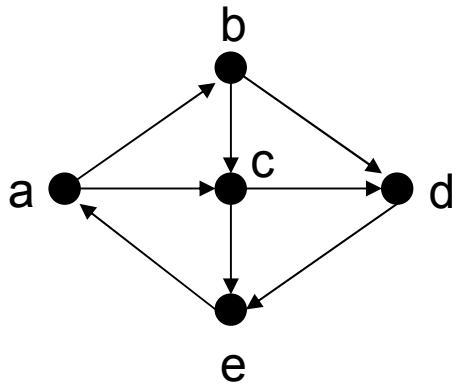
Logische Sprachen

Anwendung mit Listen:

Städtegraph:

```
kante(a,b).
kante(a,c).
kante(b,c).
kante(b,d).
kante(c,d).
kante(c,e).
kante(d,e).
kante(e,a).
weg(X,Y):-kante(X,Z),weg(Z,Y).
```



Weg von a nach d z.B. a – c – e – a – b – d

Logische Sprachen

Anwendung mit Listen:

Besser: mit Liste der bereits besuchten Orte

```
weg(X,X,Besucht).
weg(X,Y,Besucht):-
    kante(X,Z), not(member(Z,Besucht)),
    weg(Z,Y,[Z|Besucht]).
pfad(X,Y):-
    weg(X,Y,[X]).
```

?- pfad(a,d) liefert true
nicht den gelaufenen Weg

Logische Sprachen

Anwendung mit Listen:

Noch besser mit 2 Listen: bereits besuchten Orte
Pfad (Reihenfolge!)

```
weg(X,X,Besucht,[X|Besucht]).
weg(X,Y,T,Besucht):-
    kante(X,Z), not(member(Z,T)),
    weg(Z,Y,[X|T],Besucht).
```

```
pfad_mit_liste(X,Y,Pfad):-weg(X,Y,[],P),reverse(P,Pfad).
```

Logische Sprachen

Anwendung mit Listen:

```
?- pfad_mit_liste(a,d,Pfad).

Pfad = [a, b, c, d]  ;

Pfad = [a, b, d] ;

Pfad = [a, c, d];

fail.
```

Logische Sprachen

sinnvolle Hilfsprädikate auf Listen (neben vielen weiteren, existieren nur z.T.):

last(L,X)	X ist letztes Element von Liste L
praefix(P,L)	P ist Präfix von L
nachbar(El1,El2,L)	El1 ist Nachbar von El2 in L
ersetze(A,AL,B,BL)	ersetzt alle A durch B in AL, Ergebnis ist BL
teilliste(L1,L2)	L1 ist Teilliste von M2
menge(L,M)	M ist L ohne doppelte Elemente
teilmenge(T,M)	T ist Teilmenge von M
union(M1,M2,M)	M ist Vereinigungsmenge von M1, M2
intersect(M1,M2,S)	S ist Schnitt von Mengen M1, M2
difference(M1,M2,D)	D ist Differenzmenge von M1 bzgl. M2
gleich(M1,M2)	M1 und M2 enthalten die gleichen Elemente

Logische Sprachen

wichtiges Hilfsprädikat auf Listen:

permutation(L,PL)	PL ist Permutation von L
sortiert(L)	beweisbar, wenn Liste L sortiert ist
sortiere(L1,L2)	L2 ist sortierte L1-Liste