Praktische Informatik 2

Sommer-Semester 2007

Prof. Dr. sc. Hans-Dieter Burkhard
www.ki.informatik.hu-berlin.de

Praktische Informatik 2

Themen

Alternative Programmiermethoden:

Deklaratives (logisches Programmieren): Prolog

Auf den Poolrechnerrn:
/usr/local/praktikum/SWIprolog/bin/pl

Abstrakte Datenstrukturen

(Listen, Graphen, Bäume ...)

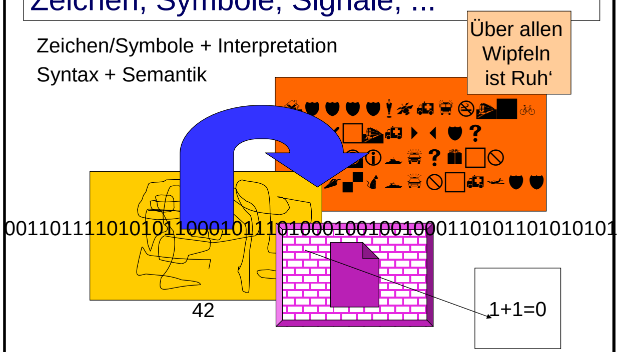
Repräsentation von „Wissen“

Lexikon
Datenbank
Briefmarkensammlung
Programme
Landkarten
Bilder
Gesetze
Regeln
. . .

Darstellung und Interpretation
- Menschlicher Nutzer
- Maschinelle Auswertung

Zeichen, Symbole, Signale, ...

Zeichen/Symbole + Interpretation
Syntax + Semantik

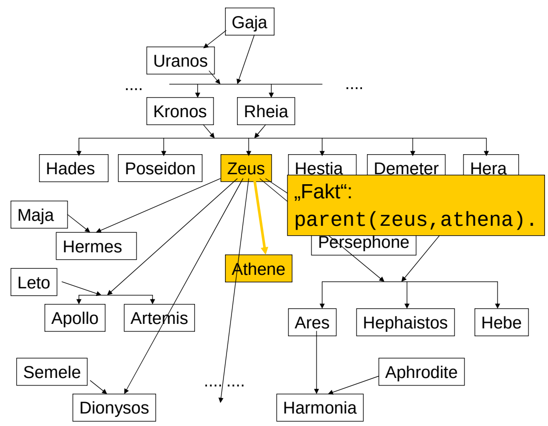


Explizites vs. Implizites Wissen

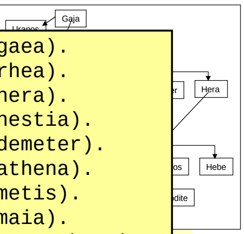
explizit: z.B. Axiome, Schluss-Regeln
Implizit: Folgerungen

Unterschiedliche Formen für
- Menschlicher Nutzer
- Maschinelle Auswertung

Inferenz: Verfahren zur Herleitung von implizitem Wissen aus explizitem Wissen (z.B. Beweisverfahren)



Griechische Götter: Fakten




```
parent(uranus, cronus).
parent(gaea,   ur
parent(gaea,   cr
pare
pare
pare
pare
pare
pare
...
parent(ze
parent(he
parent(zeus,    hephaestus
parent(hera,    hephaestus
...
```

```
male(uranus)
male(cronus)
male(zeus).
male(hades).
male(hermes)
male(apollo)
male(dionysi
male(hephaes
male(poseido
```

```
female(gaea).
female(rhea).
female(hera).
female(hestia).
female(demeter).
female(athena).
female(metis).
female(maia).
female(persephone).
female(aphrodite).
female(artemis).
female(leto).
```

```
parent(demeter, persephone).
```


Fakten in Prolog

Ein Fakt hat die Form `funktor(argumente).`

funktor ist der Name einer n-stelligen Relation (Prädikat).
n ist die Anzahl der Argumente.

```
male(zeus).
parent(zeus,athena).
parent(zeus,hera,ares).
plus(3,4,7).
kleiner(6,9).
```

male/1
parent/2
parent/3
plus/3
true/0
fail/0

Definition einer Relation durch Aufzählung

Aufzählung: Die Beziehungen werden explizit aufgezählt (Faktenmenge, Datenbank)

```
parent(uranus, cronus).
parent(gaea,
parent(gaea,
parent(rhea,
parent(cronus,
parent(rhea,
parent(cronus,
parent(cronus,
parent(rhea,
parent(cronus,
parent(rhea,
parent(zeus,    hermes).
parent(maia,    hermes).
...
```

Hinzufügen:
`asserta(male(caesar)).`
`assertz(male(augustus)).`

Löschen:
`retract(male(zeus)).`

```
male(dionysius).
male(hephaestus).
male(poseidon).
```

```
female(gaea).
female(rhea).
e(persephone).
e(aphrodite).
e(artemis).
female(leto).
```


Anfragen

Eine Anfrage hat die Form `?-funktor(argumente).`

funktor ist der Name einer n-stelligen Relation (Prädikat).

```
?- male(zeus).
yes.
?- parent(zeus,athena).
yes.
?-parent(hera,athena).
no.
```

```
parent(uranus, cronus).
parent(gaea,   cronus).
parent(gaea,   rhea).
parent(rhea,   zeus).
parent(cronus, zeus).
parent(rhea,   hera).
parent(cronus, hera).
parent(cronus, hades).
parent(rhea,   hades).
parent(cronus, hestia
parent(rhea,   hestia
parent(zeus,   hermes
parent(maia,   hermes
```

```
female(gaea).
female(rhea).
female(hera).
female(hestia).
female(demeter).
female(athena).
female(metis).
female(maia).
female(persephone).
female(aphrodite).
```

```
male(uranus).
male(cronus).
male(zeus).
male(hades).
male(hermes).
male(apollo).
male(dionysius).
male(hephaestus).
male(poseidon).
```


CWA: Closed World Assumption (1a)

Bedeutung der Antwort „no"?

Oder:
Welche Antwort gibt Prolog
für nicht gespeicherte Fakten?
`?-vater(zeus,athena).`

„Optimistische" Variante: `yes.`
„Pessimistische" Variante: `no.`

CWA: Closed World Assumption (1a)

(Vorläufige) Bedeutung der Antwort „no":
Der Fakt ist in der Datenbank nicht enthalten.

```
?-parent(hera,athena).
no.
?-parent(zeus,hera,ares).
no.
?-kleiner(3,7).
no.
```

```
parent(uranus, cronus).
parent(gaea,   cronus).
parent(gaea,   rhea).
parent(rhea,   zeus).
parent(cronus, zeus).
parent(rhea,   hera).
parent(cronus, hera).
parent(cronus, hades).
parent(rhea,   hades).
parent(cronus, hestia
parent(rhea,   hestia
parent(zeus,   hermes
parent(maia,   hermes
```

```
female(gaea).
female(rhea).
female(hera).
female(hestia).
female(demeter).
female(athena).
female(metis).
female(maia).
female(persephone).
female(aphrodite).
```

```
male(uranus).
male(cronus).
male(zeus).
male(hades).
male(hermes).
male(apollo).
male(dionysius).
male(hephaestus).
male(poseidon).
```

Anfragen nach Existenz

Anfrage enthält Variable `?-funktor(argumente).`

Variablen-Bezeichner beginnen mit Großbuchstaben

```
?- parent(zeus,X).
X=hermes?
;
X=athena?
;
X=ares?
;
no.
```

```
?- parent(zeus,X).
X=hermes?
;
X=athena?
.
yes.
```

```
parent(maia,   hermes
...
male(hades).
male(hermes).
male(apollo).
male(dionysius).
```

„ ; “ erwartet weitere Antworten. „ .“ schließt Anfrage ab.

CWA: Closed World Assumption (1b)

Bedeutung der Antwort „no“ ?

(Vorläufige) Bedeutung der Antwort „no“:
Es gibt keinen passenden Fakt in der Datenbank.

```
?-father(X,Y).
no.
?-kleiner(M,N).
no.
?-parent(X,gaea).
no.
```

```
parent(rhea,   zeus).
parent(cronus, zeus).
parent(rhea,   hera).
parent(cronus, hera).
rent(cronus, hades).
rent(rhea,   hades).
rent(cronus, hestia
rent(rhea,   hestia
rent(zeus,   hermes
rent(maia,   hermes
.
```

```
female(demeter).
female(athena).
female(metis).
female(maia).
female(persephone).
female(aphrodite).
```

```
male(uranus).
male(cronus).
male(zeus).
male(hades).
male(hermes).
male(apollo).
male(dionysius).
male(hephaestus).
male(poseidon).
```


Relationen definieren durch Regeln

Eine Regel hat die Form

`kopf:-körper.`

`goal:-subgoals.`

Mit der intuitiven Bedeutung:

„goal“ gilt (ist beweisbar),
falls alle „subgoals“ gelten (beweisbar sind).

Relationen definieren durch Regeln

Definition der Relation „Vater-Kind“:

```
father(Vater,Kind)
      :-parent(Vater,Kind), male(Vater).
```

```
father(X,Y):-parent(X,Y),male(X).
mother(X,Y):-parent(X,Y),female(X).
```

```
parent(X,Y,Z) :-father(X,Z),mother(Y,Z).
```

```
son(Sohn,Elternteil):-
parent(Elternteil,Sohn),male(Sohn).
```

```
grandfather(X,Z):-father(X,Y),parent(Y,Z).
grandmother(X,Z):-mother(X,Y),parent(Y,Z).
```

```
grandchild(X,Y):-grandfather(Y,X).
grandchild(X,Y):-grandmother(Y,Z).
```


Regel als logische Formel

Die Variablen in Regeln sind universell quantifiziert.
Die Relationen der rechten Seite sind konjunktiv verknüpft.

goal($X_1,\ldots,X_n$) :- subgoal$_1$($X_1,\ldots,X_n$) ,..., subgoal$_m$($X_1,\ldots,X_n$).

Wird aufgefasst als

$\forall X_1 \ldots \forall X_n$
[subgoal$_1$($X_1,\ldots,X_n$) $\wedge \ldots \wedge$ subgoal$_m$($X_1,\ldots,X_n$) $\rightarrow$ goal($X_1,\ldots,X_n$)]

oder

$\forall X_1 \ldots \forall X_n$
[$\neg$subgoal$_1$($X_1,\ldots,X_n$) $\vee \ldots \vee$ $\neg$subgoal$_m$($X_1,\ldots,X_n$) $\vee$ goal($X_1,\ldots,X_n$)]

Regel als logische Formel

Hornklausel:
Alternative mit maximal
einem nicht-negierten Literal

$\forall X_1 \ldots \forall X_n$
[$\neg$subgoal$_1$($X_1,\ldots,X_n$) $\vee \ldots \vee$ $\neg$subgoal$_m$($X_1,\ldots,X_n$) $\vee$ goal($X_1,\ldots,X_n$)]

Regel als logische Formel

Variable, die nur im Regelkörper auftreten, können als existentiell quantifizierte Variable **innerhalb des Regelkörpers** (!) betrachtet werden:

$\forall X_1 \ldots \forall X_n \forall Y_1 \ldots \forall Y_k$ [
$subgoal_1(X_1,\ldots,X_n,Y_1,\ldots,Y_k) \wedge \ldots \wedge subgoal_m(X_1,\ldots,X_n,Y_1,\ldots,Y_k)$
$\rightarrow goal(X_1,\ldots,X_n)$]

ist logisch äquivalent zu

$\forall X_1 \ldots \forall X_n$ [
$\exists Y_1 .. \exists Y_k [subgoal_1(X_1,..,X_n,Y_1,..,Y_k) \wedge \ldots \wedge subgoal_m(X_1,..,X_n,Y_1,..,Y_k)]$
$\rightarrow goal(X_1,\ldots,X_n)$]

```
grandfather(X,Z):-father(X,Y),father(Y,Z).
```


Regel als logische Formel

Intuitive Bedeutung:

„goal" gilt (ist beweisbar)
falls alle „subgoals" gelten (beweisbar sind).

entspricht anschaulich der Abtrennungsregel (modus ponens)

$$\frac{H_1 \rightarrow H_2 \ ,\ H_1}{H_2}$$

$goal(X_1,\ldots,X_n)$:- $subgoal_1(X_1,\ldots,X_n)$,..., $subgoal_m(X_1,\ldots,X_n)$.

Unbenannte/anonyme Variable

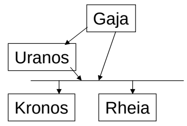
Unbenannte/anonyme Variable: _ für „beliebig"

```
mother_in_law(X,Y):-
        mother(X,Z),parent(Y,Z,_).
```

```
mother_in_law(X,Y):-
        mother(X,Z),parent(Z,Y,_).
```


Anfrage/Beweis

```
?- mother_in_law(gaea,gaea).
```



Zu beweisen:

```
mother_in_law(gaea,gaea).
```

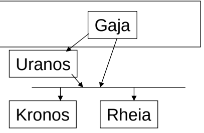
verfügbare Klausel:

```
mother_in_law(X,Y):-
        mother(X,Z),parent(Z,Y,_).
```

Klausel mit Bindung `X=gaea, Y=gaea, Z=Z[1]`

```
mother_in_law(gaea,gaea):-
     mother(gaea,Z[1]),parent(Z[1],gaea,_).
```


Anfrage/Beweis



zu beweisen:

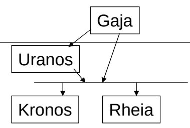
```
mother_in_law(gaea,gaea):-
     mother(gaea,Z[1]),parent(Z[1],gaea,_).
```

dafür zu beweisen

1) `mother(gaea,Z[1]).`

2) `parent(Z[1],gaea,_).`

Anfrage/Beweis



zu beweisen:

1) `mother(gaea,Z[1]).`

Verfügbare Klausel:

```
mother(X,Y):-parent(X,Y),female(X).
```

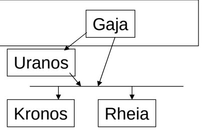
Klausel mit Bindung `X=gaea, Y=Z[1]` :

```
mother(gaea,Z[1]):-parent(gaea,Z[1]),female(gaea).
```

zu beweisen 1.1) `parent(gaea, Z[1] ).`

zu beweisen 1.2) `female(gaea).`

Anfrage/Beweis



zu beweisen:

1.1) parent(gaea, $Z_{[1]}$).

verfügbarer Fakt ermöglicht Bindung $Z_{[1]}$ =uranus :

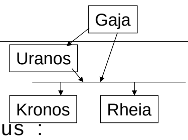
parent(gaea,uranus).

1.2) female(gaea).

verfügbarer Fakt:

female(gaea).

Anfrage/Beweis



zu beweisen

unter Beachtung der Bindung $Z_{[1]}$=uranus :

2) parent(uranus,gaea,_).

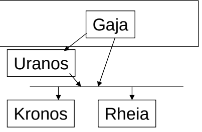
Verfügbare Klausel:

parent(X,Y,Z) :-father(X,Z),mother(Y,Z).

Klausel mit Bindungen :

parent(uranus,gaea,$Z_{[2]}$) :-
father(uranus,$Z_{[2]}$),mother(gaea,$Z_{[2]}$).

Anfrage/Beweis



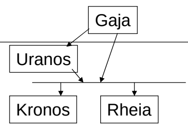
zu beweisen

parent(uranus,gaea,$Z_{[2]}$) :-
father(uranus,$Z_{[2]}$),mother(gaea,$Z_{[2]}$).

zu beweisen 2.1) father(uranus,$Z_{[2]}$).

zu beweisen 2.2) mother(gaea,$Z_{[2]}$).

Anfrage/Beweis



zu beweisen:

2.1) father(uranus,$Z_{[2]}$).

Verfügbare Klausel:

father(X,Y):-parent(X,Y),male(X).

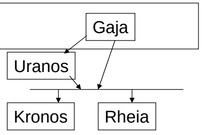
Klausel mit Bindung X=uranus, Y=$Z_{[2]}$:

father(uranus,$Z_{[2]}$):-
parent(uranus,$Z_{[2]}$),male(uranus).

zu beweisen 2.1.1) parent(uranus, $Z_{[2]}$).

zu beweisen 2.1.2) male(uranus).

Anfrage/Beweis



zu beweisen:

2.1.1) parent(uranus,$Z_{[2]}$).

verfügbarer Fakt (z.B.), bewirkt Bindung $Z_{[2]}$=cronus :

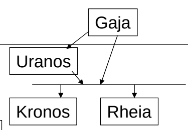
parent(uranus,cronus).

2.1.2) male(uranus).

verfügbarer Fakt:

male(uranus).

Anfrage/Beweis



zu beweisen

mit erfolgter Bindung $Z_{[2]}$=cronus

2.2) mother(gaea,cronus).

Verfügbare Klausel:

mother(X,Y):-parent(X,Y),female(X).

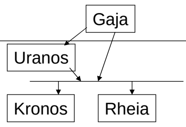
Klausel mit Bindung X=gaea, Y=cronus :

mother(gaea,cronus):-
parent(gaea,cronus),female(gaea).

zu beweisen 2.2.1) parent(gaea,cronus),

zu beweisen 2.2.2) female(gaea).

Anfrage/Beweis



zu beweisen:

2.2.1) `parent(gaea,cronus).`

verfügbarer Fakt:

`parent(gaea,cronus).`

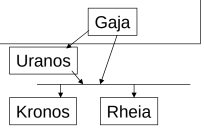
2.2.2) `female(gaea).`

verfügbarer Fakt:

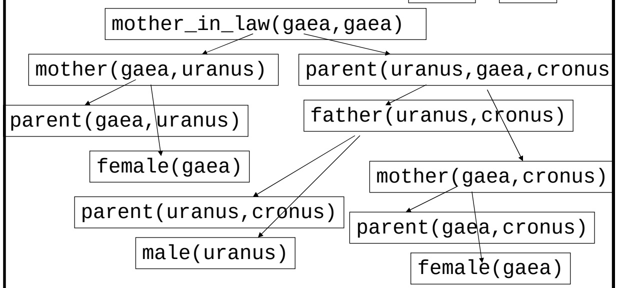
`female(gaea).`

Fertig

Anfrage/Beweis-**Baum**




```
?- mother_in_law(gaea,gaea).
Yes.
```



(existenzielle) Anfragen mit Regeln

```
?- father(zeus,athena).
yes.
?- father(zeus,X).
X=hermes?
;
X=athena?
;
X=ares?
.
yes.
```

```
rent(uranus, cronus).
rent(gaea,   cronus).
rent(gaea,   rhea).
rent(rhea,   zeus).
rent(cronus, zeus).
rent(rhea,   hera).
rent(cronus, hera).
rent(cronus, hades).
rent(rhea,   hades).
rent(cronus, hestia
rent(rhea,   hestia
rent(zeus,   hermes
rent(maia,   hermes
...
```

```
female(gaea).
female(rhea).
female(hera).
female(hestia).
female(demeter).
female(athena).
female(metis).
female(maia).
female(persephone).
female(aphrodite).
```

```
male(uranus).
male(cronus).
male(zeus).
male(hades).
male(hermes).
```

```
father(Vater,Kind)
        :-parent(Vater,Kind), male(Vater).
```


Prolog-Programm

Programme bestehen aus Klauseln.

```
father(X,Y):-parent(X,Y),male(X).
mother(X,Y):-parent(X,Y),female(X).

parent(X,Y,Z) :-father(X,Z),mother(Y,Z).

son(X,Y):-parent(X,Y),male(Y).

grandfather(X,Z):-father(X,Y),parent(Y,Z).
grandmother(X,Z):-mother(X,Y),parent(Y,Z).

grandchild(X,Y):-grandfather(Y,X).
grandchild(X,Y):-grandmother(Y,Z).
```

```
parent(uranus, cronus).
parent(gaea,   cronus).
parent(gaea,   rhea).
parent(rhea,   zeus).
parent(cronus, zeus).
parent(rhea,   hera).
parent(cronus, hera).
parent(cronus, hades).
parent(rhea,   hades).
parent(cronus, hestia
parent(rhea,   hestia
parent(zeus,   hermes
parent(maia,   hermes
...
```

```
female(gaea).
female(rhea).
female(hera).
female(hestia).
female(demeter).
female(athena).
female(metis).
female(maia).
female(persephone).
female(aphrodite).
```

```
male(uranus).
male(cronus).
male(zeus).
male(hades).
male(hermes).
male(apollo).
male(dionysius).
male(hephaestus).
male(poseidon).
```

Klauseln sind Fakten oder Regeln.

Fakt als Regel: `fakt:-true`

mit Prädikat true/0

Prolog-Programm

Klauseln definieren Relationen (Prädikate)

```
father(Vater,Kind)
        :-parent(Vater,Kind), male(Vater).
```

Intuitive Bedeutung:

Linke Seite gilt, wenn alle Prädikate der rechten Seite bei entsprechender Belegung/Bindung der Variablen gelten.

Es gilt: `father(zeus,athena).`

weil gilt: `parent(zeus,athena).`

`male(zeus).`

Prolog-Programm: Prozeduren

Klauseln mit gleichem Kopf-Funktor (Name,Stelligkeit) bilden eine **Prozedur**

Die Klauseln einer Prozedur bieten Alternativen für die Prolog-Beweise

```
grandfather(X,Z):-father(X,Y),father(Y,Z).
grandfather(X,Z):-father(X,Y),mother(Y,Z).
```

```
parent(uranus, cronus).
parent(gaea,   cronus).
parent(gaea,   rhea).
parent(rhea,   zeus).
...
```

Prolog-Programm: Prozeduren

Klauseln mit gleichem Kopf-Funktor
(Name,Stelligkeit) bilden eine **Prozedur**

Redundanzen sind **logisch** unproblemastisch

```
grandfather(X,Z):-father(X,Y),father(Y,Z).
grandfather(X,Z):-father(X,Y),mother(Y,Z).
grandfather(X,Z):-father(X,Y),parent(Y,Z).
grandfather(cronus, ares).
grandfather(cronus, athena).
```

Redundanzen bieten zusätzliche Beweisvarianten

PROLOG: evtl. unerwünschte Auswirkungen

Softwaretechnologie: Minimalitätsprinzip

Prolog-Programm

Anfragen an Programme haben die Form

$? - goal(X_1,\ldots,X_n).$

im Sinne von „gilt ...?" („ist ... beweisbar?"):

$\exists X_1 .. \exists X_k \; [\; goal(X_1,\ldots,X_n) \;]$

Oder allgemeiner

$? - goal_1(X_1,\ldots,X_n) ,\ldots, goal_m(X_1,\ldots,X_n).$

im Sinne von „gilt ...?" („ist ... beweisbar?"):

$\exists X_1 .. \exists X_k \; [goal_1(X_1,\ldots,X_n) \wedge \ldots \wedge goal_m(X_1,\ldots,X_n) \;]$

Prolog-Interpreter

Laufzeit-System für Prolog,
das Antworten auf Anfragen an ein Programm gibt,
d.h. Beweise sucht und ausführt (Theorem-Beweiser).

Vorstellung:
Man gibt Fakten und Zusammenhänge als Programm ein
(z.B. Fahrplantabelle)
und erhält Antworten bzgl. aller Folgerungen
(z.B. billigste Verbindungen von A nach B) .

Prolog-Interpreter

Fahrplan-Fakten

```
s_bahn(alexanderplatz,jannowitzbrücke,6:09,6:11, 103).
s_bahn(jannowitzbrücke,ostbahnhof,     6:11,6:13, 103).
...
```

Suchprogramm

```
erreichbar(Start,Ziel,Zeit)
:- s_bahn(Start,Zwischenziel,Abfahrt,Ankunft,_),
   erreichbar(Zwischenziel,Ziel,Zeit1),
   berechneZeit(Zeit1,Ankunft,Abfahrt,Zeit).
...
```


Prolog und Logik

Prolog-Programm P
= Axiome

Anfrage Q
= Frage nach Beweisbarkeit von Q aus P

Antwort
= Ergebnis eines Beweises bzw. Fehlschlag

Trace
= Verlauf des Beweisversuchs

Prolog-Interpreter

Ziel:
Prolog-Interpreter als universelles Verfahren im PK1

Insbesondere
Vollständigkeit:
Interpreter liefert alle Folgerungen aus dem Programm

Korrektheit:
Interpreter liefert nur Folgerungen aus dem Programm

Situation im PK1

Q folgt aus Formelmenge $\{P_1,..,P_n\}$
gdw. Q ist aus Formelmenge $\{P_1,..,P_n\}$ syntaktisch ableitbar
gdw. $P_1 \wedge ... \wedge P_n \rightarrow Q$ ist allgemeingültig

Allgemeingültigkeit ist axiomatisierbar/aufzählbar:
Falls ein Ausdruck H allgemeingültig ist,
so ist das in endlich vielen Schritten feststellbar.
Genauer: Es gibt dafür ein universelles Verfahren.

Algorithmus/Programm
„Theorembeweiser“

Situation im PK1

Q folgt aus Formelmenge $\{P_1,..,P_n\}$
gdw. Q ist aus Formelmenge $\{P_1,..,P_n\}$ syntaktisch ableitbar
gdw. $P_1 \wedge ... \wedge P_n \rightarrow Q$ ist allgemeingültig

Allgemeingültigkeit ist nicht entscheidbar:
Es gibt **kein universelles** Verfahren, das für beliebige H **entscheidet**, ob H allgemeingültig ist.

Falls ein Ausdruck H **nicht allgemeingültig** ist,
so ist das eventuell nicht feststellbar
(Verfahren kommt evtl. nicht zum Abbruch).
Genauer: Es gibt dafür **kein** universelles Verfahren.

Situation im PK1

Q folgt aus Formelmenge $\{P_1,..,P_n\}$
gdw. Q ist aus Formelmenge $\{P_1,..,P_n\}$ syntaktisch ableitbar
gdw. $P_1 \wedge ... \wedge P_n \rightarrow Q$ ist allgemeingültig

Falls ein Ausdruck H **allgemeingültig** ist,
so ist das in endlich vielen Schritten feststellbar.
Genauer: Es gibt dafür ein **universelles Verfahren**.

Falls ein Ausdruck H **nicht allgemeingültig** ist,
so ist das nicht allgemein feststellbar.
Genauer: Es gibt dafür **kein universelles Verfahren**.

Konsequenzen aus Situation im PK1

Kein Entscheidungsprogramm im PK1 möglich.

Spezielle Situation für Prolog-Interpreter:
- eingeschränkt durch Horn-Klauseln
- eingeschränkt durch Beweisstrategie im Interpreter
  (Frage nach Vollständigkeit/Korrektheit!)

Entscheidungsverfahren existieren prinzipiell für
- aussagenlogische Programme oder
- Programme über endlichen Relationen

Nichtdeterministische Suche nach Beweisbaum

Ausgangspunkt und Zwischenzustände:

Menge von „offenen“ (zu beweisenden) Teilzielen:
subgoals = { $subgoal_1$(...) ,..., $subgoal_m$(...) }

Die Teilziele $subgoal_1$(...) haben die Form funktor(t_1,...,t_n) .
Dabei bezeichnet funktor ein n-stelliges Prädikat,
dessen Argumente jeweils an Terme t_i gebunden sind.

Terme (Strukturen) sind Variablen, Konstante oder komplexere Strukturen, die wiederum Terme enthalten können.

Nichtdeterministische Suche nach Beweisbaum

Ziel (Ende des Verfahrens):

subgoals ={} , d.h. alle Teilziele sind bewiesen
dabei Antwort „yes“ bzw.
Angabe der Terme,
an die die Variablen der Anfrage gebunden wurden.

oder:
kein weiterer Beweisversuch möglich, dabei Antwort „no“

Nichtdeterministische Suche nach Beweisbaum

Zwischenschritte:

Wähle ein zu beweisendes Teilziel:

$funktor(t_1,...,t_n) \in subgoals$

Wähle eine passende Klausel der zugehörigen Prozedur:

$funktor(x_1,..,x_n)$:- $funktor^1(x^1_1,..,x^1_{n1}),...,funktor^m(x^m_1,..,x^m_{nm})$

Unifikation

des Kopfes $funktor(x_1,...,x_n)$ mit Teilziel $funktor(t_1,...,t_n)$ ergibt eine Variablensubstitution σ

(Ersetzung von Variablen durch Terme)

Neuer Zwischenzustand:

$subgoals := \sigma(\ (subgoals - \{ funktor(t_1,...,t_n) \} \ )$

$\cup \{ funktor^1(t^1_1,..,t^1_{n1}),...,funktor^m(t^m_1,..,t^m_{nm})\} \ )$

Nichtdeterministische Suche nach Beweisbaum

Die Suche ist erfolgreich, wenn

- in jedem Zwischenschritt eine passende Klausel gewählt wird, bei der die Unifikation gelingt,
- am Ende subgoals={} gilt.

In jedem Schritt i erfolgen Substitutionen σ_i von (allen) Variablen.

Die Substitutionen ergeben in ihrer Gesamtheit die Terme, an die die Variablen X der Anfrage gebunden wurden: „Antwort-Substitution":

$\sigma(X) = \sigma_k \ (\sigma_{k-1} (... \ \sigma_1 (X) \ ...) \ )$

Interpreter für Standard-Prolog

Idee:

Systematische Suche nach Beweismöglichkeiten

(Reihenfolge für „wähle Teilziel/Klausel")

Reihenfolge innerhalb einer Prozedur

(Alternativen für Beweis)

oben vor unten

Reihenfolge innerhalb einer Klausel

(alle subgoals müssen erfüllt werden)

links vor rechts

Interpreter für Standard-Prolog

Backtracking:

Alternativen für den Beweis eines Teilziel werden markiert („Backtrack-Punkte").

Beim Fehlschlagen eines Beweisversuchs wird am jüngsten Backtrack-Punkt ein alternativer Beweis gestartet („chronologisches Backtracking").
Dabei werden zwischenzeitliche Variablenbindungen zurückgenommen.

Eingabe von „;" bei Antworten auf existentielle Anfragen wirkt wie Fehlschlag (löst Backtracking aus) .


```
grandfather(X,Z):-father(X,Y),father(Y,Z).
grandfather(X,Z):-father(X,Y),mother(Y,Z).
```

```
?-grandfather(X,ares).
```

Beweisversuch mit 1. Klausel (ggf. neue Variablennamen!)

```
grandfather(X1,Z):-father(X1,Y),father(Y,Z).
```

Backtrackpunkt für 2. Klausel:

```
grandfather(X,Z):-father(X,Y),mother(Y,Z).
```

Substitution σ (X1) = X σ (Z) = ares

```
grandfather(X,ares):-
        father(X,Y),father(Y,ares).
```

zu beweisen:

```
father(X,Y).    father(Y,ares).
```



```
father(X,Y):-parent(X,Y),male(X).
```

zu beweisen:

```
father(X,Y).
```

Beweisversuch mit Klausel (neue Variablennamen):

```
father(X2,Y1):-parent(X2,Y1),male(X2).
```

(Keine Alternativen – kein Backtrackpunkt)

Substitution σ (X2) = X σ (Y1) = Y

```
father(X,Y):-parent(X,Y),male(X).
```

zu beweisen:

```
parent(X,Y).   male(X).     father(Y,ares).
```

zu beweisen:

`parent(X,Y).`

Beweisversuch mit 1. Fakt

`parent(uranus, cronus).`

Backtrackpunkt für Alternativen

`parent(gaea, cronus).`

```
parent(uranus, cronus).
parent(gaea,   cronus).
parent(gaea,   rhea).
parent(rhea,   zeus).
parent(cronus, zeus).
parent(rhea,   hera).
parent(cronus, hera).
parent(cronus, hades).
...
```

Substitution σ (X) = uranus σ (Y) = cronus

`parent(uranus,cronus).`

Ist Fakt, d.h. keine neuen Teilziele.

Zu beweisen:

`male(uranus).` `father(uranus,ares).`

zu beweisen:

`male(uranus).`

Beweis mit Fakt

`male(uranus).`

gelingt:

`male(uranus).`

```
male(uranus).
male(cronus).
male(zeus).
male(hades).
male(hermes).
male(apollo).
male(dionysius).
male(hephaestus).
male(poseidon).
```

Keine neuen Substitutionen.
Keine neuen Teilziele.

Zu beweisen:

`father(uranus,ares).`

`father(X,Y):-parent(X,Y),male(X).`

zu beweisen:

`father(uranus,ares).`

Beweisversuch mit Klausel (neue Variablennamen)

`father(X3,Y2):-parent(X3,Y2),male(X3).`

(Keine Alternativen – kein Backtrackpunkt)

Substitution σ (X3) = uranus σ (Y1) = ares

```
father(uranus,ares):-
        parent(uranus,ares),male(uranus).
```

zu beweisen:

`parent(uranus,ares).` `male(uranus).`

zu beweisen:

`parent(uranus,ares).`

Es gibt keinen solchen Fakt

Beweisversuch fehlgeschlagen.

```
parent(uranus, cronus).
parent(gaea,   cronus).
parent(gaea,   rhea).
parent(rhea,   zeus).
parent(cronus, zeus).
parent(rhea,   hera).
parent(cronus, hera).
parent(cronus, hades).
...
```

Rückkehr zum jüngsten Backtrack-Punkt

`parent(gaea,    cronus).`

beim Beweis für

`parent(X,Y).` `male(X).` `father(Y,ares).`

Weitere Fehlschläge folgen bei Beweisversuchen mit

```
parent(uranus, cronus).
parent(gaea,   cronus).
parent(gaea,   rhea).
parent(rhea,   zeus).
parent(cronus, zeus).
parent(rhea,   hera).
parent(cronus, hera).
parent(cronus, hades).
...
```


zu beweisen:

`parent(X,Y).`

Beweisversuch mit Fakt

`parent(cronus, zeus).`

Backtrackpunkt für Alternativen.

`parent(rhea, hera).`

```
parent(uranus, cronus).
parent(gaea,   cronus).
parent(gaea,   rhea).
parent(rhea,   zeus).
parent(cronus, zeus).
parent(rhea,   hera).
parent(cronus, hera).
parent(cronus, hades).
...
```

Substitution σ (X) = cronus σ (Y) = zeus

`parent(cronus,zeus).`

Ist Fakt, d.h. keine neuen Teilziele.

Zu beweisen:

`male(cronus).` `father(zeus,ares).`

```
male(uranus).
male(cronus).
male(zeus).
male(hades).
male(hermes).
male(apollo).
male(dionysius).
male(hephaestus).
male(poseidon).
```

zu beweisen:

`male(cronus).`

Beweis mit Fakt

`male(cronus).`

gelingt:

`male(cronus).`

Keine neuen Substitutionen.

Keine neuen Teilziele.

Zu beweisen:

`father(zeus,ares).`

`father(X,Y):-parent(X,Y),male(X).`

zu beweisen:

`father(zeus,ares).`

Beweisversuch mit Klausel (neue Variablennamen)

`father(X3,Y2):-parent(X3,Y2),male(X3).`

(Keine Alternativen – kein Backtrackpunkt)

Substitution σ (X3) = zeus σ (Y1) = ares

```
father(zeus,ares):-
        parent(zeus,ares),male(zeus).
```

zu beweisen:

`parent(zeus,ares).` `male(zeus).`


```
parent(uranus, cronus).
parent(gaea,   cronus).
parent(gaea,   rhea).
parent(rhea,   zeus).
parent(cronus, zeus).
parent(rhea,   hera).
parent(cronus, hera).
parent(cronus, hades).
...
```

zu beweisen:

`parent(zeus,ares).`

Beweis mit Fakt

`parent(zeus,ares).`

gelingt:

`parent(zeus,ares).`

Keine neuen Substitutionen.

Keine neuen Teilziele.

Zu beweisen:

`male(zeus).`


```
male(uranus).
male(cronus).
male(zeus).
male(hades).
male(hermes).
male(apollo).
male(dionysius).
male(hephaestus).
male(poseidon).
```

zu beweisen:

`male(zeus).`

Beweis mit Fakt

`male(zeus).`

gelingt:

`male(zeus).`

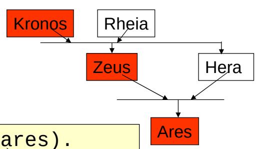
Keine neuen Substitutionen.

Keine neuen Teilziele.

Beweisversuch gelungen mit Substition: σ (X) = cronus

`?-grandfather(X,ares).`

Antwort `X = cronus?`



Beweisbaum:

`grandfather(cronus,ares).`

`father(cronus,zeus).` `father(zeus,ares).`

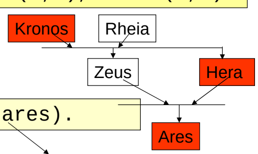
`male(cronus).`

`parent(cronus,zeus).` `male(zeus).`

`parent(zeus,ares).`


```
?-grandfather(X,ares).
X = cronus?
;
X = cronus?
```

`grandfather(X,Z):-father(X,Y),mother(Y,Z).`



Zweiter Beweisbaum:

`grandfather(cronus,ares).`

`father(cronus,hera).` `mother(hera,ares).`

`male(cronus).`

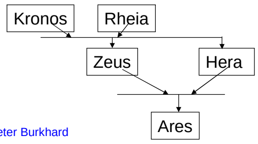
`parent(cronus,zeus).` `female(hera).`

`parent(hera,ares).`

Redundanzen ...

... führen wegen der systematischen Durchmusterung aller Beweisversuche zu Wiederholungen von Resultaten

```
?-grandfather(X,ares).
        grandfather(X,Z):-father(X,Y),father(Y,Z).
X = cronus?
;       grandfather(X,Z):-father(X,Y),mother(Y,Z).
X = cronus?
```



Unifikation (matching, instantiation)

Terme unifizieren:

Durch geeigneten **Unifikator** (Variablen-Substitution σ) als Zeichenkette identisch machen.

```
parent(X,ares).        parent(zeus,Y).
```

σ (X) = zeus σ (Y) = ares

```
parent(zeus,ares).
```

„Instantiation“:

Resultierender Term bei Substitution

Unifikation (matching, instantiation)

in Prolog: Beweisen eines Teilziels erfordert „Matchen“ von Teilziel und Klauselkopf

```
 father(zeus,ares).
father(X3,Y2):-parent(X3,Y2),male(X3).
father(zeus,ares):-
          parent(zeus,ares),male(zeus).
```

Analogie: „Prozedur-Aufruf“

Unifikation (matching, instantiation)

```
 father(zeus,ares).
father(X3,Y2):-parent(X3,Y2),male(X3).
father(zeus,ares):-
          parent(zeus,ares),male(zeus).
```

Analogie: „Prozedur-Aufruf“

Parameterübergabe durch Instantiierung:

Bindung von Variablen für gesamte Klausel

σ(X)=zeus
σ(Y)=ares

Variable „gehören“ den Klauseln:

Lebensdauer bis zum Backtracking

Sichtbarkeit innerhalb der Klausel

Kein Überschreiben von Werten

Unifikationsregeln

Terme t_1 und t_2 sind unifizierbar, falls

- t_1 und t_2 sind identische Konstanten   oder
- t_1 (bzw. t_2) ist eine Variable:
  t_1 (bzw. t_2) wird an t_2 (bzw. t_1) gebunden   oder
- t_1= funktor(t_{11},...,t_{1n}) und t_2 =funktor(t_{21},...,t_{2n})
  sind Strukturen mit identischem funktor (Name, Stelligkeit) und die Argumente t_{1i} t_{2i} sind paarweise unifizierbar.

Rekursive Definition,

die Substitution σ ergibt sich schrittweise.

Unifikationsregeln

dreieck(P,punkt(X,Y),punkt(1,X)).
dreieck(punkt(2,3),punkt(1,X),punkt(Y,Z)).

Variablenseparierung:

dreieck(P,   punkt(X,Y), punkt(1,X)).
dreieck(punkt(2,3),punkt(1,C), punkt(A,B)).

σ(P)= punkt(2,3)
σ(X)=1
σ(Y)=C
σ(A) =1
σ(B) =1
σ(C)=C

σ

dreieck(punkt(2,3),punkt(1,C), punkt(1,1)).

Prolog-Operatoren „=“ und „==“

Das Ziel `term1 = term2` ist erfüllt,
falls `term1` und `term2` unifizierbar sind.

– Variable in `term1` und `term2` werden ggf. instantiiert.

Das Ziel `term1 \= term2` ist erfüllt,
falls `term1` und `term2` nicht unifizierbar sind.

Das Ziel `term1 == term2` ist erfüllt,
falls `term1` und `term2` identisch sind.

– nicht unifizierte Variable sind nicht identisch

Das Ziel `term1 \== term2` ist erfüllt,
falls `term1` und `term2` nicht identisch sind.

Prolog-Operatoren „=“ und „==“

```
?- dreieck(P,punkt(X,Y),punkt(1,X))
   = dreieck(punkt(2,3),punkt(1,X),punkt(Y,Z)).
```

```
?- dreieck(P,punkt(X,Y),punkt(1,X))
   == dreieck(punkt(2,3),punkt(1,X),punkt(Y,Z)).
```


Occur-Check, $\sigma(X)$ = funktor(..., X, ...)

Beispiel: Programm-Klausel `p(X,f (X))`

Unterschiedliche Reaktionen von Prolog-Systemen auf Anfragen

```
? - p(Y,Y).
?- p(Y,Y), write(Y).
?- p(Y,Y), p(Z,Z),Y=Z.
```

aufwendiger Test.
Ignorieren?

Problemzerlegung

Zerlege ein Problem P in einzelne Probleme $P_1,...,P_n$

Löse jedes Problem P_i

Füge die Lösungen zusammen zu P

Beispiele:
Ungarischer Würfel
Kurvendiskussion
Integralrechnung

Problemzerlegung

Klausel als Problemzerlegung

goal(X_1,...,X_n) :- subgoal$_1$(X_1,...,X_n) ,..., subgoal$_m$(X_1,...,X_n).

„goal“ gilt (ist beweisbar)
falls alle „subgoals“ gelten (beweisbar sind).

um „goal“ zu beweisen,
beweise alle „subgoals“

Problemzerlegung

goal(X_1,...,X_n) :- subgoal$_1$(X_1,...,X_n) ,..., subgoal$_m$(X_1,...,X_n).

um „goal“ zu beweisen,
beweise alle „subgoals“

Problemzerlegung

```
erreichbar(Start,Ziel,Zeit)
:- s_bahn(Start,Zwischenziel,Abfahrt,Ankunft,_),
   erreichbar(Zwischenziel,Ziel,Zeit1),
   berechneZeit(Zeit1,Ankunft,Abfahrt,Zeit).
```

Problemzerlegung

`berechneZeit(Zeit1,Ankunft,Abfahrt,Zeit).`

Beabsichtigte Bedeutung:
Zeit = Zeit1 + (Ankunft - Abfahrt)

(Spezielle Notationen für Zeitangaben beachten)

Problemzerlegung:

```
berechneZeit(Zeit1,Ankunft,Abfahrt,Zeit)
:- addiereZeit(Zeit1,Differenz,Zeit),
   subtrahiereZeit(Ankunft,Abfahrt,Differenz).
```

Mit geeignet definierten Prädikaten
`addiereZeit/3, subtrahiereZeit/3`

Unterschied zu prozeduralem Denken

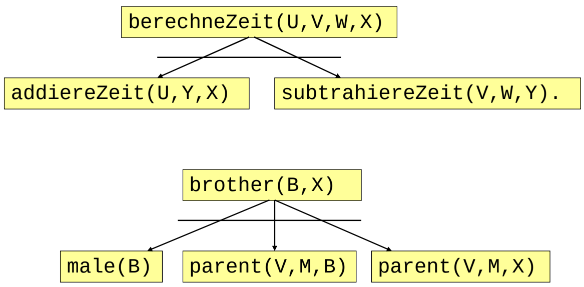
```
berechneZeit(Zeit1,Ankunft,Abfahrt,Zeit)
:- addiereZeit(Zeit1,Differenz,Zeit),
   subtrahiereZeit(Ankunft,Abfahrt,Differenz).
```

Logische Sicht:
Berechnung von „Differenz“ kann später erfolgen
(Bindung statt Wertzuweisung).

Die in Standard-Prolog eingebaute Arithmetik erfordert aus Effizienzgründen allerdings gebundene Eingangsparameter.

Problemzerlegung

Graphische Darstellung



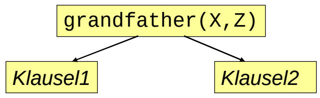
Alternativen für Problemzerlegung

Zerlegung des Problems P in Probleme $P_1,\dots,P_n$
oder in Probleme $P'_1,\dots,P'_{n'}$
oder ...

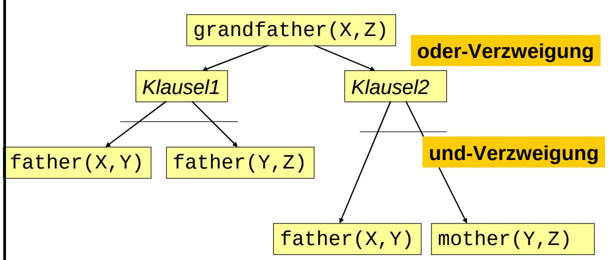
Klauseln einer Prolog-Prozedur bieten Alternativen

```
grandfather(X,Z):-father(X,Y),father(Y,Z).
grandfather(X,Z):-father(X,Y),mother(Y,Z).
```

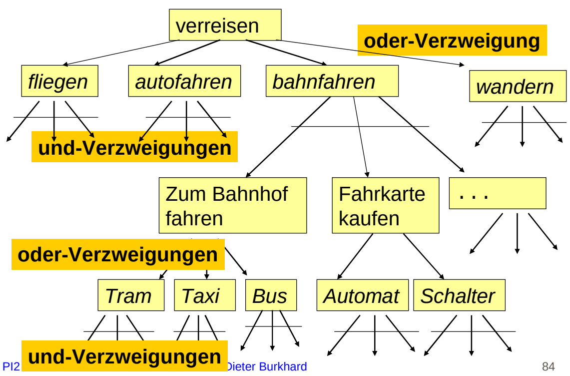
Graphische Darstellung



Alternativen für Problemzerlegung



Kombinierte Verzweigungen



Und-Oder-Baum

Ein und-oder-Baum besteht (abwechselnd) aus
- Knoten mit oder-Verzweigungen und
- Knoten mit und-Verzweigungen

Modell für Problemzerlegungen:
- oder-Verzweigungen für alternative Möglichkeiten zur Problemzerlegung
- und-Verzweigungen für Teilprobleme

Modell für Prolog-Programm:
- oder-Verzweigungen für alternative Klauseln einer Prozedur
- und-Verzweigungen für subgoals einer Klausel

Und-Oder-Baum

Anfrage

Startknoten („**Wurzel**") modelliert Ausgangsproblem

Knoten ohne Nachfolger („**Blätter**") sind unterteilt in
- terminale Knoten („primitive Probleme") modellieren unmittelbar lösbare Probleme

Fakt

- nichtterminale Knoten modellieren nicht zu lösende Probleme

Unerfüllbares Subgoal (keine unifizierende Klausel)

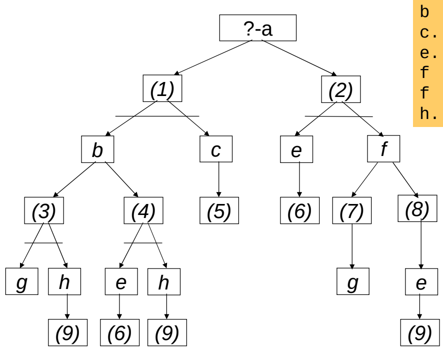
Innere Knoten sind unterteilt in
- Knoten mit und-Verzweigung
- Knoten mit oder-Verzeigung

Und-Oder-Baum

```
a :- b,c.      % (1)
a :- e,f.      % (2)
b :- g,h.      % (3)
b :- e,h.      % (4)
c.             % (5)
e.             % (6)
f :- g.        % (7)
f :- e         % (8)
h.             % (9)
```


Und-Oder-Baum

```
a :- b,c. % (1)
a :- e,f. % (2)
b :- g,h. % (3)
b :- e,h. % (4)
c.        % (5)
e.        % (6)
f :- g.   % (7)
f :- e    % (8)
h.        % (9)
```



Lösbare/unlösbare Knoten

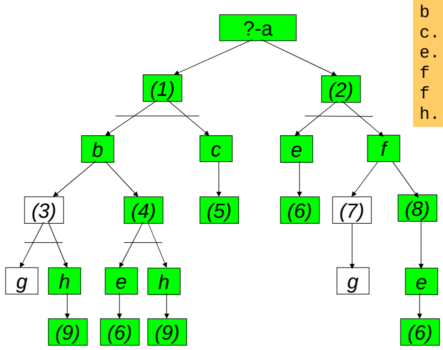
Lösbare Knoten:
1. terminale Knoten,
2. Knoten mit Und-Verzweigung: falls alle Nachfolger lösbar sind
3. Knoten mit Oder-Verzweigung: falls mindestens ein Nachfolger lösbar ist

Unlösbare Knoten:
1. nichtterminale Knoten,
2. Knoten mit Und-Verzweigung: falls mindest. ein Nachfolger unlösbar,
3. Knoten mit Oder-Verzweigung: falls alle Nachfolger unlösbar sind

Für endliche Bäume ergibt sich bei Festlegung für die Blätter eine eindeutige Zerlegung in lösbare/unlösbare Knoten (Bedingungen sind jeweils komplementär)

Lösbare/unlösbare Knoten

```
a :- b,c. % (1)
a :- e,f. % (2)
b :- g,h. % (3)
b :- e,h. % (4)
c.        % (5)
e.        % (6)
f :- g.   % (7)
f :- e    % (8)
h.        % (9)
```



Bottom-up-Konstruktionsalgorithmus

Anfang: $M_{LÖSBAR}$:= terminale Knoten
$M_{UNLÖSBAR}$:= nichtterminale Knoten

Zyklus:
Solange nicht alle Knoten untersucht wurden:
Wähle einen Knoten k, dessen Nachfolger alle untersucht wurden.
Falls bei k und-Verzweigung und alle Nachfolger von k in $M_{LÖSBAR}$ oder
falls bei k oder-Verzweigung und ein Nachfolger von k in $M_{LÖSBAR}$:
$M_{LÖSBAR} := M_{LÖSBAR} \cup \{k\}$,
andernfalls:
$M_{UNLÖSBAR} := M_{UNLÖSBAR} \cup \{k\}$.

Bottom-up-Konstruktionsalgorithmus

Ergebnis für endliche Und-oder-Bäume:
Zerlegung der Knoten in
lösbare Knoten ($M_{LÖSBAR}$) und
unlösbare Knoten ($M_{UNLÖSBAR}$)

Das Ausgangsproblem
kann genau dann durch Problemzerlegung gelöst werden,
wenn der Startknoten in $M_{LÖSBAR}$ ist.

Falls das Ausgangsproblem lösbar ist, kann ein Lösungsbaum konstruiert werden.