LOGIKPROGRAMMIERUNG

Abgabevoraussetzung

In Prolog geschriebene Programme können als logische Formeln gelesen werden.
→ Logische Formeln können als deutsche Sätze gelesen werden.

Beim Schreiben von Programmen werden implizite Annahmen explizit formuliert.
→ Prolog hilft Dinge genauer zu analysieren und zu formulieren

Deklarative Programmiersprache:
- Beschreibung des Resultats
- Keine Beschreibung der Ausführung / Rechenschritte

- Prolog hat aus Effizienzgründen einen einfachen / unvollständigen Beweismechanismus
→ Auf diesen muss beim Schreiben von Programmen Rücksicht genommen werden (Prozedurale Semantik)

Grundelemente: Fakten, Regel, Anfrage / Zusicherung

Konstante / Atom: Beginnt mit Kleinbuchstaben bzw. in Apostroph ('Domänenelemente')

Funktion / Struktur: können Argumente gruppieren (Funktionssymbole)

→ Atome und Strukturen ergeben Terme

Relationen / Prädikate: Beziehung zwischen Termen, haben Stelligkeit (Prädikatensymbole)

Sprechende Prädikatsnamen: Beschreibung der Argumentlisten
Verkürzung der Namen
Beziehung zwischen Argumenten
Abkürzungen

Fakten: Elemente einer Relation
Grundfakten enthalten keine Variablen
Datenbasis: Programm welches nur aus Grundfakten besteht

Regeln: Beschreibung von Elementen einer Relation
Zusammenfassung von zusammengehörigen Anfragen

Anfragen: Überprüfung, ob das Ziel erfüllbar ist
Auflistung möglicher Variablenbindungen

Anfragen ... Ziel wird auf Ableitbarkeit geprüft
→ Zusicherung

Gründe für Scheitern einer Anfrage:
- Programm beschreibt Wirklichkeit unvollständig
- Gegenteil der Anfrage gilt (negation as finite failure)

→ negative Zusicherung prüft, ob Ziel mit vorhandenem Wissen nicht ableitbar ist; inkonsistente Daten vermeiden

Anfragen mit Variablen: existenziell quantifiziert
Antworten sind Variablenbindungen

Zusammengesetzte Anfragen: Mehrere Ziele konjugiert
Verbindung durch gemeinsame Variablen

Regeln ... definieren von konkreten Dingen unabhängige Beziehungen

Aufbau: Regelkopf ← Regelkörper

→ Regel ist ableitbar, wenn alle Ziele aus Regelkörper ableitbar sind

Lösungsmenge vs. Lösungssequenz:
redundante Lösungen in Lösungssequenz durch interne Variablen oder verschiedene Regeln eines Prädikats mit gleichen Lösungen

Lesarten

informelles Lesen: lesen eines Programms als deutschen Satz möglich da Logik als deutsche Sätze gelesen werden kann

→ bei komplexen Prädikaten sehr unverständlich da Schwierigkeit auf die selben Objekte / Elemente verweisen

Deklarative Lesart: Was beschreibt das Programm (Klausel),
Analyse eines Prädikats: jede Regel eines Prädikats beschreibt einen Teil der Lösungsmenge

→ Einzelne Regeln können durch Ziel falsch „ausgeschaltet" werden
→ Eine Regel kann die Fehler einer Anderen ausmachen

↳ Spezialisierung durch Ausschalten von Regeln

Analyse einer Regel: jedes Ziel einer Regel reduziert die Lösungsmenge des Prädikats ein

-> Ziele können durch * ausgeschaltet werden
-> Wenn bereits Verallgemeinerung einer Regel gelöst kann Regel nicht richtig sein

↳ Verallgemeinerung durch Ausschalten von Zielen

Schlussfolgerndes Lesen: Zudecken des Kopfs einer Regel
-> „Erraten," was beschrieben wird

Fehlersuche:
-> Prädikat für Anfrage erfüllt welche scheitern sollte: zumindest eine Regel / Klausel zu allgemein

-> Prädikat scheitert für Anfrage die erfüllt sein sollte: zumindest ein Ziel zu speziell

Prozedurale Lesart: Wie wird Programm ausgeführt

Inferenz: Ziel wird durch Regelkörper einer Klausel des Prädikats erreicht.

-> Wird für am weitesten links stehendes Ziel durchgeführt
-> Ziele einzeln betrachten / aufdecken
↳ Variablenbindungen analysieren

Termination

-> PROLOG ist Turing-vollständig, kann deshalb nicht in jedem Fall terminieren
↳ Außerdem wird ein effizienter, aber unvollständiger, Beweisalgorithmus verwendet

-> Endlosschleifen werden durch (in) direkte rekursive Regeln verursacht (Programme die nur direkte Rekursion enthalten sind strukturiert)

-> Es ist möglich, dass Lösungen gefunden werden, das Ziel aber nicht terminiert

- Terminationsüberlegungen durch prozedurale Lesart:
↳ Welche Variablenbindungen können Rekursion durch scheitern auffallen?
↳ Welche Variablen werden „durchgereicht"?

- Ursache für Nicht-Termination
↳ Unendlich große Lösungsmenge
↳ Probleme durch Prologs Beweismechanismus

- Terminationsvermutung: Zuweisungsmaß
:- Ziel. Es werden Lösungen gefunden
:- Ziel, fehler. Ziel terminiert

Effizienzüberlegungen

- Größe der Lösungsmenge
  Abhängig von Argumentbelegung (bekannt vs Variable)

- Anzahl der Inferenzen
  Ähnliche Vorgehensweise wie Terminationsüberlegungen
  Vernachlässigen von nicht-rekursiven Regeln
  Vernachlässigen von „durchgereichten" Variablen

- Termgröße
  Ausführzeit proportional zur Größe verwendeter Terme (eff)

Schreiben von Programmen

- Bedeutende Bezeichnungen
- Typen bestimmen: is_-Prädikate
- Aufspalten von Relationen
- Prädikatsnamen bestimmen
- Zusicherungen anschreiben (Fehlermeldung mit !=)
- Prädikat definieren

→ Negative Definitionen vermeiden
  - zu weit gefasst
  - höherer Wartungsaufwand
  - schlechter erweiterbar

Allgemeine Terme

Strukturen ... gruppieren zusammengehörige Argumente
  - Vereinfacht Verwendung von Prädikaten
  - Bessere Wiederverwendbarkeit / Erweiterbarkeit
  → Strukturen sind Name eigener Terme
  → Strukturen nur in Argumenten möglich / sichtbar.

Unifikation ... übernimmt die Aufgabe, Terme zu vergleichen, zu erzeugen und zu selektieren
  → Durch „Gleichsetzung" der Terme
  → Entspricht oft (nicht immer) „matching"

Vorgehensweise: Gleichzusetzende Terme werden in mehrere Gleichungen zerlegt
  Gleichungen werden schrittweise aufgelöst

Occurs-Check: Unifikation von $X = f(X)$ müsste eigentlich scheitern
  → dieser Occurs Check wird aus Effizienzgründen aber nicht durchgeführt
    ↳ daraus ergeben sich „infinite Terme"

Ungleichheit: möglich durch vordefiniertes Prädikat dif/2

Termarithmetik

→ Darstellung der natürlichen Zahlen mit Strukturen
↳ Nachfolger als Struktur
↳ Terme können beliebige Nachfolger von 0 darstellen

Listen ... spezielle Strukturen mit äußerlich unterschiedlicher Syntax

- `!.!(a,[])` = `[a]` = `[a|[]]`

  (Listenkonstruktor (Struktur))

- Unvollständige Liste: `[a,b,c|Xs]`

  (Listenkopf: `[a,b,c]`, Listenrest: `Xs`)

→ zur Darstellung von Sequenzen von Elementen verwendet

→ Texte / Strings sind Listen von Zeichen
↳ Liste von Atomen der Länge 1
↳ `"abcdef"` = `[a,b,c,d,e,f]`