Prolog

Idee (Kowalski 1979)
Algorithmus = Logik + Kontrolle
Logik: was ist das Problem?
Kontrolle: wie löse ich es geschickt?

Sichtweisen:
- Theorembeweiser (Resolution)
- Datalog (Datenbankanfragesprache)
- Programmiersprache

Hier:
- Programmiersprache kennenlernen,
- Suchverfahren darin implementieren
- Hintergrund: Beweisen

Modellieren in Prolog

Der Problembereich wird durch
- Fakten und
- Regeln ausgedrückt.

Ein Problem wird durch eine
- Frage ausgedrückt.

Die Kontrolle stellt das Prologsystem zur Verfügung.
Es gibt nur wenige Möglichkeiten für die Programmiererin, Kontrollstrukturen zu explizieren.

Fakten

Ein Fakt ist ein positives Literal.
Es besteht aus
- einem Prädikatsymbol (beginnend mit Kleinbuchstaben) und
- Argumenten.

Die Anzahl der Argumente ist die Stelligkeit.
Dasselbe Prädikatsymbol bei verschiedener Stelligkeit bezeichnet verschiedene Prädikate.

vater (X, maria).	**kinder (uta, maria, mario).**	**kinder/3**
mutter (ulf, mario).	**kinder (uta, [maria, mario]).**	**kinder/2**
vater (_v, anna).	**kinder (maria, anna).**	**kinder/2**

kinder/3 und **kinder/2** sind verschiedene Prädikate!

Argumente eines Prädikates

Ein Argument ist
- eine Variable (beginnend mit Großbuchstaben oder _),
- eine Konstante (Zahl oder Buchstaben beginnend mit Kleinbuchstaben),
- eine Funktion oder
- eine Liste.

kinder (uta, maria, mario).	**kinder/3**
kinder (uta, [maria, mario]).	**kinder/2**

Funktionen und Listen

Funktionen
meist arithmetisch, wobei eine Argumentstelle für den Eingabewert, die andere für das berechnete Ergebnis reserviert ist.
sin (X, Y)   bedeutet sin (X) = Y
Um Eingabe- und Ausgabeposition zu markieren, wird in der Dokumentation meist + und - verwendet.
% sin/2, sin (+, -)

:- ensure_loaded (library (math)).

Listen

Listen sind intern auch Funktionen.

. (Kopf, Rest)

geschrieben als

[Kopf | Rest]

kinder (uta, [maria | X]).

Regeln

Hornklauseln sind Klauseln mit nur einem positiven Literal.
Dies ist der Klauselkopf.

Der Klauselkörper ist eine Disjunktion negierter Literale.
{oma (X, Y), ¬ mutter (X, Z), ¬ mutter (Z,Y)}

Hornprogramme sind eine Konjunktion von Hornklauseln.
{oma (X, Y), ¬ mutter (X, Z), ¬ mutter (Z,Y)}
{oma (X, Y), ¬ mutter (X, Z), ¬ vater (Z,Y)}

Prolog-Notation

oma (X, Y) :- mutter (X,Z), mutter (Z,Y).
oma (X, Y) :- mutter (X,Z), vater (Z,Y).

Fragen

Fragen bestehen aus einer nicht-leeren Menge negativer Literale.
{¬ mutter(uta, maria), ¬ mutter(maria, X)}
{¬ mutter(uta, X), ¬vater (X, Y)}

Prolog-Notation

?- mutter(uta, maria), mutter(maria,X). („?-“ ist der Prompt des Interpreters)
:- mutter(uta, X), vater (X, Y). (als Befehl in Dateien)

Beschreibung eines Problembereichs (interpretiert)

Fakten eintragen (interaktiv im Interpreter)

?- assert(vater(ulf, maria)).
?- assert(mutter(uta, maria)).
?- assert(mutter(uta, mario)).
?- assert(mutter(maria, anna)).

Regeln eintragen
?- assert((oma(X, Y) :- mutter(X, Z), mutter(Z, Y))).
?- assert((oma(X, Y) :- mutter(X, Z), vater(Z, Y))).

Beschreibung eines Problembereichs (compiliert)

Fakten und Regeln in eine Datei eintragen.
emacs verwandschaft.pl

```
% Familie von Maria mit vater/2, mutter/2
vater( ulf, maria).
mutter( uta, maria).
mutter( uta, mario).
mutter( maria, anna).
oma( X, Y) :- mutter( X, Z), mutter( Z,Y).
oma( X, Y) :- mutter( X, Z), vater( Z,Y).
```

Prolog aufrufen mit **prolog**
?- [verwandschaft].

Problemlösen

Wer ist die Oma von Anna?

?- oma (Oma, anna).

Oma = uta

Wer sind die Kinder von Uta?

?- mutter(uta, X).

X = maria ;
X = mario ;
no.

Resolution

¬ oma (Oma, anna) **oma (uta, anna), ¬ mutter (uta, maria), ¬ mutter(maria, anna)**

¬ mutter (uta, maria), ¬ mutter(maria, anna) **mutter (uta, maria)**

¬ mutter(maria, anna) **mutter(maria, anna)**

□

Klappt nur, wenn **Oma** mit **uta** unifiziert wird.

Substitution

Eine Substitution ist eine endliche Menge
{ V1 / t1, V2 / t2, ..., Vn / tn }, wobei **Vi≠ Vj** für alle **i≠j**
Vi / ti bedeutet, daß die Variable **Vi** an den **Term ti** gebunden wird.

Eine Substitution anwenden, heißt, alle Vorkommen der Variablen gleichzeitig durch die betreffenden Terme zu ersetzen.

oma (Oma, anna) σ **= oma (uta, anna) mit** σ**: { Oma / uta}**

Identitätssubstitution **{ }**

Unifikation

Eine Substitution σ wird Unifikator genannt, wenn
für eine endliche Menge von Literalen L
die Anwendung der Substitution
eine Menge mit nur einem Element ergibt.

L:
vater (X, mario).
vater (Y, mario).
vater (ulf, Z).

σ **: { X / ulf, Y / ulf, Z / mario}**

Lσ **: { vater (ulf, mario) }**

Allgemeinster Unifikator

Ein Unifikator σ ist der allgemeinste, wenn für jeden anderen Unifikator τ eine Substitution η existiert, so dass
τ = σ η
Wenn man den allgemeinsten Unifikator durch eine weitere Substitution spezialisiert, erreicht man einen anderen, nicht allgemeinsten Unifikator.

unterhalt(ulf, maria, X).
unterhalt(Y, maria, Z).
unterhalt(Y, maria, alimente(Y, V)).

σ **: { Y / ulf, X / alimente(ulf, V), Z / alimente(ulf, V)}**
τ **: {Y / ulf, X / alimente(ulf,1000), Z / alimente(ulf, 1000), V/1000}**
η **: { V / 1000}**

Unifikationsalgorithmus

Eingabe: nicht-leere Menge L von Literalen
σ **:= { }**

Solange L mehr als ein Literal enthält:
gehe von links nach rechts alle Literale durch und suche nach Unterschieden.
Wenn keines der unterscheidenden Zeichen eine Variable ist, halte an "nicht unifizierbar".

Sei X ist Variable und t der im anderen Literal beginnende Term.
Wenn X in t vorkommt, halte an "nicht unifizierbar".
Sonst σ:= σ **{X/t}**

Gib aktuelles σ **als allgemeinsten Unifikator aus.**

Beispiel

L:
unterhalt(ulf, maria, X).
unterhalt(Y, maria, Z).
unterhalt(Y, maria, alimente(Y, V)).

1. Unterschied
σ**:= {Y/ulf}**

Lσ:
unterhalt(ulf, maria, X).
unterhalt(ulf, maria, Z).
unterhalt(ulf, maria, alimente(ulf, V)).

2. Unterschied
σ:= σ **{X/alimente(ulf, V), Z/alimente(ulf, V)}**

Ergebnis

Lσ:
unterhalt(ulf, maria, X).
unterhalt(ulf, maria, Z).
unterhalt(ulf, maria, alimente(ulf, V)).

2. Unterschied
σ:= σ **{X/alimente(ulf, V), Z/alimente(ulf, V)}**

Lσ:
unterhalt(ulf, maria, alimente(ulf, V))

Kontrollfluß

- Prolog arbeitet von **links nach rechts**.
- Erst wird ein **Kopfliteral** gesucht, das mit der Frage unifiziert.
- Dann wird das erste **Körperliteral** versucht, zu beweisen.
- Gelingt dies, wird das nächste Körperliteral versucht, zu beweisen.
- Gelingt dies nicht, geht Prolog zum nächstliegenden linken Literal zurück und versucht einen anderen Beweis für dies Literal.
- Wenn es keine Alternativen mehr für ein Literal gibt, das nicht bewiesen werden konnte, so wird folglich bis zum Kopf zurückgegangen, d.h. eine andere Klausel versucht, mit der Frage zu unifizieren.

Folglich: wenn das am weitesten rechts stehende Literal zu beweisen ist, dann auch die Frage.

Beispiel

?- oma(Oma, anna) .

vater (ulf, maria).
mutter (uta, maria).
mutter (uta, mario).
mutter (maria, anna).
oma (X, Y) :- mutter (X,Z), mutter (Z,Y).
oma (X, Y) :- mutter (X,Z), vater (Z,Y).

σ**: {X/Oma, Y/anna)** →

oma (Oma, anna) , ¬ mutter (Oma, Z), ¬ mutter (Z, anna)

σ**: {Oma/uta, Z/maria}**

oma (Oma, anna) , ¬ mutter (Oma, maria), ¬ mutter (maria, anna)

Zurückziehen

- Kann ein Literal im Klauselkörper nicht bewiesen werden, geht Prolog zurück zum links nächsten Literal.
- Bereits vorgenommene Unifikationen werden zurückgezogen.
- Unifikation mit einem anderen Literal wird versucht.

Beispiel

?- oma(Oma, anna).

vater (ulf, maria).
mutter (uta, mario).
mutter (uta, maria).
mutter (maria, anna).
oma (X, Y) :- mutter (X,Z), mutter (Z,Y).
oma (X, Y) :- mutter (X,Z), vater (Z,Y).

σ: **{X/Oma, Y/anna)** →

oma (Oma, anna) , ¬ mutter (Oma, Z), ¬ mutter (Z, anna)

σ: **{Oma/uta, Y/anna, Z/mario}**

oma (Oma, anna) , ¬ mutter (Oma, maria), ¬ mutter (mario, anna)

σ: **{Oma/uta, Y/anna, Z/maria }**

Rekursion

Wie immer:
- Abbruchbedingung (Induktionsbasis)
- Rekursiver Aufruf (Induktionsschritt)

```
edge(X, Y).
connected(X,X).                                   % Abbruch
connected (X, Z) :- edge (X, Y), connected (Y, Z). % Aufruf
```

Immer Endrekursion verwenden!
Sonst keine Reduktion, sondern unendlicher Zirkel.

In anderen Worten:

- Um ein Ziel zu erfüllen, wird nach einer passenden Klausel gesucht.
- Ob eine Klausel passt, wird anhand des ersten Arguments des Prädikats festgestellt (*first argument indexing*).
- Gibt es mehrere passende Klauseln wird ein *choice point* erzeugt.
- Ein Prädikat heißt deterministisch, wenn nach Erzeugen der ersten Lösung kein choice point bleibt.

Box-Modell

Call G: erster Aufruf des Zieles G

Exit S: erfolgreich bewiesen, Rückgabe der Substitution, evtl. verbleibt ein choice point

Fail G: Fehlschlag, G kann nicht bewiesen werden

Redo G: Sprung zurück zum nächsten choice point

Debugging

trace: Beobachtung des Kontrollflusses (?- trace.)

notrace: Abschalten der Beobachtung

spy(P): Beobachtung des Prädikats P

nospy(P): Abschalten der Beobachtung

nospyall: Abschalten der Beobachtung aller Prädikate

debug: Normale Abarbeitung bis zum Erreichen des ersten Beobachtungspunktes

Cut !

! sorgt dafür, dass für ein Ziel kein alternativer Beweis versucht wird.

! sorgt dafür, dass alle choice points vor dem ! gelöscht werden.

! vermeidet unnötige Beweisversuche.

$$A :- B_1, ..., B_k, !, B_{k+1}, ..., B_n$$

- Alternative Lösungen für $B_1, ..., B_k$ suchen.
- Alternative Lösungen für $B_{k+1}, ..., B_n$ suchen.
- Während versucht wird, $B_{k+1}, ..., B_n$ zu erfüllen, werden die choice points für $A, B_1, ..., B_k$ gelöscht.
- Kann B_{k+1} nicht erfüllt werden, soll ! von rechts nach links überschritten werden, kann A nicht erfüllt werden.

Fail

fail ist immer falsch,
erzwingt Rücksetzen.

```
student(tom). student (uta). student(paul). student(anne).

alle_studenten :- student(X), write (X), nl, fail.
alle_studenten.
```

Kombination

- Ausschluss einer Verwendungsweise einer Klausel.

```
lebewesen (mensch).
lebewesen (hund).
tier (mensch) :- !, fail.
tier (X) :- lebewesen (X).
```