Schritt 1 Richtung Resolution: Substituieren

Wegen impliziter Allquantifizierung der Variablen gilt:

- $P(x)$, $\neg P(y)$ widersprüchlich;
- $P(x)$ $\neg P(f(a))$ widersprüchlich;

aber *nicht*

- $P(x)$, $\neg P(f(x))$!

Für vollständige Inferenzverfahren müssen wir semantisch gleiche Literale auch textuell gleich machen, unifizieren können!

Eine **Substitution** ist eine Folge von Ersetzungen x/t einer Variablen x durch einen Term t, wobei x in t nicht vorkommt.

Bemerkung: Durch Anwendung einzelner Ersetzungen auf den ersetzenden Term früherer Ersetzungen erzeuge Reihenfolgeunabhängigkeit der Ersetzungen

➥ Substitutionen werden *Mengen* von x/t-Paaren.

Schritt 2 Richtung Resolution: Unifizieren

Eine Substitution, die eine Menge von Literalen textuell gleich macht bis auf ihr Vorzeichen, heißt **Unifikator**.

Unifikatoren sind nicht eindeutig! Beispiel:
Für $\neg P(x,y)$ und $\neg P(a,z)$ Unifikator $[x/a]\ [y/z]$ (➥ $\neg P(a,z)$)
aber auch $[x/a]\ [z/y]$ (➥ $\neg P(a,y)$) und $[x/a][y/\boldsymbol{a}][z/\boldsymbol{a}]$ (➥ $\neg P(a,a)$)

Ein Unifikator θ einer Menge $\mathcal{L}$ von Literalen heißt **allgemeinster Unifikator** (*most general unifier*, MGU), falls es für jeden Unifikator θ‘ von $\mathcal{L}$ eine Substitution σ gibt mit θ‘= θσ.

Unifikationssatz von Robinson

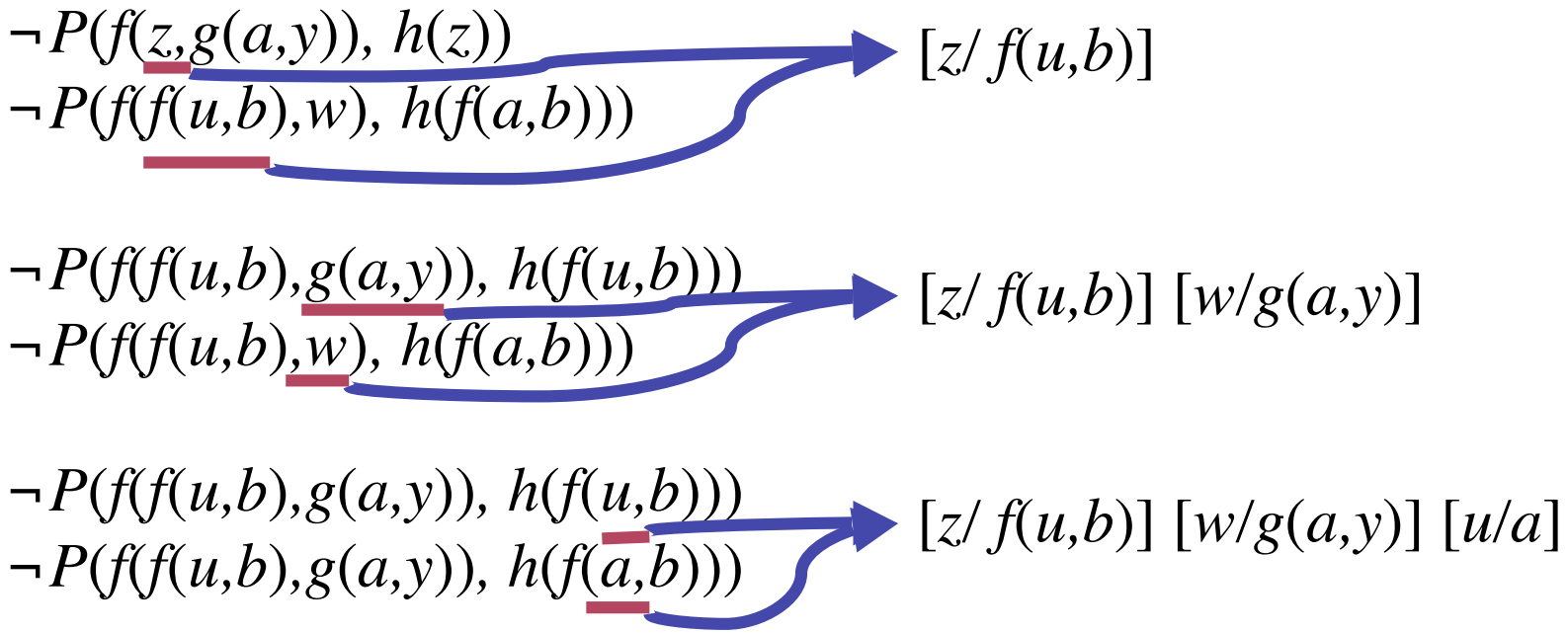
Jede unifizierbare Literalmenge hat einen MGU.

Bemerkung: Der MGU ist eindeutig bis auf Variablen-Umbenennungen.

Beweis: Konstruktiv durch Angabe und Analyse des Algorithmus UNIFY

UNIFY am Beispiel eingeführt

UNIFY setzt Terme von links nach rechts komponentenweise gleich, bis fertig oder zwei verschiedene Funktionen auftreten.



Komplexität der Unifizierung

- „Naive“ Unifizierung (UNIFY) quadratisch wegen OCCUR-CHECK (Test auf „wobei x in t nicht vorkommt“)
- Es gibt Unifizierungsalgorithmen mit linearer Laufzeit in der Länge der Terme (mit relativ großen Konstanten!)

Manche Implementierungen, z.B. typische PROLOG-Interpreter(!), lassen den OCCUR-CHECK aus (Effizienz)

Beispiel SWI-PROLOG:

```
p(X,X).
?- p(f(Y),Y).
   Y = f(**)
   Yes
```

Für Programmierung ist das praktisch i.d.R. irrelevant.

Resolution, prädikatenlogische Variante

Die Klausel R ist **Resolvente** zweier Klauseln K_1 und K_2, gdw:

1. Es gibt Substitutionen σ_1, σ_2, sodass $K_1\sigma_1$ und $K_2\sigma_2$ keine gemeinsamen Variablen enthalten.
2. Es gibt positive $L_1, \ldots, L_m \in K_1\sigma_1$ $(m \geq 1)$ und negative $L'_1, \ldots, L'_n \in K_2\sigma_2$ $(n \geq 1)$, sodass $\mathcal{L} = \{L_1, \ldots, L_m, L'_1, \ldots, L'_n\}$ unifizierbar ist mit MGU θ.
   (Für $m>1$ oder $n>1$: Literale einer Klausel werden **faktorisiert**.)
3. $R = [(K_1\sigma_1 - \{L_1, \ldots, L_m\}) \cup (K_2\sigma_2 - \{L'_1, \ldots, L'_n\})]\theta$

Beispiele

$\{P(x,y), P(f(z),a)\}$ $\{\neg P(u,x)\}$

$\sigma_2=[x/v]$

$\{P(f(z),a)\}$

$\theta=[x/u][y/v]$

$\{P(x,y), P(f(z),a)\}$ $\{\neg P(u,x)\}$

$\sigma_2=[x/v]$

$\square$

$\theta=[x/f(z)][y/a][u/f(z)][v/a]$

Korrektheit der prädikatenlogischen Resolution

Satz

Sei R die Resolvente zweier Klauseln K_1 und K_2. Sei für eine Klausel K der *Allabschluss* $\forall K$ die Formel $\forall x_1. \ldots \forall x_n.K$, wobei $x_1, \ldots, x_n$ alle in K vorkommenden Variablen sind.
Dann gilt: $\forall R$ ist eine Folgerung aus $\forall K_1 \wedge \forall K_2$.

Beweisidee: Analog dem Beweis für die aussagenlogische Resolution: Ein Modell für $\forall K_1 \wedge \forall K_2$ muss nach Streichung der durch Resolution wegfallenden Literale mindestens einen der Allabschlüsse der Restklauseln $\forall K'_1$ oder $\forall K'_2$ erfüllen, und damit auch $\forall R$.

Folgerung

Insbesondere gilt: Ist $\square \in \mathrm{Res}^*(S)$, dann ist S inkonsistent.

Vollständigkeit der Resolution: Beweisgang

Jede Menge $\mathcal{F}$ von Formeln ist als Klauselmenge S darstellbar.
Sei S inkonsistent.

Satz von Herbrand

Menge S' von Grundklauseln ist inkonsistent.

Resolutionssatz der AL

AL-Resolution leitet $\square$ aus S' ab.

„Lifting-Lemma“: noch zu zeigen!

Es gibt eine Widerlegung von S mit PL-Resolution

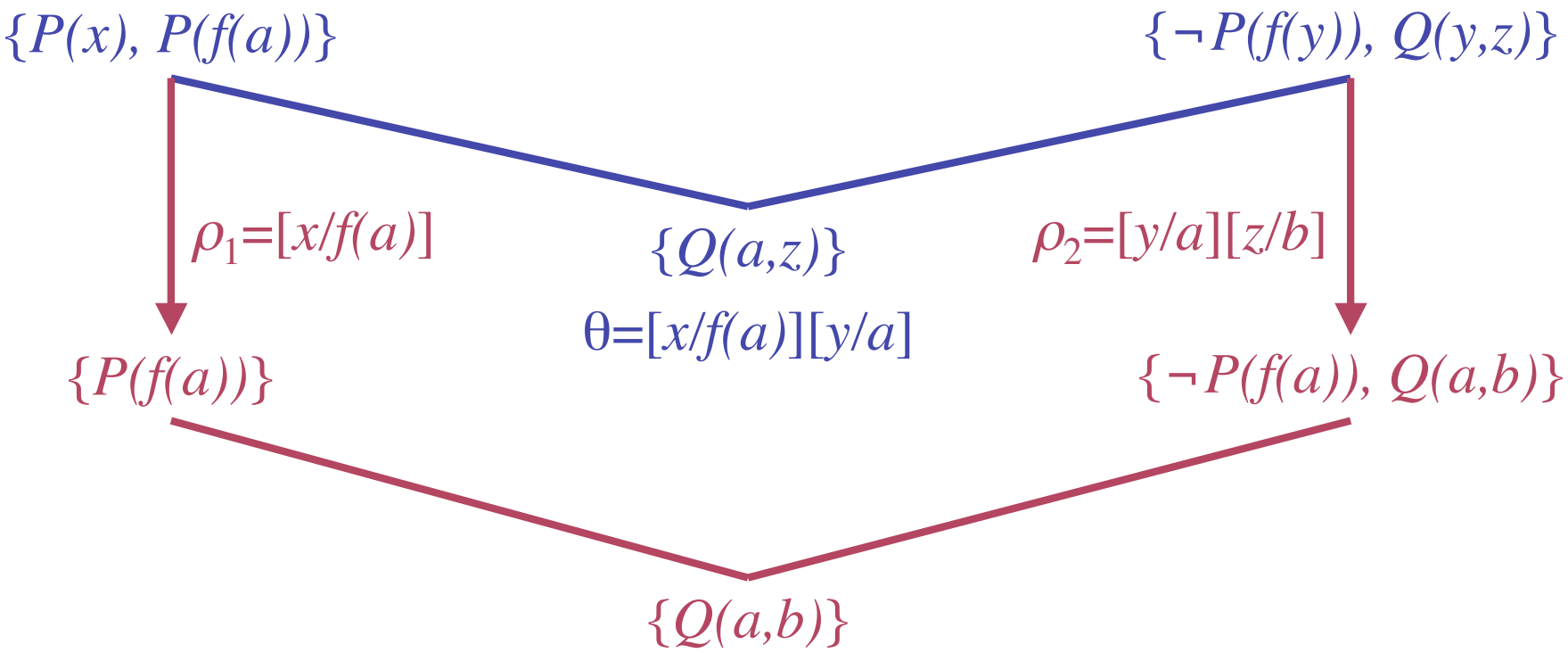
Das Lifting Lemma

Seien K_1 und K_2 Klauseln mit disjunkten Variablen;
seien $K'_1 = K_1\rho_1$ und $K'_2 = K_2\rho_2$ beliebige Grundinstanzen davon, die aussagenlogisch resolvierbar sind mit Resolvente R'.
Dann gibt es eine prädikatenlogische Resolvente R von K_1 und K_2, zu der R' eine Grundinstanz ist.

Beweisidee: Verwende als Unifikator in der PL-Resolution diejenigen Substitutionen aus ρ_1, ρ_2, welche die entsprechenden Literale aus K_1 und K_2 unifizieren. Diese muss es geben.

Beispiel für Lifting-Lemma-Konstruktion

Prädikatenlogik-Ebene blau, Aussagenlogik rosa



Vollständigkeit der Resolution, Resolutionssatz

Vollständigkeitssatz

Sei $\mathcal{F}$ eine inkonsistente Formel in Skolemform mit quantorenfreiem Teil $\mathcal{F}^*$ in KNF. Dann ist $\square \in \text{Res}^*(\mathcal{F}^*)$ für prädikatenlogische Resolution.

Beweisgang: s. vorletzte Folie.

Resolutionssatz der Prädikatenlogik

Sei $\mathcal{F}$ eine Formel in Skolemform mit quantorenfreiem Teil $\mathcal{F}^*$ in KNF. $\mathcal{F}$ ist inkonsistent, gdw. $\square \in \text{Res}^*(\mathcal{F}^*)$.

Beweis: Zusammenfassung von Korrektheit und Vollständigkeit.

Bemerkung zur Faktorisierung

Resolution *ohne* Faktorisierung ist unvollständig!

Beispiel

1. $\{\neg P(x), \neg P(y)\}$
2. $\{P(u), P(v)\}$

Unterschiedliche Formulierungen von Resolution nehmen Faktorisierung entweder in die Resolutionsregel mit auf (wie in der Definition oben) oder definieren sie als eigene Inferenzregel.

Spezialisierungen der Resolution

… sind dieselben wie in der AL!
Prinzip: Beschränkung der Auswahlmöglichkeiten für die Elternklauseln K_i, K_j

Insbesondere gibt es:

- Stützmengen-Resolution (*set of support*)
- Einsklausel/Unit-Resolution
- Input-Resolution
- SLD-Resolution

Alle Spezialisierungen „erben“ Korrektheit!

Vollständigkeit (ggf. mit Einschränkungen) beweise jeweils mit Hilfe des Lifting Lemma!

Beispiel Resolution (1/3: Formalisierung)

Problembeschreibung

Prämisse: Jeder, der Kapital spart, bekommt eine Rendite.
Folgere: Wenn es keine Rendite gibt, dann spart keiner Kapital.

Eine(!) mögliche Formalisierung

$S(x,y)$: x spart y; $K(x)$: x ist Kapital; $R(x)$: x ist Rendite; $B(x,y)$: x bekommt y

Prämisse: $\forall x.[\ \exists y.(S(x,y) \wedge K(y)) \Rightarrow \exists r.(R(r) \wedge B(x,r))]$

Folgerung: $\neg \exists r.R(r) \Rightarrow \forall x.\forall y.(S(x,y) \Rightarrow \neg K(y))$

Beispiel Resolution (2/3: Klauselform)

Prämisse: $\forall x.[\ \exists y.(S(x,y) \wedge K(y)) \Rightarrow \exists r.(R(r) \wedge B(x,r))]$
Folgerung: $\neg \exists r.R(r) \Rightarrow \forall x.\forall y.(S(x,y) \Rightarrow \neg K(y))$

Formulierung in Klauselform

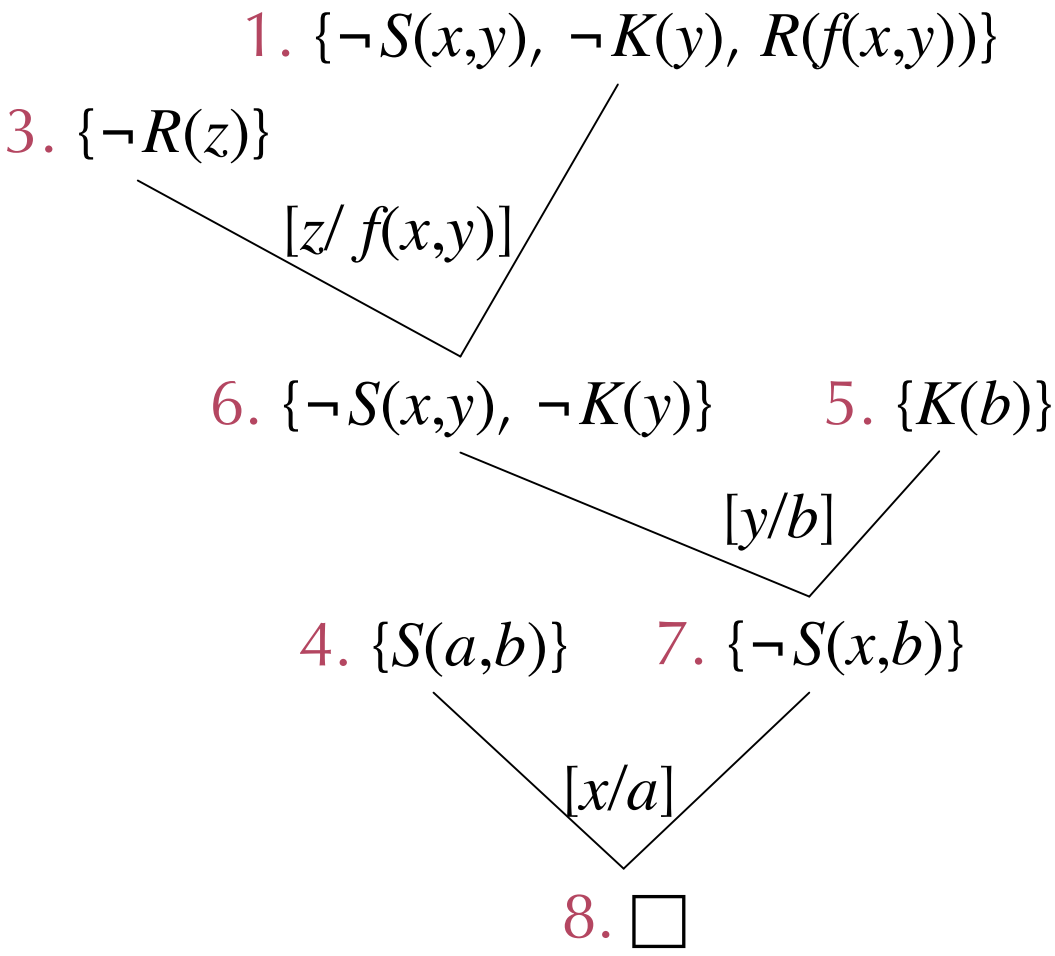
1. $\{\neg S(x,y),\ \neg K(y),\ R(f(x,y))\}$
2. $\{\neg S(x,y),\ \neg K(y),\ B(x, f(x,y))\}$
3. $\{\neg R(z)\}$
4. $\{S(a,b)\}$
5. $\{K(b)\}$

} Prämisse

} Negation der Folgerung

Beispiel Resolution (3/3: Widerlegung)

1. $\{\neg S(x,y),\ \neg K(y),\ R(f(x,y))\}$
2. $\{\neg S(x,y),\ \neg K(y),\ B(x, f(x,y))\}$
3. $\{\neg R(z)\}$
4. $\{S(a,b)\}$
5. $\{K(b)\}$



Frage: Ist das eine …

- *set of support*-Ableitung?
  (Wenn ja: welches SOS?)
- Unit-Ableitung?
- Input-Ableitung?
- SLD-Ableitung?

Demodulation

Inferenzregel für Formelmengen, in denen Gleichheit zwischen Termen (Prädikat „=“) definiert ist und Gleichungen Gegenstand von Ableitungen sein können.
Seien α Teilklausel, $L[t]$ Literal, in dem Term t vorkommt; seien t_1, t_2 Terme; seien t_1, t mit Substitution θ unifizierbar:

$$\frac{t_1 = t_2 \mid L[t] \vee \alpha}{(L[t_2] \vee \alpha)[\theta]}$$

Beispiel

1. $\{f(a) = b\}$
2. $\{\neg S(x,y),\ B(x, f(x))\}$

$[x/a]$

$\{\neg S(a,y),\ B(a,b)\}$

Verallgemeinerung Paramodulation: „=“-Literal darf Element einer längeren Klausel sein.

Praktisch verwende Gleichheit als Termersetzung: Ausschließlich der linke Term wird durch den rechten ersetzt!

OTTER (1/2)

Organized Techniques for Theorem proving and Effective Research

procedure OTTER (*sos*,*usable*)

inputs: *sos*, a set of support

usable, background knowledge clauses deemed relevant for proof

repeat

clause ← the lightest member of *sos*

move *clause* from *sos* to *usable*

PROCESS(INFER(*clause*, *usable*), *sos*)

until *sos* = [] **or** a refutation has been found

Heuristik zur Bewertung von Klauseln,
z.B. Einsklauseln sind „leicht“, geringe Termtiefe ist „leicht“.

OTTER (2/2)

PROCESS(INFER(*clause, usable*), *sos*)

function INFER (*clause,usable*) **returns** clauses
resolve *clause* with each member of *usable*
return the resulting clauses after applying FILTER

Heuristik für „uninteressante" Klauseln

procedure PROCESS (*clauses,sos*)
for each *clause* **in** *clauses* **do**
    clause ← SIMPLIFY(*clause*)
    merge identical literals
    discard clause if it is a tautology
    sos ← [*clause* | *sos*]
    if *clause* has no literals **then** a refutation has been found
    if *clause* has one literal **then** look for a unit refutation

Vereinfachung durch Termersetzung (Demodulation)

URL des OTTER-Systems: http://www-unix.mcs.anl.gov/AR/otter/