Logische Sprachen

Anwendung mit Listen: Expertensysteme

Wissen der Form:
„Wenn es eine Pflanze ist, und diese Pflanze einen Stamm hat und sehr gross ist, dann ist es ein Baum.“
„Wenn die Pflanze klein ist und bunt, dann ist es eine Blume.“

abbilden in Regeln (Fakten):
(Regelnr, Prämisse,Konklusion, [Liste von Bedingungen])

```
regel(1,pflanze,baum,[stamm,gross]).
regel(2,pflanze,blume,[klein,bunt]).
…
```

Logische Sprachen

Anwendung mit Listen: Expertensysteme

zu Bedingungen Anfragen formulieren
(Liste von Worten):

```
anfrage(stamm,[hat,die,pflanze,einen,stamm]).
anfrage(gross, [ist, die, pflanze, sehr, gross]).
anfrage(bunt, [ist, die, pflanze, bunt]).
…
frage(Bedingung):-anfrage(Bedingung,Text),
            schreibe_Liste(Text),write(‘?‘), nl,
            read(Antwort),….,
            Antwort==ja.
```

Logische Sprachen

Anwendung mit Listen: Expertensysteme

zu Prämisse passende Regel heraussuchen, alle Bedingungen abfragen:

```
erkenne(Praemisse):-regel(Nr,Praemisse,Konkl,Bed),
      ueberpruefe(Bed),
      merke(Nr),
      drucke_Erg(Konkl),
      erkenne(Konkl).

ueberpruefe([]).
ueberpruefe([B|WeitereBed]):-frage(B),
                                    ueberpruefe(WeitereBed).
```

Logische Sprachen

Anwendung mit Listen: Expertensysteme

```
drucke_Erg(Konkl):-write(´Die Pflanze ist ein/eine ´),
                   write(Konkl),nl.
```

Wissensbasis erweitern, um erfolgreiche Regel temporär zu merken:

```
merke(Nr):-assert(gefunden(Nr)).
```

gibt es nach Abarbeiten aller regel(..)-Klauseln Fakten gefunden(…), so gab es mind. eine Konklusion, jetzt aber keine weiteren →

```
erkenne(_):-gefunden(_),write(‘Mehr weiss ich nicht.‘),nl.
```

Logische Sprachen

Anwendung mit Listen: Expertensysteme

Kein gefunden(..)-Faktum: keine regel-Klausel passte
→ erkenne(_):-not(gefunden(_)),
write(‘Die Pflanze kenne ich nicht‘),nl.

(In SWI-Prolog: mit Trick, s. wissen2.pl)

Wichtig: mit assert ergänzte Fakten nach Klassifikation wieder löschen!