Lernzettel

Logische Programmierung und Prolog

Universität: Technische Universität Berlin
Kurs/Modul: Softwaretechnik und Programmierparadigmen
Erstellungsdatum: September 19, 2025

Zielorientierte Lerninhalte, kostenlos!
Entdecke zugeschnittene Materialien für deine Kurse:

https://study.AllWeCanLearn.com

Softwaretechnik und Programmierparadigmen

Lernzettel: Logische Programmierung und Prolog

(1) Grundideen der logischen Programmierung. In der logischen Programmierung modelliert man Probleme durch Fakten und Regeln. Eine Abfrage (Query) sucht nach Belegungen der Variablen, die durch die Regeln sinnvoll ableitbar sind. Der Lösungsweg basiert auf univikation und Backtracking, um alle mögliche Belegungen zu explorieren.

(2) Prolog-Programmiersprache im Überblick. Ein Prolog-Programm besteht aus drei Bausteinen: Facts (Fakten), Rules (Regeln) und Queries (Abfragen). Die Ableitung erfolgt schrittweise durch Rückwärtsverkettung (Backward Chaining) und Unifikation von Termen.

(3) Struktur von Facts und Rules. Facts:

```
parent(john, mary).
parent(mary, sue).
```

Regeln:

```
grandparent(X,Y) :- parent(X,Z), parent(Z,Y).
```

Abfrage:

```
?- grandparent(john, Who).
```

(4) Unifikation. Unifikation entspricht dem mechanischen Abgleichen von Termen durch Variablenbindung. Sie erzeugt eine Substitution, die Terme äquivalent macht.

$$X = a$$
$$Y = f(X)$$

(5) Terme, Listen und Strukturen. In Prolog werden Listen als $[Head|Tail]$ dargestellt, wobei Head das erste Element und Tail die Restliste ist. Beispiele:

$[head|tail]$ mit *head* als erstem Element, tail als Rest

Fakten über Listen können so aussehen:

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

(6) Rekursion und Beweise mit Prolog. Viele Probleme lassen sich rekursiv in Prolog lösen, z. B. das Aneinanderreihen oder das Finden von Pfaden.

```
% Beispiel: append/3
append([], L, L).
append([H|T], L2, [H|R]) :- append(T, L2, R).
```

(7) Typische Beispiele und Abfragen. Beispielwissen:

```
ancestor(X, Y) :- parent(X, Y).
ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).

parent(john, mary).
parent(mary, sue).
```

Beispiele für Abfragen:

```
?- ancestor(john, Y).
?- ancestor(X, sue).
```

(8) Negation, Schnitt und Kontrolle. - Negation als Failure: P bedeutet, dass P nicht ableitbar ist. - Schnitt-Operator: ! dient dazu, Alternativen abzubrechen und die Suche zu determinieren.

```
not_parent(X) :- \+ parent(X, _).
```

(9) Praktische Hinweise und Best Practices. - Schreibe klare Facts und modulare Regeln, vermeide unnötige Backtracking-Punkte. - Nutze Denotationen wie $[Head|Tail]$ sinnvoll, um Listen effizient zu verarbeiten. - Prüfe Kantenfälle (leere Listen, unbekannte Variablen) robust ab. - Dokumentiere Sinn und Zweck von Regeln, damit spätere Änderungsarbeiten leichter fallen.

(10) Verifikation und Qualitätssicherung (Hinweis zur Vorlesung). Beispiele und Tests helfen, die richtige Logik zu überprüfen. In der Vorlesung werden eventuell Konzepte wie formale Korrektheit (vgl. Hoare-Kalkül) als Bezüge herangezogen, um Eigenschaften von Programmen zu diskutieren. Praktisch bedeutet das: schreibe kleine Testschnipsel, prüfe Abfragen gegen erwartete Ergebnisse, sanftes Refactoring von Regeln.

(11) Kleine Aufgaben zum Üben. - Definiere Facts für eine Familienrelation (parent, mother, father) und eine Regel für siblings. - Implementiere eine Regel für grandparents mit einer passenden Abfrage. - Schreibe eine Regel, die prüft, ob zwei Listen dieselben Elemente in derselben Reihenfolge enthalten.