Anwendungen der Künstlichen Intelligenz

Sebastian Iwanowski
FH Wedel

Kap. 2:
KI-Logik (Grundlagen und Grenzen)

Literatur zu Prolog

Lehrbücher:

Ivan Bratko: *PROLOG, Programming for Artificial Intelligence*,
2nd Edition, Pearson 1990, ISBN 0-201-41606-9
3rd Edition, Pearson 2001, ISBN 0-201-40375-6
4th Edition, Pearson 2011, ISBN 0-321-41746-6
Companion website mit Prolog code: www.pearsoned.co.uk/bratko

Peter Bothner / Wolf-Michael Kähler: Programmieren in *PROLOG,*
Eine umfassende praxisgerechte Einführung,
Vieweg 1991, ISBN 3-528-05158-2

P. Blackburn, J. Bos, K. Striegnitz: *Learn Prolog Now!*,
Texts in Computing Vol. 7, King's College Publications. 2006, ISBN 1-904987-17-6.
Companion website mit On-line-Version: www.learnprolognow.org

Seminararbeit:

Max Rohde: *Eignung logischer Programmiersprachen für Spiele-KI am Beispiel Prolog,*
FH Wedel, Iwanowski, SS 2007, Informatik-Seminar zur Spiele-KI
↳ gibt auch einen Überblick über Prolog und enthält weiterführende Literaturliste

Bausteine von PROLOG

Elementarbausteine:

- **Zahlen**

  Unterschieden wird zwischen ganzen und gebrochen rationalen Zahlen.

- **Atome**

  Name, dessen erstes Zeichen Kleinbuchstabe ist.

- **Variable**

  Name, dessen erstes Zeichen Großbuchstabe ist. Ausnahme: _

- **Listen**

  [] oder [Term | Liste]
  Kurzschreibweise: [1,2,3,4] für [1 | [2 | [3 | [4 | []]]]]

- **Terme**

  Zahlen, Atome, Variable, Listen oder Atom(Term) oder Atom(Term,Term) oder ...

- **Prädikate**

  Terme der Form Atom, Atom(Term) oder Atom(Term,Term) oder ...
  2 Prädikate gelten als gleich, wenn sie mit demselben Atom benannt sind und dieselbe Anzahl von Parametern haben.

Bausteine von PROLOG

Logische Operatoren zwischen Prädikaten:

- **Konjunktion**

  a , b entspricht: a ∧ b

- **Implikation**

  a :- b entspricht: b → a

- **Äquivalenz**

  a = b entspricht: b ↔ a

- **Nichtäquivalenz**

  a \= b entspricht: b ↮ a

- **versionsspezifische Operatoren**

  not, member, length, ...

Bausteine von PROLOG

Arithmetische Operatoren

- **+, -, *, /, div, mod**

  Arithmetische Ausdrücke werden in Infix-Notation gebildet.

Auswertung arithmetischer Operationen

- **nicht automatisch!**
- **durch Zuweisung an Variable**

  `Varname` **`is`** `Arithmetischer Ausdruck`
  weist der Variable Varname das Ergebnis des arithmetischen Ausdrucks zu.

- **durch logische Operatoren mit Auswertungsfunktion**

  <, =<, > >=. =:=, =\= wertet arithmetische Ausdrücke auf beiden Seiten aus.
  (in manchen Implementierungen nur auf einer Seite)

Bausteine von PROLOG

Wissen in Form von Klauseln

- **Fakten**

  `Prädikat.`
  Derartige Prädikate werden in der Wissensbasis als wahr vorausgesetzt.

- **Regeln**

  `Prädikat :- Konjunktion von Prädikaten.`
  Derartige Terme werden in der Wissensbasis als wahr vorausgesetzt, wenn die rechte Seite als wahr vorausgesetzt werden muss.
  Es kann für dasselbe Prädikat als Konklusion mehrere Regeln geben.

- **Fragen**

  `?- Konjunktion von Prädikaten.`
  Prolog versucht, eine Frage aus den bekannten Fakten und Regeln herzuleiten. Falls das gelingt, kommt als Antwort `yes` mit der dafür notwendigen Unifikation für die Variablen, anderenfalls `no`.

Funktionsweise eines PROLOG-Interpreters

PROLOG ist wissensbasiert:

- **Wissensbasis**

  Fakten und Regeln, dynamisch erweiterbar

- **Inferenzmaschine**

  Automatische Herleitung neuer Fakten und Regeln mit
  Resolution und Unifikation

- **Dialogkomponente**

  Eingabe: Fragen
  Ausgabe: yes / no, Angabe der Unifikation im Erfolgsfall, Write als „Seiteneffekt“

  Yes: Das Prädikat der Frage folgt aus der Wissensbasis.
  No: Das Prädikat der Frage folgt nicht aus der Wissensbasis.
  No impliziert nicht, dass das Prädikat als falsch abgeleitet werden kann.

Funktionsweise eines PROLOG-Interpreters

Arbeitsweise der Inferenzmaschine:

- **Zerlegung eines Ziels in Unterziele**

  Erstes Ziel ist die Frage.
  Versuch, das Ziel durch Unifikation mit Prädikaten aus der Wissensbasis zu erreichen.
  Auswertung von Regeln führt zu Unterzielen.

- **Auswertungsreihenfolge**

  Alle Daten der Wissensbasis werden **von oben nach unten** ausgewertet.
  Konjunktionen in Regelvoraussetzungen werden **von links nach rechts** ausgewertet.
  Die Auswertungsreihenfolge wird *nicht* getrennt nach Fakten und Regeln vorgenommen.

- **Instanziierung von Variablen**

  Variablen werden nur zum Zweck der Unifikation mit Werten instanziiert.
  Die Instanziierung wird nach Misserfolg des gegenwärtigen Suchzweigs wieder aufgehoben.

- **Backtracking**

  Nach dem Misserfolg einer gegenwärtigen Instanziierung wird eine neue Instanziierung versucht.
  Tiefes Backtracking: Anderer Wert zur Erfüllung derselben Klausel.
  Seichtes Backtracking: Anderer Wert zur Erfüllung einer anderen Klausel für dasselbe Prädikat.

Einfache Beispiele

- **Prädikatenwelt aus Erstsemestervorlesung:**

Deklarative Lösung ohne die Probleme mit symmetrischen Prädikaten: XSB
http://xsb.sourceforge.net/

Wissensbasis:

```
vater(sven, georg).
bruder(holger, anna).
verheiratet(sven, anna).

maennlich(X) :- vater(X,Y).
maennlich(X) :- bruder(X,Y).

onkel(X,Y) :- vater(Z,Y), bruder(X,Z).
onkel(X,Y) :- mutter(Z,Y), bruder(X,Z).
mutter(X,Y) :- vater(Z,Y), verheiratet(X,Z).
weiblich(X) :- verheiratet (X,Z), maennlich(Z).
verheiratet (X,Y) :- verheiratet (Y,X).
```

In ISO-Prolog funktioniert das nicht!

Besser:

```
verh(X,Y) :- verheiratet(X,Y).
verh(X,Y) :- verheiratet(Y,X).
```

Fragen:

```
?-weiblich(X).
?-maennlich(X).
?-onkel(holger,X).
?-verheiratet(X,Y).
```

```
?- verh(X,Y).
```

Logische Programmiersprachen

- **Input:**
  Spezifikation des Problems mit logischer Beschreibungssprache

- **Output:**
  Antwort in logischer Beschreibungssprache

- **automatisch:**
  Generierung des Outputs aus Input

- **Zur Effizienzverbesserung:**
  Beeinflussung der Generierung des Outputs aus Input

Logische Programmiersprachen

Aufgabe für den Interpreter:

Eigentliches Ziel: Konstruktionsaufgabe ***erst recht nicht lösbar für allgemeine Formeln***

Gegeben eine Menge $\mathscr{F}$ von logischen Formeln. Bestimme alle Formeln F, die aus $\mathscr{F}$ folgen.

Hilfsziel: Verifikationsaufgabe ***nicht lösbar für allgemeine Formeln***

Gegeben eine Menge $\mathscr{F}$ von logischen Formeln und eine (neue) logische Formel F.
Bestimme, ob F aus $\mathscr{F}$ folgt.

Äquivalente Formulierungen zur Verifikationsaufgabe:

1) Gegeben eine Menge $\mathscr{F}$ von logischen Formeln und eine (neue) logische Formel F. Bestimme, ob die Formelmenge $\{\neg F\} \cup \mathscr{F}$ widersprüchlich ist.
2) Gegeben eine Menge $\mathscr{F}$ von logischen Formeln. Bestimme, ob sie erfüllbar ist.

entspricht Erfüllbarkeitsproblem: nicht lösbar für allgemeine Formeln

Lösung der Verifikationsaufgabe
entspricht Lösung der Erfüllbarkeitsaufgabe!

Wdh.: Aussagenlogische Formeln

- **Eine aussagenlogische Formel ist eine Verknüpfung von endlich vielen Literalen mit aussagenlogischen Operatoren.**
  - Die Literale sind als Variable aufzufassen, die genau einen von 2 Werten annehmen können
- **Eine Belegung einer Formel ist eine Zuweisung von Wahrheitswerten an die Literale derart, dass dieselben Literale immer denselben Wahrheitswert erhalten.**
- **Eine Formel heißt erfüllbar, wenn es eine Belegung gibt derart, dass die Formel wahr ist.**
  - Das Erfüllbarkeitsproblem ist in der Aussagenlogik immer lösbar, da man alle (endlich vielen) Belegungsmöglichkeiten der Variablen nur nacheinander auszuprobieren braucht.
  - Leider dauert das Lösen durch Ausprobieren sehr lange (exponentiell in der Anzahl der Variablen). Es ist bis heute kein effizienterer Algorithmus bekannt.

Das Problem ist NP-vollständig !

Wdh.: Prädikatenlogik

Die Prädikatenlogik (1. Stufe) erweitert die Aussagenlogik um folgende Elemente:

- **Prädikate**
  - Aussagen, die von Variablen abhängen
    (wenn es von k Variablen abhängt,
    dann heißt das Prädikat k-stellig)
- **Variable**
  - entsprechen den Literalen der Aussagenlogik,
    können aber beliebig viele Werte annehmen
- **Funktionen**
  - eindeutige Zuordnungen, die von Variablen abhängen
    (wenn sie von k Variablen abhängt,
    dann heißt die Funktion k-stellig)
  - 0-stellige Funktionen sind Konstante
- **Quantoren**
  - Existenzquantor (∃) und Allquantor (∀)
  - Quantoren werden nur auf Variablen angewendet (sonst nicht 1. Stufe)

Prädikatenlogische Formeln im Detail

Variablen, Prädikate, Funktionen

- **Variablen** sind Ausdrücke der Form x_i, i = 1,2,3,…,v
- **Prädikate** sind Ausdrücke der Form $P_i(t_1,\ldots,t_{k(i)})$, i = 1,2,3,…,p, wobei die $t_1,\ldots t_{k(i)}$ Terme sind
- **Funktionen** sind Ausdrücke der Form $f_i(t_1,\ldots,t_{k(i)})$, i = 1,2,3,…,f, wobei die $t_1,\ldots t_{k(i)}$ Terme sind

k(i)=0 ist auch erlaubt

Terme

- Jede Variable ist ein Term.
- Falls f_i eine Funktion ist und $t_1,\ldots t_{k(i)}$ Terme, dann ist $f_i(t_1,\ldots,t_{k(i)})$ ein Term.

Formeln

- Falls P_i ein Prädikat ist und $t_1,\ldots t_{k(i)}$ Terme, dann ist $P_i(t_1,\ldots,t_{k(i)})$ eine Formel.
- Für jede Formel F ist auch ¬F eine Formel.
- Für alle Formeln F, G sind auch (F∧G) und (F∨G) Formeln.
- Falls x eine Variable ist und F eine Formel, dann sind auch ∃F und ∀F Formeln.

Literatur: Schöning: Logik für Informatiker

Prädikatenlogische Formeln im Detail

Bsp.: Formel $\varphi = \forall x \, (R(\mathbf{y}, \mathbf{z}) \wedge \exists y \, (\neg P(\mathbf{y}, \mathbf{x}) \vee R(\mathbf{y}, \mathbf{z})))$

Grüne Vorkommen von *y* und *z* sind **frei**.
Rote Vorkommen von *x, y* und *z* sind **gebunden**.

Geschlossene Formeln:

Formeln, die keine freien Variablen enthalten.

Offene Formeln:

Formeln, die keine gebundenen Variablen enthalten.

Atomare Formeln:

Formeln, die nur aus einem Prädikat abhängig von Termen bestehen.

Wdh.: Prädikatenlogische Formeln

- **Eine Belegung einer Formel ist eine Zuweisung von *Werten aus festgelegten Definitionsbereichen an die freien Variablen* derart, dass dieselben Variablen immer denselben Wert erhalten.**

- **Eine Formel heißt erfüllbar, wenn es eine Belegung gibt derart, dass die Formel wahr ist.**

  - Das Erfüllbarkeitsproblem ist in der Prädikatenlogik nicht entscheidbar, d.h. kein Algorithmus kann jemals in der Lage sein, von jeder Formel zu entscheiden, ob sie erfüllbar ist oder nicht.

Das allgemeine Problem ist unlösbar !

Gibt es dennoch einen Ausweg ?

Ja, löse ein spezielleres Problem !

Die logische Programmiersprache PROLOG

PROLOG akzeptiert nicht beliebige Formeln:

- ***beliebige Variablenbereiche und Funktionen zugelassen***
- ***aber keine Quantoren***
- ***in KNF müssen alle Klauseln Hornklauseln sein:***

¬p ∨ ¬q ∨ . . . ∨ ¬r ∨ x	*Maximal ein Literal ist positiv.*

Regelschreibweise von Hornklauseln:

p ∧ q ∧ . . . ∧ r → x

Regeln (Hornklauseln)

In der Voraussetzung darf nur eine Konjunktion von positiven Literalen stehen.

Satz (Vollständigkeit der Beweisfindung auf Hornklauseln):

Für jede Menge von Hornklauseln und eine neue Hornklausel kann Prolog nach endlicher Zeit entscheiden, ob die neue Hornklausel aus der alten Menge folgt ***oder nicht***

☹ ***Anmerkung: „Endliche Zeit“ kann „sehr lange“ heißen !***