IV. Logische Programmierung

- 1. Grundkonzepte der logischen Programmierung
- 2. Syntax von Prolog
- **3. Rechnen in Prolog**

Unifikation

```
add(X, zero, X).
add(X, succ(Y), succ(Z)) :- add(X, Y, Z).
```

```
?- add(succ(zero),succ(zero),U).
```

Unifikation

Existiert eine Substitution σ von Variablen mit Termen, so dass
σ (Anfrage) = σ (Faktum) oder σ (Anfrage) = σ (Klauselkopf)?

σ(add(succ(zero), succ(zero), U)) = σ(add(X, zero, X)) ?

false

σ(add(succ(zero), succ(zero), U)) = σ(add(X, succ(Y), succ(Z)))?

σ = {X = succ(zero), Y = zero, U = succ(Z)}

Unifikation

- **Substitution σ :** Abbildung von Variablen auf Terme
- s und t sind **unifizierbar**, falls es Substitution gibt mit σ(s) = σ(t)
- σ heißt dann **Unifikator** von s und t

```
σ( add(succ(zero), succ(zero), U) ) = σ( add(X, zero, X) ) ?
false

σ(add(succ(zero), succ(zero), U)) = σ(add(X, succ(Y), succ(Z)))?

σ = {X = succ(zero), Y = zero,  U = succ(Z)}
```

Unifikation

```
σ( add(succ(zero),succ(zero),U       ) ) =
σ( add(X          ,succ(Y)   ,succ(Z)) )
```

Allgemeinster Unifikator (MGU)

```
σ = {X = succ(zero), Y = zero,  U = succ(Z)}
```

```
σ = {X = succ(zero), Y = zero,  U = succ(zero),    Z = zero}
```

```
σ = {X = succ(zero), Y = zero,  U = succ(succ(W)), Z = succ(W)}
```

Unifikator μ ist MGU wenn alle anderen Unifikatoren σ aus μ entstehen, indem man Terme für seine Variablen einsetzt.

Formal:
Für alle anderen Unifikatoren σ muss eine Substitution τ existieren mit $\sigma = \tau \circ \mu$.

Berechnung des MGU

Eingabe: s und t
Ausgabe: MGU oder Fehlschlag

1. Falls s und t gleiche Variablen sind, dann $\sigma = \{\ \}$.

2. Falls s Variable ist und t enthält s nicht, dann $\sigma = \{s = t\}$.

3. Falls t Variable ist und s enthält t nicht, dann $\sigma = \{t = s\}$.

4. Falls $s = f(s_1,\ldots,s_n)$ und $t = f(t_1,\ldots,t_n)$, dann:

   a) Sei $\sigma_1 = MGU(s_1,t_1)$.

   b) Für alle $2 \leq i \leq n$ sei

      $\sigma_i = MGU(\ \sigma_{i-1}(\ \ldots\ (\sigma_1(s_i))\ \ldots\ ),\ \sigma_{i-1}(\ \ldots\ (\sigma_1(t_i))\ \ldots\ )\ )$.

   c) Falls alle σ_i existieren, dann $\sigma = \sigma_n \circ \ldots \circ \sigma_1$.

5. Sonst sind s und t nicht unifizierbar.

Beweisverfahren von Prolog: Resolution

Algorithmus SOLVE

Eingabe: Anfrage ?- $G_1, \ldots, G_m$
Ausgabe: Antwortsubstitution σ oder Fehlschlag

1. Wenn m = 0, dann terminiere mit $\sigma = \{\ \}$.

2. Sonst: Suche nach der nächsten Programmklausel

   bei n=0: Faktum → H :- $B_1, \ldots, B_n$. ← Var. in Prog-Klausel werden umbenannt, sodass sie frisch sind.

   so dass G_1 und H unifizierbar (mit MGU μ) sind.
   Gibt es keine, dann terminiere mit Fehlschlag.

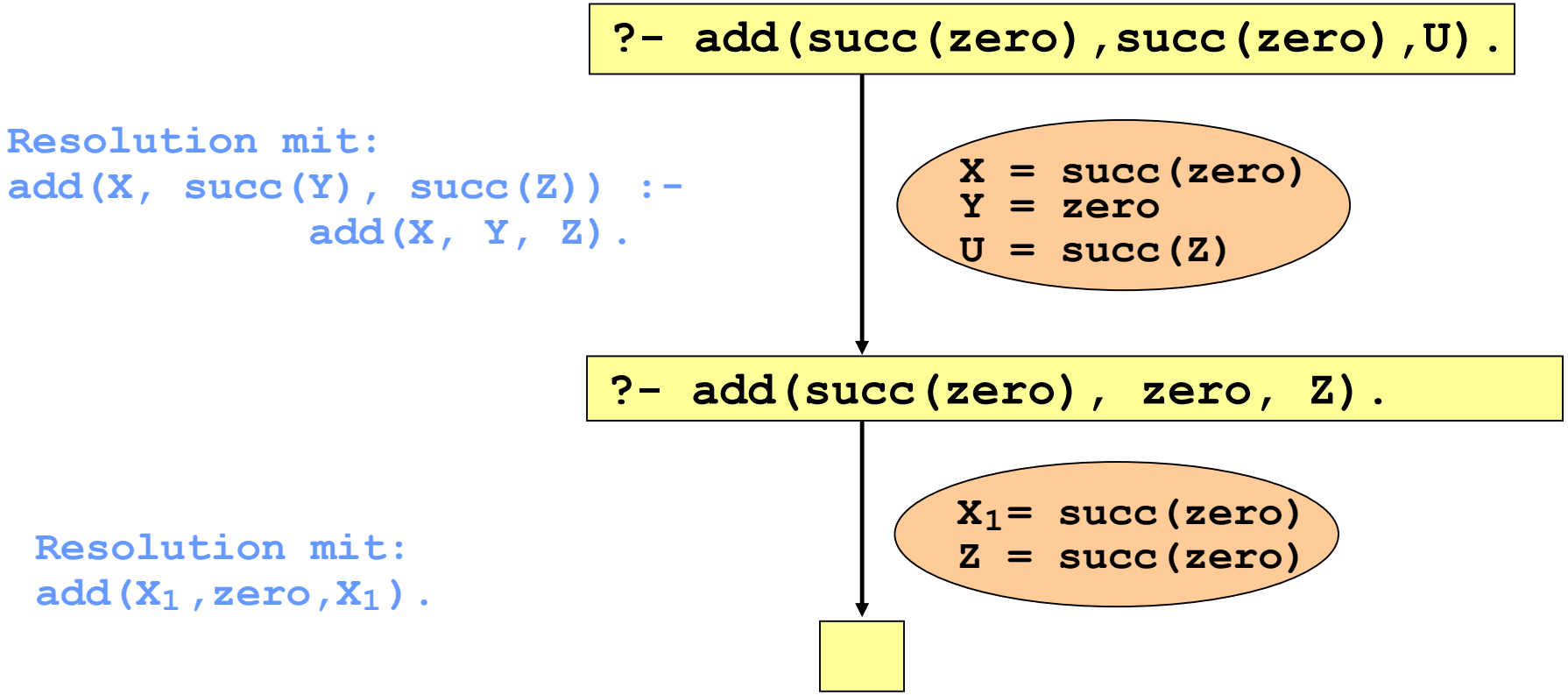
3. Rufe SOLVE mit der folgenden Anfrage auf:

   ?- $\mu(B_1), \ldots, \mu(B_n), \mu(G_2), \ldots, \mu(G_m)$.

4. Falls dieser Aufruf Antwortsubst. τ berechnet,
   dann: Terminiere mit $\sigma = \tau \circ \mu$. ← nur der Teil, der Var. in $G_2, \ldots, G_m$ betrifft
   sonst: Gehe zurück zu Schritt 2.

Beweisbaum

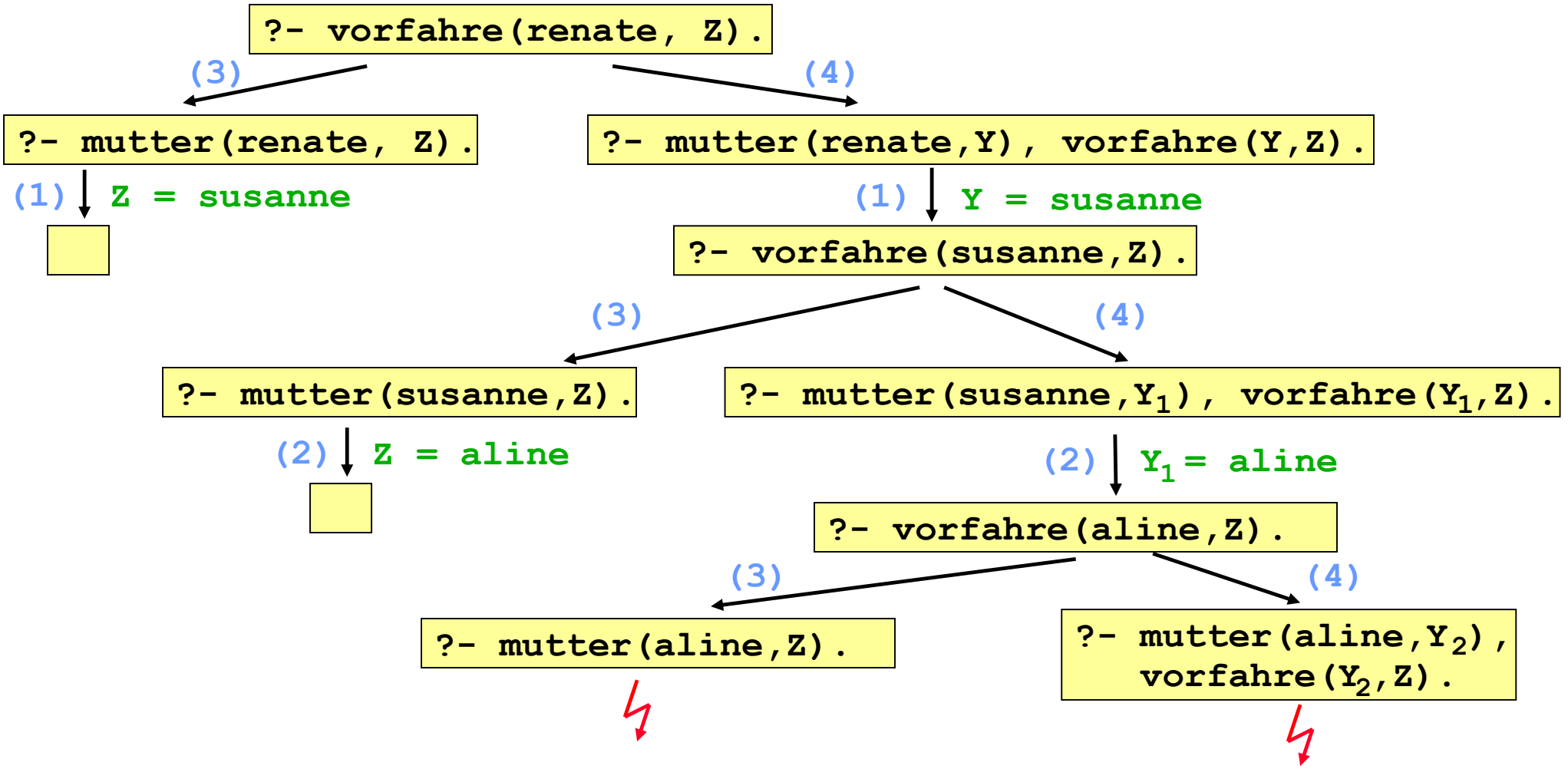
```
add(X, zero, X).
add(X, succ(Y), succ(Z)) :- add(X, Y, Z).
```



Antwortsubstitution: {U = succ(succ(zero))}

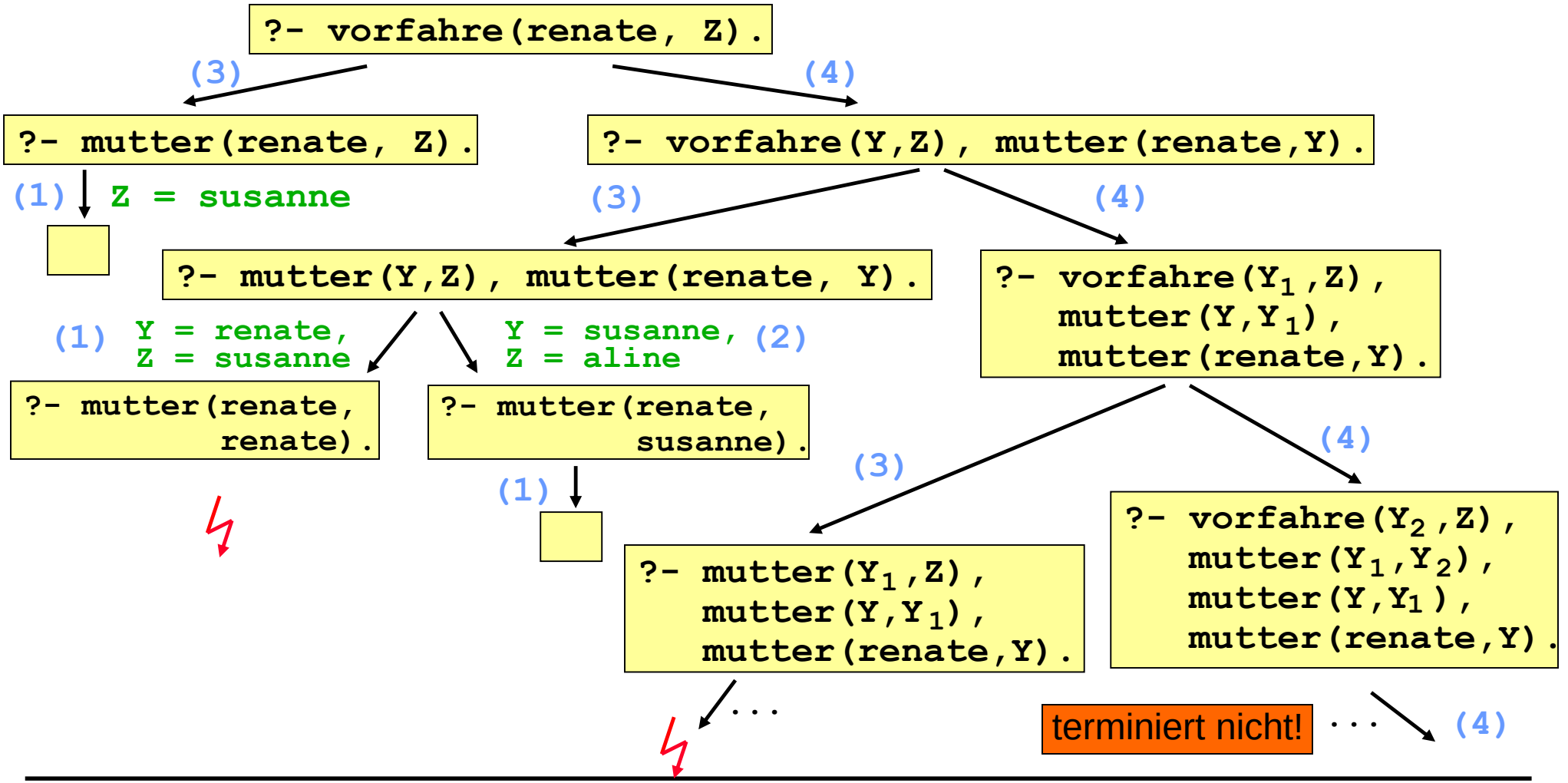
Beweisbaum

```
(1) mutter(renate,susanne).   (3) vorfahre(V,X) :- mutter(V,X).
(2) mutter(susanne,aline).    (4) vorfahre(V,X) :- mutter(V,Y),
                                                   vorfahre(Y,X).
```



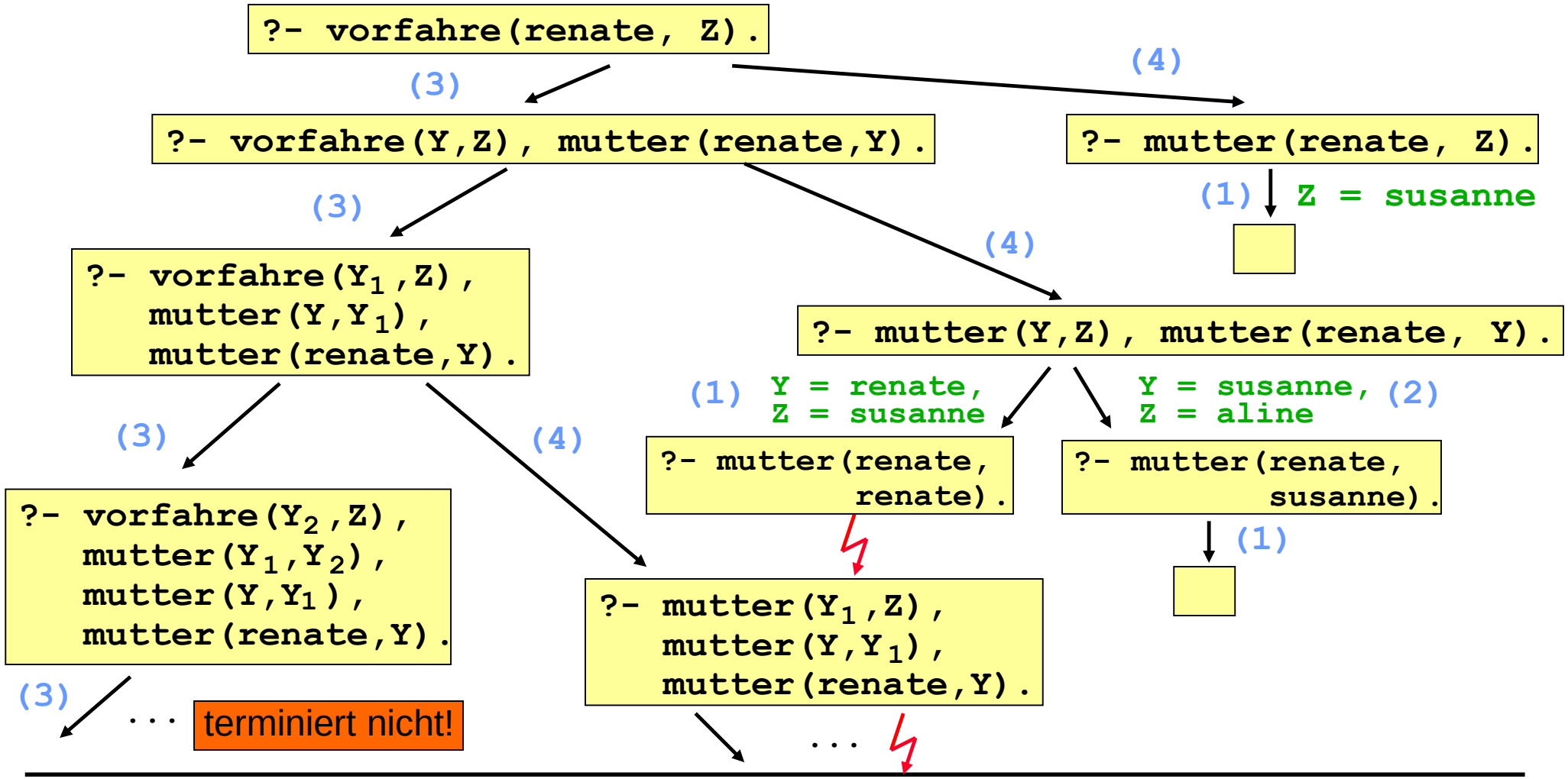
Beweisbaum

```
(1) mutter(renate,susanne).   (3) vorfahre(V,X) :- mutter(V,X).
(2) mutter(susanne,aline).    (4) vorfahre(V,X) :- vorfahre(Y,X),
                                                   mutter(V,Y).
```



Beweisbaum

```
(1) mutter(renate,susanne).(3) vorfahre(V,X) :- vorfahre(Y,X),
                                                mutter(V,Y).
(2) mutter(susanne,aline).  (4) vorfahre(V,X) :- mutter(V,X).
```



Gleichheit in Prolog: =

```
gleich(X,X).
```

Ist in Prolog vordefiniert, heißt dort "="

```
?- succ(X) = succ(succ(Y)).
X = succ(Y)
```

```
?- .(a,L) = [X,b|K].
X = a, L = [b|K]
```

?- s = t berechnet MGU von s und t

```
mem(X, [Y|_]) :- X = Y.
mem(X, [_|L]) :- mem(X,L).
```

```
?- mem(X, [1,2,3]).
X = 1 ;
X = 2 ;
X = 3
```

```
mem(X, [X|_]).
mem(X, [_|L]) :- mem(X,L).
```

Gleichheit in Prolog: `is`

```
?- X = 2 + 5.
X = 2 + 5
```

```
?- 7 = 2 + 5.
false
```

"=" berechnet nur syntaktische Unifikation

Vordefiniertes Prädikat **"is":** **`s is t`**

- **`t`** muss vollständig instantiierter arithmetischer Ausdruck sein
- rechne erst **`t`** aus
- dann wird das Ergebnis der Auswertung mit **`s`** unifiziert

```
?- X is 2 + 5.
X = 7
```

```
?- 7 is 2 + 5.
true
```

```
?- 2 + 5 is 7.
false
```

```
len([], zero).
len([_ | Rest], succ(N)) :- len(Rest, N).
```

```
len([], 0).
len([_ | Rest], M) :- len(Rest, N), M is N + 1.
```

Vorlesung "Programmierung"

- **Inhalt der Vorlesung**
  - Was ist ein Programm?
  - Was sind grundlegende Programmierkonzepte?
  - Wie konstruiert (entwickelt) man ein Programm?
  - Welche Programmiertechniken und -paradigmen gibt es?

- **Teil I: Einleitung und Grundbegriffe**
- **Teil II: Imperative & objektorientierte Programmierung (*Java*)**
- **Teil III: Funktionale Programmierung (*Haskell*)**
- **Teil IV: Logische Programmierung (*Prolog*)**