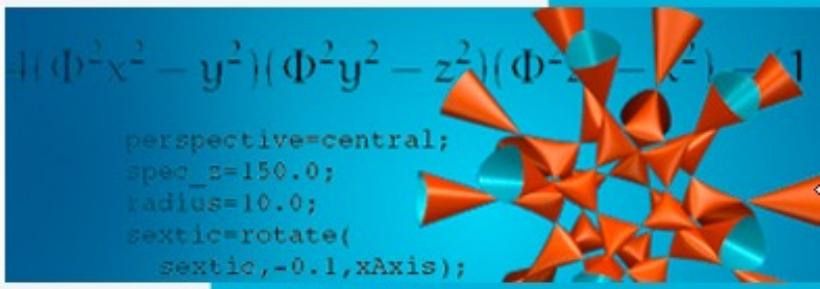
Philipps Universität Marburg

Deklarative Programmierung

Sommersemester 2022

Prof. Christoph Bockisch
(Programmiersprachen und –werkzeuge)

Axel Nowak, Felix Ludwig, Kevin Mankel, Mohamed Eldakar,
Sibelius Kruse, Stefan Störmer, Yevheniia Makara



[The art of Prolog: Einleitung – 1.7]

Logikprogrammierung

- Ausgangspunkt:
  - Wie will ich Probleme/Lösungen beschreiben?
  - Nicht: Welche Beschreibungen versteht die Maschine?
  - Ähnlich wir funktionale Programmierung

- Logik Programm
  - Wissen explizit ausdrücken: **logische Axiome**
  - Problem als logische Aussage beschreiben (“**Zielaussage**“)
  - Programm: Menge von Axiomen
  - Programmausführung: konstruktiver Beweis der Zielaussage über dem Programm

Logikprogrammierung

- Zielaussage typischerweise existential quantifiziert:
  “Es gibt eine Liste X, sodass das Sortieren der Liste [3, 1, 2] die Liste X ergibt.“

- Ergebnis des Programms
  - Folgt die Zielaussage aus den Annahmen? - Ja/Nein
  - Konstruktiver Beweis: Angabe von Werten für die Unbekannten für die die Aussage wahr wird.

- Beispiel: angenommen, ausreichende Axiome über “Sortieren“ wurden definiert:
  - X = [1, 2, 3]

Geschichte von Prolog

- Entwickelt 1970er (Kowalski, Colmerauer)
- PROrammieren in LOGik
- Beeinflusste viele Entwicklungen:
  - 5th Generation Project
  - Deductive Databases
  - Constraint Logic Programming
- Prolog-Implementierung:
  - Empfehlung SWI Prolog

SWI Prolog

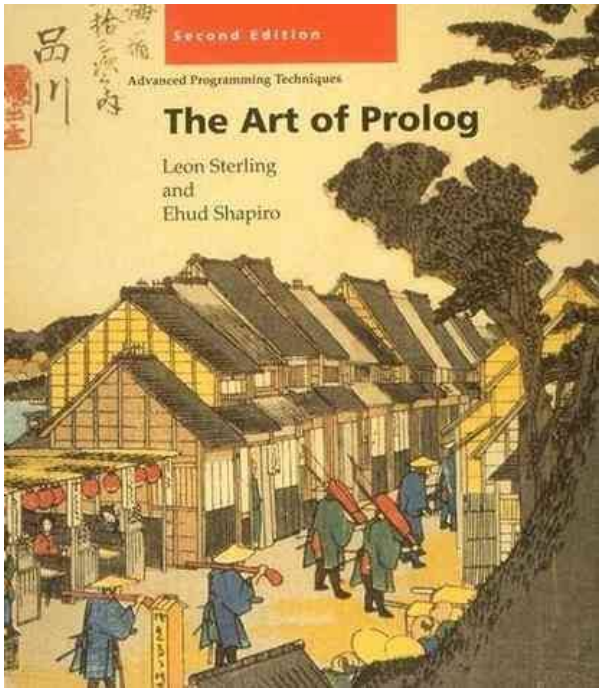
- Einfachste Möglichkeit (erfordert Internetzugang):
  - https://swish.swi-prolog.org
- Für Offline-Variante:
  - SWI Prolog Interpreter (Installieren Sie die 32-Bit Variante)
    - https://www.swi-prolog.org/download/stable
  - Editor, z.B. für Windows
    - https://arbeitsplattform.bildung.hessen.de/fach/informatik/swiprolog/indexe.html
    - Nach der Installation muss noch
      - der Pfad zu Prolog gesetzt werden (Menü „Fenster“ -> „Konfiguration“)
      - bei „Prolog Verzeichnis“ das Root-Verzeichnis von Prolog angeben
  - Andere Entwicklungsumgebungen: https://www.swi-prolog.org/IDE.html

TuProlog

- Keine Installation nötig
- Herunterladen von:
  http://apice.unibo.it/xwiki/bin/view/Tuprolog/Download
- Minimale Entwicklungsumgebung (ohne Unterstützung zur Fehlersuche)
- Achtung: Syntax weicht in einigen Fällen von SWI Prolog ab. Im Zweifel gilt die SWI-Prolog Syntax!

Literatur

- Leon Sterling, Ehud Shapiro. The Art of Prolog
- http://www.learnprolognow.org



Programmieren in Prolog

- Programmieren in Prolog bedeutet
  - Tatsachen (Fakten) über Objekte und deren Beziehungen zu deklarieren
  - Regeln über Objekte und deren Beziehungen zu definieren
  - Fragen zu den Objekten und Beziehungen zu stellen

Sokrates ist ein Mensch.

Alle Menschen sind sterblich.

Ist Sokrates sterblich?

- Nur **eine Datenstruktur: logischer Term**

Fakten

- Aussage: eine Relation (Prädikat) gilt zwischen Objekten
- Objekte werden als atomarer Bezeichner dargestellt
- Beispiel
  - father(abraham,isaac).

Die Relation “father“ gilt zwischen abraham und isaac

Relation

Atome

- Konvention: Relationen und Atome beginnen mit kleinem Buchstaben

Fakten

- (Arithmetische) Operation lassen sich als Fakten darstellen
  - Hier unvollständig

- Beispiel
  - plus(0,0,0). plus(0,1,1). plus(0,2,2). plus(0,3,3).
    plus(1,0,1). plus(1,1,2). plus(1,2,3). plus(1,3,4).

- Eine endliche Menge an Fakten definiert ein Programm

Fakten

- father(terach,abraham).
  father(terach,nachor).
  father(terach,haran).
  father(abraham,isaac).
  father(haran,lot).
  father(haran,milcah).
  father(haran,yiscah).
- mother(sarah,isaac).
- male(terach).
  male(abraham).
  male(nachor).
  male(haran).
  male(isaac).
  male(lot).
- female(sarah).
  female(milcah).
  female(yiscah).

Abfragen

- Abfragen von Information aus einem Logikprogramm
- Abfragen, ob eine Relation zwischen Objekten erfüllt ist
- Beispiel
  - ?- father(abraham, isaac).
  - *true*

Gilt die Relation father zwischen abraham und isaac?

Abfragen

- Abfragen und Fakten haben dieselbe Syntax
  - Aus dem Kontext ergibt sich, ob es eine Abfrage oder ein Fakt ist

- **Fakt**
  P.
  - Aussage, dass das Ziel P wahr ist

- **Abfrage**
  ?- P.
  - Frage, ob P wahr ist

- Eine einfache Abfrage besteht aus einem einzigen Ziel

Abfragen

- Beantworten einer Abfrage bezüglich eines Programms
  - Ist die Abfrage eine logische Konsequenz des Programms?
- Logische Konsequenz wird bestimmt durch Anwenden von **Deduktionsregeln**
- Vorgehen vereinfacht:
  - Suche einen Fakt, der die Abfrage impliziert
  - Wird so ein Fakt gefunden, ist die Abfrage erfüllt (Antwort Ja)
  - Die Antwort ist Nein, wenn kein solcher Fakt gefunden wird
- Eine erste Deduktionsregel - ***Identität***:
  - P impliziert P

Abfragen

- Antwort Nein bedeutet
  - Die Aussage konnte aus dem Programm heraus nicht bewiesen werden
- Antwort Nein bedeutet nicht,
  - Dass die Aussage tatsächlich falsch ist
- Beispiel
  - plus(0,0,0). plus(0,1,1). plus(0,2,2). plus(0,3,3).
  - plus(1,0,1). plus(1,1,2). plus(1,2,3). plus(1,3,4).
  - ?- plus(1, 4, 5).
    - Ergibt *false*, weil die Fakten über die Addition unvollständig sind

Logische Variablen

- **Variable**: unspezifiziertes Objekt
- **Abstraktion**, die für mehrere Abfragen steht
- Beispiel: Von wem ist abraham der Vater?
  - Alle Objekte durchgehen,
    und feststellen, bei welchen die Abfrage „Ja“ ergibt.
    - ?- father(abraham, lot).
    - ?- father(abraham, milcah).
    - ...
    - ?- father(abraham, isaac).
  - Gebrauch von logischen Variablen
    - ?- father (abraham, X).

Logische Variablen

- **Variable**: unspezifiziertes Objekt
- **Abstraktion**, die für mehrere Abfragen steht
- Beispiel: Von wem ist abraham der Vater?
  - Alle Objekte durchgehen,
    und feststellen, bei welchen die Abfrage „Ja“ ergibt.
    - ?- father(abraham, lot).
    - ?- father(abraham, milcah).
    - ...
    - ?- father(abraham, isaac).
  - Gebrauch von logischen Variablen
    - ?- father (abraham, X).

Ist abraham Vater von irgendeinem Objekt?

Programmausführung: konstruktiver Beweis. Daher ist das Ergebnis: Ja mit X=isaac

Logische Terme

- Induktive Definition: Logischer Term
  - Konstanten und Variablen sind logische Terme
  - Zusammengesetzte Terme (“Strukturen“) sind Terme

- Zusammengesetzter Term
  - Funktor + Sequenz von mindestens einem Argument
  - Argumente sind Terme
  - Funktor ist charakterisiert durch:
    - Name
    - Arität (Anzahl der Argumente)

Fakten und Abfragen sind zusammengesetzte Terme

Zusammengesetzte Terme

- Terme, die keine Variablen enthalten werden "**Grundterm**" genannt
- Beispiele
  - s(0).
  - hot(milk).
  - name(john,doe).
  - list(a,list(b,nil)).
  - tree(tree(nil,3,nil),5,R).
  - foo(X).

Name=s, Arität=1, Funktor- s/1

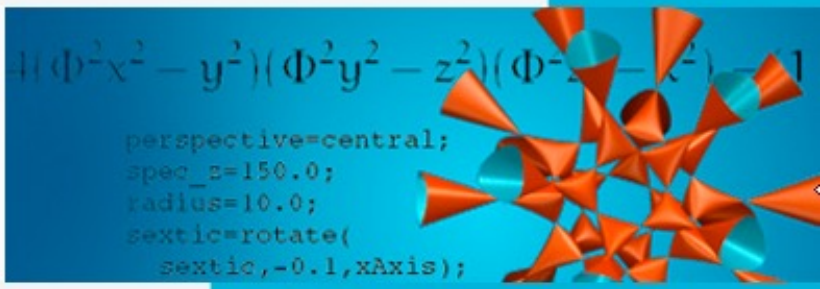
Grundterm

Strukturen sind rekursiv

Kein Grundterm (**Nicht-Grundterm**)

Philipps Universität Marburg

Eine Auswertungsstrategie für Prolog



Substitution

- **Definition**: Eine Substitution ist eine endliche (evtl. leere) Menge von Paaren:
  - $X_i = t_i$
  - X_i ist eine Variable
  - t_i ist ein Term
  - Alle X_i sind verschieden:
    - Für alle i und j mit $i \neq j$ gilt $X_i \neq X_j$
  - Keine Variable, die ersetzt wird, kommt in einem der Terme vor, durch die ersetzt wird:
    - Für alle i und j gilt X_i kommt nicht in t_j vor
- Wir schreiben für eine Substitution typischerweise: θ
- Anwenden der Substitution θ auf den Term A ergibt $A\theta$

Substitution

- Beispiel:
  - θ = {X = isaac}
  - A = father(abraham,X).
  - Dann ist $A\theta$ = father(abraham,isaac).

Instanz

- **Definition**: Ein Term *A* ist eine **Instanz eines Terms** *B*,
  - Wenn es eine Substitution *θ* gibt, sodass
  - *A* durch Substitution aus *B* hervorgeht: $A = B\theta$

- Beispiel
  - father(abraham,isaac).
    - Ist eine Instanz von father(abraham,X).
    - Mit der Substitution {X = isaac}
  - mother (sarah, isaac).
    - Ist eine Instanz von mother(X, Y).
    - Mit der Substitution {X = sarah, Y = isaac}

Existenz-Abfragen

- Variablen in Abfragen sind "**existentiell quantifiziert**":
  - Die Abfrage bedeutet: existieren Terme, sodass die Abfrage erfüllt ist, wenn jede Variable durch einen dieser Terme ersetzt wird?

- Allgemein
  - Abfrage ?- $p(T_1, T_2, \ldots, T_n)$. mit den Variablen $X_1, X_2, \ldots, X_k$
    - Gibt es $X_1, X_2, \ldots, X_k$ sodass $p(T_1, T_2, \ldots, T_n)$.?

- Beispiel:
  - ?- father(abraham, X).
    - "Existiert ein X, sodass abraham der Vater von X ist?"

Existenz-Abfragen: Generalisierung

- **Deduktionsregel**: ***Generalisierung***
  - Eine Existenz-Abfrage ist die logische Konsequenz ihrer Instanzen
  - Gegeben eine Abfrage *P*
  - *P* ist erfüllt, wenn es eine Instanz von *P* mit beliebiger Substitution *θ* gibt
  - Und *Pθ* erfüllt ist

- Beispiel
  - Der Fakt father(abraham,isaac).
  - Impliziert, dass es ein X gibt,
  - Sodass ?- father(abraham, X). erfüllt ist, nämlich für *θ*={X = isaac}

Existenz-Abfragen: Generalisierung

- Bedeutung eines **nicht-Grundterms** (Generalisierung)
  - Gegeben eine Abfrage
    von Fakten
  - Suche einen Fakt, der
    - Existiert so ein Fakt, d
    - Repräsentation der Lö
    - Gibt es keinen entspre

Nicht-Grundterm: ein Term, der mindestens eine Variable enthält.

- Im Allgemeinen hat eine Existenz-Abfrage mehrere Lösungen
- Beispiel
  - ?- father(haran,X). hat die Lösungen {X = lot}, {X = milcah}, {X = yiscah}
  - ?- plus (X, Y, 4). hat die Lösungen {X = 0, Y = 4}, {X = 1,Y = 3}, ...

Existenz-Abfragen: Generalisierung

- Bedeutung eines **nicht-Grundterms** (**Generalisierung**)
  - Gegeben eine Abfrage mit logischen Variablen und ein Programm von Fakten
  - Suche einen Fakt, der eine Instanz von der Abfrage ist
    - Existiert so ein Fakt, dann ist diese Instanz die Lösung
    - Repräsentation der Lösung durch Substitution die zu der Instanz führt
    - Gibt es keinen entsprechenden Fakt, ist das Ergebnis Nein

- Im Allgemeinen hat eine Existenz-Abfrage mehrere Lösungen
- Beispiel
  - ?- father(haran,X). hat die Lösungen {X = lot}, {X = milcah}, {X = yiscah}
  - ?- plus (X, Y, 4). hat die Lösungen {X = 0, Y = 4}, {X = 1,Y = 3}, ...

Universelle Fakten

- Variablen können auch in Fakten vorkommen
  - Auch hier: Abstraktion durch Wiederverwendung von Gemeinsamkeiten
  - Variablen fassen mehrere Fakten zusammen

- Beispiel:
  - Anstatt
    - likes(abraham,pomegranates).
    - likes(sarah,pomegranates).
  - Universeller Fakt
    - likes(X, pomegranates).

- Zusammenfassung von Regeln
- Jede Zahl mit 0 multipliziert ergibt 0: times(0, X, 0).

Universelle Fakten

- Variablen in Fakten sind "**universell quantifiziert**"
- Allgemein
  - Ein Fakt $p(T_1, ..., T_n)$. mit den Variablen $X_1, \ldots, X_k$ bedeutet, dass der p für alle Werte der Variablen $X_1, \ldots, X_k$ gilt.
- Beispiel
  - likes(X, pomegranates).
    - "Für alle X gilt: X mag Granatäpfel."

Einschränkungen über Variablen

- Variablennamen können mehrfach verwendet werden
- Vorkommen von Variablen müssen **konsistent** durch dieselben Objekte ersetzt werden
- Beispiel
  - Abfragen
    - ?- plus(X, X, 4).
      - Existiert ein X, sodass X + X gleich 4? – Ja mit {X = 2}
  - Fakten
    - plus(0,X,X).
      - Für alle Werte von X ist 0 + X gleich X

Universelle Fakten

- Bedeutung eines **Grundterms** (**Instantiierung**)
  - Gegeben eine Grund-Abfrage u
    quantifizierten Fakten
  - Suche einen Fakt von dem die A

Grundterm: ein Term, der keine Variable enthält.

Grund-Abfrage: eine Abfrage, die keine Variable enthält.

- Beispiel
  - ?- plus(0, 2, 2).
    - Ja, denn es ist eine Instanz des universellen Fakts plus(0,X,X).

Universelle Fakten

- Bedeutung eines **Grundterms** (**Instantiierung**)
  - Gegeben eine Grund-Abfrage und ein Programm von universell quantifizierten Fakten
  - Suche einen Fakt von dem die Abfrage eine Instanz ist

- Beispiel
  - ?- plus(0, 2, 2).
    - Ja, denn es ist eine Instanz des universellen Fakts plus(0,X,X).