Programmiertechnik II

Prolog

Überblick

- Deklaratives Programmieren (“Programmieren in Logik”)
- 1972 entwickelt von A. Colmerauer
- vor allem für KI-Systeme verwendet
  - Kern-Sprache des japanischen “Fifth Generation Computer Systems Project” (1982-1992)
- basiert auf Prädikatenkalkül
  - Einschränkung auf Horn-Klauseln (Disjunktionen mit höchstens einem positiven Literal: Implikationen)
- Clocksin, Mellish: Programming in Prolog

Objekte und Beziehungen

- Objekte: Abstrakte Dinge, die an Beziehungen teilnehmen
  - hier nicht Objekte aus OO-Sprachen, also nicht durch Zustand, Verhalten, Identität beschrieben, keine Klassen, keine Polymorphie
- Aussagen: Verknüpfungen von Objekten über Beziehungen
  - Hans besitzt das Buch.
  - Der Edelstein ist wertvoll.
  - Anna ist die Schwester von Hans.
- Fragen: Test, ob bestimmte Aussagen erfüllt sind
  - eventuell mit Bindung von Variablen: Lösungen
  - Besitzt Hans das Buch?
  - Wer ist die Schwester von Hans?

Programmieren in Prolog

- Formulierung von Prädikaten:
  - Aussagen (facts): Beziehungen zwischen (konkreten) Objekten
  - Regeln (rules): Beschreibung weiterer Eigenschaften auf Basis existierender Beziehungen
  - Fragen (questions/queries): Formulierung des zu lösenden Problems
- besitzt(hans, buch).
- schwester(X, Y) :- weiblich(X), vater(X, V1), vater(Y, V2), V1=V2.
- ?- besitzt(hans, buch).
- ?- schwester(X, hans).

Fakten

- Relation mit Parametern
- Name der Relation: Buchstabe/Ziffern/_, beginnend mit Kleinbuchstaben
  - Namen haben den Datentyp “Atom”
  - Name hat für Prolog-System keine weitere Bedeutung (Ausnahme: vordefinierte Prädikate)
    - Assoziation von Objektname und “wirklichem” Objekt durch Entwickler/ Anwender
- Parameter in Klammern
- Fakt endet mit Punkt (.)
- Parameter: Atome, Zahlen, Strukturen, Listen, ...
- Gesamtheit aller Fakten: Wissensbasis

```
besitzt(hans, buch).
```

Fragen

- An interaktivem Prompt (?-) einzugeben
- Syntax wie Fakten
- Anfrage wird mit Wissenbasis verglichen (“Unifikation”)
  - Name und Stelligkeit der Frage muss mit dem Fakt übereinstimmen
  - Parameter müssen in Übereinstimmung gebracht werden
- Antwort “yes”, wenn Übereinstimmung erfolgreich
  - ;<enter> sucht nach weiteren Lösungen
- Antwort “no”, wenn keine passenden Fakten gefunden wurden
  - Closed World Assumption (CWA): Annahme, dass Faktenliste vollständig ist

?- besitzt(werner, buch).

no

Variablen

- Variablennamen beginnen mit Großbuchstaben
- Sichtbarkeit beschränkt auf Anfrage oder Regel
- Verwendung in Anfragen: Platzhalter für Lösung
  - Ist X die Schwester von Hans?
- Verwendung in Regeln: Platzhalter für Quantifizierung
  - X ist eine Großstadt, wenn X eine Stadt ist und die Zahl der Einwohner von X größer als 100000 ist.
- Variablenname _ ist anonyme Platzhaltervariable

?- besitzt(X, buch).

X=hans

yes

Konjunktionen

- Überprüfen mehrerer Prädikate
- einzelne Prädikate komma-verknüpft
  - Komma ausgesprochen “and”

  ?- likes(john, mary), likes(mary, john).
- Überprüfung von links nach rechts
  - Variablen werden in jedem Term gebunden

    ?- likes(mary, X), likes(john, X).
- Falls ein Prädikat scheitert, werden weitere Lösungen des vorigen Prädikats probiert: Backtracking
  - Bindung gebundener Variablen wird wieder aufgehoben; bei neuer Lösung erfolgt neue Bindung

Regeln

- Allgemeine Aussagen über Mengen von Objekten
  - Implizite All-Quantifizierung durch Variablen
- Regelkopf: prädikat(muster, muster, ...)
  - Prädikat, für das eine Regel definiert werden soll
- Regelkörper: Konjunktion von Prädikaten, die erfüllt sein müssen
  - Getrennt vom Kopf durch :-
  - Beendet mit Punkt (.)
- Wenn-dann-Aussagen
  - Ich benutze einen Regenschirm wenn es regnet
- Definitionen
  - Ein Vogel ist ein Tier das Federn hat

  X ist ein Vogel wenn:
  X ist ein Tier, und
  X hat Federn
  - X ist Schwester von Y wenn:
    - X weiblich ist und
    - X und Y die gleichen Eltern haben

Regeln: Beispiele

- John mag jeden der Wein mag
  - likes(john, X) :- likes(X, wine).
- Konjunktionen: Variablenbindungen sind für alle Prädikate gleich:
  - John mag jeden, der Wein und Essen mag

  likes(john, X) :- likes(X, wine), likes(X, food).

Syntax

- Programme bestehen aus Termen:
  - Konstanten, Variablen, Strukturen
- Konstante: Atome, Zahlen
  - Atome: Bezeichner mit Kleinbuchstaben, oder ‘text ...’
    - Atome der Länge 1: Zeichen
  - Zahlen: ganze, Gleitkommazahlen
- Variablen: beginnen mit Großbuchstaben
  - Sonderfall: _ muss keine konsistente Interpretation haben

Strukturen

- Tupel von Termen
- name(parameter, ...)

```
besitzt(hans, buch(der_fremde_gast, autor(charlotte, link)).
besitzt(hans, buch(staub, autor(patricia, cornwell)).
?- besitzt(hans, buch(_, autor(_, link)).
```

Rekursive Strukturen

- Strukturen, deren Terme gleichartige Strukturen sind
  - knoten(knoten(blatt(3), blatt(4)), blatt(7))
  - Regeln könnten sich auf Strukturmuster beziehen:

"Ein Baum ist geordnet, wenn für alle Knoten gilt: die Elemente im linken Teilbaum sind kleiner als die im rechten."

```
geordnet(blatt(_)).
geordnet(knoten(L, R)) :- geordnet(L), geordnet(R), ganzrechts(L, ML), ganzlinks(R, MR), ML < MR.

ganzrechts(blatt(X), X).
ganzrechts(knoten(_, R), MR) :- ganzrechts(R, MR).

ganzlinks(blatt(X), X).
ganzlinks(knoten(L, _), ML) :- ganzlinks(L, ML).
```

Operatoren

- Binäre Operatoren: zweistellige Terme
  - +(10,5), <(8,9)
- Infixnotation für binäre Operatoren
  - 10+5, 8<9
- Viele Operatoren vordefiniert
- Weitere Operatoren nutzerdefinierbar
  - Vorrang und Assoziativität Teil der Operatordefinition

Gleichheit und Unifikation

- Infix-Prädikat =
  ?- knoten(X) = knoten(4).
  - Ungleich (not unifiable): \=
- Unifikation: Linke und rechte Seite von = werden in Übereinstimmung gebracht
  - Zwei Atome oder Zahlen unifizieren, wenn sie den gleichen Wert haben
  - Zwei Strukturen unifizieren, wenn sie den gleichen Strukturnamen und die gleiche Stelligkeit haben und die Parameter unifizieren
  - Freie Variablen unifizieren mit beliebigen Werten
    - Im Ergebnis wird die Variable an den Wert gebunden
    - Zwei verschiedene freie Variablen unifizieren so, dass sie immer noch frei, aber aneinander gekoppelt (shared) sind.

Arithmetik

- Unifikation gilt insbesondere auch für binäre Operatoren

  ?- 3+4 = 2+5.

  no
- Ausweg: Operator “is” berechnet rechten Operanden, unifiziert Ergebnis mit linkem Operanden

  ?- X is 3+4.

  X = 7

  yes
- Vordefinierte Operatoren: +, -, *, /, // (Integer-Division), rem, mod, ...
- Vordefinierte Funktionen: sin, cos, log, ...
- Vordefinierte Prädikate: <, =<, >=, =:= (numerisch gleich), =\=, between, ...

Listen

- Aufgebaut aus Paaren .(Head, Tail)
  - leere Liste []
- Kurznotation: [A, B, C, D]
  - für .(A, .(B, .(C, .(D, []))))
- Kurznotation: [H | T]
  - für .(H, T)
  - entsprechend: [rot, gruen, blau | Rest]
- Unifikation von Listen: entsprechend Paar-Struktur .(H, T)

Listenoperationen

- member(X, L): Ist X in L enthalten?

```
member(X, [ X | _ ]).
member(X, [ _ | T ]) :- member(X, T).
```

- append(List1, List2, List3): List3 unifiziert mit der Verkettung von List1 und List2
  - alternativ: auch List1 oder List2 können ungebunden sein

```
append([], L1, L1).
append([X | L1], L2, [X | L3]) :- append(L1, L2, L3).
```

- length(List, Int): List enthält Int Elemente

```
length([], 0).
length([H | T], Len) :- length(T, Len1), Len is Len + 1.
```

  - Alternative Definition mit Akkumulator (erlaubt tail recursion)

Generatoren

- Generatoren: Prädikate, die mehrere Lösungen haben
- Listenoperationen können auch als Generatoren verwendet werden
  - member(X, List) zählt (durch Backtracking) alle Elemente von List auf
  - append(X, Y, List) führt (durch Backtracking) alle möglichen Zerlegungen von List durch
  - permutation(List1, List2) testet, ob List2 eine Permutation von List1 ist; fallst List2 ungebunden ist, werden der Reihe nach alle Permutationen generiert

Generate-and-test

- Aufgaben der Art: Gesucht ist ein X, für das P(X) gilt
- Lösungsansatz:
  - generiere alle möglichen X
  - teste, ob sie P(X)
- Beispiel: Sortieren

```
trivial_sort(In, Out) :- permutation(In, Out), is_sorted(Out).

is_sorted([]).
is_sorted([ _ ]).
is_sorted([A, B | Rest]) :- smaller(A,B), is_sorted([B | Rest]).
```

Cut

- Beschneidung des Suchraums: ! (cut-Operator)
  - Syntax wie 0-stelliges Prädikat
  - Lösung des Prädikats gelingt sofort, hat genau eine Lösung
  - beschneidet das Backtracking: Ziele vor dem Cut-Operator liefern keine weiteren Lösungen
- Reihenfolge von Prädikaten in Alternativen und von Regeln einer Prozedur wird relevant

```
?- p(X), !, q(Y).     % hat andere Lösungen als
?- q(Y), !, p(X).
```

Anwendungen des Cut: Ausschließen von Lösungen

- für bestimmte Parameter soll nur eine eingeschränkte Definition des Prädikats gelten
- Beispiel: Buchausleihe; Nutzer, die ihre Leihfrist überzogen haben, dürfen nur die Elementarfunktionen benutzen

```
facility(Pers, Fac) :-
    book_overdue(Pers, Book), !, basic_facility(Fac).
facility(Pers, Fac) :- general_facility(Fac).

basic_facility(reference).
basic_facility(enquiries).

additional_facility(borrowing).
additional_facility(inter_library_loan).

general_facility(X) :- basic_facility(X).
general_facility(X) :- additional_facility(X).
```

Anwendungen des Cut: Optimierung

- Falls bekanntlich nur eine Lösung existiert, kann die Suche nach weiteren Lösungen unterdrückt werden.
- Beispiel: max(X, Y, Max)
- max(X, Y, X) :- X >= Y.
  max(X, Y, Y) :- X < Y.
  - u.U. müssen X und Y zweimal verglichen werden
- max(X, Y, X) :- X >= Y, !.
  max(X, Y, Y).

Anwendungen des Cut: Optimierung (2)

trivial_sort(In, Out) :- permutation(In, Out), is_sorted(Out), !.

- Problem: “bekanntlich nur eine Lösung” gilt oft nur nur für bestimmte Parameterrichtungen
  - trivial_sort(X, [1,3,5,6]) müsste eigentlich alle Permutationen der sortierten Liste generieren, hört aber schon nach der ersten Lösung auf
  - Dokumentation von Prozeduren erläutert, welche Parameter Eingabe(+), Ausgabe(-) oder beides(?) sein können.

Anwendungen des Cut: cut-fail

- Bestimmte Lösungen sollen ausgeschlossen werden
- Prädikat fail scheitert sofort
- Beispiel: “Der durchschnittliche Steuerzahler ist ein Bürger, der ein Einkommen zwischen 2000 und 20000 hat”
  - aber: statt “ist Bürger” steht nur “ist Ausländer” zur Verfügung

```
average_taxpayer(X) :- foreigner(X), !, fail.
average_taxpayer(X) :-
    gross_income(X, Inc),
    2000 < Inc, 20000 > Inc.
```

Anwendungen des Cut: Negation

- Prologs Negation: negation-as-failure
- Historisch: Prädikat not

  not(P)
  - oft auch in Operatorschreibweise: not P
- ISO Prolog: Operator \+
- Hilfsprädikat call: wertet einen Term als Prädikat aus

```
\+P :- call(P), !, fail.
\+P.
```

Negation und freie Variablen

- ledigerStudent(X) :- \+ verheiratet(X), student(X).
  student(peter).
  verheiratet(klaus).
- ?- ledigerStudent(peter). yes
- ?- ledigerStudent(X). no

Weitere Prolog-Funktionen

- Ein-Ausgabe
  - von Zeichenfolgen
  - von Prolog-Termen
- Termklassifikation (Variablen, Zahlen, Atome, Listen)
- Exceptions
- Definition kontextfreier Grammatiken
- SWI-Features:
  - Online-Hilfe (help, apropos)
  - Systembibliotheken (einschließlich GUI)
    - Foreign Language Interface
  - Debugger
  - Profiler
  - Constraint Handling Rules (CHR)
  - Koroutinen
  - Multi-Threading
  - ...