

# ***Künstliche Intelligenz***

Sebastian Iwanowski  
FH Wedel

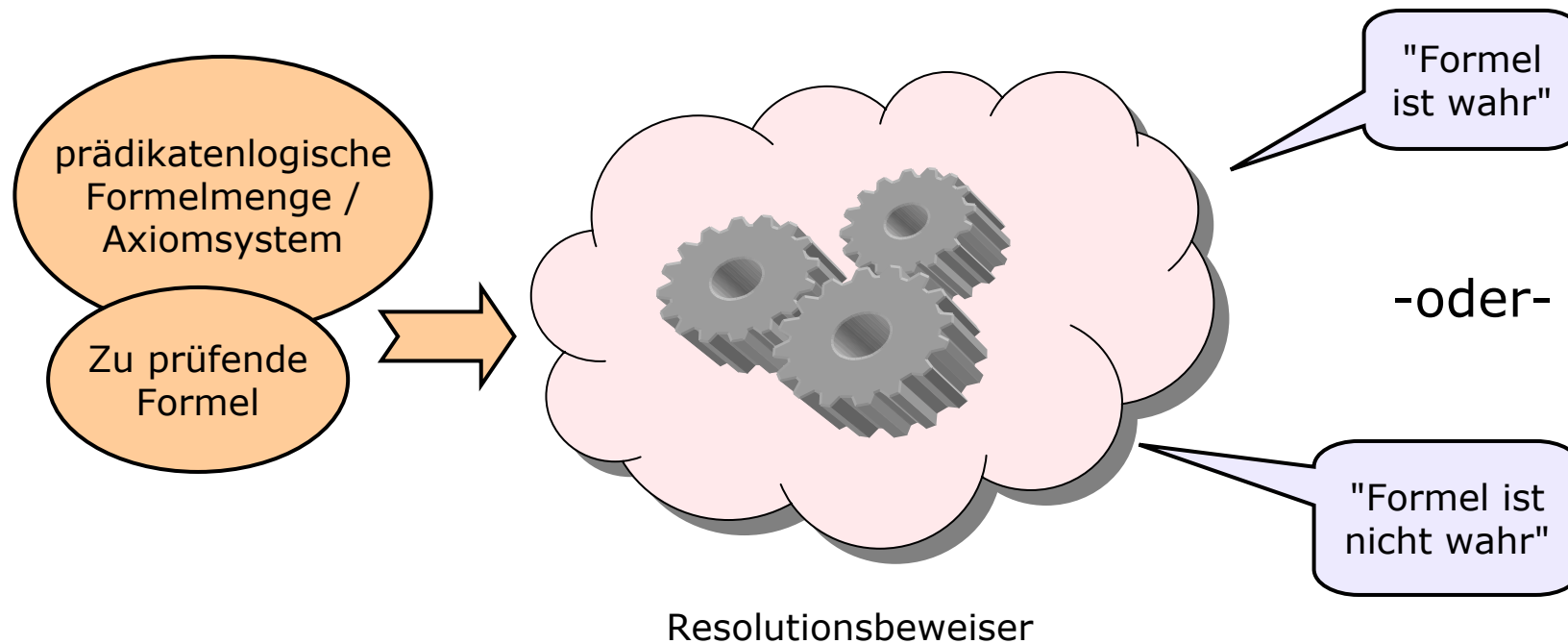
**Kap. 2:**  
KI-Logik

2.3: Funktionsweise eines Resolutionsbeweisers

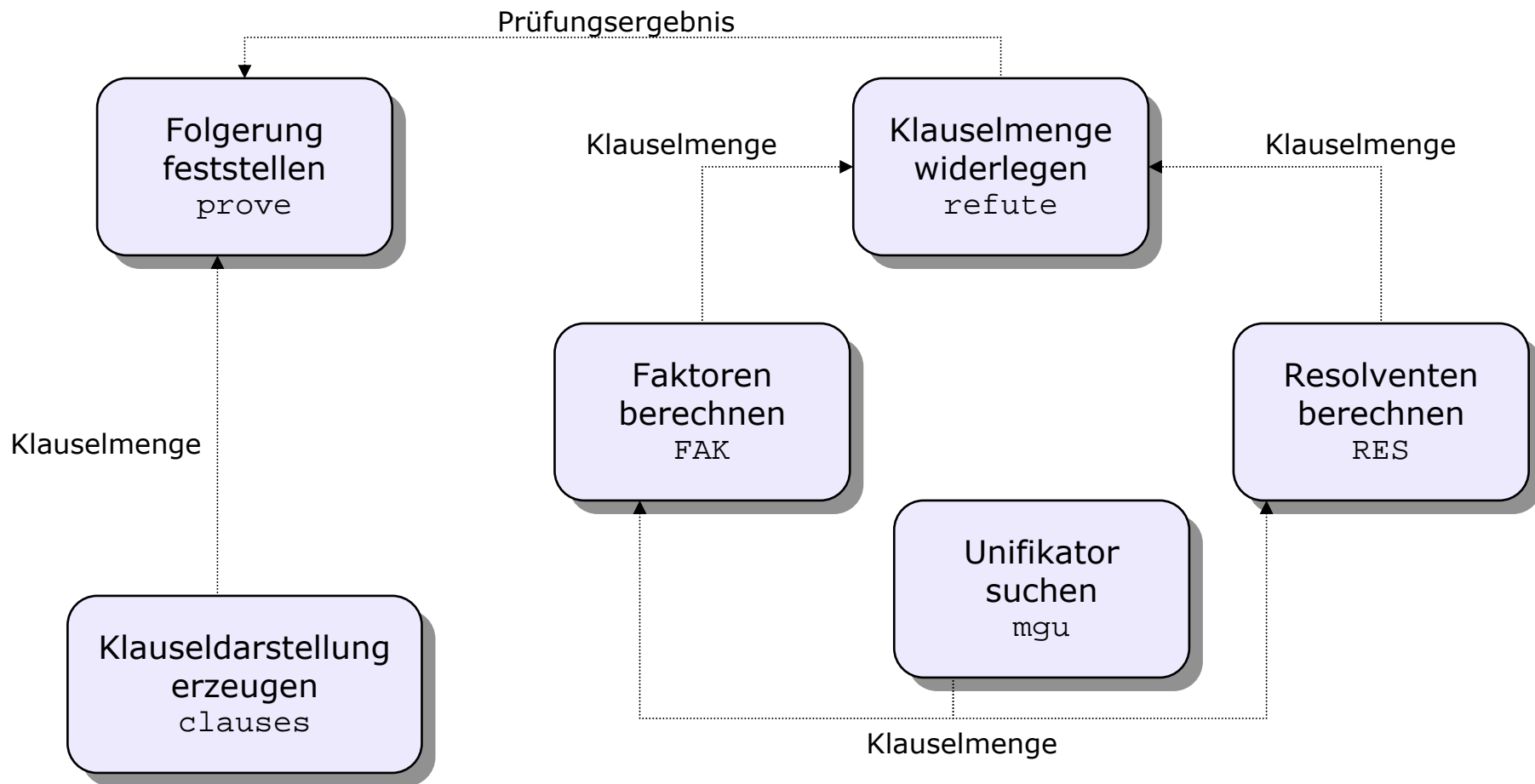
*Die Folien dieser Vorlesung beruhen im Wesentlichen  
auf Folien des Seminarvortrags von Daniel Dittmann, gehalten im SS 2005.*

# Ziel des Resolutionsbeweisers

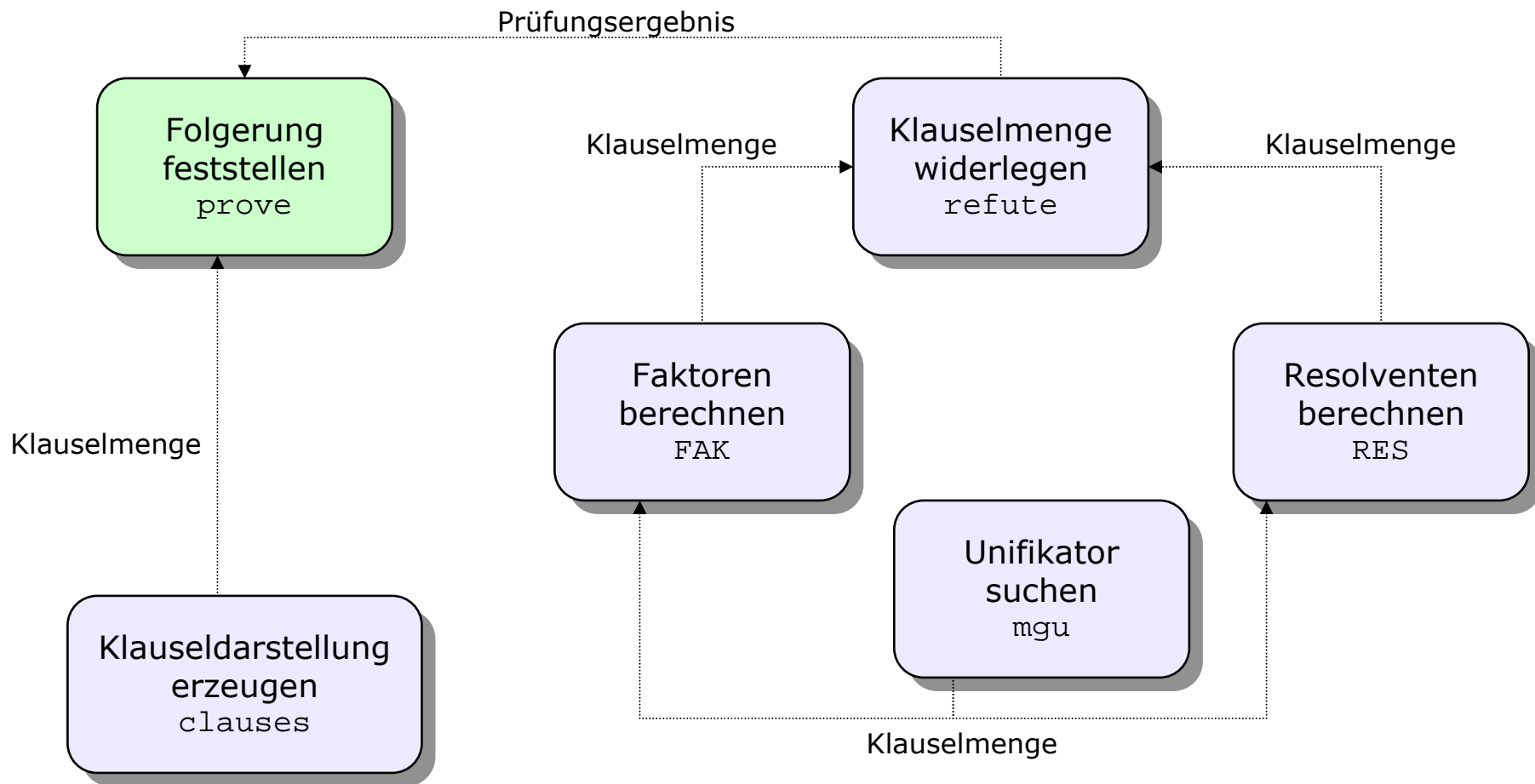
- Entwicklung eines Softwaresystems zur automatischen Beweisführung mit Hilfe des Resolutionskalküls



# Modularisierung und Datenflüsse im Resolutionsbeweiser

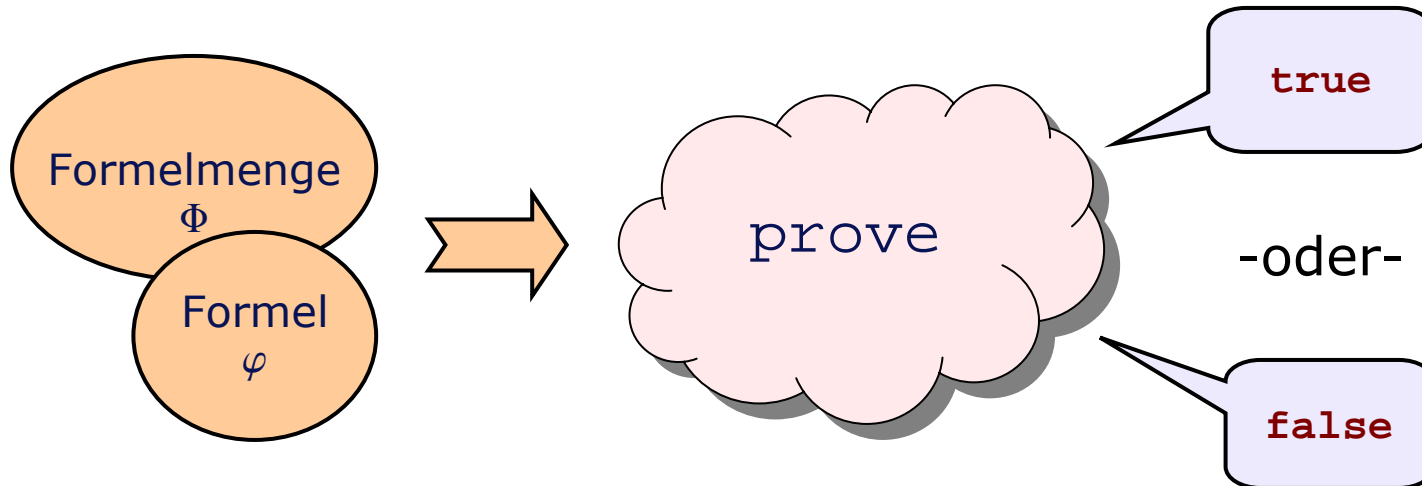


# Modularisierung und Datenflüsse im Resolutionsbeweiser



# prove: Anforderungen und Funktionalität

- Schnittstelle des Systems
- Darstellung der Inputgrößen in Prädikatenlogik 1. Stufe (Nutzung des Resolutionskalküls transparent)



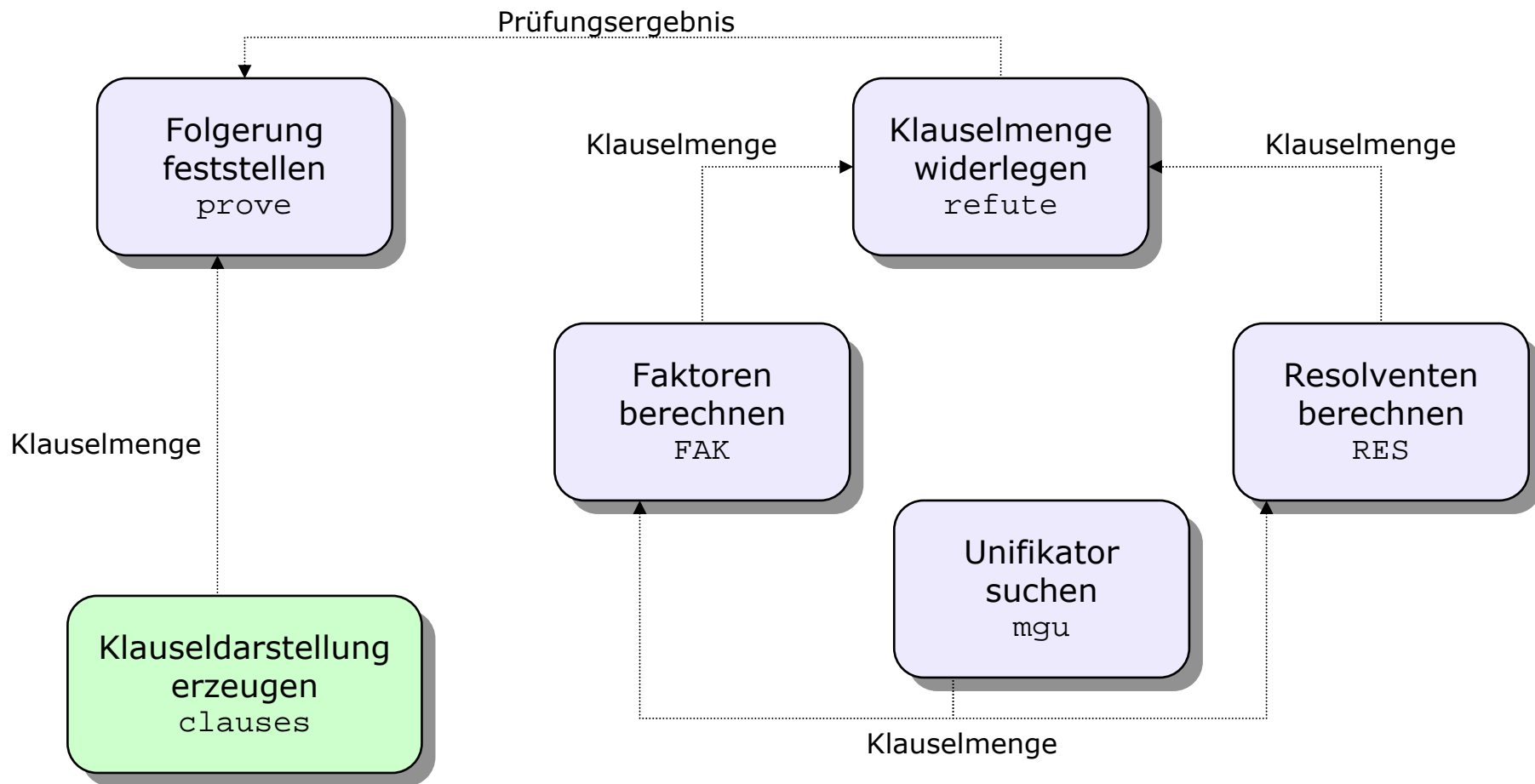
- Aufgabe:  
Suche die Belegungen für die freien Variablen, unter denen die Folgerung true ist.

# prove: Funktionsweise

1. Umformung der Inputs in Klauseldarstellung
2. Negation der zu überprüfenden Formel mit Axiomsystem vereinigen
3. Versuch, neu gebildete Klauselmenge zu widerlegen

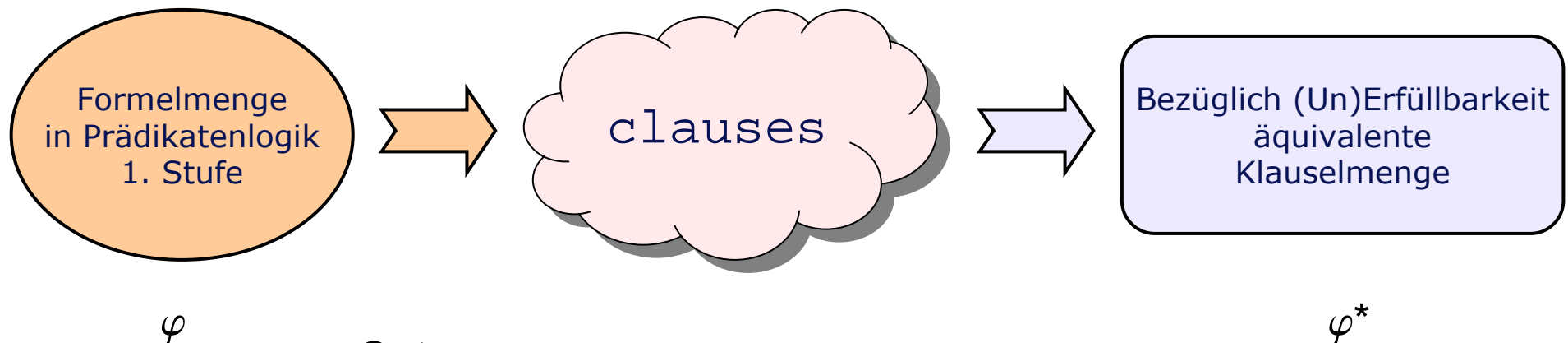
```
function prove( $\Phi \in 2^{\mathcal{F}}$ ,  $\varphi \in \mathcal{F}$ ) : bool  
   $\mathcal{S}_{\mathcal{A}} := \text{clauses}(\Phi)$   
   $\mathcal{S}_{\mathcal{T}} := \text{clauses}(\{\neg\varphi\})$   
   $r := \text{refute}(\mathcal{S}_{\mathcal{A}} \cup \mathcal{S}_{\mathcal{T}})$   
  return( $r$ )  
end
```

# Modularisierung und Datenflüsse im Resolutionsbeweiser



# clauses: Anforderungen und Funktionalität

- Umformung prädikatenlogischer Formeln in Klauseln
- Grundlage für Nutzbarkeit des Resolutionskalküls durch `prove`
- Äquivalente Darstellung nicht möglich, daher Beschränkung auf (Un)Erfüllbarkeit



**Satz:**

$\varphi$  ist erfüllbar  $\Leftrightarrow \varphi^*$  ist erfüllbar  
 $\varphi$  ist unerfüllbar  $\Leftrightarrow \varphi^*$  ist unerfüllbar



# clauses: Funktionsweise

## Definition:

Eine Formel  $\varphi$  ist in Pränexer Normalform (PNF):

$\varphi = \Delta x_1 \Delta x_2 \dots \Delta x_k \psi$  mit  $\Delta$  ist Quantor und  $\psi$  ist quantorenfreie Formel.

## Definition:

Sei  $\varphi = \forall x_1 \forall x_2 \dots \forall x_k \exists y \psi$  eine prädikatenlogische Formel.

Dann ist  $\varphi^* = \forall x_1 \forall x_2 \dots \forall x_k \psi [y/f(x_1, \dots, x_k)]$  die Formel, in der jedes Vorkommen von  $y$  durch  $f(x_1, \dots, x_k)$  ersetzt wurde.

$\varphi^*$  heißt Skolemisierung von  $\varphi$  und  $f(x_1, \dots, x_k)$  die zugehörige Skolemfunktion.

### 1. Umformung in PNF

z.B.  $\forall x P(x,y) \wedge \exists y: Q(y)$  wird zu  $\forall x \exists z P(x,y) \wedge Q(z)$

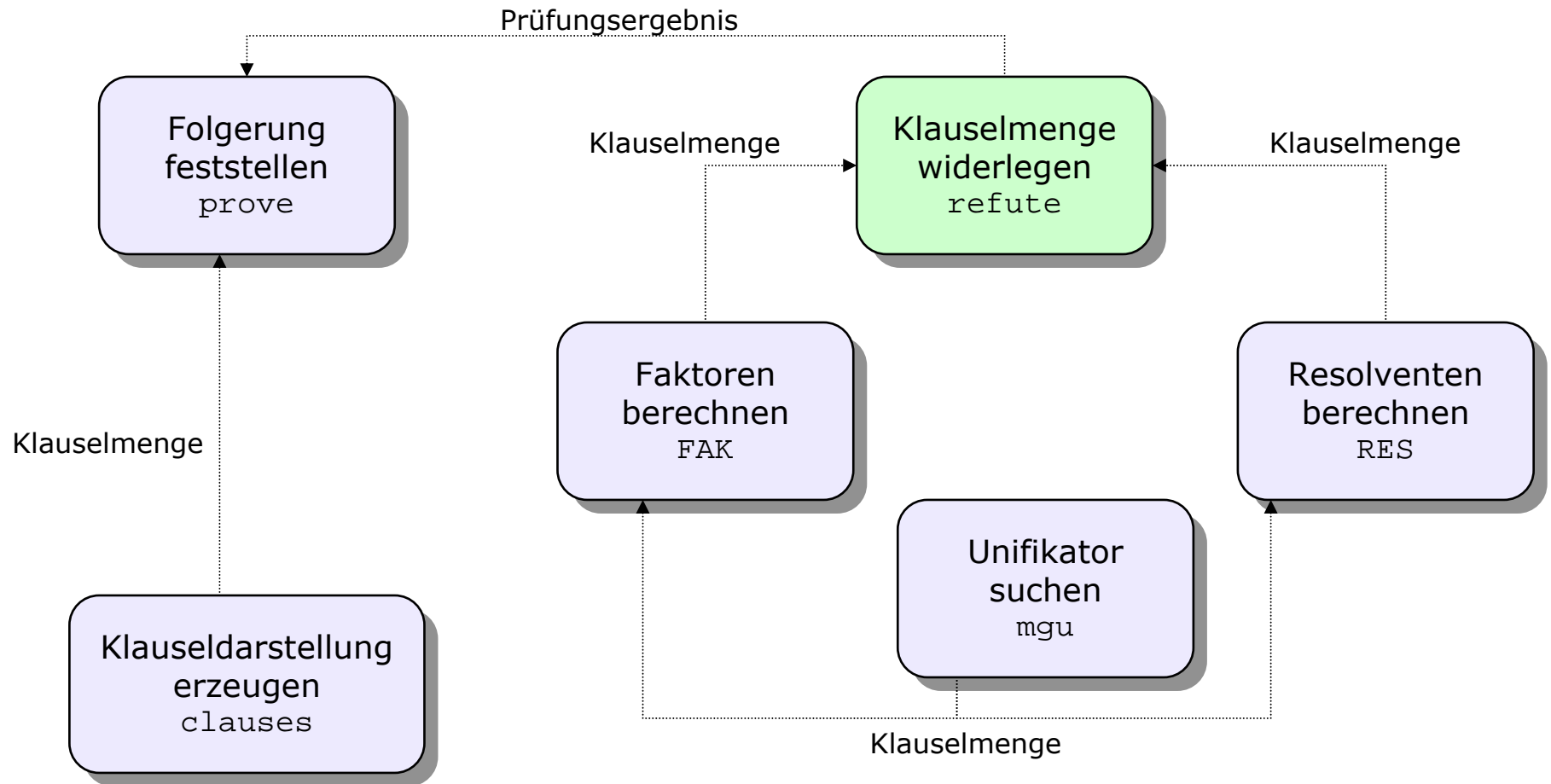
### 2. Eliminierung von Existenzquantoren durch Skolemisierung

z.B.  $\forall x \exists y: P(x,y)$  wird zu  $\forall x P(x, f(x))$

### 3. Umformung in KNF in Mengendarstellung ohne Allquantoren

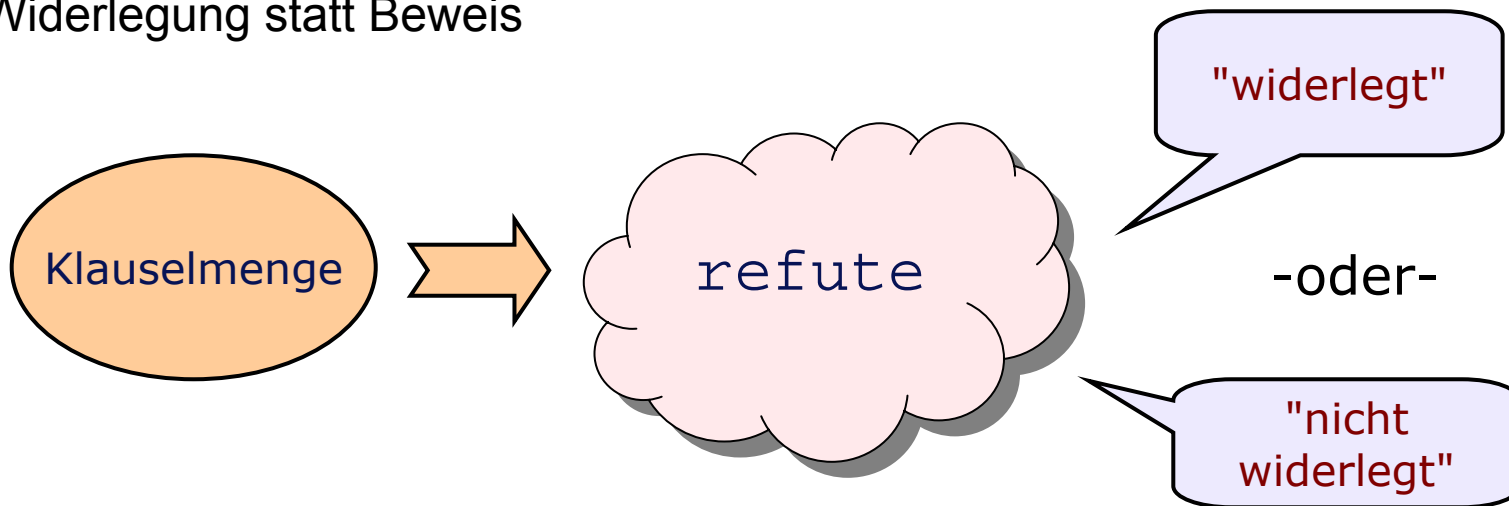
z.B.  $\forall x, y: P(x) \vee Q(x)$  wird zu  $\{P(x), Q(x)\}$

# Modularisierung und Datenflüsse im Resolutionsbeweiser



# refute: Anforderungen und Funktionalität

- `prove` als Überbau und Schnittstelle (koordinierende Instanz)
- `refute` als Systemkern (austauschbar)
- Implementation des Resolutionskalküls
- Widerlegung statt Beweis



- Aufgabe:  
Suche die Belegungen für die Variablen, unter denen die Klauselmenge widerlegt ist.

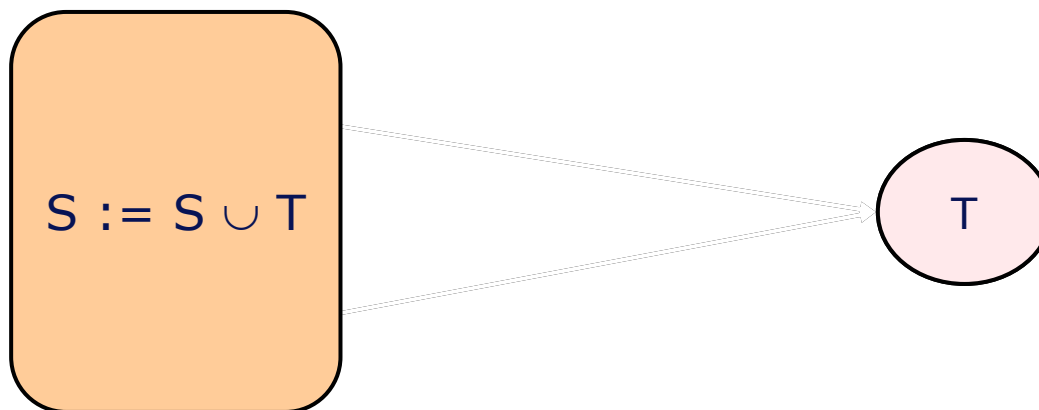
# refute: Funktionsweise

Datenstrukturen:

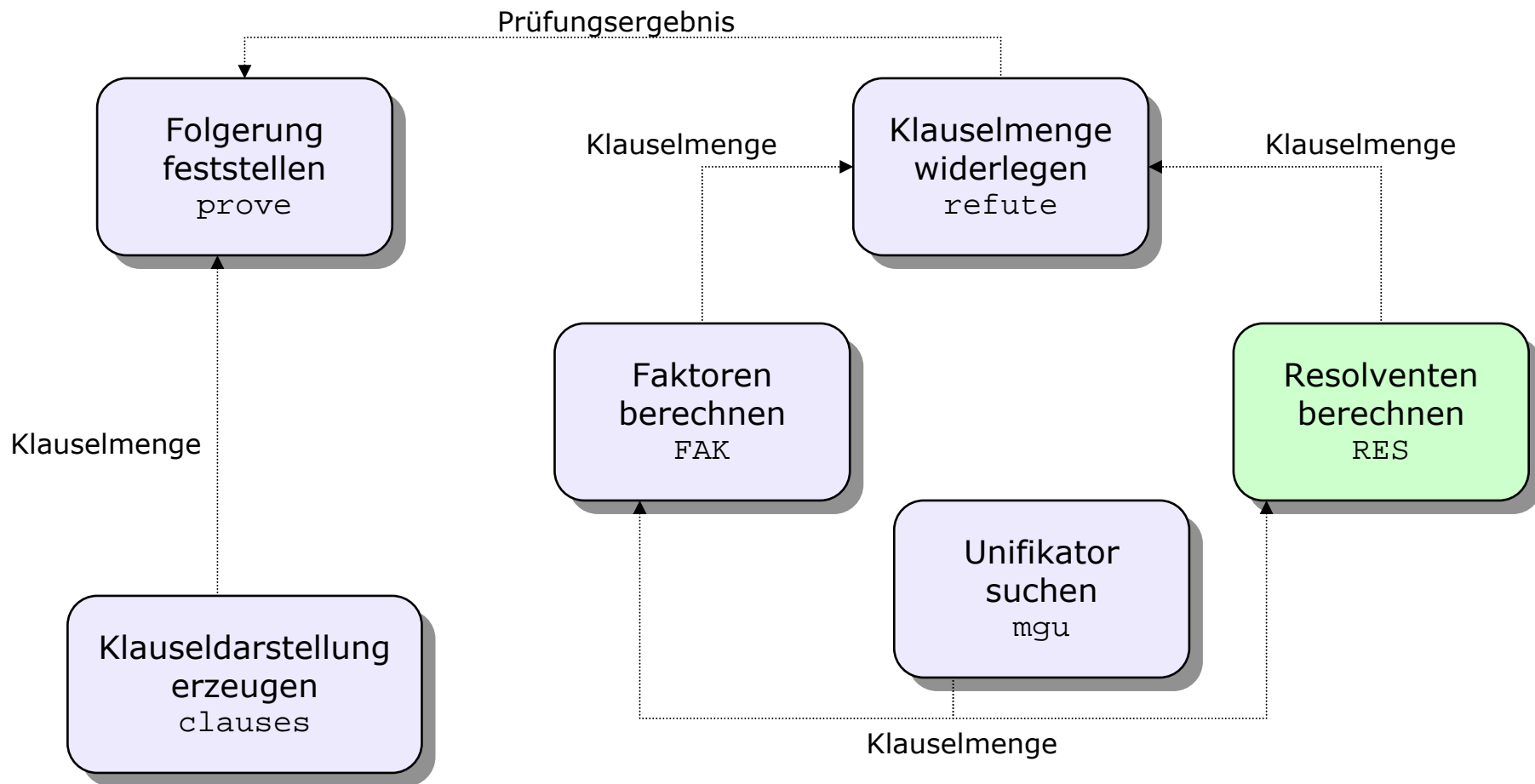
- Akkumulator S für sämtliche Klauseln der bisherigen Herleitung
- Behälter T für Klauseln der letzten Generation

Vorgehen:

- Zuletzt erzeugte Klauseln (T) mit bisherigen Klauseln resolvieren
- S um T ergänzen, neu resolvierte Klauseln werden neues T
- Stopp wenn leere Klausel hergeleitet oder keine Resolutionsmöglichkeiten mehr

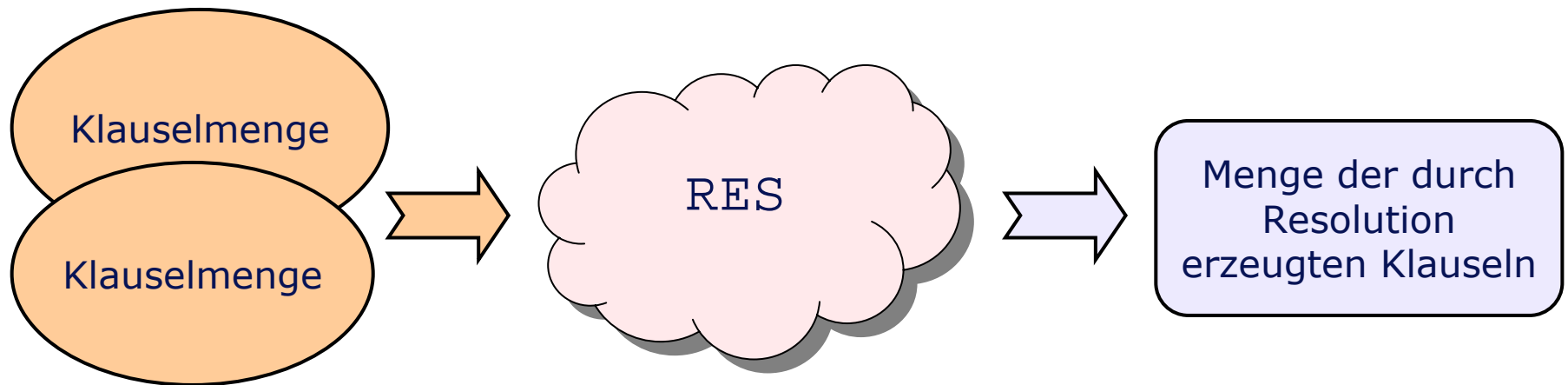


# Modularisierung und Datenflüsse im Resolutionsbeweiser



# RES: Anforderungen und Funktionalität

- Berechnung aller Resolventen, die zwischen den Literalen zweier Klauselmengen möglich sind
- Stopp falls  $\perp$  hergeleitet



# RES: Resolution in der Prädikatenlogik

## Definition:

Seien  $C_1$ , und  $C_2$  Klauseln,  $L_1$  Literal der Klausel  $C_1$ ,  $L_2$  Literal der Klausel  $C_2$  und  $\sigma$  eine Variablensubstitution.

Nach Durchführung von  $\sigma$  seien die Literale  $L_1$  und  $L_2$  komplementär

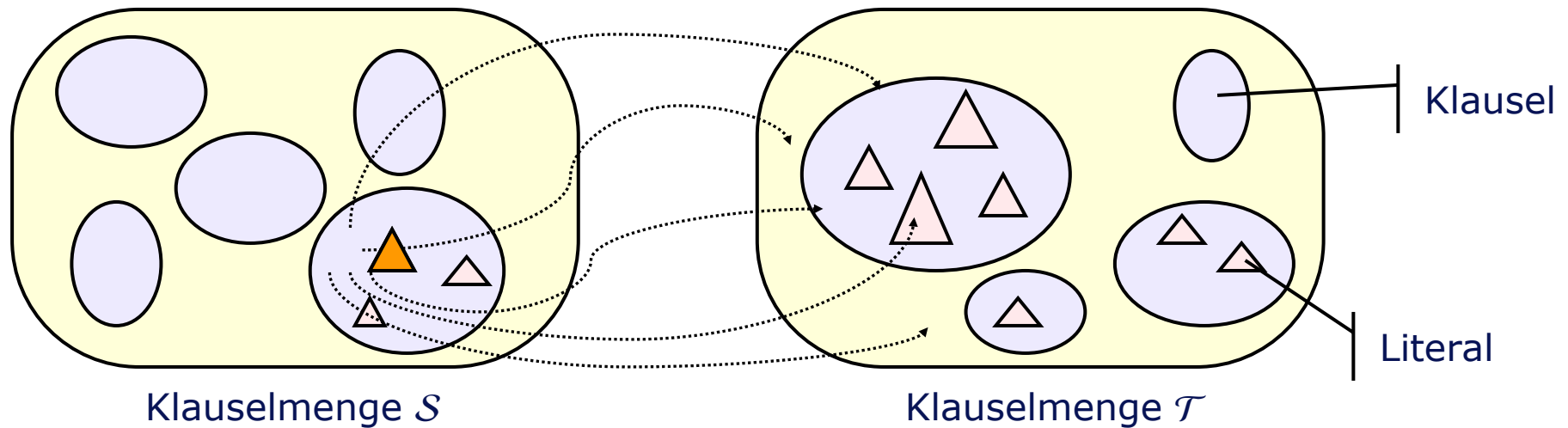
Dann ist **Res( $C_1, L_1, C_2, L_2, \sigma$ )** die Resolvente nach Durchführung der Ersetzung  $\sigma$

Bsp.:

$$\begin{array}{lll} C_1 = \{P(x), \neg Q(f(y))\} & C_2 = \{Q(z), R(g(z))\} & \\ L_1 = \neg Q(f(y)) & L_2 = Q(z) & \sigma = [z/f(y)] \\ \text{Res}(C_1, L_1, C_2, L_2, \sigma) = \{P(x), R(g(f(y)))\} & & \end{array}$$

# RES: Grundsätzliche Funktionsweise

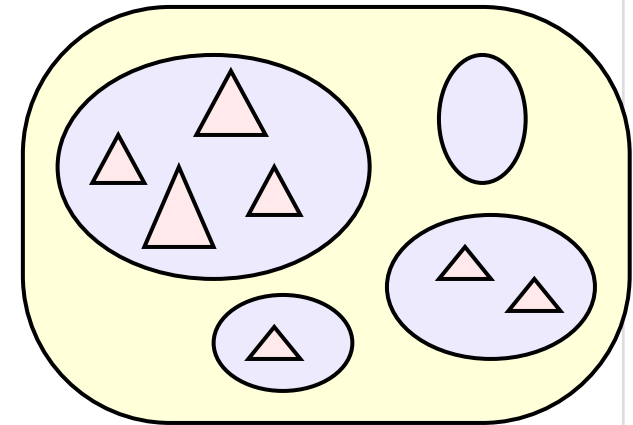
- Jede Klausel der einen wird mit jeder Klausel der anderen Menge auf Resolvierbarkeit getestet.
- Jedes Literal der einen wird mit jedem Literal der anderen Klausel auf Resolvierbarkeit getestet.
- Ggf. Resolventenbildung



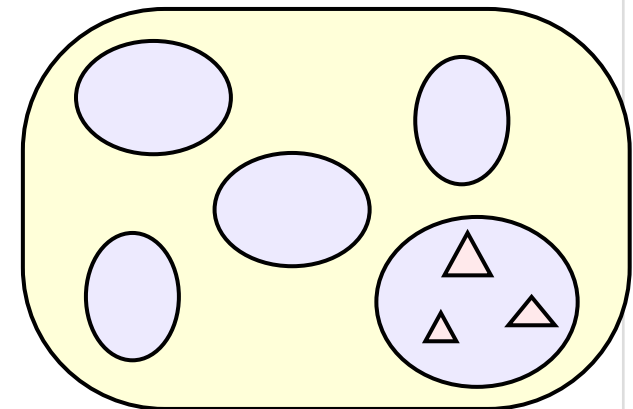


# RES: Implementierung im Detail

```
function RES( $S, \mathcal{T} \in 2^{\mathcal{L}}$ ) :  $2^{\mathcal{L}}$   
   $\mathcal{R}^* := \emptyset$   
   $\mathcal{T}' := \mathcal{T}$   
  for all  $C \in S$  do  
     $\mathcal{T}' := \mathcal{T} \setminus \{C\}$   
    for all  $L \in C$  do  
      for all  $D \in \mathcal{T}'$  do  
        for all  $\mathcal{K} \in D$  do  
          if  $L$  komplementär  $\mathcal{K}$  then  
             $\sigma := \text{mgu}(|L|, |\mathcal{K}|)$   
            if  $\sigma \neq \text{failed}$  then  
               $R := \sigma(C - L) \cup \sigma(D - \mathcal{K})$   
               $\mathcal{R}^* := \mathcal{R}^* \cup \{R\}$   
              if  $R = \perp$  then return  $\mathcal{R}^*$  fi  
          fi fi done done done done  
        return  $\mathcal{R}^*$   
      end  
    end  
  end
```

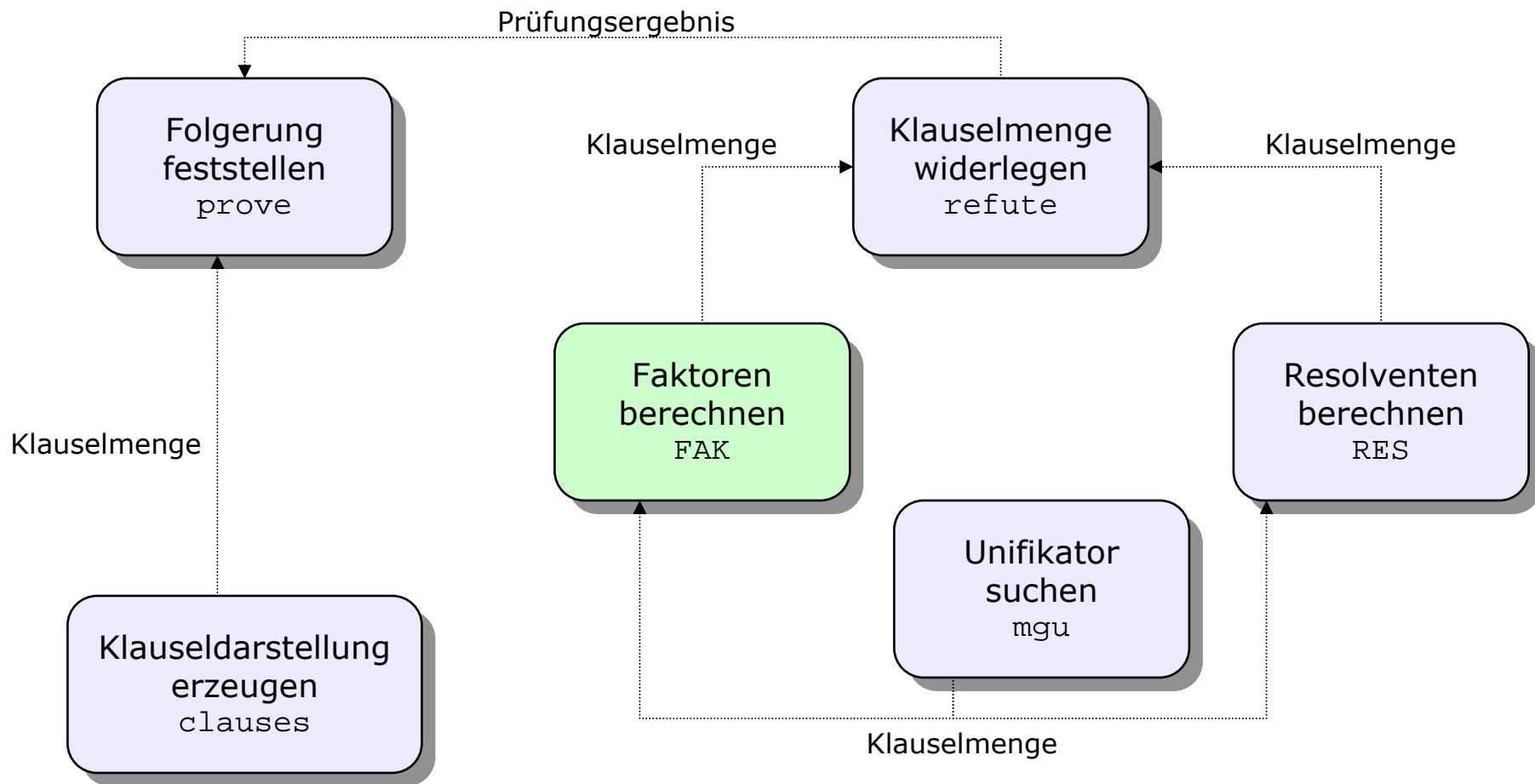


Klauselmenge  $\mathcal{T}$



Klauselmenge  $S$

# Modularisierung und Datenflüsse im Resolutionsbeweiser



# FAK: Motivation

## Russellsche Antinomie:

„Der Barbier rasiert eine Person genau dann, wenn sie sich nicht selbst rasiert.“

$\forall x [\text{Rasiert}(\text{barbier}, x) \Leftrightarrow \neg \text{Rasiert}(x, x)]$

Enthält Widerspruch, der sich aber durch RES alleine nicht herleiten lässt:

$\{\neg \text{Rasiert}(\text{barbier}, x), \neg \text{Rasiert}(x, x)\} \quad \{\text{Rasiert}(y, y), \text{Rasiert}(\text{barbier}, y)\}$

$[y/\text{barbier}, x/\text{barbier}]$

RES leitet her:  $\{\neg \text{Rasiert}(\text{barbier}, \text{barbier}), \text{Rasiert}(\text{barbier}, \text{barbier})\}$

Wir wollen herleiten:  $\{\neg \text{Rasiert}(\text{barbier}, \text{barbier})\} \quad \{\text{Rasiert}(\text{barbier}, \text{barbier})\}$

*entnommen aus:*

*Studentenvorlesung von Björn Peemöller und Stefan Roggensack, gehalten im WS 2007/2008*

# FAK: Faktoren in der Prädikatenlogik

## Definition:

Sei  $C$  eine Klausel, und  $L$  eine Menge von Literalen dieser Klausel  $C$ .

$\sigma$  sei der mgu von  $L$  (kann leer sein).

Dann ist **Fak(C, L)** =  $\sigma(C)$  der Faktor von  $C$  bezüglich  $L$ .

Bsp.:  $C = \{P(x), P(f(y)), Q(z)\}$

$L = \{P(x), P(f(y))\}$

$\sigma = [x/f(y)]$

**Fak(C, L)** =  $\{P(f(y)), Q(z)\}$

# FAK: Anforderungen und Funktionalität

- Berechnung aller möglichen Faktoren einer Klauselmenge  $S$
- $S$  Teil des Outputs (auch Faktorisierung über  $\epsilon$  möglich)



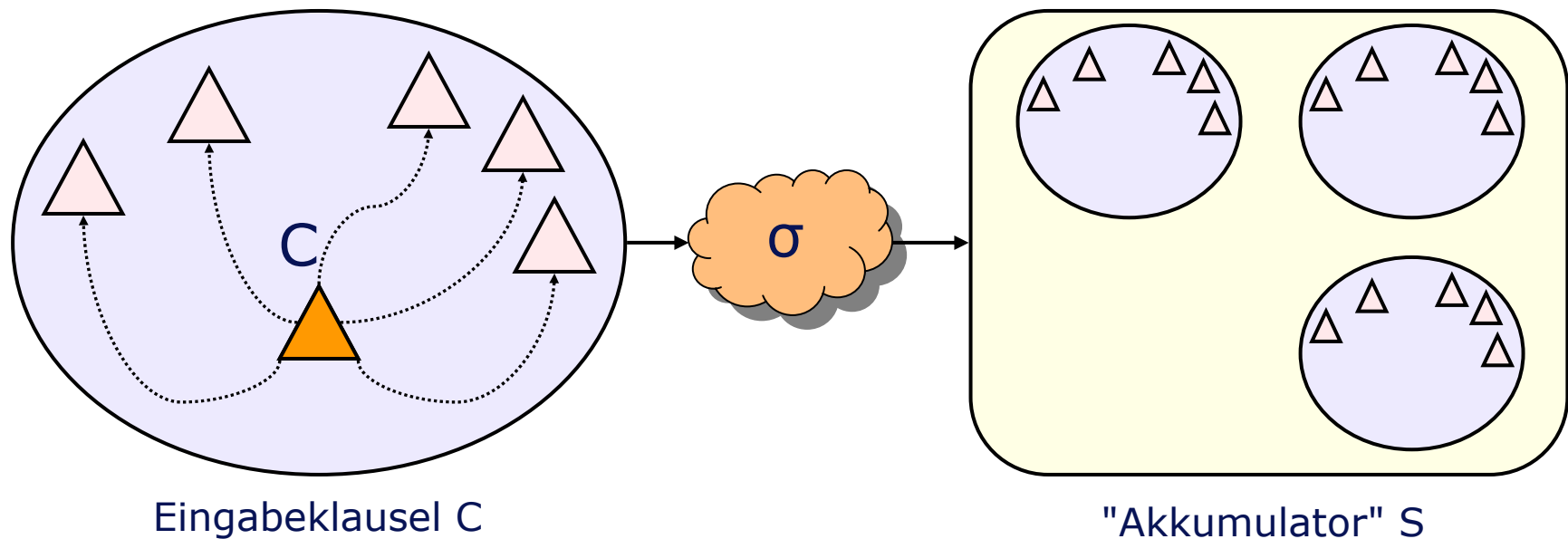
# FAK: Grundsätzliche Funktionsweise

- Faktoren haben nur eine Elternklausel
- Teilalgorithmus  $f_{ak}$  für eine Klausel
- Iteration über alle Elemente der Input-Menge

```
function FAK( $S \in 2^{\mathcal{L}}$ ) :  $2^{\mathcal{L}}$   
     $\mathcal{F} := \emptyset$   
    for all  $C \in S$  do  
         $\mathcal{F} := \mathcal{F} \cup f_{ak}(C)$   
    done  
    return  $\mathcal{F}$   
end
```

# FAK: Funktionsweise von Teilalgorithmus fak

- Jedes Literal mit jedem auf Unifizierbarkeit überprüfen
- Anwendung des gefundenen Unifikators auf gesamte Klausel (=Faktorisierung)
- Sammlung der Teilergebnisse



# FAK: Detailimplementierung von Teilalgorithmus fak

```
function fak( $C \in \mathcal{L}$ ) :  $2^{\mathcal{L}}$ 
   $S := \emptyset$ 
   $\mathcal{D} := C$ 
  for all  $\mathcal{L} \in C$  do
     $\mathcal{D} := \mathcal{D} - \mathcal{L}$ 
    for all  $\mathcal{K} \in \mathcal{D}$  do
      if not  $\mathcal{L}$  komplementär  $\mathcal{K}$  then
         $\sigma := \text{mgu}(|\mathcal{L}|, |\mathcal{K}|)$ 
        if  $\sigma \neq \text{failed}$  then
           $S := S \cup \text{fak}(\sigma(C))$ 
        fi
      fi
    done
  done
  return  $S \cup \{C\}$ 
end
```



# Zusammenfassung und Ausblick

## Resolutionsbeweiser:

Implementierung eines Systems aus sechs Algorithmen

- mgu
- RES
- FAK
- clauses
- refute
- prove

Teilalgorithmen in beispielhaften Grundversionen

## Ausblick:

Für praktische Implementierungen Optimierungen nötig

Ableitungsstrategien – sehr viele Möglichkeiten