


- Variablenbelegung bei Unifikation *in globaler Scope*
- *kein Prolog* Programmmentwurf
 - eventuell Fehlersuche / Erweitern
 - "custom Funktionen" nicht verwenden (bsp. Prädikat suffix) - wird vorgegeben in Klausur
- Bei Fragen "welche Antwort?":
 - *Fehler* berücksichtigen!!
 - Endlosschleifen sind auch Fehlermeldungen (out of local stack)

Formalisierung

 **Junktoren binden stärker als Klammern**

Tableau

- *UND* Regel: überprüfen, ob *alle* erzeugten Terme bereits in Spalte sind
- *ODER*-Regel: überprüfen, ob *eins* der erzeugten Terme bereits in Spalte ist
→ wenn das der Fall ist: REGEL NICHT ANWENDBAR
- *Abschlusssatz*: "Jeder Pfad enthält einen Clash, deswegen ist φ unerfüllbar."
- gegebenen Term *nicht* vereinfachen !!!
- verschiedene Pfade sind unabhängig voneinander zu behandeln

Tableau-Algorithmus

Testet logische Formel auf Erfüllbarkeit
Liefert ein Modell $M \models \varphi$, wenn kein Clash erzeugt wurde

Verfahren

1. Formel in NNF
2. Tableau durchführen
 1. Work depth-first
 2. Heuristiken für Reihenfolge:
 1. \wedge Regel
 2. \exists Regel
 3. \vee Regel
 4. \forall Regel
3. enthält jede Spalte einen Clash:
 1. φ ist *unerfüllbar*
4. Ist auf einem Pfad keine Regel mehr anwendbar:
 1. Suche Unifikationen, und Belegungen, die auf Clash-freiem Pfad liegen

- Wenn \exists innerhalb Allquantor-Scope ist, *zuerst* \forall Regel anwenden, um nicht komplex skolemisieren zu müssen

Unifikation

- $\cdot // \cdot$ Syntax erklären

- ANGABE VON UNIFIKATOR REICHT!!!!
 \Rightarrow es muss nicht jeder Schritt angegeben werden

Seien φ, ψ zwei ungleiche Atome

① Benenne Variablen in ψ so um, dass φ und ψ keine gleichnamigen Variablen mehr enthalten

② Enthalten φ und ψ unterschiedliche Prädikate? \rightarrow ja: nicht unifizierbar

③ Gleiche φ und ψ von links nach rechts an:

↳ Variablen darf auf beliebigen Term abgebildet werden

↳ Bilde Variablen so ab, dass $\varphi_i = \psi_i$

↳ Für jede Abbildung τ :

↳ Occurs-Check (Seits nicht möglich)

↳ ersetze alle Vorkommen der Variable

\rightarrow ⚠ Variable darf es jetzt nicht mehr vorkommen!!

⚠ Nur VARIABLEN können abgebildet werden!! (Funktionen nicht)

④ Wenn $\varphi = \psi$: Unifikator gefunden

↳ Wenn $\varphi \neq \psi$

Resolution



- Nicht Unifikation und Faktorisierung gleichzeitig machen
- Für Unifikation nur die Abbildungen, die wirklich notwendig sind

- mit Einheitsklauseln ($\{\neg K(a)\}$) arbeiten, um andere Klauseln kleiner zu machen!
- bereits generierte Klauseln verwenden!!!!

1. FORMEL IN UNF (mit Quantoren)

1.1 Eliminiere \rightarrow und \leftrightarrow

1.2 De-Morgan

1.2.1 Auch Für Quantoren !!!

$$\begin{cases} \neg \forall x P(x) \Leftrightarrow \exists x \neg P(x) \\ \neg \exists x P(x) \Leftrightarrow \forall x \neg P(x) \end{cases}$$

1.3 Eliminiere unnötige Negationen

2. SKOLEMISIERUNG

2.0: Optionale Schritte:

2.0.0 Alle generifizierten Variablen verschieden benennen \rightarrow verhindert falsches Zusammenfassen

2.0.1 Miniscoping \rightarrow optimiert Laufzeit von Algorithmus

2.1 Eliminiere \exists -Quantoren durch Skolem-Funktionen

2.2 Yest \forall -Quantoren

Δ ggf. Variablen umbenennen (falls nicht in 2.0.0 gemacht)

\Rightarrow keine Quantoren mehr

3. FORMEL IN UNF

Use Distributivgesetz

4. RESOLUTION

Miniscoping

Annahme: x in φ, ψ aber x nicht in π

$$\forall x (\varphi \wedge \pi) \Leftrightarrow \forall x \varphi \wedge \pi$$

$$\exists x (\varphi \wedge \pi) \Leftrightarrow \exists x \varphi \wedge \pi$$

$$\forall x (\varphi \vee \pi) \Leftrightarrow \forall x \varphi \vee \pi$$

$$\exists x (\varphi \vee \pi) \Leftrightarrow \exists x \varphi \vee \pi$$

$$\forall x (\varphi \wedge \psi) \Leftrightarrow \forall x \varphi \wedge \forall x \psi$$

$$\exists x (\varphi \vee \psi) \Leftrightarrow \exists x \varphi \vee \exists x \psi$$

Hier von innen nach außen bearbeiten!

Prolog Prädikate

Suchbäume

- ist ein **Search Tree** (ref. Info-LK)

- Seiten sind Relevant: was **zuerst** bearbeitet wird, ist linker Teilbaum
- Verzweigungspunkte einzeichnen, AUCH WENN NACH ERSTER ANTWORT AUFGEHÖRT WERDEN KANN!!!
- Immer nur **einen** Schritt pro Verzweigung

Prädikate

Vergleichsprädikate

Arithmetik	Prolog
$x < y$	<code>X < Y</code>
$x \leq y$	<code>X =< Y</code>
$x = y$	<code>X == Y</code>
$x \neq y$	<code>X \= Y</code>
$x \geq y$	<code>X >= Y</code>
$x > y$	<code>X > Y</code>

Funktionsweise

- **Beide** Seiten werden arithmetisch ausgewertet
- Ergebnisse werden verglichen
- Abhängig von **Ergebnissen** ist Goal erfolgreich oder scheitert

Warning

- **Beide** Seiten müssen korrekte arithmetische Terme sein.
- **Keine** Seite darf ungebundene Variablen enthalten
- **Vorsicht!** es heißt `=<` und nicht `<=` (BS lol)

Prädikat Member

`member(T, L)`

returns true, wenn $T \in L$, sonst false.

- zweistellig
- **1. Argument:** Term T
- **2. Argument:** Liste L
- Argumente können **Variablen sein** → belegt Variable der Reihe nach mit **allen Elementen** der Liste aus

Funktionsweise

- Rekursive Prädikate

```
member(X, [X|_]).
member(X, [_|T]) :- member(X, T).
```

- Implementation mit anonymer Variable je Basisklausel verhindert verzweigungspunkte.
⇒ Unifikation findet in **Regelkopf** statt

Beispiel

```
?- member(a,[a]).
true.
?- member(a,[b,c,d]).
false.
?- member(d,[b,c,d]).
true.
?- member(a, [[a],b]).
false.
?- member(b, [[a],b]).
true.
```

```
?- member(X,[a,b,[c,d],father(butch),Y]).
X = a ;
X = b ;
X = [c,d] ;
X = father(butch) ;
X = Y.
?- member(X,[]).
false.
?- member(X,a).
false.
```

Prädikat length

```
length(List, Length)
```

- *erstes Argument*: Liste
- *zweites Argument*: Länge der Liste

Funktionsweise

[Leere Liste](#)

[Ganzzahl-Arithmetik > Variablenzuweisung](#)

```
length([], 0).
length([_|T], X) :- length(T, Y), X is Y + 1.
```

Beispiel

```
?- length([],X).
X = 0.
?- length([a,b,c],X).
X = 3.
?- length([a,[b,c]],X).
X = 2.
```

```
?- length([a,b],2).
true.
?- length(X,3).
X = [_G578, _G584, _G590].
```

Prädikat append

returns true wenn alle Argumente eine [Liste](#) sind und

- *1. Argument*: Liste *A*
- *2. Argument*: Liste *B*
- *3. Argument*: Liste *C*, mit `C = A.append(B)`.

Funktionsweise

```
append([], L, L).
append([H|T], L, [H|R]) :- append(T, L, R).
```

vgl. Prädikat `add/3` in [Übung Rekursive Addition](#)

Suchbaum

Beispiel 5.69

Wissensbank

```
append([ ],L,L).
append([H|T],L,[H|R]):-
    append(T,L,R).
```

```
A = [a|R1]
  = [a|[b|R2]]
  = [a|[b|[c|R3]]]
  = [a|[b|[c|[1,2,3]]]]
  = [a,b,c,1,2,3]
```

```
append([a,b,c],[1,2,3],A)
```

```
  H1 = a, T1 = [b,c]
  L1 = [1,2,3], A = [a|R1]
```

```
append([b,c],[1,2,3],R1)
```

```
  H2 = b, T2 = [c]
  L2 = [1,2,3], R1 = [b|R2]
```

```
append([c],[1,2,3],R2)
```

```
  H3 = c, T3 = [ ]
  L3 = [1,2,3], R2 = [c|R3]
```

```
append([ ],[1,2,3],R3)
```

```
  L4 = [1,2,3], R3 = L4
```



```
A = [a,b,c,1,2,3]
```