Universität Bielefeld

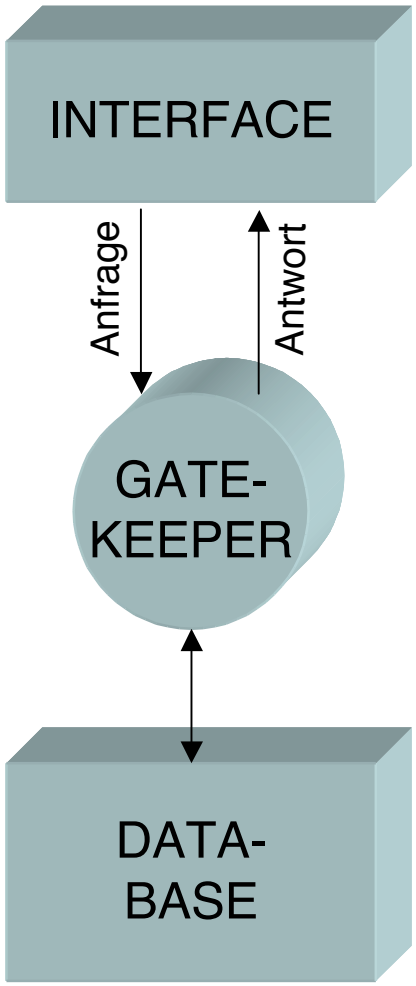
3 Logik und Inferenz

8. Vorlesung: Unifikation; Vorwärts- und Rückwärts-verkettung, Goal Trees

Methoden der Künstlichen Intelligenz

Ipke Wachsmuth WS 2008/2009

Schlussfolgern durch Theorembeweisen



`query` **fügt eine Frage-Formel in DATABASE ein und versucht, darauf passende Antworten zu deduzieren**

Idee: Eine Anfrage wird als zu beweisendes Theorem betrachtet.

(a) Direkter Beweis

- **bei den Axiomen (d.h. hier: den assertierten Formeln) beginnen**
- **Schlussregeln anwenden, bis zu beweisendes Theorem dasteht**

(b) Indirekter Beweis: durch Widerlegung des Gegenteils

- **Nimm an, die Verneinung (das Negat) der Anfrage sei <u>wahr</u>**
- **Zeige, dass die Axiome und das Negat zusammen etwas als <u>wahr</u> bezeichnen, das nicht stimmt**
- **Folgere, dass das Negat <u>nicht wahr</u> sein kann, da ein Widerspruch folgt**
- **Schließe, dass die Anfrage <u>wahr</u> sein muss, da ihr Negat nicht wahr sein kann**

Einschub: Schlussfolgern in der Aussagenlogik

Ein Blick auf die deduktiven Schluss-regeln der Aussagenlogik (AL)

Klauseldarstellung

Modus ponens

$$\frac{A \rightarrow B \qquad A}{B} \qquad \frac{\neg A \vee B \qquad A}{B}$$

Modus tollens

$$\frac{A \rightarrow B \qquad \neg B}{\neg A} \qquad \frac{\neg A \vee B \qquad \neg B}{\neg A}$$

Resolutionsregel

$$\frac{A \vee B \qquad \neg B \vee C}{A \vee C}$$

Resolution ist eine allgemeinere Schlussregel; sie beinhaltet zugleich Modus ponens und Modus tollens.

Beispiel: Resolution in der Aussagenlogik

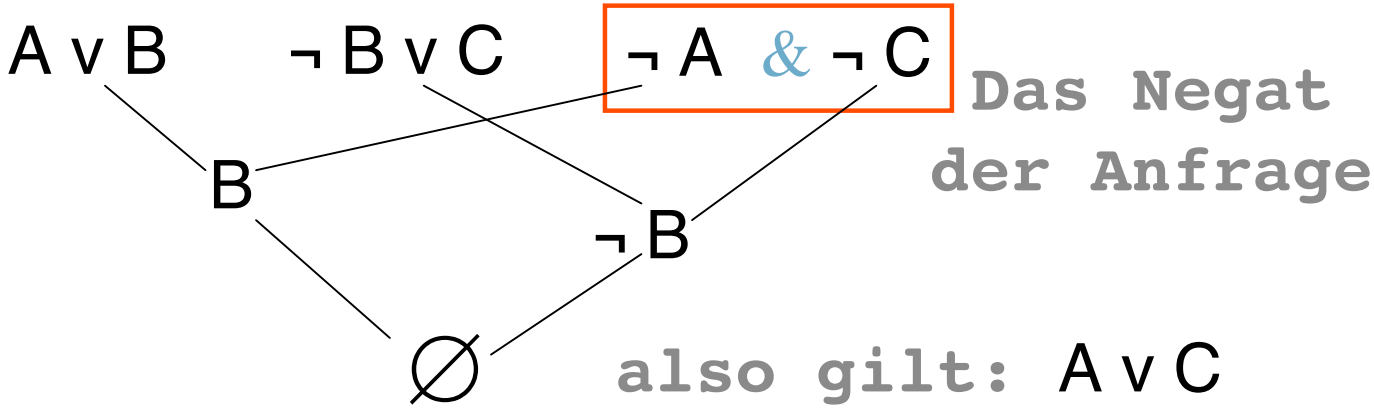
gegeben: Ich trage die Sonnenbrille (A) ODER Es regnet (B)

gegeben: Es regnet nicht (¬B) ODER Ich nehme den Schirm mit (C)

Schluss: Ich trage die Sonnenbrille (A) ODER Ich nehme den Schirm mit (C)

Gilt A v C ?

Beweis durch Widerlegung des Gegenteils



PL-Formeln mit Variablen: 2. Weg

(PL: Prädikatenlogik)

- <u>Unifikation</u>: Sie ermöglicht Inferenzen mit Formeln der PL, die gleiche Prädikate, aber verschiedene Argumente haben.

=> Verallgemeinerung (zunächst) des Modus ponens für PL

(I) Modus Ponens erneut betrachtet:

Aus ***p*** **und** **`(if p q )`** **inferiere** ***q***

- Problem:

  Was tun, wenn hier statt ***p*** eine Formel ***p'*** steht, welche andere Argumente hat als in ***p*** ?

- Idee:

$$\overbrace{\texttt{(inst x canary)}}^{p} \text{ und } \overbrace{\texttt{(inst tweety canary)}}^{p'}$$

würden gleich bei der Substitution: `{x = tweety}`

Verallgemeinerter Modus Ponens

Statt (I) Modus Ponens betrachte folgende allgemeinere Regel:

(III) Unifikations-Inferenz **(Modus Ponens verallgemeinert)**

Aus p' **und** `(if` p q`)` **inferiere** q'

wobei p **mit** p' **unifizieren muss:** $p'\theta = p\theta$

und die resultierende Substitution θ **auf** q **angewandt wird,**

wodurch man q' **erhält als:** $q' = q\theta$

Beispiel

`(if p q)` **`assert: (if (inst ?x canary)(color ?x yellow))`**

`p'` **`assert: (inst tweety canary)`**

`(inst ?x canary)`
`(inst tweety canary)`
unifizieren mit der Substitution: `{x = tweety}`

„theta“ θ

und durch Anwendung dieser Substitution auf `(color ?x yellow)`

`q'` **ist `(color tweety yellow)` das Ergebnis.**

Unifikation

<u>Unifikation</u> heißt hier:

Substituiere Werte von Variablen so, dass zwei verschiedene Ausdrücke gleich werden. (Werte können auch andere Variablen sein.)

- **Allgemein ist eine Substitution eine <u>Menge von Variable-Wert-Paaren</u>.**
- **Jedes solche Paar heißt <u>Variablenbindung.</u>**
- **Entsprechende Variablen heißen <u>durch die Substitution gebunden</u>.**

Die <u>Anwendung</u> einer Substitution auf eine Formel heißt, jedes Vorkommen einer dadurch gebundenen Variablen durch ihren Wert zu ersetzen.

- **Es können auch mehrere Ausdrücke (Formeln oder Terme) einer gemeinsamen Substitution unterzogen werden!**

Anderes Beispiel für Unifikation

- Können Variablensubstitutionen gefunden werden, die die Ausdrücke gleich machen (unifizieren)?

p `(freund ?x,hans)`

p' `(freund fritz,?y)`

`(freund fritz,hans)`

unifiziert mit der Variablensubstitution {`x=fritz,y=hans`} θ

p und *p'* unterscheiden sich in den Argumenten des `freund`-Prädikats

`(freund ?x,?x)`

`(freund fritz,hans)`

Gegenbeispiel

?

unifiziert *nicht* (`x` kann nicht zwei Dinge zugleich sein)

Verallgemeinerter Modus Ponens *da capo*

Die Inferenzregel (III) ist ein Spezialfall der Resolutionsregel in der Prädikatenlogik. Hauptgewinn:

- **verschiedene separate Inferenz-Mechanismen werden in einem Mechanismus zusammengefasst.**

- **forward chaining für eine assertierte Formel *p* durchzuführen, heißt dann:**

  Finde Assertionen `(if p' q )`
  deren Antezedenten *p'* mit der Assertion *p* unifizieren,
  (die mit der Assertion *p* "resolvieren").

Weitere Notationen und Begriffe

Benennung von Substitutionen durch griechische Buchstaben, z.B.:

„theta“ → θ = `{ x = a, y = (surface ?z) }`

Für p **Formel oder Term: notiere** $p\theta$ **für das Resultat der Substitution von** θ **in** p **.**

Beispiel:

p: `(touching (side ?x) ?y)`

θ = `{ x = a, y = (surface ?z) }`

$p\theta$: `(touching (side a) (surface ?z))`

Begriff der Subsumtion (auch: Subsumption)

Man sagt, eine Formel p **<u>subsumiert</u> (schließt ein) eine andere Formel** q **, wenn es eine Substitution gibt mit** $p\theta = q$ **.**

Beispiel:

`(not (divine ?x))` **subsumiert**

`(not (divine fred))` **mit** θ = `{ x = fred }`

Subsumtionsinferenz, Varianten

Mit der Subsumtion erhält man eine einfache weitere Inferenzregel:

(IV) Subsumtions-Inferenz

Aus *p* subsumiert *q* schließe

q* folgt aus *p

1) Zwei Formeln, die sich gegenseitig subsumieren, heißen <u>Varianten</u> (bedeuten das gleiche).

Beispiel: `(p ?x ?x)` **und** `(p ?y ?y)` **sind Varianten**

2) *p* <u>subsumiert</u> *q* <u>echt</u>, wenn gilt:

***p* subsumiert *q* , aber *q* subsumiert nicht *p* .**

Beispiel: `(p ?x ?y)` **subsumiert echt** `(p ?x ?x)`

Unifikator

(*engl:* unifier) "Vereinheitlicher"

Eine Substitution θ ist ein <u>Unifikator</u> für eine Menge von Formeln oder Termen $\{ p_1, p_2, ..., p_n \}$

wenn $p_1\theta = p_2\theta = ... = p_n\theta = P$

(das heißt also, dass die p_i durch die Substitution θ gleich werden).

Beispiel:

`(above ?z floor-19)` und der Antezedent von

`(if (above ?x ?y) (below ?y ?x))` haben als Unifikator:

θ `={ x = table-21, y = floor-19, z = table-21 }`

dann folgt `(below floor-19 table-21).`

Allgemeinster Unifikator

Most General Unifier (MGU)

Ein Unifikator ist ein MGU (für eine Menge von Formeln oder Termen), wenn das produzierte *P* nicht echt subsumiert wird von dem *P*, das irgendein anderer Unifikator produziert.

Beispiel:

`(above ?z floor-19)` **und der Antezedent von**

`(if (above ?x ?y) (below ?y ?x))` **haben als**

1. Unifikator (s.o. - kein MGU!)

θ_1 ={ x = table-21, y = floor-19, z = table-21 }

(dann folgt `(below floor-19 table-21)` **)**

und als

2. Unifikator (ein MGU!!)

θ_2 ={ x = ?z, y = floor-19}

(dann folgt `(below floor-19 ?z)` **).**

Es ist günstig, einen möglichst allgemeinen Unifikator zu finden: damit nur so viele Spezialisierungen erfolgen, wie eben nötig sind.

(Vor allen Dingen bleiben Variablen erhalten, was oft wichtig ist.)

MGUs sind gut fürs chaining!

Übersicht Unifikation

Unifikation:

das Substituieren von Variablen so, dass zwei Ausdrücke (Terme oder Formeln) gleich werden.

Unifikator:

Die Substitution θ, die zwei Ausdrücke (Terme oder Formeln) gleich macht. (D.h. ein Unifikator ist eine spezielle Substitution!)

Der MGU ist *eindeutig* (bis auf Varianten)

Das ist der beste!

Subsumtion:

Eine Formel p subsumiert eine Formel q, wenn q aus p durch eine Variablensubstitution hervorgeht. (Formeln, die sich gegenseitig subsumieren, heißen Varianten.)

Allgemeinster Unifikator (MGU):

Diejenige unifizierende Substitution, die am wenigsten Spezialisierungen vornimmt. Man kann zeigen: *Für jede unifizierbare Menge von Formeln existiert ein MGU!*

siehe z.B. Lloyd: Foundations of Logic Programming

Vorwärts- und Rückwärtsverkettung

Forward Chaining

Aus p' und (if p q)
inferiere q'

wobei p mit p' unifiziert
mit MGU θ und $q' = q\ \theta$

- Inferenz zur Assertion Time
- "Die Assertion resolviert mit der Implikation."

Backward Chaining

(if p q) sei assertiert.

Wenn nach q' gefragt wird (als goal)
und q, q' haben MGU θ
wird $p' = p\ \theta$ als subgoal aufgeworfen.

- Inferenz zur Query Time
- "Das goal resolviert mit der Implikation."

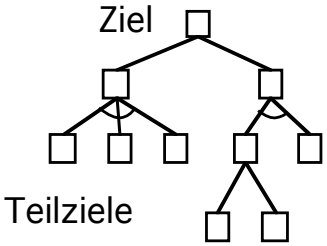
Rückwärtsverkettung: zielorientiertes Inferenzverfahren
(bei konjunktiven (sub-)goals in Verbindung mit Goal Tree-Suche)

Rückwärtsverkettung – Idee

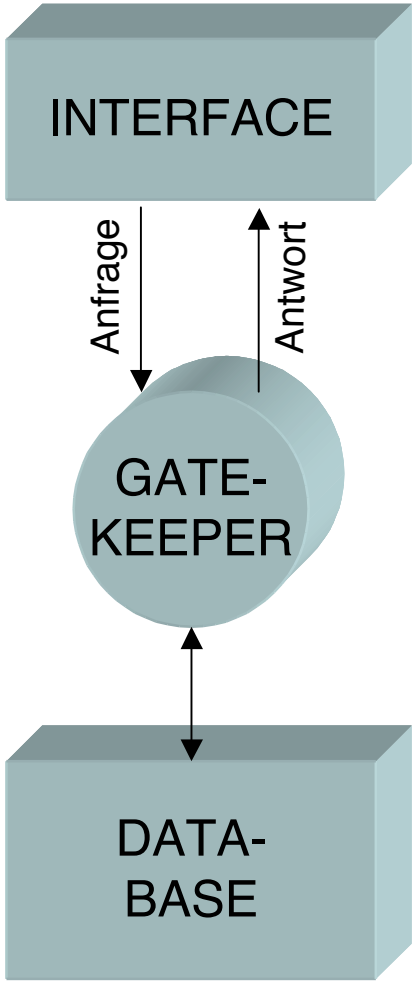
Die meisten Inferenzen in einem deduktiven System werden zur Query Time vorgenommen.

- **GATEKEEPER wird eine Implikation `(if` *`p q`*`)` zwar assertieren, aber nichts damit anstellen, bis eine Frage (query) der Form *`q'`* gestellt wird. Dann wird subquery (subgoal) *`p'`* gestellt.**
- **Das Stellen von subgoals, sub-subgoals etc. wird durch backward chaining (Rückwärtsverkettung) bewerkstelligt.**
- **Der Basismechanismus des Backward Chaining bezieht sich auf *nichtkonjunktive goals:* Verkettung von Implikationen (rückwärts).**
- **Als Suchstruktur bei *konjunktiven goals* von der Form `(if (and` *`p1 p2`*`) `*`q`*`)` werden Goal Trees eingesetzt.**

Goal Trees
(Zielbäume)



Goal-Formeln kennzeichnen



Verwende `(Show: *formula* )` zur Kennzeichnung von goal-Formeln.

Hier am Beispiel `(Show: (color tweety yellow))`
bzw. subgoal `(Show: (inst tweety canary))`
(bezogen auf früheres Beispiel)

- **`Show:` ist kein logischer Junktor! (manchmal Pseudo-Junktor genannt)**
- **hängt aber zusammen mit dem Junktor `not` (kommt später)**

Wenn eine Anfrage (query) an DATABASE gestellt wird, muss die entsprechende Formel gekennzeichnet werden, um sie von den "beliefs" (assertierten Formeln) unterscheiden zu können.

Goal-Formeln skolemisieren

Für goals werden die Konventionen fürs Skolemisieren umgedreht (entsprechend dem Verfahren bei not):

<u>Logisch gesehen</u> ist `Show:` gleichbedeutend mit `not` (die entsprechenden Aus-drücke werden in gleicher Weise skolemisiert)

<u>pragmatisch gesehen</u> entspricht `Show:` einer Annahme (probeweise gemachte Assertion)

```
(Show: (exists (y) (color y yellow)))
```

wird skolemisiert als

```
(Show: (color ?y yellow))
```

```
(Show: (forall (y) (color y yellow)))
```

wird skolemisiert als

```
(Show: (color sk-8 yellow))
```

intuitiv: Wenn es für ein gewisses `sk-8` gezeigt werden kann, kann es für beliebige Instanzen gezeigt werden.

Resolutionsregel – grobe Idee

`(Show: (color tweety yellow))` – goal – kann als Versuch verstanden werden, `(color tweety yellow)` durch Widerspruch zu beweisen:

- `(not (color tweety yellow))`
  wird probeweise assertiert – *als Annahme;*
- Schon vorhandene "geglaubte" Assertionen:
  `(inst tweety canary)`
  `(if (inst ?x canary)(color ?x yellow))`
  was gleichbedeutend ist mit:
  `(or (not(inst ?x canary))(color ?x yellow))`
- Das negierte goal resolviert mit der Implikation zu
  `(not (inst tweety canary))`

WIDERSPRUCH! **Also gilt das ursprüngliche goal.**

Allgemeine Resolutionsregel

Klauseldarstellung!

mit forward und backward chaining als Spezialfällen:

Schreibe `(if p q)` **als** `(or (not p) q)`

Chaining	Aus	und	inferiere
Forward	`p'`	`(or (not p) q)`	`q'`
Backward	`(not q')`	`(or q (not p))`	`(not p')`
Allgemein	`not m'`	`(or m n)`	`n'`
Forward	`m =(not p)`		`n = q`
Backward	`m = q`		`n =(not p)`

Weitere Details siehe Abschnitt 6.6 in Charniak/ McDermott

mit unifizierender Substitution analog zu (III)

(beide Regel-Antezedenten dürfen disjunktiv sein)

(V) Allgemeine Resolutionsregel

Aus `(or` n_1 `(not m'))` **und** `(or m` n_2`)`

inferiere `(or` n_1' n_2'`)`

Zentral: Antwortsubstitution

Allgemein:

Gegeben das goal `(Show:`q'`)` mit `(if` p q`)` in DATABASE und q, q' haben MGU θ.

Produziere subgoal `(Show:`$p\theta$`)` und falls das eine Antwort ψ hat, ist $\theta \cup \psi$ eine Antwort auf das ursprüngliche goal.

„theta“ *„psi“*

Die beim backward-chaining vorgenommenen Substitutionen liefern eine spezifische Antwort!

Beispiel: Assertiert seien

(i) `(inst tweety canary)`

(ii) `(if (inst ?x canary)(color ?x yellow))`

`(Show: (color ?y yellow))` als goal

resolviert mit (ii) zum subgoal

`(Show: (inst ?y canary))` θ = `{x = ?y}`

Dies unifiziert mit (i) zu

`(inst tweety canary)` ψ = `{y = tweety}`

und damit bestätigt sich das ursprüngliche goal zu

`(color tweety yellow)` $\theta \cup \psi$ = `{x = tweety}`

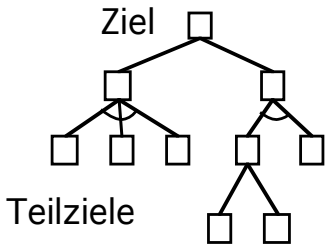
D.h. die Frage *"Existiert etwas, das gelb ist?"* führt zu der Antwort *"Ja,* `tweety` *ist gelb":* `{x = tweety}`

Antwortsubstitution

Theorembeweiser – Goal Tree

Bei konjunktiven (sub)goals
z.B. `(if (and r1 r2) q)`

Goal Trees
(Zielbäume)



Leseempfehlung heute:

- Charniak & McDermott, Kap. 6, 351-360; 378ff.

Bei konjunktiven Goals sind Antworten für ein Konjunkt zu finden, die auch Antworten für die anderen Konjunkte sind.

Algorithmus: Theorembeweiser
("Deductive Retriever")

---> Goal Tree-Suche

Die UND-Knoten eines Goal Trees enthalten *constraints:* Randbedingungen an die Lösungen für jede ihrer Komponenten.

Hier sind das die folgenden: Die Variablen-bindungen für gleich benannte Variablen in den Teillösungen müssen identisch sein!

Ein PROLOG-Beispiel

PROLOG: "PROgramming in LOGic" (Basis: Hornformeln der PL)

Eine Definition von "Großvater":

Für alle `x`, für alle `y`, für alle `z` gilt:
`x` ist Großvater von `y`, wenn
`x` Vater von `z` ist und `z` Vater von `y`.

oder ganz formal:

```
∀x∀y∀z (grossvater x y)
      ← (vater x z)(vater z y)
```

Die entsprechende Definition "über die Mutter":

```
∀x∀y∀z (grossvater x y)
      ← (vater x z)(mutter z y)
```

"klassische" Clocksin/Mellish–Syntax:

```
grossvater(X Y)
:- vater(X Z),vater(Z Y)
```

In PROLOG gelten alle Variablen als implizit all-quantifiziert (matchen mit beliebigen Termen); Quantoren werden weggelassen:

```
   (grossvater ?x ?y)1
← (vater ?x ?z)(vater ?z ?y)
```

```
   (grossvater ?x ?y)2
← (vater ?x ?z)(mutter ?z ?y)
```

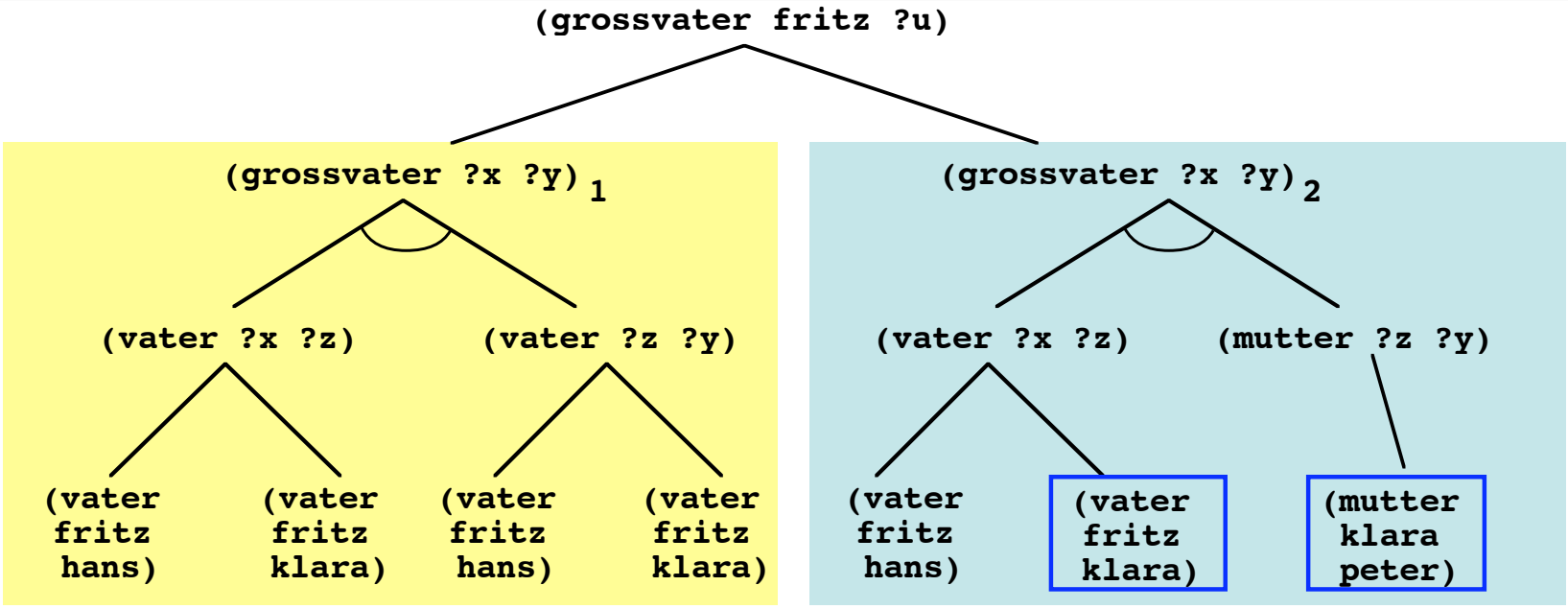
Seien dazu nun noch folgende Fakten assertiert:

```
(vater fritz hans)
(vater fritz klara)
(mutter klara peter)
```

Show: Ist Fritz ein Großvater?

```
(grossvater fritz ?u)
>(grossvater fritz peter)
```

Goal-Tree-Suche in PROLOG




```
show:    (grossvater fritz ?u)
show:    (vater fritz ?z)
success: (vater fritz hans)
show:    (vater hans  ?y)
redo:    (vater hans  ?y)
fail:    (vater hans  ?y)
redo:    (vater fritz ?z)
success: (vater fritz klara)
show:    (vater klara ?y)
redo:    (vater klara ?y)
fail:    (vater klara ?y)
```

```
redo:    (grossvater fritz ?u)
show:    (vater fritz ?z)
success: (vater fritz hans)
show:    (mutter hans ?y)
fail:    (mutter hans ?y)
redo:    (vater fritz ?z)
success: (vater fritz klara)
show:    (mutter klara ?y)
success: (mutter klara peter)
success: (grossvater fritz peter)
```

FRAGE: Was sind hier die Constraints an den UND-Knoten?

ANTWORT: Dass die Variablen-bindungen gleich benannter Variablen in den Teil-lösungen auch tatsächlich gleich sind.