argumente haben, bleibt es weiterhin sehr mühsam das gesamte Wissen eines PROLOG Programms zu definieren.
Deshalb hat man in PROLOG Regeln eingeführt, die erlauben logische Sachverhalte zu formulieren.

Regeln

bestehen aus einem Kopf und einem Rumpf, dazwischen sind die Zeichen :-
Fakten kann man auch Regeln ohne Rumpf bezeichnen.
Mit Regeln können also vorher definierte Fakten oder Gruppen von Fakten zusammengefasst werden.

Regeln

Die Aussage:

Wenn jemand im ersten Semester ist, dann hört er Einführung in die Programmierung für Informatiker kann übergeführt werden in die Prolog Regel:

```
ist_im_ersten_semester
:- veranstaltung(einfprog).
```

Die Aussage vor dem falls wird der Kopf der Regel, die Bedingung der Rumpf der Regel.

Regeln

Regeln können im Kopf auch Objekte oder Variablen beinhalten, aber keine anderen Regeln. Die leben im gesamten Rumpf einer Regel.

```
ist_im_ersten_semester(Wer) :-
veranstaltung(einfprog,Wer).
moni_ist_im_ersten_semester :-
veranstaltung(einfprog,moni).
moni_ist_im_semester(1) :-
veranstaltung(einfprog,moni),semester(1,einfprog).
```

Anfrage von einer Regel mit definiertem Objekt

Nun fragen wir das System an, ob eine Regel wahr ist :
`ist_im_ersten_semester(franz).`
In einer prozeduralen Programmiersprache könnte der Mechanismus der Berechnung des Ergebnisses wie folgt gelesen werden.

```
if veranstaltung(einfprog,franz) then print
'true' else print 'false'
```

Anfrage von einer Regel mit definiertem Objekt

Das PROLOG System kennt die Regel:

```
ist_im_ersten_semester(Wer) :-
veranstaltung(einfprog,Wer).
```

es testet die Regel

```
ist_im_ersten_semester
```

mit dem Objekt

```
franz
```

und findet im Rumpf die Relation

```
veranstaltung(einfprog,Wer).
```

die es aufzulösen gilt: Die Variable `Wer` bekommt den Wert `franz` eingesetzt und PROLOG sucht weiter in seinem *Wissen*: Gibt es eine Regel oder ein Fakt

```
veranstaltung(einfprog,franz).
```

Prolog findet ein Faktum, hat die Anfrage komplett aufgelöst und gibt die Antwort

`true` aus.

Anfrage von einer Regel mit freier Variable

Bislang diskutierte Fragen nicht so interessant. Wir bekommen dieselbe Information zurück, die wir hineingetan haben.
Soll PROLOG alle möglichen Lösungen selbst ermitteln, dann werden anstatt festdefiniertem Objekt Variablen eingeführt.
Variablen in Regeln oder Fragen sind Platzhalter für alle Objekte die zu dieser Regel oder zu diesem Faktum passen. In Prolog wird keine Einschränkung an den Typ der Variable vorgenommen. Eine Variable kann als Wert alle Arten von Objekten annehmen. Es muss lediglich die Regel oder das Faktum mit diesem Objekt erfüllt sein. PROLOG unterscheidet Variablen von Objekten dadurch, dass Variablen mit einem Großbuchstaben, Objekte mit einem Kleinbuchstaben beginnen.
Wir wollen das System auffordern, alle Menschen zu ermitteln, die ihrem Wissen entsprechend im ersten Semester studieren.


Anfrage von einer Regel mit freier Variable

Das Argument der Anfrage darf kein definiertes Objekt mehr sein, sondern eine Variable. Die Anfrage lautet:

```
ist_im_ersten_semester(Wer).
```

Der Abarbeitungsmechansimus von PROLOG durchsucht alle Regeln, die mit der Anfrage übereinstimmen und ermittelt die Objekte, die die Regel erfüllen.

```
ist_im_ersten_semester(Wer) :-
veranstaltung(einfprog,Wer).
```

Anfrage von einer Regel mit freier Variable

In einer prozeduralen Programmiersprache könnte der Mechanismus der Berechnung der Werte wie folgt gelesen werden.

```
forall Wer
if veranstaltung(einfprog,Wer)
then
print Wer=
else
print 'false'
```

Bemerkung:

Die fiktive Wiederholungsanweisung `forall` generiert alle Objekte die im Programm für diese Regel zur Verfügung stehen.
In unserem Beispiel wird also `Wer` gebunden mit den drei verfügbaren Objekten: `franz`
`moni`
`ali`
da nur diese Einführung in die Programmierung hören und somit im ersten Semester sind:

```
veranstaltung(einfprog,franz).
veranstaltung(einfprog,moni).
veranstaltung(einfprog,ali).
```

die anderen beiden Fakten kommen nicht in Betracht, da die Veranstaltungen `morphologie` und `hoehereprog` nicht mit `einfprog` übereinstimmt.

```
veranstaltung(morphologie,susi).
veranstaltung(hoehereprog,heini).
```

Bemerkung

Für jeden einzelnen Aufruf, wird die Variable an dieses Objekt gebunden und es muss der Rumpf der Regel mit diesem Objekt getestet werden.
Zum Beispiel:

```
ist_im_ersten_semester(Wer,Veranstaltung) :-
    semester(1,Veranstaltung),
veranstaltung(Veranstaltung,Wer).
ist_im_ersten_semester(franz,einfprog) :-
    semester(1,einfprog),
veranstaltung(einfprog,franz).
```

VARIABLEN IN PROLOG

Variablennamen beginnen in Prolog mit einem Grossbuchstaben.
Benutzt Prolog eine Variable, so kann diese instantiert sein oder nicht.
Instantiert heisst: ein Objekt ist an die Variable gebunden, für das die Variable steht.
Beim Durchsuchen der Datenbank erlaubt Prolog der Variable mit jedem anderen Argument an der gleichen Position im Fakt zu matchen.

Ein weiteres Beispiel

```
likes(mary,flowers).
likes(mary,john).
likes(mary,mary).
likes(paul,mary).
?- likes(mary,X).
yes. X=flowers.
```

Prolog sucht top down. Findet ein Prädikat `likes`, matcht `mary` und instantiert `X` mit `flowers`. Alles matcht also `yes`.

Ausgabe mehrerer Lösungen

Nach einem Match markiert Prolog die entsprechende Stelle in der Datenbank. Durch Eingabe des Strichpunktes ' ; ' wird eine neue Suche ausgelöst.
Prolog kehrt in die Datenbank zurück und sucht ab der letzten Markierung topdown.
So findet es die anderen Lösungen.

```
X=john
X=mary
```

Konjunktionen

- Do John and Mary like eachother?
- zwei Fragen: mag John Mary und mag Mary John.
- In Prolog zwei goals werden durch die Konjunktion ’ , ’verbunden.
- `?-likes(john, mary), likes(mary, john)`
- In unserer Datenbank: `no`

Kombination von Variablen und Konjunktionen

```
likes(mary,food).
likes(mary,wine).
likes(john,wine).
likes(john,mary).
```

- Gibt es etwas das sowohl `john` als auch `mary` mögen?
- Informal: irgendein`X` das `mary` mag $\wedge$ `john` mag `X`.
- `?- likes(mary,X), likes(john,X).`
- Erfülle das erste Ziel, versuche das zweite Ziel zu erfüllen.
- !Im Backtrackingprozess hat jedes Ziel seinen eigenen Placemarker!

Kombination von Variablen und Konjunktionen

```
likes(mary,food).
likes(mary,wine).
likes(john,wine).
likes(john,mary).
?- likes(mary,X), likes(john,X).
```

- Erstes Ziel erfüllt, `X` wird mit `food` instantiert: `X=food`, placemarker gesetzt.
- Zweites Ziel: `likes (john, food).` `X` Unifikation. `fail`
- Backtracking: gehe unter den placemarker von goal 1 und versuche dieses anders zu erfüllen.

Kombination von Variablen und Konjunktionen

- Erstes Ziel erfüllt, `X` wird mit `wine` instantiert: `X=wine`
- Zweites Ziel wird: `likes(john,wine).` `X` Unifikation. Dritter Fakt matcht.
- Beide Ziele der Konjunktion erfüllt. Gesamte Anfrage: `yes, X=wine`
- Erzwungenes Backtraking: ';'
- Bottom up: Fakt 4 matcht nicht, Versuch goal 1 anders zu lösen (resolve); keine weitere Möglichkeit goal 1 zu lösen also `no`.

VARIABLENINSTANTIERUNG IN ZIELKONJUNKTIONEN

1. Ziel	Fakt
`likes(mary,X)`	`likes(mary,wine)`

2. Ziel

`likes(john,X)`

Sobald eine Variable instantiert wird (hier mit `wine`) wird sie an allen Stellen instantiert, also auch im zweiten Ziel.

VARIABLENINSTANTIERUNG: BACKTRACKING

Resatisfy: uninstantiere alle Variablen, die in dem Ziel instantiert wurden, das durch ’ ; ’ auf andere Art gelöst werden soll. Dasselbe gilt für Teilziele die durch Backtracking rückwärts zu Alternativen gezwungen werden.
Dies ist gerade der Vorteil von Prolog: so eine Buchhaltung selbst zu Programmieren ist durchaus anspruchsvoll.