Logische Programmierung

050026 VO Theoretische Informatik

Wolfgang Dvořák

Theory and Applications of Algorithms Gruppe,
Fakultät für Informatik, Universität Wien

Sommersemester 2016

Programmierparadigmen

Programmierparadigmen

- sind fundamentale Prinzipien nach denen Programmiersprachen aufgebaut sind.
- sollen Programmierer bei der Erstellung von „gutem“ Code unterstützen.
- unterscheiden sich durch die Art
  - wie Sachverhalte modelliert und
  - „Funktionen“ berechnet werden.

Eine konkrete Programmiersprache kann aber mehreren Paradigmen gleichzeitig folgen.

Programmierparadigmen

- Imperative Programmierung: Eine Folge von Befehlen gibt vor was in welcher Reihenfolge vom Computer getan werden soll.
- Objektorientierte Programmierung: Daten und darauf arbeitende Routinen werden zu Objekten zusammengefasst.
- Parallele Programmierung: Hat Methoden die es erlauben Programmteile nebenläufig auszuführen.
- Deklarative Programmierung: Es wird nicht angegeben wie etwas ausgerechnet werden soll, sondern es wird spezifiziert was ausgerechnet werden soll.
  - Logische Programmierung: nutzt logische Aussagen
  - Funktionale Programmierung: nutzt math. Funktionen
  - Mengen-Orientierte Abfragesprachen: z.B. SQL für Datenbanken
- ...

Vorteile Deklarativer Programmierung

- Die Spezifikation ist schon das Programm.
- Der Berechnungsmechanismus ist nicht Teil des Programms.
  $\hookrightarrow$ kann leicht ausgetauscht werden.
- Deklaratives „Denken“ ist oft einfacher als prozedurales „Denken“.
- Bei Logischer Programmierung ist der Output eine logische Konsequenz des Programms.
- Deklarative Programme sind (meist) sehr flexibel bezüglich der Fragestellung.

Als ein Beispiel für Deklarative Programmierung betrachten wir:

- Logische Programmierung – Sprache: Prolog

Logische Programmierung

Konventionelle Programmierung

- Prozedurale Denkweise als Grundlage
- Program = Algorithmus + Datenstruktur

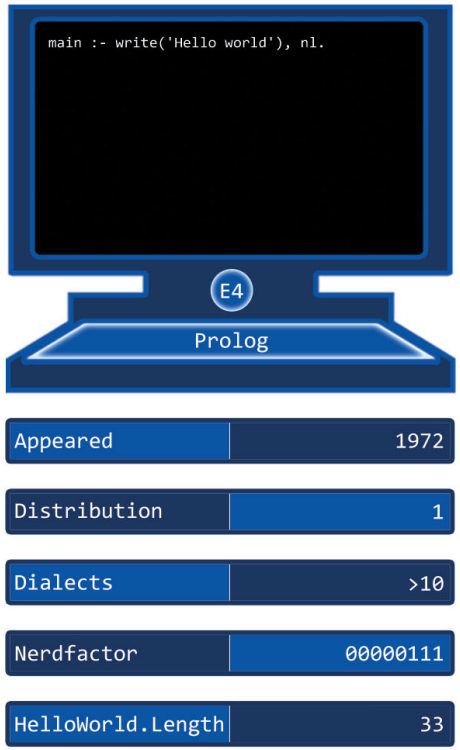
Logische Programmierung

- Algorithmus = Logik + Steuerung
- Logik definiert Wissen und Zielsetzung
- Steuerung definiert die Strategie dieses Wissen einzusetzen.
- Programm = Logik + Datenstruktur + Steuerung
- Der Programmierer sieht aber nur die Logik

Prolog

Programmiersprache **Prolog**:

- Populärste logische Programmiersprache.
- Der Name kommt vom Französischen „Programmation en Logique“.
- In den 1970ern maßgeblich von Alain Colmerauer entwickelt.
- Query-oriented: Berechnungen werden durch eine Frage in Form einer Formel gestartet.
- Prolog Implementierungen sind für die meisten Plattformen frei verfügbar z.B.: `http://www.swi-prolog.org/`.



Johann Weiher – `http://codequartett.de`

Teile eines Prolog-Programms

- Prolog-Programme bestehen aus Aussagen und nicht aus Anweisungen.
- Die Reihenfolge der Aussagen hat keine Auswirkung auf die Semantik/Ausgabe des Programms.
- Atome werden aus Prädikaten gebildet.
- Aussagen werden als Horn Klauseln codiert:
  - Fakten
  - Regeln
  - Anfragen (Zielklausel)

Teile eines Prolog-Programms - Fakten

Die simpelsten Bestandteile von Prolog Programmen sind **Fakten**, auch „allgemeine Tatsachen“ genannt. Diese stellen elementare Tatsachen dar, oft wird damit die konkrete Probleminstanz/Eingabe codiert.

Hierzu verwendet man Prädikate und Objekte.

Beispiel

- Aussage: „Otto ist lieb“
  Prolog: `lieb(otto)`
- Aussage: „Otto ist Vater von Karl“
  Prolog: `istvater(otto, karl).`

Erinnerung: Bei Prädikaten ist die Reihenfolge der Argumente wichtig. *istvater*(*otto*, *karl*) ist also nicht das gleiche wie *istvater*(*karl*, *otto*).

Prolog Notation: Prädikate und Konstanten werden klein geschrieben. Variablen beginnen mir einem Großbuchstaben.

Teile eines Prolog-Programms - Regeln

Regeln erlauben neue Fakten abzuleiten und haben die Form

```
a:-b1, ..., bn.
```

a ist der Kopf (head) und `b1,...,bn` der Rumpf (body) der Regel.[1]

```
Kopf :- Rumpf
```

Die Regel liest sich als

Wenn alle `bi` gelten dann gilt auch `a`.

und „entspricht“ damit der Implikation

$$a \leftarrow b1 \wedge \cdots \wedge bn$$

[1] :- ist das „neck“-Symbol.

Teile eines Prolog-Programms - Regeln

Beispiel

Regel: Wenn ein Mann Vater eines Kindes ist und dieses Kind Vater eines Kindes ist, dann ist er Großvater.

In Prolog:

```
grossvater(Mann):-istvater(Mann,Kind),istvater(Kind,Enkel).
```

`grossvater, istvater` sind Prädikate.
`Mann, Kind, Enkel` sind Variablen.

Das entspricht der logischen Formel:

$$\forall x, y, z(Grossvater(x) \leftarrow (Istvater(x, y) \wedge Istvater(y, z)))$$

Um das Prädiktat grossvater vollständig zu definieren braucht man noch:

```
grossvater(Mann):-istvater(Mann,Kind),istmutter(Kind,Enkel).
```

Teile eines Prolog-Programms - Rekursive Regeln

Wir wollen eine Relation definieren die alle Vorfahren kennt:

```
vorfahre(A,B) :- elternteil(A, B).
vorfahre(A,B) :- elternteil(A, X), elternteil(X, B).
vorfahre(A,B) :- elternteil(A, X), elternteil(X, Y),
                                   elternteil(Y, B).
vorfahre(A,B) :- elternteil(A, X), elternteil(X, Y),
                  elternteil(Y, Z), elternteil(Z,B).
...
```

Mit diesem Schema bräuchten wir (unendlich) viele Regeln.
Besser ist wir verwenden eine rekursive Regel.

```
vorfahre(A, B) :- elternteil(A, B).
vorfahre(A, B) :- elternteil(A, X), vorfahre(X, B).
```

Wichtig: Der rekursive Aufruf muss auf der rechten Seite stehen.
Da Prolog Regeln von Links nach Rechts abarbeitet.

Teile eines Prolog-Programms - Rekursive Regeln

Rekursive Regeln können lange Schleifen oder sogar Endlos-Schleifen erzeugen.

Faustregeln zum Umgang mit Rekursionen:

- Zuerst eine Regel definieren die dem Rekursionsanfang entspricht
- Im Rumpf der rekursiven Regel:
  - Die nicht-rekursiven Prädikate zuerst.
  - Die rekursiven Prädikate ganz rechts

Teile eines Prolog-Programms - Anfragen / Queries

Berechnungen werden durch **Anfragen** (Queries) gestartet. Diese haben die Form

```
?-b1, ..., bn.
```

Die Anfrage liest sich als

Gilt b1, b2, . . . ,und bn.

Beispiel

Frage: Ist fritz Großvater?
Prolog: `?-grossvater(fritz)`

Frage: Wer ist Großvater?
Prolog: `?-grossvater(X)`

SWI-Prolog

Freies Prolog System für Windows, Linux, und Mac OS:
`http://www.swi-prolog.org`

Aufruf mit `swipl` oder `prolog` (UNIX) / `swipl-win.exe` (Windows)
Startet in einem Modus in dem nur Anfragen gestellt werden können.

Programm laden: (für Programmdatei program.pl)

- Beim Start mit `prolog -f program.pl`
- Im laufenden Betrieb mit `consult(program.pl).` oder `[program.pl].`

Programm zur Laufzeit schreiben:

- Mit `consult(user).` oder `[user].` starten
- Programm eingeben
- mit Strg+D (Ctrl+D) beenden

Es gibt auch eine online Version:
`http://pengines.swi-prolog.org/apps/swish/index.html`

Typisches Prolog-Programm

Ein typische Prolog-Programm besteht aus

- Der **Wissensbasis**:
  - Aufstellen von Fakten
  - Aufstellung von Regeln
- Ergänzenden **Kommentaren** (`/* Kommentar */`)
- **Anfragen** zum starten von Berechnungen.

Eine Wissensbasis:

```
/* Fakten */
istvater(fritz, paul). istvater(paul, karl).
istvater(zeus, herkules). istmutter(alkmene,herkules).
/* Regeln */
grossvater(Mann):-istvater(Mann,Kind),istvater(Kind,Enkel).
```

Anfragen wären:

```
?- grossvater(zeus).
?- grossvater(X).
?- grossvater(X), istvater(X, paul).
```

Antwort: false.
Antwort: X=fritz
Antwort: X=fritz

Anfragen Auswerten I

Beispiel

Anfrage: `?- grossvater(zeus).`

Wir haben nur eine Regel um grossvater abzuleiten:
`grossvater(Mann):-istvater(Mann,Kind),istvater(Kind,Enkel).`

Wir setzen `Mann=zeus`
`grossvater(zeus):-istvater(zeus,Kind),istvater(Kind,Enkel).`

Wir suchen einen Wert für `Kind` sodass `istvater(zeus,Kind)` wahr ist.
Die einzige Möglichkeit ist `Kind=herkules`
`grossvater(zeus):-istvater(zeus,herkules),istvater(herkules,Enkel).`

Nun gibt es keinen Wert für `Enkel` sodass `istvater(herkules,Enkel)` wahr ist.
$\hookrightarrow$ **Antwort**: false.

Anfragen Auswerten II

Beispiel

Anfrage: `?- grossvater(X).`

`grossvater(X):-istvater(X,Kind),istvater(Kind,Enkel).`

Wir haben 3 Möglichkeiten `istvater(X,Kind)` wahr zu machen.

1. (X,Kind)=(fritz, paul):
   `grossvater(fritz):-istvater(fritz,paul),istvater(paul,Enkel).`
   Die einzige Möglichkeit für `istvater(paul,Enkel)` ist `Enkel=karl`
   ↪ `X=fritz` ist Lösung
2. (X,Kind)=(paul, karl):
   `grossvater(paul):-istvater(paul,karl),istvater(karl,Enkel).`
   `istvater(karl,Enkel)` ist immer falsch
   ↪ `X=paul` ist keine Lösung
3. (X,Kind)=(zeus, herkules): analog zu (2) `X=zeus` ist keine Lösung

↪ **Antwort**: X=fritz; false

Anfragen Auswerten III

Beispiel

Anfrage: `?- grossvater(X), istvater(X, paul).`

Zuerst werten wir `grossvater(X)` aus und bekommen wieder `X=fritz`:
`?- grossvater(fritz), istvater(fritz, paul).`

Da `istvater(fritz, paul)` wahr ist erhalten wir

Antwort: X=fritz; false

Closed World Assumption

Prolog folgt bei Anfragen der **Closed World Assumption**:

- Wenn sich etwas nicht aus Fakten und Regeln ableiten lässt ist es falsch.

Sie kennen dieses Prinzip vielleicht von

- Datenbanken
- Zugfahrplänen
- Vorlesungsverzeichnissen

Eine Alternative wäre die Open World Assumption, bei der ein Atom nur falsch ist wenn sich das auch ableiten lässt.

Variablen bei Anfragen

Eine Wissensbasis:

```
/* Fakten */
frau(alkmene). frau(aphrodite). frau(harmonia). frau(hera).
mann(zeus). prim(zwei).
```

Wollen wir alle Frauen (oder eine beliebige Frau) ermitteln könnten wir alle Objekte einzeln testen: `?-frau(zwei).`, `?-frau(alkmene).`, ...

Es geht aber natürlich einfacher: `?-frau(X).` (`X` ist eine Variable)

Als Ausgabe erhalten wir:

```
X=alkmene;
X=aphrodite;
X=harmonia;
X=hera;
false.
```

In SWI Prolog wird zunächst nur die erste Antwort ausgegeben. Weitere Antworten können mit ";“ abgerufen werden.

Konjunktion bei Anfragen

Eine **Wissensbasis**:

```
mag(utta, otto). mag(utta,milch).
mag(otto,essen). mag(otto,utta). mag(otto,milch).
```

In Anfragen können wir Konjunktion nutzen.

- **Frage:** Mag Otto Utta und mag Utta essen ?
  Prolog: `?- mag(otto,utta), mag(utta,essen).` **Antwort**: false.
- **Frage:** Mag Otto Utta und mag Utta Milch?
  Prolog: `?- mag(otto,utta), mag(utta,milch).` **Antwort**: true.

Das hätten wir aber auch leicht mit zwei getrennten Anfragen herausfinden können. Ein interessanteres Beispiel:

- **Frage:** Gibt es etwas das sowohl Otto als auch Utta mögen
  Prolog: `?- mag(otto,Etwas), mag(utta,Etwas).`
  Antwort: Etwas = milch.

Prolog - Beispiel

Eine Wissensbasis:

```
/* Fakten */
maennlich(uranus). maennlich(kronos).
weiblich(gaia). weiblich(rhea).
/* eltern(X,Y,Z) - X,Y sind Eltern von Z */
eltern(uranus,gaia,kronos). eltern(uranus,gaia,rhea).
```

Wir wollen eine Relation `istbruder` definieren:

```
/* Regel */
istbruder(Bruder,X):-maennlich(Bruder),eltern(Y,Z,Bruder),
                                    eltern(Y,Z,X),Bruder\==X.
```

Anfragen:

```
?- istbruder(kronos,rhea).
```
Antwort: true.

```
?- istbruder(kronos,Person).
```
Antwort: Person=rhea

```
?- istbruder(kronos,X),istbruder(X,kronos).
```
Antwort: false.

Prolog - Beispiel (Anfrage 1 auswerten)

Beispiel

Anfrage: `?- istbruder(kronos,rhea).`

Wir haben nur eine Regel um `istbruder` abzuleiten:
`istbruder(Bruder,X):-maennlich(Bruder),eltern(Y,Z,Bruder), eltern(Y,Z,X),Bruder\==X.`

Wir setzen `(Bruder,X)=(kronos,rhea)`
`istbruder(kronos,rhea):-maennlich(kronos),eltern(Y,Z,kronos), eltern(Y,Z,rhea),kronos\==rhea.`

`maennlich(kronos)` ist wahr. Wir müssen `eltern(Y,Z,kronos)` wahr machen. Daher setzen wir `(Y,Z)=(uranus,gaia)`:

`istbruder(kronos,rhea):-maennlich(kronos), eltern(uranus,gaia,kronos), eltern(uranus,gaia,rhea), kronos\==rhea.`

Jetzt sind auch `eltern(uranus,gaia,rhea)` und `kronos\==rhea` wahr.
$\hookrightarrow$ **Antwort**: true.

Prolog - Beispiel (Anfrage 2 auswerten)

Beispiel

Anfrage: `?- istbruder(kronos,Person).`

Wir haben nur eine Regel um `istbruder` abzuleiten:
`istbruder(Bruder,X):-maennlich(Bruder),eltern(Y,Z,Bruder), eltern(Y,Z,X),Bruder\==X.`

Wir setzen `Bruder=kronos`
`istbruder(kronos,X):-maennlich(kronos),eltern(Y,Z,kronos), eltern(Y,Z,X),kronos\==X.`

`maennlich(kronos)` ist wahr. Wir müssen `eltern(Y,Z,kronos)` wahr machen. Daher setzen wir `(Y,Z)=(uranus,gaia)`:

`istbruder(kronos,rhea):-maennlich(kronos), eltern(uranus,gaia,kronos), eltern(uranus,gaia,X),kronos\==X.`

Es gibt zwei Möglichkeiten `eltern(uranus,gaia,X)` wahr zu machen `X=kronos` und `X=rhea`. Im ersten Fall ist `kronos\==kronos` falsch im zweiten `kronos\==rhea` wahr.
↪ **Antwort**: Person=rhea; false.

Prolog - Beispiel (Anfrage 3 auswerten)

Beispiel

Anfrage: `?- istbruder(kronos,X),istbruder(X,kronos).`

Wir betrachten zuerst `istbruder(kronos,X)` und bekommen wie bei Anfrage 2 `X=rhea; false.`. Wir setzen also `X=rhea`.

```
?- istbruder(kronos,rhea),istbruder(rhea,kronos).
```

Jetzt betrachten wir `istbruder(rhea,kronos)`.

```
istbruder(rhea,kronos):-maennlich(rhea),eltern(Y,Z,rhea),
eltern(Y,Z,kronos),rhea\==X.
```

`maennlich(rhea)` ist falsch und damit auch `istbruder(rhea,kronos)`

$\hookrightarrow$ **Antwort**: false.

Mehr Beispiele

Beispiel (Verwendung von Regeln)

Nehmen wir an, wir wollen folgende Tatsache angeben:

„Otto mag Essen.“

Wir können schreiben:
„Otto mag Brot.“ und “Brot is essbar.“
„Otto mag Wurst.“ und “Wurst is essbar.“
„Otto mag Käse.“ und “Käse is essbar.“

Besser: Eine allgemeine Regel der Form:

„Wenn ein Objekt *o* essbar ist, dann mag Otto *o*.“

In Prolog schaut dass dann so aus:

```
essbar(brot). essbar(wurst). essbar(kaese).
mag(otto, X) :- essbar(X).
```

Mehr Beispiele

Beispiel (Verwendung von Regeln)

```
essbar(brot). essbar(wurst). essbar(kaese).
mag(otto, X) :- essbar(X).
```

- `?- mag(otto,brot).` **Antwort**: true.
- `?- mag(otto,X).` **Antwort**: X=brot; X=wurst; X=kaese; false.
- `?- mag(otto,brot),mag(otto,kaese).` **Antwort**: true.
- `?- mag(otto,brot),mag(otto,otto).` **Antwort**: false.

Mehr Beispiele

Beispiel (Verkehrsmittel)

Wir betrachten eine verschiedene Verkehrsmittel in einer Stadt:

- Bus, Bahn, Fahrrad, Auto, ...

Wir wollen öffentliche Verkehrsmittel von privaten unterscheiden.

```
/* Fakten */
oeffentlich_vm(bus). oeffentlich_vm(bahn).
privat_vm(fahrrad). privat_vm(auto).
```

Fahrrad fahren und öffentliche Verkehrsmittel sind in der Stadt billig.

```
billig(fahrrad).
/* Regeln */
billig(X):-oeffentlich_vm(X).
```

Mehr Beispiele

Beispiel (Verkehrsmittel)

Jetzt können wir verschiedene Fragen stellen:

- Ist Autofahren billig?
  `?- billig(auto).` **Antwort**: false.
- Welche Verkehrsmittel sind billig?
  `?- billig(X).` **Antwort**: X=bus; X=bahn; X=fahrrad; false.
- Welche privaten Verkehrsmittel sind billig?
  `?- privat_vm(X), billig(X).` **Antwort**: X=fahrrad; false.

Prolog - Generate & Test

Beispiel

Drei Buben Fritz, Hans und Karl rudern mit ihren Booten.

- Die Boote haben die Farben rot, grün und blau.
- Der Bub im roten Boot ist der Bruder von Fritz.
- Hans sitzt nicht im grünen Boot.
- Der Bub im grünen Boot hat Streit mit Fritz.

Wer sitzt in welchem Boot?

Lösungsparadigma Generate & Test:

- Die Lösungen werden in ein Prädikat `loesung(`$\vec{X}$`)` gespeichert.
- In einem ersten Teil einer Regel `generate(`$\vec{X}$`)` generieren wir alle Lösungskandidaten
- In einem zweiten Teil der Regel `test(`$\vec{X}$`)` testen wir alle Bedingungen die eine Lösung erfüllen muss.

`loesung(`$\vec{X}$`):- generate(`$\vec{X}$`), test(`$\vec{X}$`).`

Prolog - Generate & Test

Drei Buben Fritz, Hans und Karl rudern mit ihren Booten.

- Die Boote haben die Farben rot, grün und blau.
- Der Bub im roten Boot ist der Bruder von Fritz.
- Hans sitzt nicht im grünen Boot.
- Der Bub im grünen Boot hat Streit mit Fritz.

Wer sitzt in welchem Boot?

1) Wir nutzen ein Prädikat `loesung(B_rot,B_gruen,B_blau)`, das sich als `B_rot` sitzt im roten Boot, etc. liest.

2) Dann generieren wir Lösungskandidaten:

Prolog:

```
bub(fritz). bub(hans). bub(karl).

loesung(B_rot,B_gruen,B_blau):-
bub(B_rot), bub(B_gruen), bub(B_blau),
```

Prolog - Generate & Test

Drei Buben Fritz, Hans und Karl rudern mit ihren Booten.

- Die Boote haben die Farben rot, grün und blau.
- Der Bub im roten Boot ist der Bruder von Fritz.
- Hans sitzt nicht im grünen Boot.
- Der Bub im grünen Boot hat Streit mit Fritz.

Wer sitzt in welchem Boot?

3) Wir testen Lösungskandidaten:

Prolog:

```
bub(fritz). bub(hans). bub(karl).

loesung(B_rot,B_gruen,B_blau):-
    bub(B_rot), bub(B_gruen), bub(B_blau),
        B_rot\==B_gruen, B_rot\==B_blau, B_gruen\==B_blau,
        B_rot\==fritz, hans\==B_gruen, B_gruen\==fritz.
```

Die Lösung bekommen wir mit `?-loesung(B_rot,B_gruen,B_blau).`

Ansätze für logische Programmiersprachen

Prolog folgt dem Ansatz basierend auf **Theorembeweisern**

1. Generiere Problem Repräsentation.
2. Die Lösung wird durch eine Ableitung einer Formel / Query gegeben.

Ansatz der **Model-Generierung**

1. Generiere Problem Repräsentation.
2. Die Lösungen werden durch die Modelle der Repräsentation gegeben.

Beispiele dafür Model-generierende Sprachen

- SATisfiability Testing: Generiert Modelle für Aussagenlogische Formeln.
- Answer-Set Programming: Generiert „stable models“ für logische Programme.

Ergänzende Materialien (Prolog)

SWI-Prolog Manual.
http://www.swi-prolog.org/

Wikibooks
Prolog.
https://en.wikibooks.org/wiki/Prolog

Logic Programming with Prolog
Max Bramer
Springer, 2013, ISBN: 978-1-4471-5486-0

Learn Prolog Now!
Patrick Blackburn, Johan Bos, and Kristina Striegnitz
http://www.learnprolognow.org/

Zusammenfassung

Programmiersprachen folgen verschiedenen Programmierparadigmen.

- Klassische Programmiersprachen: Programmiererin gibt an wie etwas berechnet werden soll
  - Imperative Programmierung
  - Objektorientierte Programmierung
  - Parallele Programmierung
  - ...
- Deklarative Programmierung: Programmiererin definiert was berechnet werden soll
  - Beispiel: Logische Programmierung (Prolog)