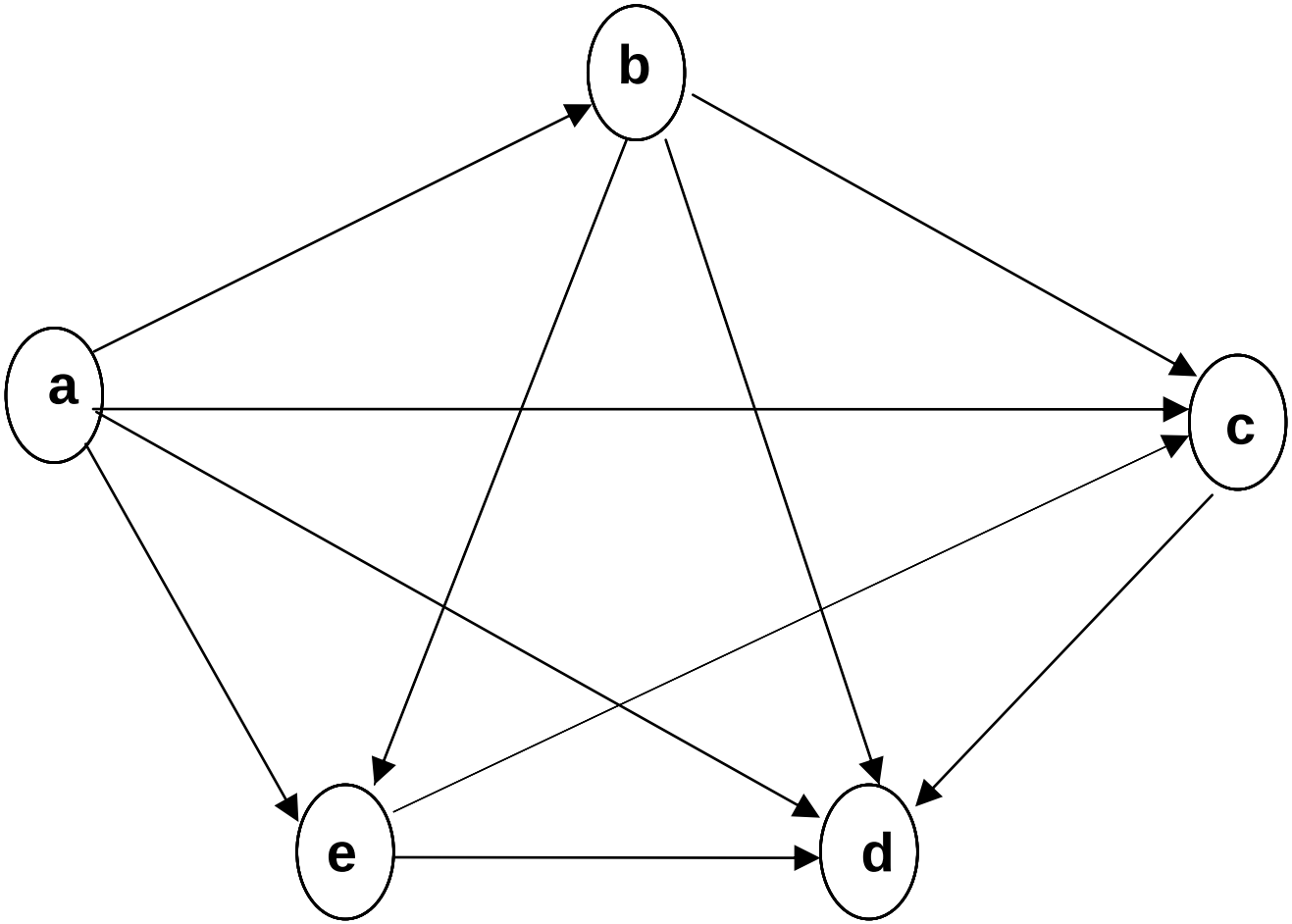
Suche nach Wegen



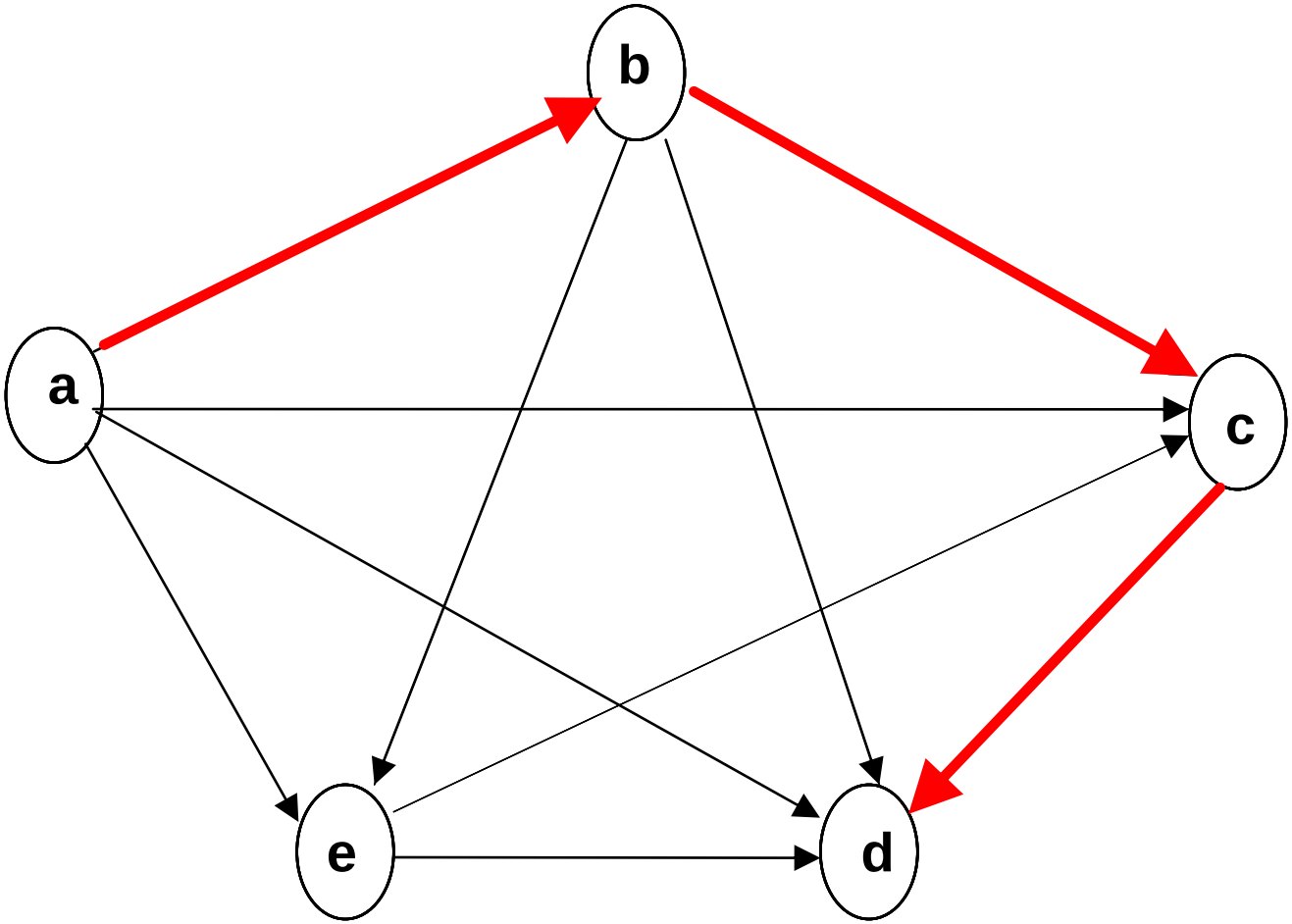
Beispielprogramm P & Query Q

```
kante(a,b).
kante(a,c).
kante(a,d).
kante(a,e).
kante(b,c).
kante(b,d).
kante(b,e).
kante(c,d).
kante(e,c).
kante(e,d).
```

```
wegstrecke(X,X,list(X,nil)).
wegstrecke(X,Y,list(X,Rests))
    :- kante(X,Irgend),
       wegstrecke(Irgend,Y,Rests).
```

```
?- wegstrecke(a,d,Zs).
```

Suche nach Wegen




```
Zs = [a,b,c,d];
```

nichtdet. Beispielrechnung

```
?- wegstrecke(a,d,Zs).                                          []
?- kante(a,Irgend), wegstrecke(Irgend,d,Z1s).
                                                  [Zs/list(a,Z1s)]
?- wegstrecke(b,d,Z1s).
                                                  [Zs/list(a,Z1s)]
?- kante(b,Irgend), wegstrecke(Irgend,d,Z2s).
                                          [Zs/list(a,list(b,Z2s))]
?- wegstrecke(c,d,Z2s).
                                          [Zs/list(a,list(b,Z2s))]
?- kante(c,Irgend), wegstrecke(Irgend,d,Z3s).
                                  [Zs/list(a,list(b,list(c,Z3s)))]
?- wegstrecke(d,d,Z3s).
                                  [Zs/list(a,list(b,list(c,Z3s)))]
```

Nehmen Fakt:

?- . [Zs/list(a,list(b,list(c,list(d,nil))))]

Bei Anwendung der Regel ginge es wie folgt weiter:

```
?- kante(d,Irgend), wegstrecke(Irgend,d,Z4s).
                          [Zs/list(a,list(b,list(c,list(d,Z4s))))]
```

Fail

Beispielprogramm

Fakten:

```
vater(abraham,isaak).          männlich(abraham).
vater(haran,lot).              männlich(isaak).
                               männlich(haran).
vater(gott,X).                 männlich(lot).

mutter(sarah,isaak).           weiblich(sarah).
```

Regeln:

```
R1: sohn(X,Y) :- vater(Y,X), männlich(X).
R2: sohn(X,Y) :- mutter(Y,X), männlich(X).
R3: tochter(X,Y) :- vater(Y,X), weiblich(X).
```

Prolog-Berechnungsalgorithmus (nichtdet.)

Eingaben: **Ein Prolog-Programm P und eine Query Q**

Ausgabe: **Antwortsubstitution sub oder Fail** (nicht erfolgreich)

out = [ ]; **// Speicher für die Antwortsubstitution**

Führe folgende Schritte solange aus

1. Nehme das am weitesten links stehende Prädikat `Q1` in Query.
2. Suche nach einer Regel (kann auch Fakt sein, dann `r` leer) `l:-r.`, so dass die linke Seite `l & Q1` **unifizierbar sind mit mgu** `sub`.
3. Ersetze in Query `Q1` durch rechte Regelseite `r`.
4. Wende die Substitution `sub` auf die in 3. erhaltene Query an.
5. out = **outsub|Z** (Einschränkung auf Variablen in Query Q)

bis **a.) Query leer ist :** Output(out)

oder **b.) Schritt 2 nicht ausführbar ist:** Output(Fail)

Problemstellung

Müssen linkstes Prädikat in Query und eine linke Regelseite durch Substitution gleich bekommen, damit Regel anwendbar.

linke Regelseite: `vater(gott,X)`

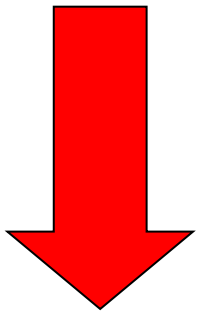
Prädikat in Query: `vater(Z,abraham)`

➡ **in beide Richtungen substituieren**

`sub=[Z/gott,X/abraham]`

➡

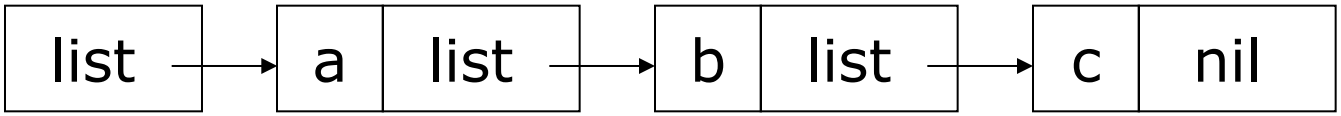
```
      vater(gott,X)sub
=     vater(gott,abraham)
=     vater(Z,abraham)sub
```



Unifikation

Wiederholung: Listenstruktur

C-Struktur:



Prolog-Term:

```
list(a,list(b,list(c,nil)))
```

Konvention für Listen-Variablen in Prolog-Programmen:

- End-s -> Liste
- ohne End-s -> Listenelement

```
list(X,Xs)
```

Unifikationsbeispiel

Listen **definiert durch:**

- **Konstante `nil`: leere Liste**
- **Funktor `list`: 1. Arg: Eintrag, 2. Arg.: Restliste**

Konkatenation (Verbindung) von Listen:

- **`app(nil,Xs,Xs).`**
- **`app(list(X,X1s),Ys,list(X,X2s))`**
  `:- app(X1s,Ys,X2s).` **P**

Query:

```
?- app(list(Z1,list(a,Z2s)),
       nil,
       list(Z3,list(Z4,nil))).
```
Q

Unifikations-Algorithmus

Eingaben: linke Regelseite P(t1,...,tn) und Query Q(s1,...sm)

Ausgaben: Nein oder (Ja, sub) // sub ist mgu von P(t1,...,tn) & Q(s1,...sm)

sub = [];

If (Prädikate verschieden oder Stelligkeit verschieden) **return (Nein);**

Lege (sm,tm),...(s1,t1) auf den Stack.

While (Stack isNotEmpty) {

Nehme oberstes Paar (s,t) vom Stack

// Fallunterscheidung über Aufbau von t & s:

1. t = X Variable

sub = sub[X/s];

Ersetze auf ganzem Stack X durch s.

kein Occur Check

2. t = a Konstante & s keine Variable

If (s != a) **return (Nein);**

3. t = f(t1,...,tk) zusammengesetzter Term & s keine Variable

If (s != f(s1,...,sk)) **return (Nein);**

Lege (sk,tk),...(s1,t1) auf den Stack

4. s = Z Variable & t keine Variable

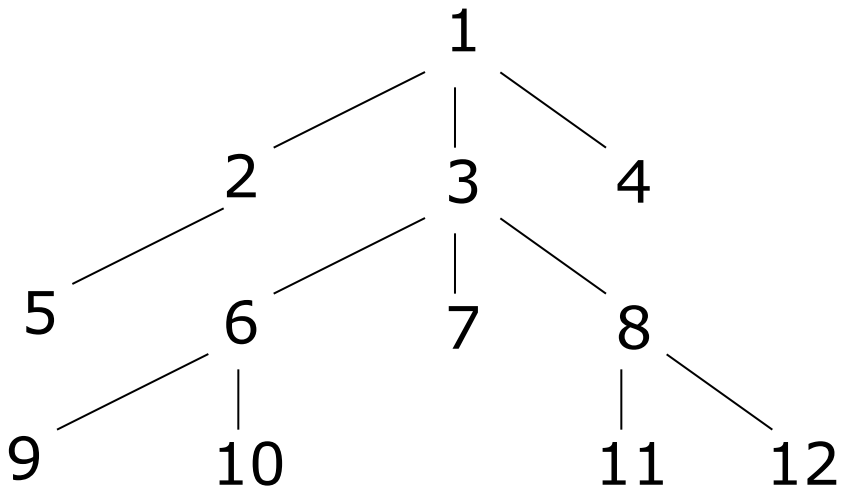
sub = sub[Z/t];

Ersetze auf ganzem Stack Z durch t. }

kein Occur Check

return (Ja, sub);

Baumdurchlauf



Breitensuche: **1 2 3 4 5 6 7 8 9 10 11 12**

Tiefensuche: **1 2 5 3 6 9 10 7 8 11 12 4**