



Implementierung eines Resolutionsbeweisers

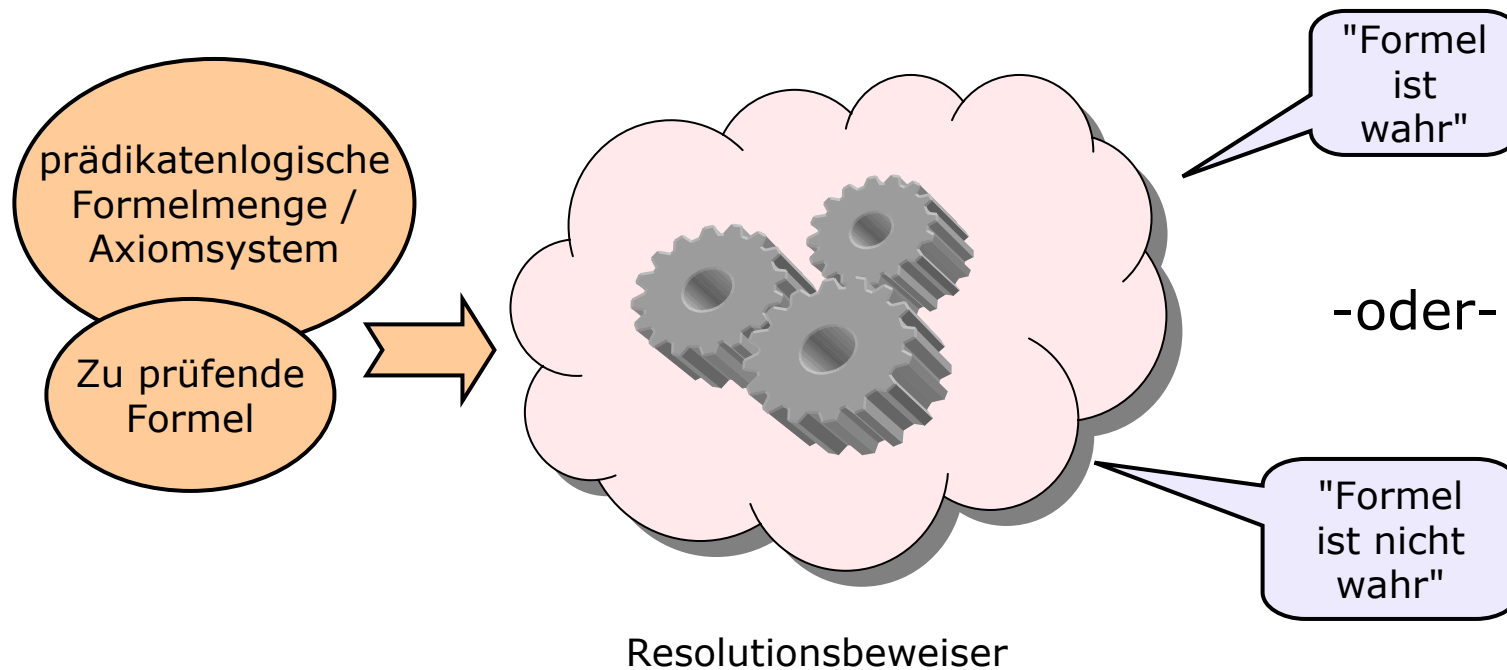
Informatikseminar SS 2005 – Künstliche Intelligenz  
Erarbeitet von: Daniel Dittmann

---

## 1.1 Motivation

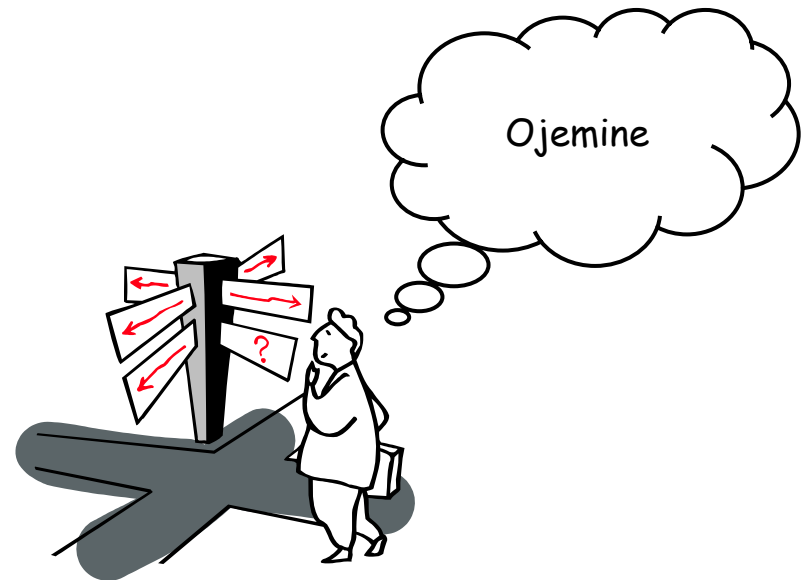
### Worum geht es?

- Entwicklung eines Softwaresystems zur automatischen Beweisführung mit Hilfe des Resolutionskalküls



## Agenda

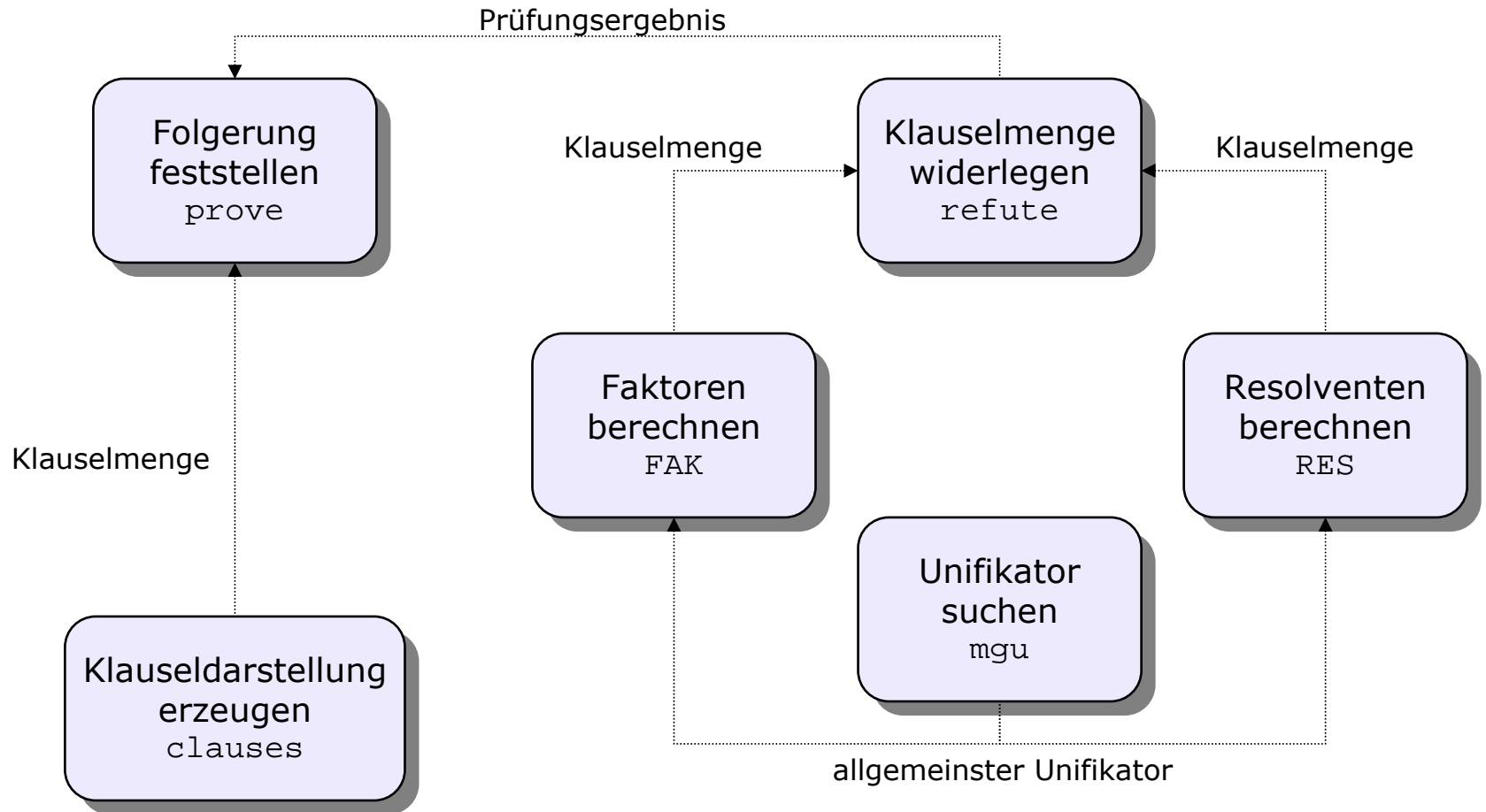
- 1 Einführung
- 2 Überblick über das System
- 3 Die Grundalgorithmen im Detail
- 4 Optimierungsansätze
- 5 Zusammenfassung und Ausblick



Zwischenfragen sind ausdrücklich erwünscht!

## 2 Überblick über das System

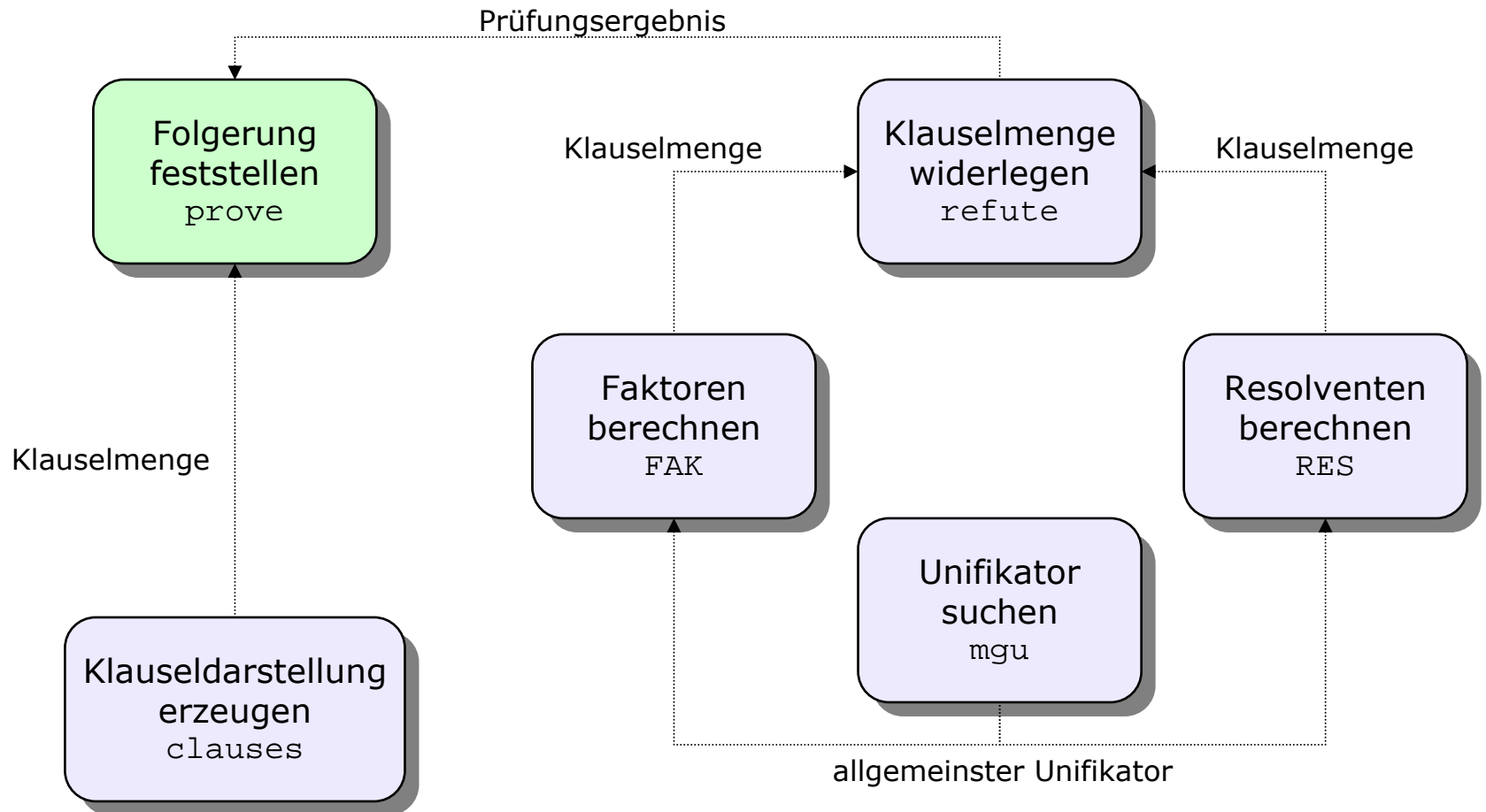
# Modularisierung und Datenflüsse



# 3 Die Grundalgorithmen im Detail

### 3.1 Der Beweisalgorithmus

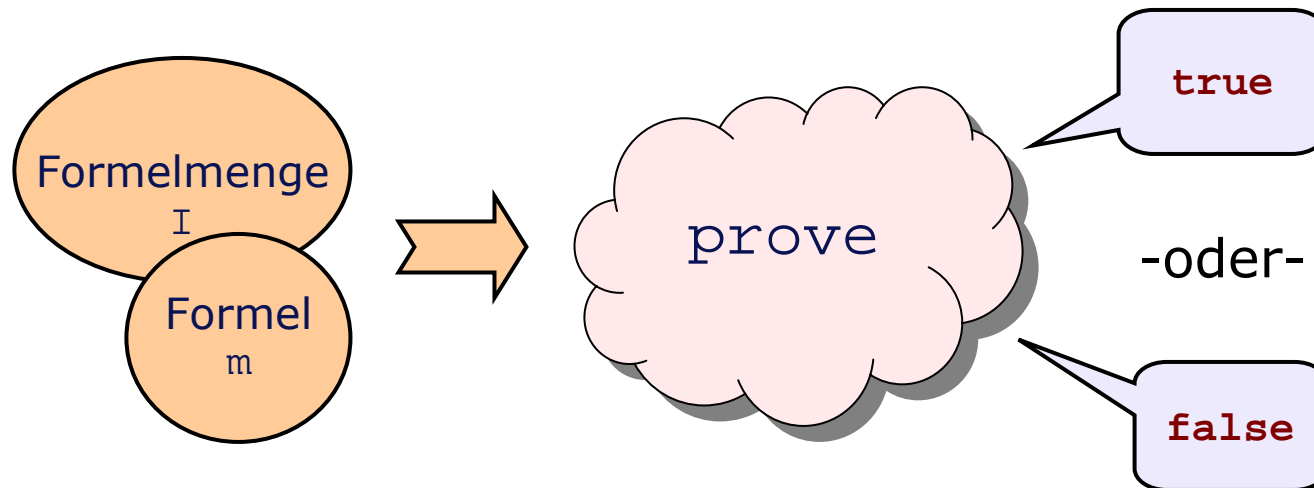
## Modularisierung und Datenflüsse



### 3.1 Der Beweisalgorithmus

## Anforderungen und Funktionalität

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



### 3.1 Der Beweisalgorithmus

---

#### Funktionsweise

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

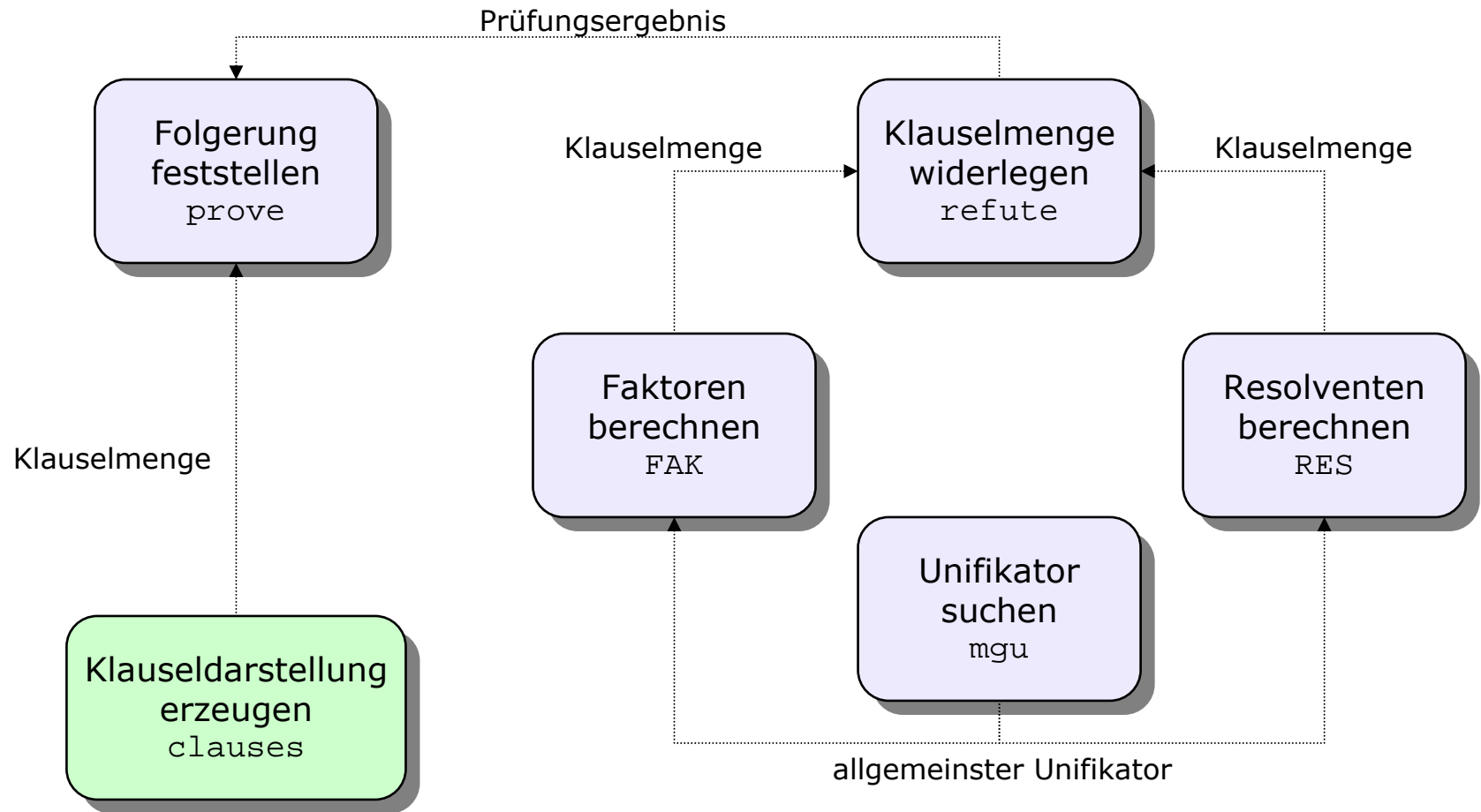
```

function prove ( I  $\mu$   $5^M$  / m  $\mu$  M ) : bool
  ZH =@ clauses ( I )
  ZL =@ clauses (  $\neg \exists m \phi$  )
  y =@ refute ( ZH + ZL )
  return ( y )
end

```

### 3.2 Berechnung von Klauselmengen

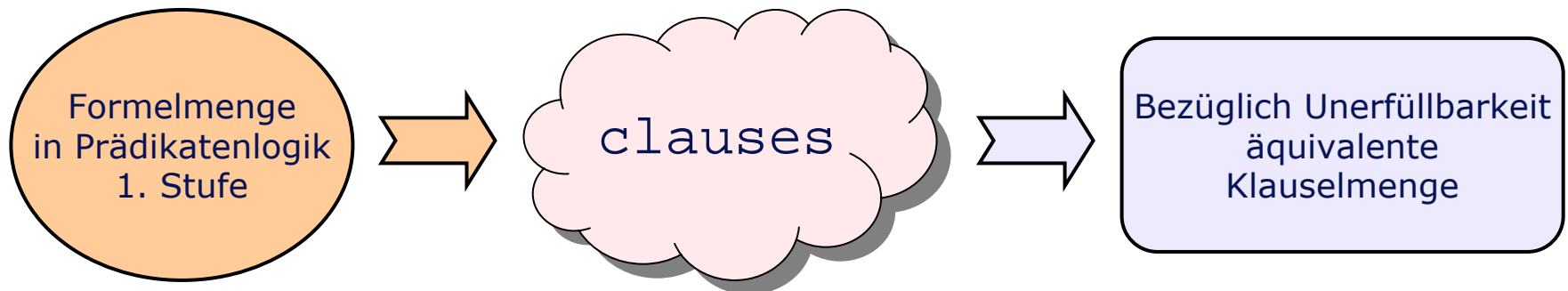
## Modularisierung und Datenflüsse



## 3.2 Berechnung von Klauselmengen

### Anforderungen und Funktionalität

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



## 3.2 Berechnung von Klauselmengen

---

### Funktionsweise

1. Umformung in Prenexe Normalform mit Matrix in KNF durch Ersetzungsregeln

z.B. de Morgan:  $\neg(\hat{m}_4 \wedge m_5) \equiv \neg\hat{m}_4 \vee \neg m_5$

2. Eliminierung von Existenzquantoren durch Skolemisierung

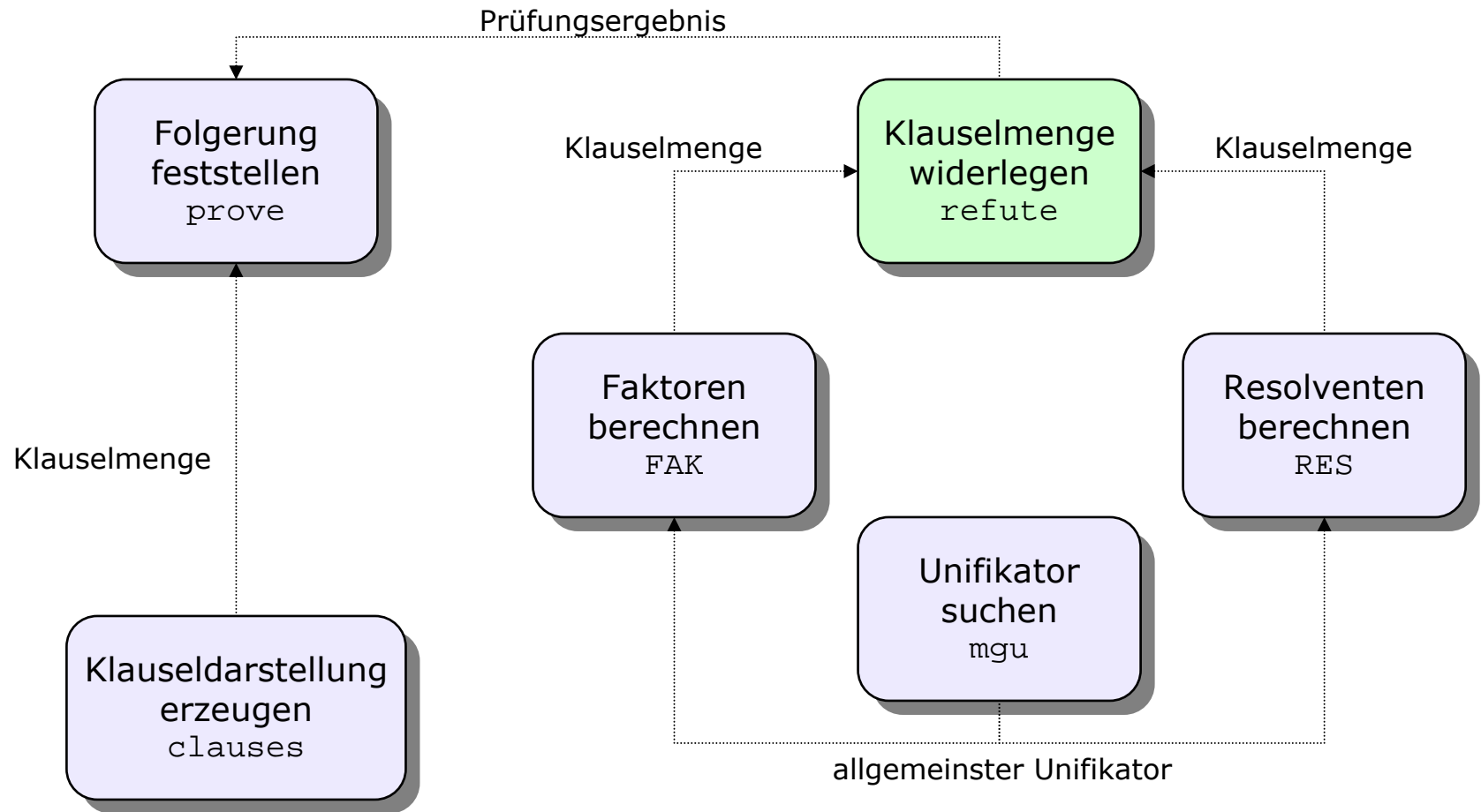
z.B.  $\forall x \exists y [P(x, y)]$  wird zu  $\forall x [P(x, f(x))]$

2. Umformung in Klauselschreibweise (Mengendarstellung)

z.B.  $\forall x, y [P(x) \wedge Q(x)]$  wird zu  $\{P(x), Q(x)\}$

### 3.3 Der Widerlegungsalgorithmus

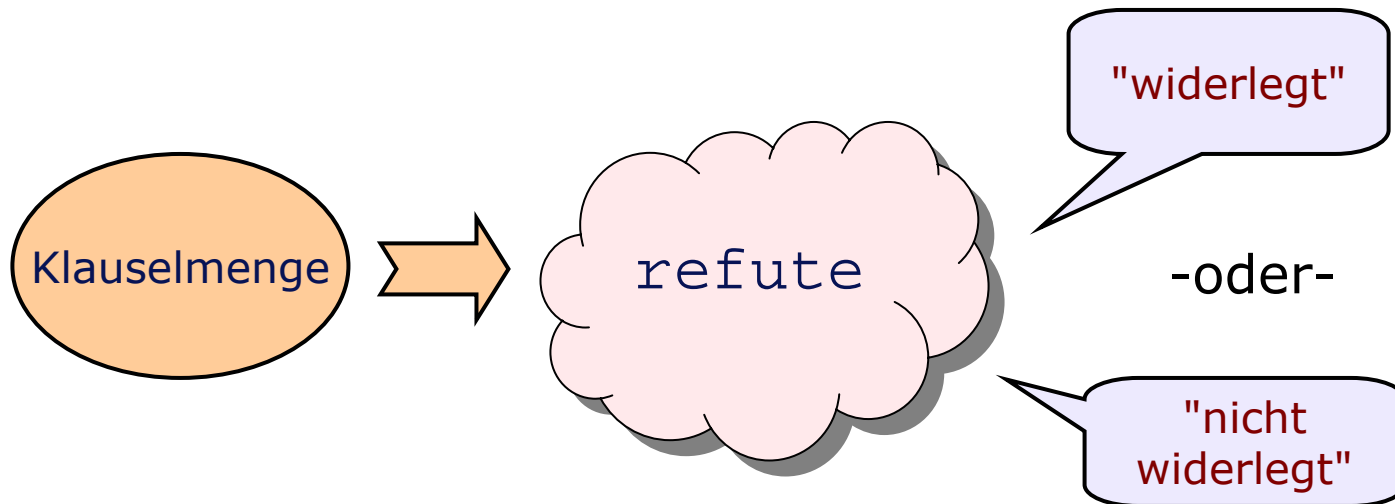
## Modularisierung und Datenflüsse



### 3.3 Der Widerlegungsalgorithmus

## Anforderungen und Funktionalität

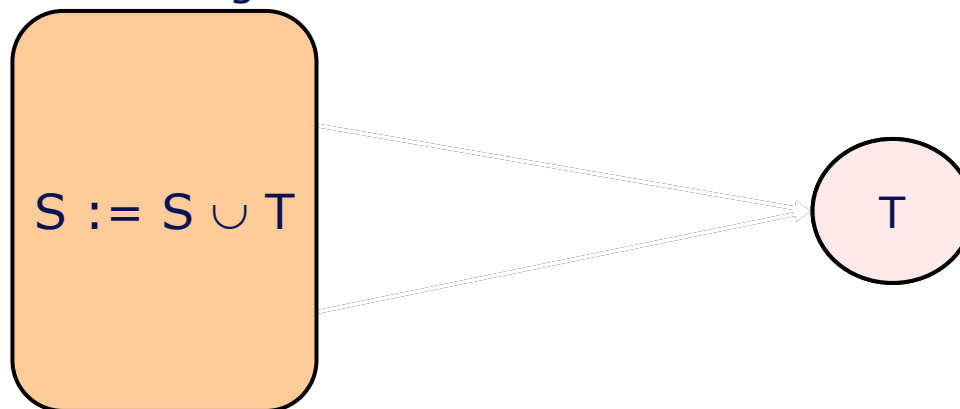
- `prove` als Überbau und Schnittstelle (koordinierende Instanz)
- `refute` als Systemkern (austauschbar)
- Implementation des Resolutionskalküls
- Widerlegung statt Beweis



### 3.3 Der Widerlegungsalgorithmus

## 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



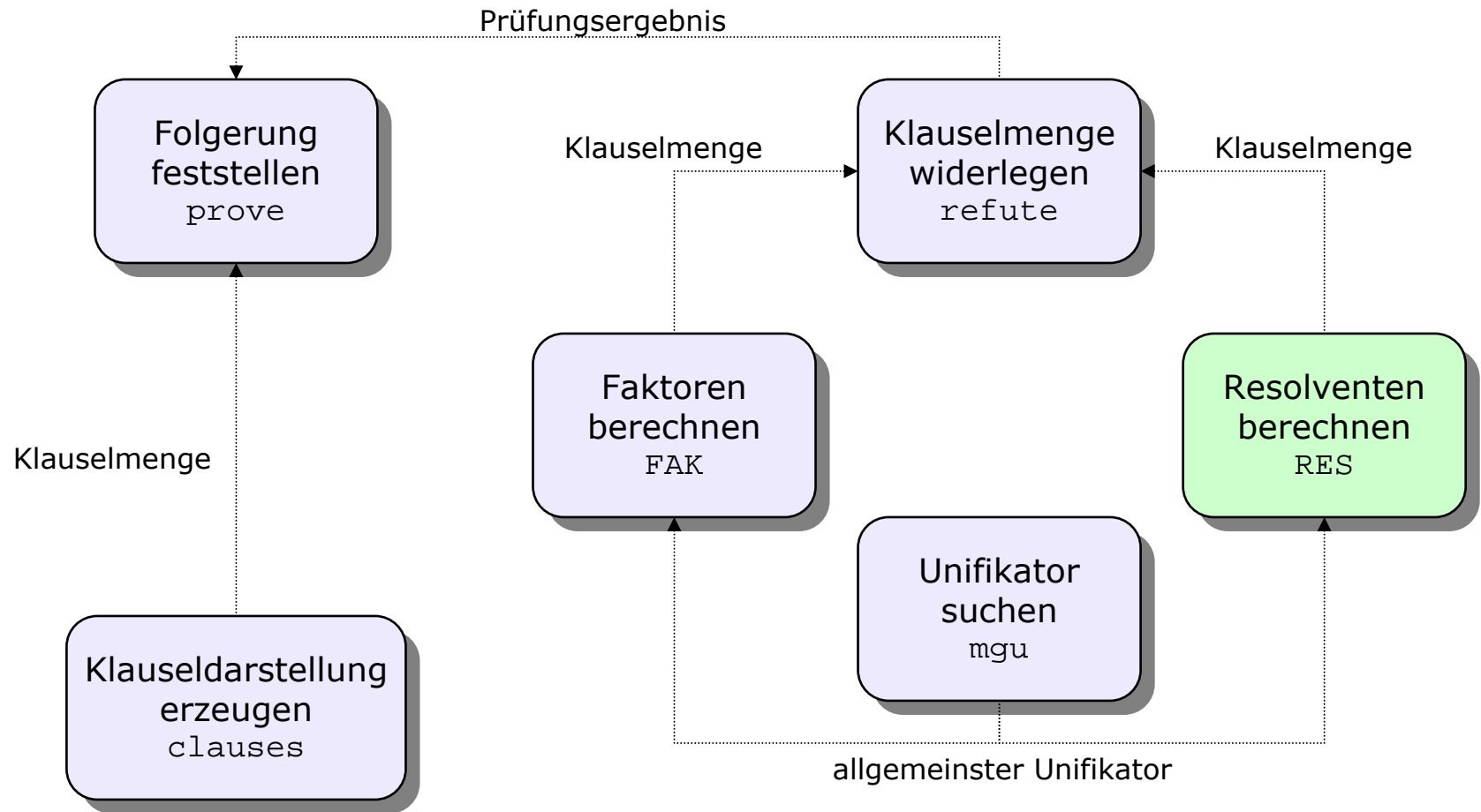
## Vollständige Implementierung

```

function refute( $Z \mu 5^S$ ) : bool
  [ @ FAK( $Z$ )
   $Z$  @ [
  while [  $\hat{u} \neq \hat{u}$  and  $\hat{u} \neq \hat{u}$  ] do
    [ @ RES( $Z/[$ )
    if  $\hat{u} \neq \hat{u}$  ] then
      [ @ FAK([ )
       $Z$  @  $Z + [$ 
    fi
  done
  return not( [ @  $\neq$  )
end
  
```

### 3.4 Der Resolutionsalgorithmus

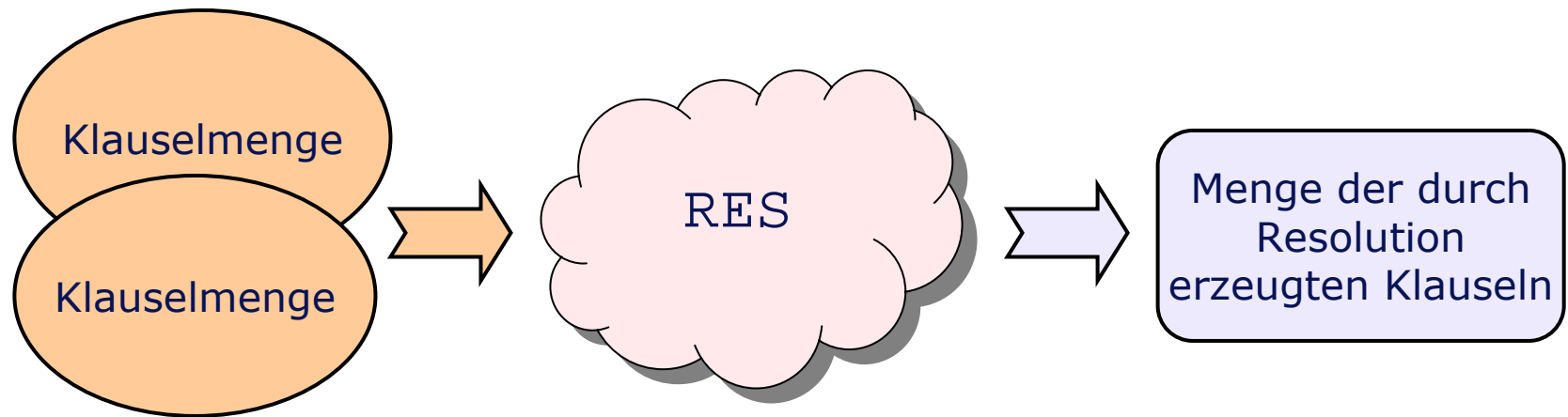
## Modularisierung und Datenflüsse



### 3.4 Der Resolutionsalgorithmus

#### Anforderungen und Funktionalität

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



### 3.4 Der Resolutionsalgorithmus

Zur Erinnerung: Resolution

•  $\text{Res}(C, L, D, K, \sigma)$

Bsp.:  $C = \{P, \neg Q\}$      $D = \{Q, R\}$   
 $L = \neg Q$              $K = Q$              $\sigma = h$

$\text{Res}(C, L, D, K, \sigma) = \{P, R\}$

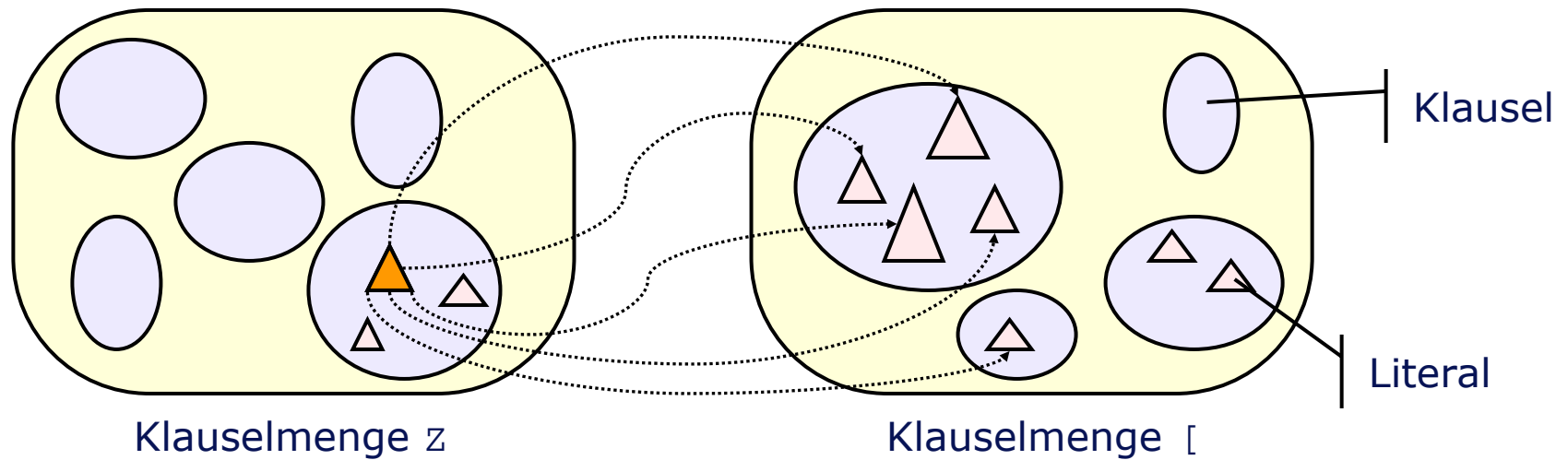
• Voraussetzungen für Resolvierbarkeit:

1. Literale komplementär
2. Jeweils zugehörige Atome unifizierbar

### 3.4 Der Resolutionsalgorithmus

#### Idee

- Jedes Literal der einen mit jedem Literal der anderen Ausgangsmenge auf Resolvierbarkeit testen
- Ggf. Resolventenbildung



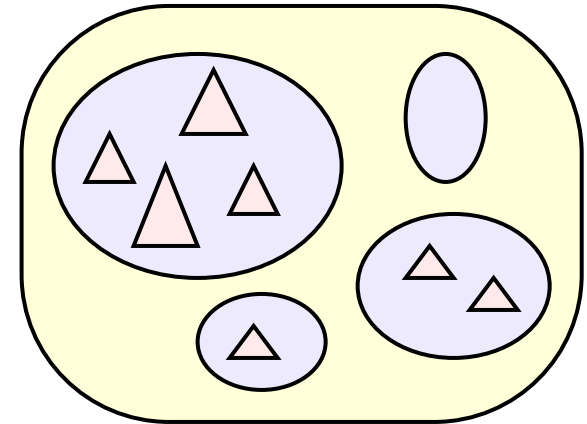
### 3.4 Der Resolutionsalgorithmus

#### Die Implementierung im Detail

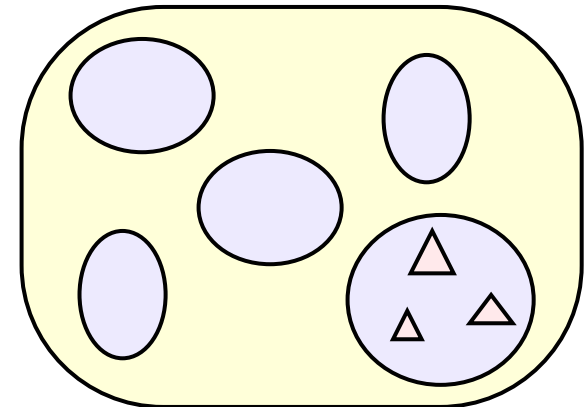
```

function RES( $Z / [ \mu 5^S ] : 5^S$ )
     $Y^* \Leftarrow \perp$ 
     $[ \Leftarrow [$ 
    for all  $J \mu Z$  do
         $[ \Leftarrow [ \setminus \sim J \Leftarrow$ 
        for all  $S \mu J$  do
            for all  $K \mu [ \Leftarrow$ 
                for all  $R \mu K$  do
                    if  $S$  komplementär  $R$  then
                         $v \Leftarrow \text{mgu}(i_S \setminus i_R)$ 
                        if  $v$   $\hat{=}$  failed then
                             $Y \Leftarrow v \setminus J \cup S, + v \setminus K \cup R,$ 
                             $Y^* \Leftarrow Y^* + \sim Y \Leftarrow$ 
                            if  $Y \Leftarrow \hat{\Delta}$  then return  $Y^*$  fi
                        fi fi done done done done
                    return  $Y^*$ 
                end
            end
        end
    end

```



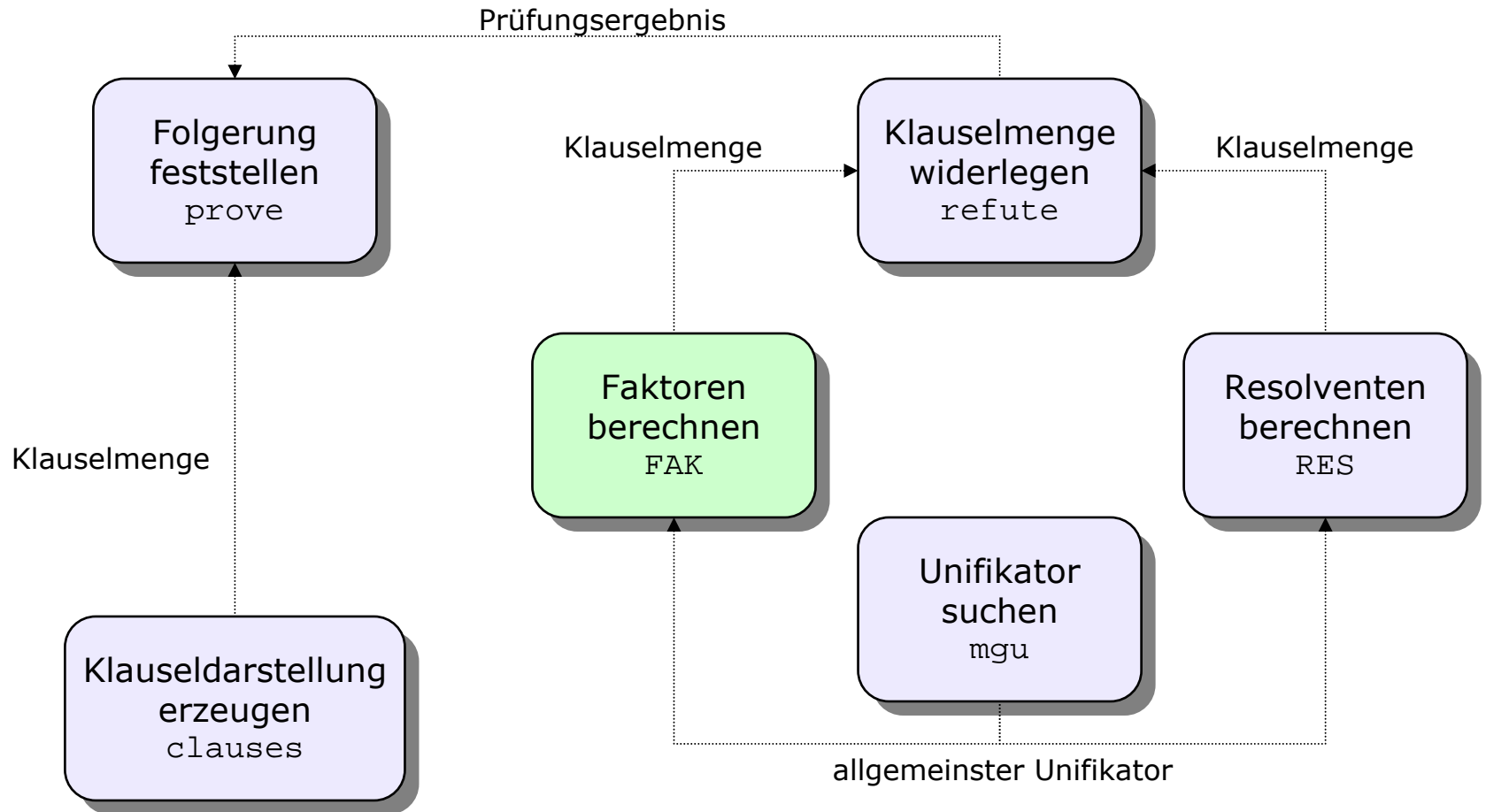
Klauselmenge  $[$



Klauselmenge  $Z$

### 3.5 Der Faktorisierungsalgorithmus

## Modularisierung und Datenflüsse



### 3.5 Der Faktorisierungsalgorithmus

#### Anforderungen und Funktionalität

- Berechnung aller möglichen Faktoren einer Klauselmenge
- S Teil des Outputs (auch Faktorisierung über  $\mathbb{h}$  möglich)



### 3.5 Der Faktorisierungsalgorithmus

#### Idee

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

```

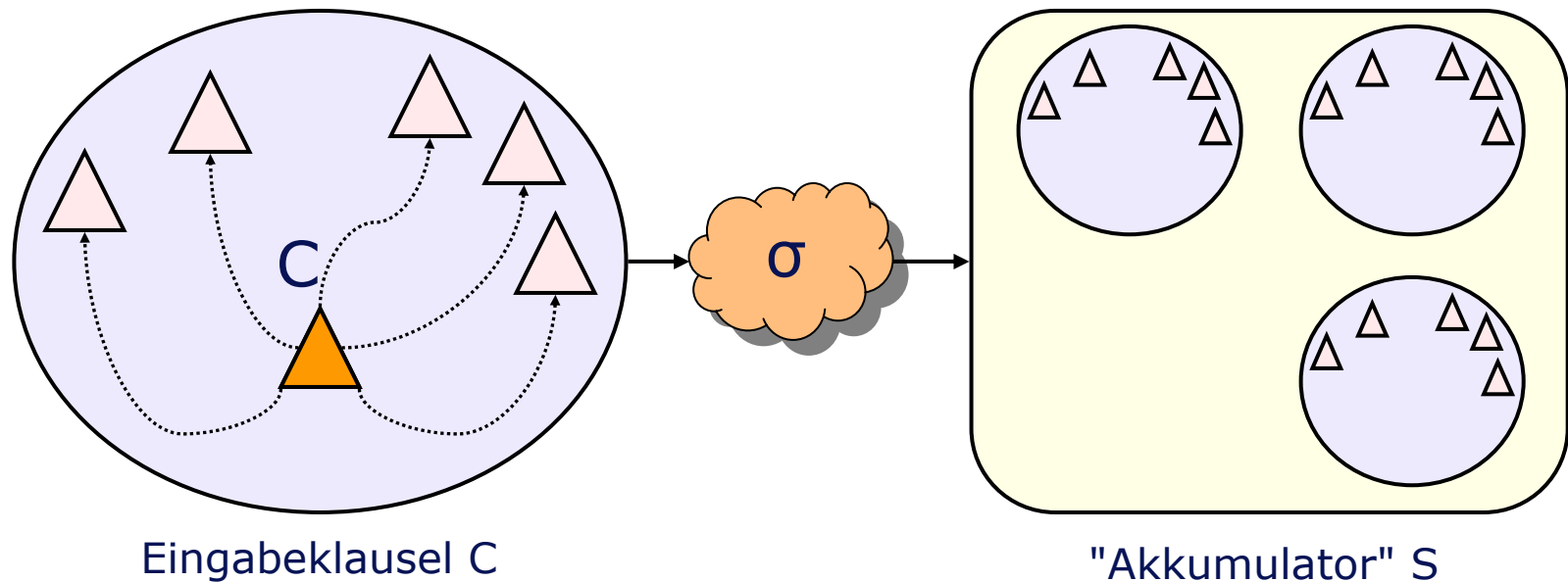
function FAK (  $Z \mu 5^S$  ) :  $5^S$ 
   $M \leftarrow 1$ 
  for all  $J \mu Z$  do
     $M \leftarrow M + f_{ak}(J)$ 
  done
  return  $M$ 
end

```

### 3.5 Der Faktorisierungsalgorithmus

#### Teilalgorithmus fak - Idee

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



### 3.5 Der Faktorisierungsalgorithmus

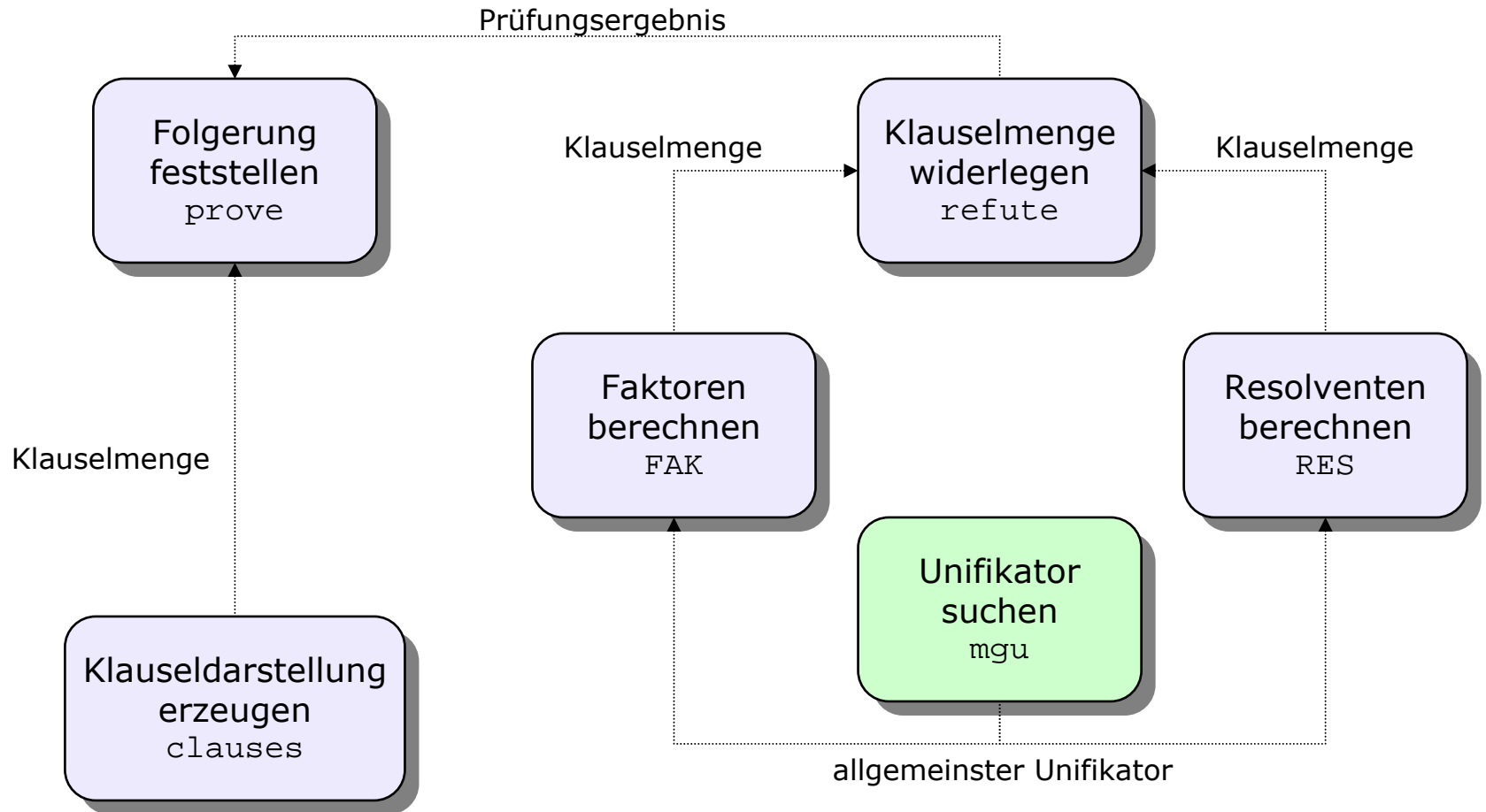
## Teilalgorithmus fak – vollständige Implementierung

```

function fak(J μ S ): 5S
  Z ← 1
  K ← J
  for all S μ J do
    K ← K ∩ S
    for all R μ K do
      if not S komplementär R then
        v ← mgu( iS / iR )
        if v ≠ failed then
          Z ← Z + fak(v + J, )
        fi
      fi
    done
  done
  return Z + ~J ∅
end
  
```

### 3.6 Der Unifikationsalgorithmus

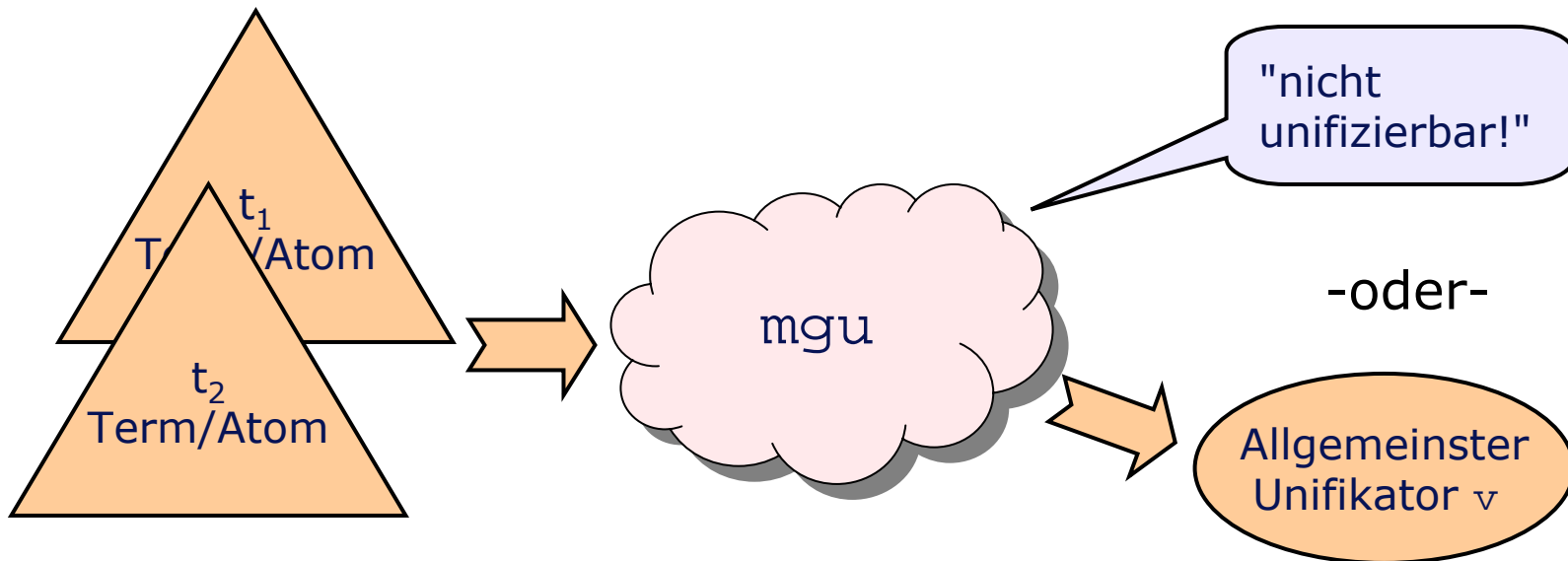
## Modularisierung und Datenflüsse



### 3.6 Der Unifikationsalgorithmus

## Der Unifikationsalgorithmus

- Zwei Aufgaben:
  - ✓ Test auf Unifizierbarkeit
  - ✓ Ggf. Berechnung eines allgemeinsten Unifikators



## Gründe für Nicht-Unifizierbarkeit

1. Occur Failure:  $t_1=x$   $t_2=..f(..x..)...$   
*"egal wie x ersetzt wird, Funktionssymbol bleibt immer erhalten"*

z.B.  $t_1=x$   $t_2=f(x)$

2. Clash Failure:  $t_1=g(q_1 \dots q_n)$   $t_2=h(r_1 \dots r_m)$  mit  $g \neq h$   
*"aus g wird nie h, egal wie ersetzt wird"*

z.B.  $t_1=f(x)$   $t_2=g(x)$

### 3.6 Der Unifikationsalgorithmus

#### Funktionsweise



- Terme gleich Fertig (h als Unifikator)
  
- $t_1$  oder  $t_2$  Variable *Occur Failure*
  - ✓ Enthalten in zweitem Term Fertig (z.B.  $t_1=x, t_2=f(y)$ )
  - ✓ Ersetzen durch zweiten Term
  
- Geschachtelte Ausdrücke  $t_1 = G(q_1 \dots q_n), t_2 = H(r_1 \dots r_m)$ 
  - ✓ Verschieden *Clash Failure*
  - ✓ Gleich sequentiell Unifikatoren für

Parameter suchen und Lösung aus diesen zusammensetzen (s. Bsp.)

### 3.6 Der Unifikationsalgorithmus

## Vollständiger Algorithmus (1)

```

function mgu(  $\{_4 / \{_5 \mu [ + H [ ' ) : SUB + \sim failed \} \}$ 
  if  $\{_4 @ \{_5$  then return  $\sim \} \}$  fi           // 1. Fall
  if  $\{_4 \mu ]$  then                                 // 2. Fall
    if  $\{_4 \mu ] + \{_5,$  then
      return failed                                 // "occur failure"
    else
      return  $\sim \{_4 \} \{_5 \}$ 
    fi
  fi
  if  $\{_5 \mu ]$  then return mgu(  $\{_5 / \{_4$  ) fi
  ...
  
```

### 3.6 Der Unifikationsalgorithmus

## Vollständiger Algorithmus (2)

```

...
assume {4@N ⊕4 Æ xu, / {5@O ⊕4 Æ yt, // 3. Fall
if N û O then return failed fi // "clash failure"
assume {4@m ⊕4 Æ xu, / {5@m ⊕4 Æ yu, / uA 3
v =@ h
  
```

```

for p=@ 4 to u do
  t=@ mgu(v ⊕p, /v ⊕p)
  assume v @ ~i4 2z4, / Æ /iu 2zu ç
  v =@ t+ ~i4 2t+z4, / Æ /iu 2t+zu, ç
done
  
```

$i_p$	x	y	z	a
$z_p$	x	y	z	a



```

return v
end
  
```

### 3.6 Der Unifikationsalgorithmus

#### Ablaufbeispiel

$$t_1 = f(x \ g(y \ h(y))), \quad t_2 = f(z \ g(z \ h(a)))$$

$$t_1' = g(y \ h(y)), \quad t_2' = g(z \ h(a))$$

$$t_1'' = h(z), \quad t_2'' = h(a)$$

$i_p$	x	y	z	a
$z_p$	?	?	?	?

v	$v \rightarrow x_p$	$v \rightarrow y_p$	t
h	x	z	{x/z}
{x/z}	g(y h(y))	g(z h(a))	{z/a, y/a}
h	y	z	{y/z}
{y/z}	h(z)	h(a)	{z/a}
h	z	a	{z/a}
{z/a}			
{z/a, y/a}			
{z/a, y/a, x/a}			

```

...
v ← h
for p ← 4 to u do
  t ← mgu(v → x_p / v → y_p)

  v ← t + {i_4 2t + z_4, ...} / {i_u 2t + z_u, ...}
done
...

```

# 4 Optimierungsansätze

## Optimierungsansätze

- Grundsystem zu ineffizient für Praxis, z.B.
  - ✓ exponentieller Klauselzuwachs in `refute`
  - ✓ Aufwändige ( $O(n*m)$ ) Resolventenbildung in `res`
- Optimierungsmöglichkeiten insbesondere für
  - ✓ `refute`
  - ✓ `prove`
  - ✓ `res`
- Beispiel: Optimierung der Resolventensuche durch [Klauselgraphen](#)

## 4.2 Klauselgraphen

---

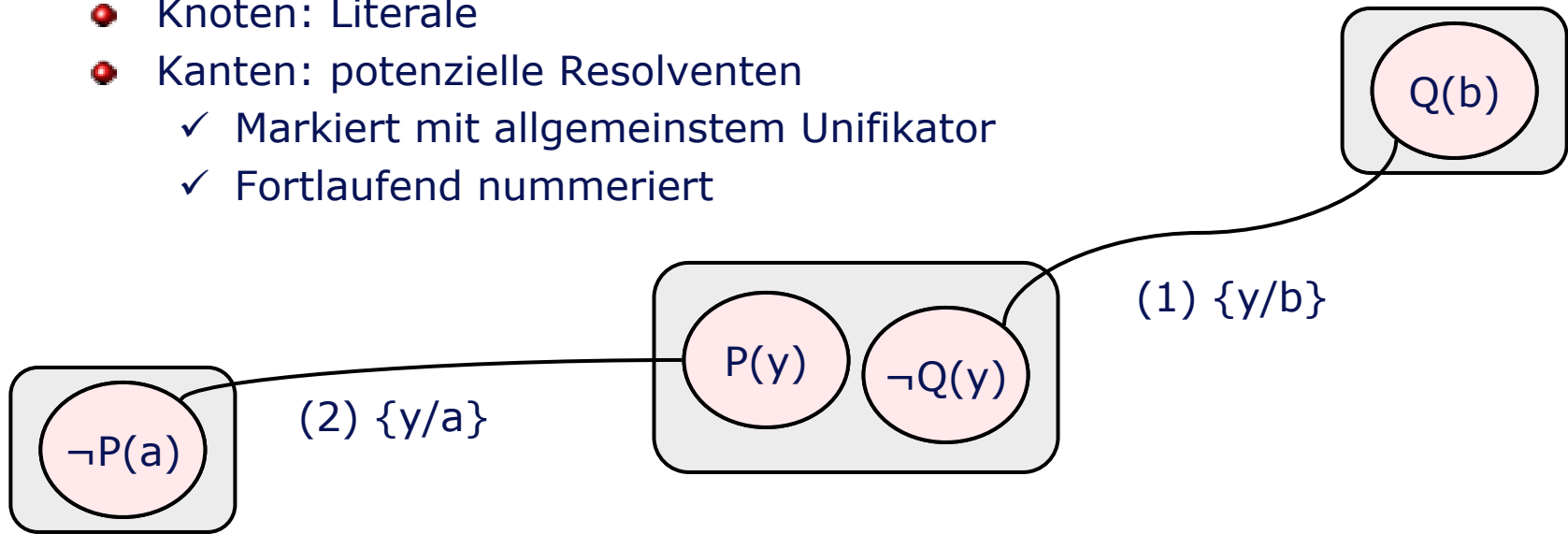
### Grundgedanke

- Aufwand von  $RES=O(n*m)$  "*jeder mit jedem*"
- Häufig nur Bruchteil der getesteten Verbindungen resolvierbar
- Problem: Ineffizienz
- Idee: nur bestimmte Verbindungen testen
- Realisierung: **Klauselgraphen**

## 4.2 Klauselgraphen

### Aufbau

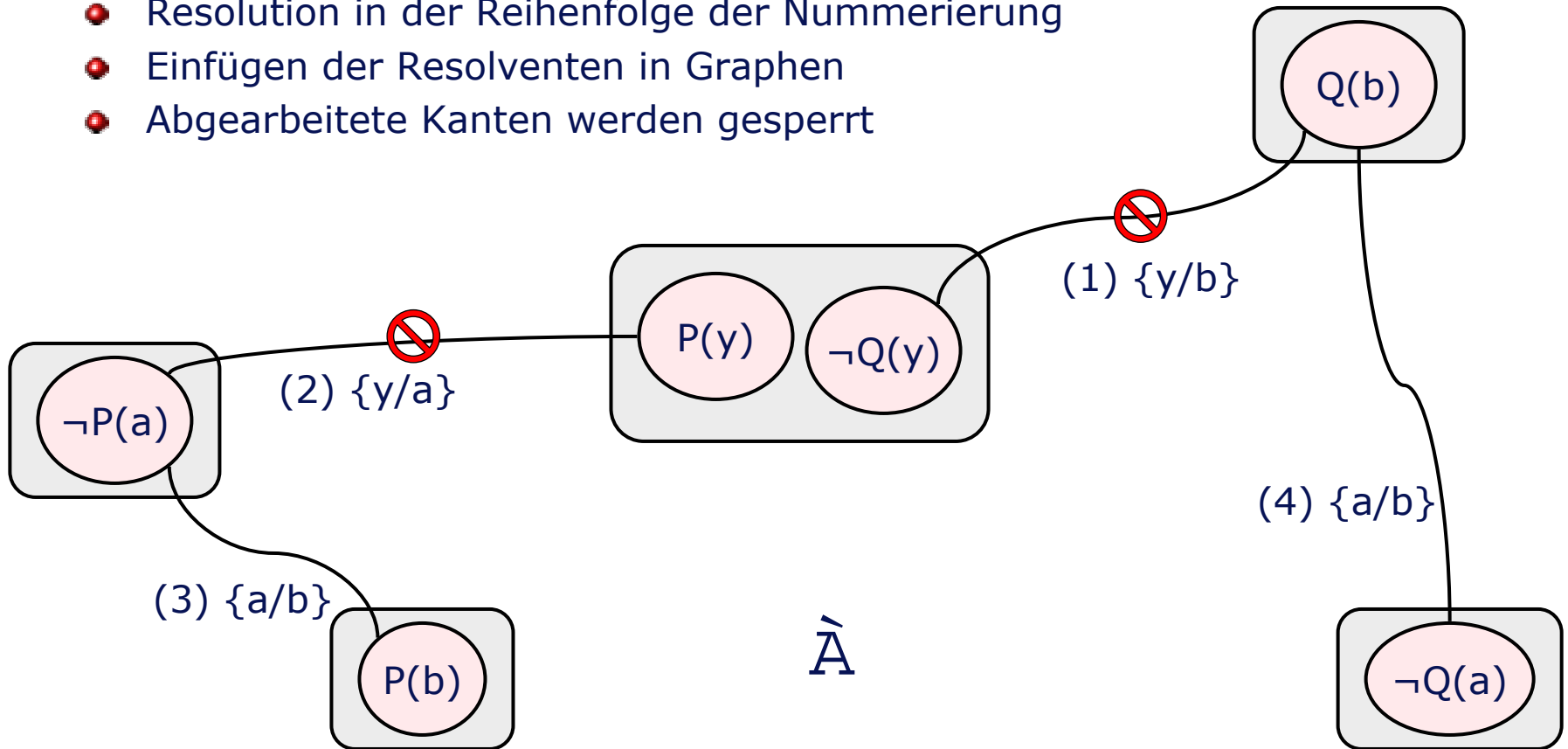
- Knoten: Literale
- Kanten: potenzielle Resolventen
  - ✓ Markiert mit allgemeinstem Unifikator
  - ✓ Fortlaufend nummeriert



## 4.2 Klauselgraphen

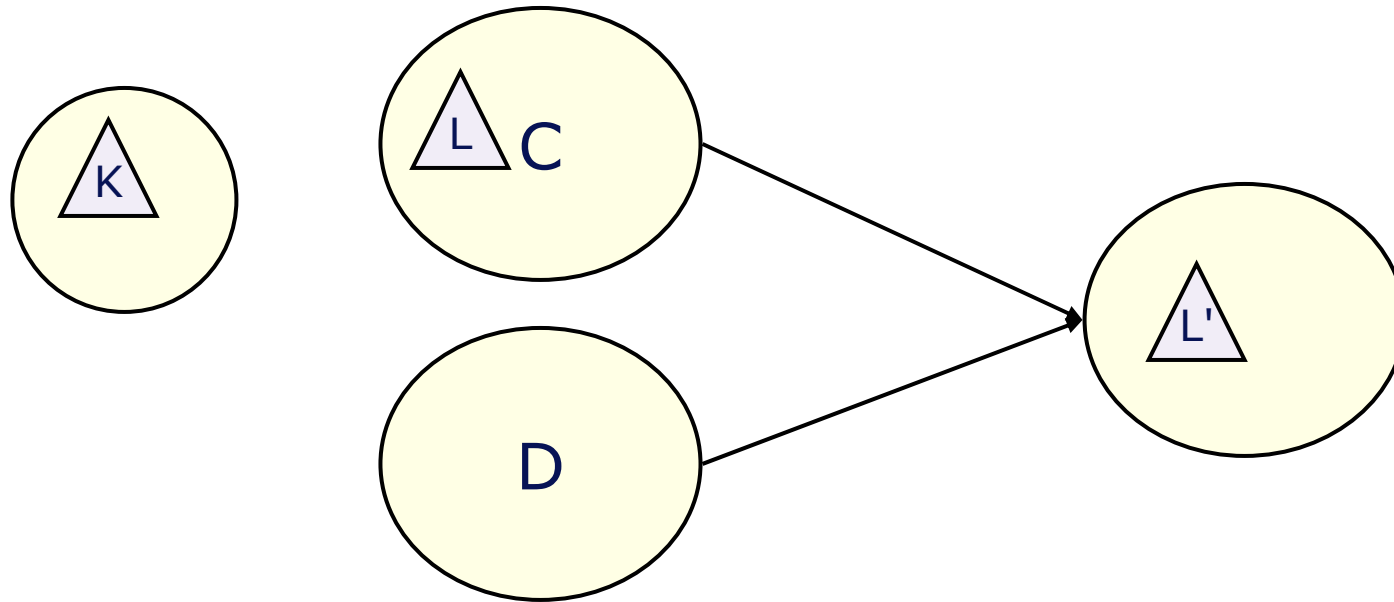
### Verfahren

- Resolution in der Reihenfolge der Nummerierung
- Einfügen der Resolventen in Graphen
- Abgearbeitete Kanten werden gesperrt



## 4.2 Klauselgraphen

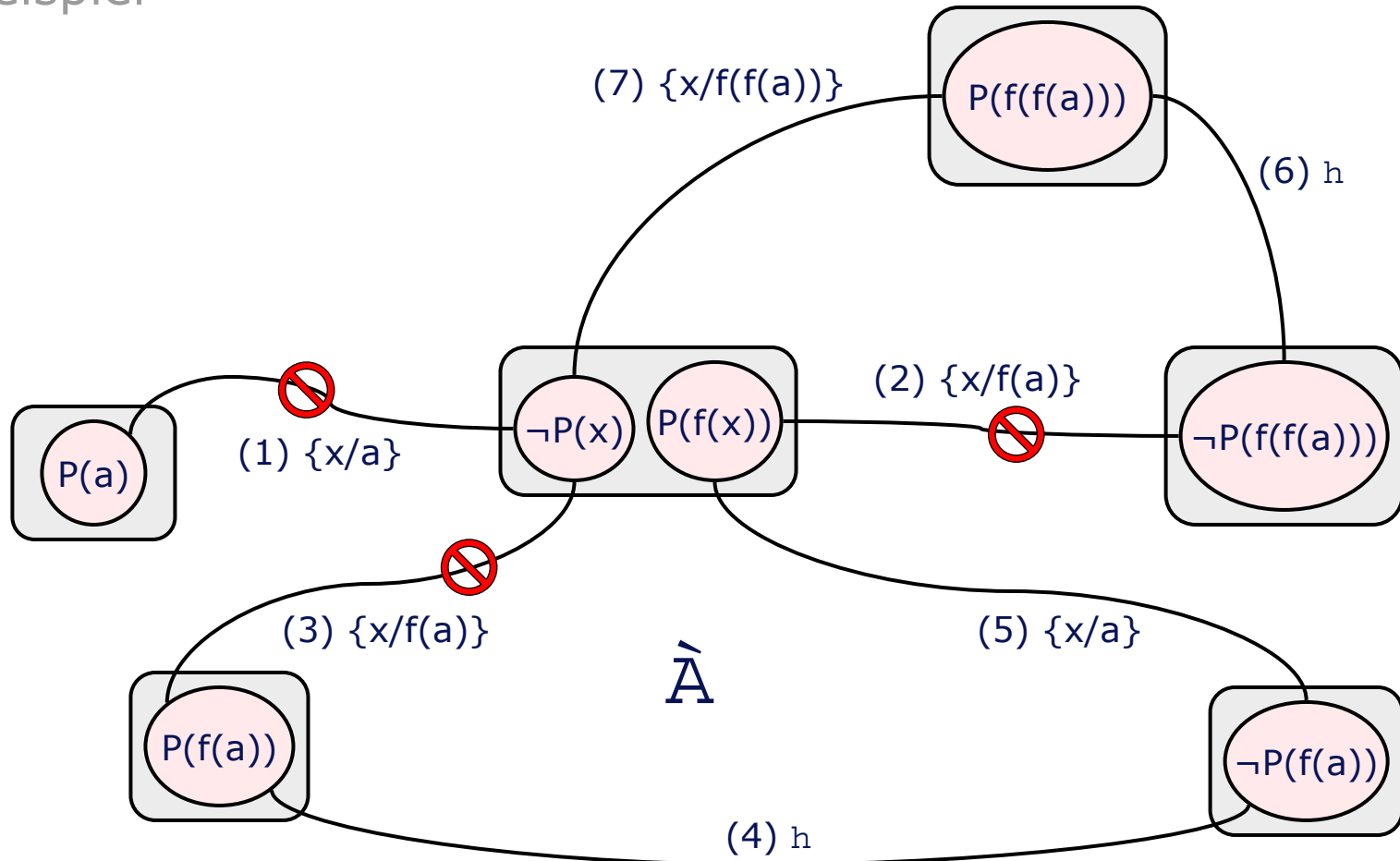
### Idee des Verbesserungsansatzes



- Einschränkung des Suchraums auf
  1. Partner der Elternliterale
  2. Literale der Elternklauseln

4.2 Klauselgraphen

Beispiel



# 5 Zusammenfassung und Ausblick

## Zusammenfassung und Ausblick

- Implementierung eines Systems aus sechs Algorithmen
  - ✓ mgu
  - ✓ RES
  - ✓ FAK
  - ✓ clauses
  - ✓ refute
  - ✓ prove
- Teilalgorithmen in beispielhaften Grundversionen
- Für praktische Implementierungen Optimierungen nötig
- Klauselgraphen als eine Möglichkeit
- Ausblick: Ableitungsstrategien – sehr viele Möglichkeiten

- Ende -



## 4.1 Löseregeln und Ableitungsstrategien

### Löseregeln und Ableitungsstrategien

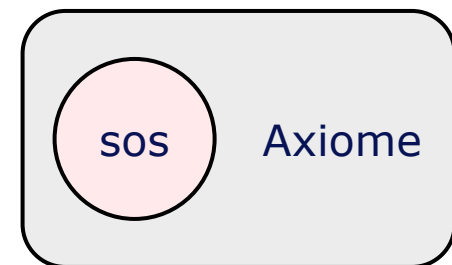
- Reduktion der durch refute erzeugten Klauseln durch Verbotskriterien zur Herleitung neuer Klauseln
- Löseregeln: unabhängig von Entstehungsgeschichte der Klausel

Bsp.: **Tautologieklauseln** ( $\{A, \neg A\} \text{ } \delta \text{ } C$ )

- Ableitungsstrategien: abhängig von Entstehungsgeschichte der Klausel

Bsp.: **set-of-support**

Prämisse: Axiomsystem erfüllbar  
 Zwischen Axiomen keinen Widerspruch suchen



Zu widerlegende Klauselmenge

## 4.1 Löseregeln und Ableitungsstrategien

---

### Löseregeln und Ableitungsstrategien

- Idee: Reduktion der Anzahl der in refute hergeleiteten Klauseln
- Ansatz: Bestimmte Klauseln nicht herleiten

Bsp.: **Tautologieklauseln** ( $\{A, \neg A\} \delta C$ )