



Hochschule für Angewandte
Wissenschaften Hamburg
Hamburg University of Applied Sciences

Intelligente Systeme

– Logik –

Prof. Dr. Michael Neitzke

Diskutieren Sie!

- Was verstehen Sie unter Logik?
- Welche Logiken kennen Sie bereits?
 - Im Detail?
 - Nur dem Namen nach?
- Was unterscheidet diese Logiken, was ist das Gemeinsame?

Logik: Definition

- Nicht formal:
„Die Wissenschaft vom exakten Denken und Schließen“
- Mathematisch:
„Die Wissenschaft von der logischen Struktur **zusammengesetzter Aussagen**. Mit Hilfe **formaler Darstellungen** werden allgemeine Aussagen darüber gewonnen, unter welchen Umständen man aus der **Gültigkeit** gewisser Aussagen (Voraussetzungen) auf die Gültigkeit anderer Aussagen (Folgerungen) **schließen** kann.“ (Schneider: "Lexikon der Informatik und Datenverarbeitung")
- Logik ist Grundlage der **Symbol-verarbeitenden** Künstlichen Intelligenz

Hieraus ergeben sich Fragen

- Was ist eine Aussage?
 - Was ist eine zusammengesetzte Aussage?
 - Wann ist eine Darstellung formal?
 - Wann ist eine Aussage gültig?
 - Was bedeutet es zu „schließen“ oder „Folgerungen“ zu machen?
 - Was ist ein Symbol?
 - Was sind die Merkmale der symbolverarbeitenden KI?
-
- **Diese Fragen sollten Sie am Ende des Logik-Kapitels beantworten können!**

Weshalb Logik?

- Wichtigster Formalismus, um Wissen zu repräsentieren
- Logiken haben beweisbare Eigenschaften, die für die Praxis von hoher Relevanz sind:
 - Korrekt, vollständig, entscheidbar
- Beschreibungslogiken (Teilmengen der Prädikatenlogik) haben große praktische Bedeutung zur Definition von Ontologien
 - Ontologiesprache OWL ist logikbasiert
- Die theoretische Basis von Prolog

Ein Problem aus der Praxis

- Eine Deutschlandkarte soll so eingefärbt werden, dass benachbarte Bundesländer eine unterschiedliche Farbe erhalten.
- **Welches Wissen müsste für eine formale Darstellung modelliert werden?**
- **Wie drückt man dies in Prolog aus?**
- Übrigens kommt man für jede beliebige Landkarte ohne Enklaven mit maximal vier Farben aus. (Vier-Farben-Satz)



- Aus welchen Bundesländern besteht Deutschland und welche sind davon benachbart?
 - `deutschland(SH, HH, NS, HB, MV, SA, BB, BE, NW, HE, TH, SS, RP, SL, BY, BW) :-
benachbart(SH, MV),
benachbart(SH, HH),
...`
- Welche Farben gibt es?
 - `farbe(blau).
farbe(rot).
...`
- Benachbarte Bundesländer dürfen nicht dieselbe Farbe erhalten:
 - `benachbart(X, Y) :- farbe(X), farbe(Y), X \= Y.`
- Es sind natürlich auch andere Modellierungen in Prolog möglich. Dies liegt in der Freiheit des Programmierers.

Wo geht es hin?

- Nach Durcharbeitung des Logik-Kapitels sollten Sie folgende Fragen beantworten können:
 - 1) Prolog ist eine ziemlich enge Umsetzung der Prädikatenlogik. Wie lautet das Prolog-Programm zur Deutschlandkarte in der Notation der Prädikatenlogik?
 - 2) In der Prädikatenlogik gibt es Quantoren. Wie sieht es damit in Prolog aus?
 - 3) Ein Prolog-Programm besteht aus Klauseln. Was sind Klauseln?
 - 4) Prolog verwendet eine spezielle Art von Klauseln, nämlich Horn-Klauseln. Was sind Horn-Klauseln?
 - 5) Es gibt drei Arten von Horn-Klauseln. Was sind deren Eigenschaften? Finden Sie die verschiedenen Arten von Horn-Klauseln in diesem Prolog-Programm.

Anleitung zur Gruppenarbeit (1)

- Bearbeiten Sie zum nächsten Vorlesungstermin die Lernmodule L1, L2, P1, L3, L4 und L5
 - Versuchen Sie die Übungsaufgaben zu lösen. Schauen Sie sich nicht eine Lösung an, ohne es zuvor selbst versucht zu haben.
- Ergänzend empfehle ich für dieses Kapitel insbesondere folgende Literatur:
 - [LC 08]
 - [Ert 09]

Anleitung zur Gruppenarbeit (2)

- Für Prolog gibt es auch Tutorials bzw. Bücher im Internet:
 - Blackburn, Bos, Striegnitz: "Learn Prolog Now!"
<http://www.learnprolognow.org/>
 - Fisher: "prolog :- tutorial"
http://www.csupomona.edu/~jrffisher/www/prolog_tutorial/contents.html
 - Merritt: "Adventure in Prolog"
<http://www.e-booksdirectory.com/details.php?ebook=2429>
- Schicken Sie mir per E-Mail bis einen Tag vor dem nächsten Vorlesungstermin, 12:00h, Antworten auf folgende Fragen:
 - Was haben wir verstanden?
 - Welche Schwierigkeiten haben wir?
 - Welche konkreten Fragen haben wir?

L1: Aussagenlogische Formeln

Lernziele L1

- V1: Aussagenlogische Sätze / Formeln erklären können
- A1: Aussagenlogische Sätze / Formeln bilden können
 - Es muss nicht eine saubere Interpretation natürlicher Sprache trainiert werden!

Erläuterungen

- Die Begriffe **Satz** und **Formel** werden im Folgenden immer synonym verwendet.

Was sind Aussagen?

- Beispiele für Aussagen:
 - In der Sahara regnet es selten.
 - Die Hauptstadt Deutschlands heißt Hamburg.
 - Rudi Carell war witzig.
 - Es gibt ein Leben nach dem Tod.
- Die Aussagenlogik ist zweiwertig: Aussagen können nur wahr oder falsch sein.
 - Innerhalb einer Modellierung durch Aussagenlogik werden Aussagen immer als wahr oder falsch angenommen, unabhängig davon, dass Menschen evt. anderer Meinung sind. (Diese Zuweisung eines Wahrheitswertes wird als Interpretation bezeichnet wie wir später sehen werden.)
- Beispiele für zusammengesetzte Aussagen:
 - Bayern München war Deutscher Meister in 2006 und Werder Bremen der Pokalsieger in 2006.
 - Wenn es regnet, ist die Straße nass.
 - Zusammengesetzt aus „Es regnet.“ und „Die Straße ist nass.“

Welche Sätze sind Aussagen?

1. Heute ist ein schöner Tag.
2. Ob das Wetter wohl morgen auch so gut wird?
3. Das glaube ich nicht!
4. Nun mach schon!
5. Ich beeile mich ja.
6. Der Duden.

Lösung

- 1, 3 und 5 sind Aussagen
 - 2 ist eine Frage, 4 eine Aufforderung, 6 fehlt ein Verb

Symbole der Aussagenlogik

■ Aussagensymbole

- zum Beispiel: P, Q, R, S
- Aussagensymbole stehen für Aussagen über die Welt, z. B. "Die Hauptstadt Deutschlands heißt Hamburg."

■ Wahrheitssymbole

- **wahr, falsch**

■ Junktoren

- \wedge Konjunktion
- \vee Disjunktion
- \neg Negation
- \Rightarrow Implikation
- \Leftrightarrow Äquivalenz

■ Uneigentliche Symbole

- (
-)

Syntax der Aussagenlogik

Satz \rightarrow *AtomarerSatz* | *KomplexerSatz*

AtomarerSatz \rightarrow **wahr** | **falsch** | *Symbol*

Symbol \rightarrow **P** | **Q** | **R** | ...

KomplexerSatz \rightarrow \neg *Satz* |
(*Satz* \wedge *Satz*) |
(*Satz* \vee *Satz*) |
(*Satz* \Rightarrow *Satz*) |
(*Satz* \Leftrightarrow *Satz*)

Semantische / logische Äquivalenzen

Kommutativität von \wedge	$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$
Kommutativität von \vee	$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$
Assoziativität von \wedge	$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$
Assoziativität von \vee	$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$
Eliminierung der doppelten Negation	$\neg(\neg\alpha) \equiv \alpha$
Kontraposition	$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$
Implikationseliminierung	$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$
Bikonditionaleeliminierung	$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$
de Morgan	$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$
de Morgan	$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$
Distributivität von \wedge über \vee	$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$
Distributivität von \vee über \wedge	$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$

Erläuterungen

- Durch die aufgeführten Zusammenhänge werden mögliche Äquivalenzumformungen für aussagenlogische (und übrigens auch prädikatenlogische) Formeln definiert.
- Die griechischen Buchstaben α , β , γ , usw. stehen für beliebige aussagenlogische Formeln.
- Das Symbol \equiv drückt logische Äquivalenz (auch semantische Äquivalenz genannt) aus.
- Der Begriff der semantischen (oder auch logischen) Äquivalenz wird weiter hinten definiert.

Präzedenzen

■ In absteigender Reihenfolge:

■ \neg

■ \wedge

■ \vee

■ \Rightarrow

■ \Leftrightarrow

■ Drücken Sie folgende Aussagen durch aussagenlogische Formeln aus:

1. *Es ist Mai und es wird wärmer.*
2. *Es ist Mai, aber es wird nicht wärmer.*
3. *Im Mai wird es wärmer.*
4. *Wir müssen bald heizen, es sei denn, es wird wärmer.*
5. *Ich heize genau dann (und nur dann) mit Holz, wenn es jemand gibt, der es mir kostenlos liefert und sauber aufstapelt.*

Anmerkung: „es gibt jemand, der es mir kostenlos liefert und sauber aufstapelt“ als eine Aussage modellieren (im Sinne von „jemand hilft“)

Lösung

1. $M \wedge W$
2. $M \wedge \neg W$
3. $M \Rightarrow W$
4. $H \Leftrightarrow \neg W$ oder $\neg (H \Leftrightarrow W)$
(Erläuterung siehe nächste Folie)
5. $H \Leftrightarrow L$

L steht hier für „Es gibt jemand, der es mir liefert und sauber aufstapelt.“

■ Zu 4)

Der Aussage leichter zu entnehmen ist folgender Zusammenhang:

$$(\neg W \Rightarrow H) \wedge (W \Rightarrow \neg H)$$

Das kann man umformen zu:

$$(\neg W \Rightarrow H) \wedge (H \Rightarrow \neg W) \quad (\text{Kontraposition})$$

$$\equiv \neg W \Leftrightarrow H$$

Lernziele L1

- V1: Aussagenlogische Sätze / Formeln erklären können
- A1: Aussagenlogische Sätze / Formeln bilden können
 - Es muss nicht eine saubere Interpretation natürlicher Sprache trainiert werden!

L2: Prädikatenlogische Formeln

Lernziele L2

- V1: Prädikatenlogische Sätze / Formeln erklären können
- A1: Prädikatenlogische Sätze / Formeln bilden können
 - Es muss nicht eine saubere Interpretation natürlicher Sprache trainiert werden!

Übung

In einer Firma arbeiten drei Freunde: ein C++, ein Java- und ein Pascal-Programmierer. Sie heißen Otto, Paul und Heinz. Der C++-Programmierer hat keine Geschwister. Er ist der Jüngste. Heinz, der mit der Schwester von Otto verheiratet ist, ist älter als der Java-Programmierer.

- **Versuchen Sie, die Zusammenhänge mit Hilfe der Aussagenlogik zu modellieren. Sie können aufhören, wenn Sie sich klar gemacht haben, wie die Lösung aussehen müsste.**

Lösung

- Die Aufgabenstellung ist mit der Aussagenlogik modellierbar, allerdings werden die Einschränkungen der Aussagenlogik hierbei deutlich, weil die Lösung sehr umständlich wird.
- Man muss nämlich für die Kombinationen aus Name, Programmiersprache, Name und Alter, Name und Geschwister eigene Aussagensymbole vergeben, also z. B. ein eigenes Symbol, sagen wir OC, für „Otto ist der C++-Programmierer“.

Nachteile der Aussagenlogik

- (In **Blau** eine prädikatenlogische Modellierung)
- Keine explizite Modellierung von Eigenschaften
 - **hat_geschwister(otto)**
- Keine explizite Modellierung von Beziehungen
 - Also kein Zugriff auf Komponenten einer Aussage
 - **programmiersprache(otto, java)**
- Keine Modellierung von Aussagen über die Instanzen einer Klasse
 - **$\forall X : \text{sprache}(X, \text{cplusplus}) \Rightarrow \text{keine_geschwister}(X)$**

Symbole der Prädikatenlogik (1)

- Konstantensymbole
 - beginnen mit einem Kleinbuchstaben
 - `otto`
- Variablensymbole
 - beginnen mit einem Großbuchstaben
 - `x`
- Funktionssymbole
 - beginnen mit einem Kleinbuchstaben
 - `schwester(otto)`
- Wahrheitssymbole
 - `wahr, falsch`

Symbole der Prädikatenlogik (2)

■ Junktoren

■ \wedge	Konjunktion
■ \vee	Disjunktion
■ \neg	Negation
■ \Rightarrow	Implikation
■ \Leftrightarrow	Äquivalenz

■ Variablenquantoren

■ \forall	Allquantor
■ \exists	Existenzquantor

■ Uneigentliche Symbole

■ (
■)

Syntax der Prädikatenlogik

<i>Satz</i>	→ <i>AtomarerSatz</i> <i>KomplexerSatz</i> <i>Quantor Variable, ... Satz</i>
<i>AtomarerSatz</i>	→ wahr falsch <i>Prädikat(Term, ...)</i>
<i>Term</i>	→ <i>Funktion(Term, ...)</i> <i>Konstante</i> <i>Variable</i>
<i>Prädikat</i>	→ verheiratet , bruder , ...
<i>Funktion</i>	→ vater , linkes_bein , ...
<i>Quantor</i>	→ \forall , \exists
<i>Konstante</i>	→ p q john ...
<i>Variable</i>	→ P Q Ehemann ...
<i>KomplexerSatz</i>	→ \neg <i>Satz</i> (<i>Satz</i> \wedge <i>Satz</i>) (<i>Satz</i> \vee <i>Satz</i>) (<i>Satz</i> \Rightarrow <i>Satz</i>) (<i>Satz</i> \Leftrightarrow <i>Satz</i>)

Erläuterungen

- Prädikate und Funktionen sehen syntaktisch gleich aus:
 - Prädikat: männlich(tim)
 - Funktion: freund(tim)
- Semantisch gibt es jedoch Unterschiede:
 - Prädikate beschreiben Eigenschaften oder Beziehungen. Sie können wahr oder falsch sein
 - Tim ist männlich: männlich(tim)
 - Tim und Struppi sind befreundet: befreundet(tim,struppi)
 - Richard und John sind Brüder: bruder(richard,john)
 - Funktionen haben hingegen einen Rückgabewert, der ein Objekt aus der modellierten Domäne ist.
 - bruder(richard) = john
 - freund(tim) = struppi
- Da Prolog nicht funktional ist, werden funktionale Ausdrücke nicht ausgewertet
- In Prolog können Prädikate nicht innerhalb von Klammern stehen. Sonst hätte man es mit der Prädikatenlogik zweiter Stufe zu tun (Prädikate über Prädikate)
- In Prolog müssen Funktionen immer innerhalb von Klammern stehen. Es wird kein Funktionswert berechnet, aber es werden die geschachtelten Strukturen verglichen.

■ Erstellen Sie prädikatenlogische Ausdrücke für folgende Aussagen:

1. Alle Menschen sind sterblich.
2. Nicht alle Menschen sind glücklich.
3. Jemand liebt Uwe.
4. Niemand liebt Uwe.
5. Manchmal ist Uwe einsam.
6. Elefanten haben vor Mäusen Angst.
7. Auf jeden Topf passt ein Deckel.

Lösung (1)

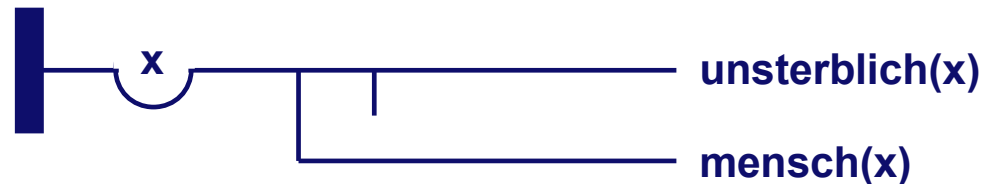
1. $\forall X: \text{mensch}(X) \Rightarrow \text{sterblich}(X)$
2. $\neg \forall X: \text{mensch}(X) \Rightarrow \text{glücklich}(X)$
oder
 $\exists X: \neg (\text{mensch}(X) \Rightarrow \text{glücklich}(X)) \equiv$
 $\exists X: \neg (\neg \text{mensch}(X) \vee \text{glücklich}(X)) \equiv$
 $\exists X: \text{mensch}(X) \wedge \neg \text{glücklich}(X)$
3. $\exists X: \text{liebt}(X, \text{uwe})$
4. $\neg \exists X: \text{liebt}(X, \text{uwe})$ oder $\forall X: \neg \text{liebt}(X, \text{uwe})$
5. $\exists T: \text{zeitraum}(T) \wedge \text{einsam}(\text{uwe}, T)$
6. $\forall E, \forall M: \text{elefant}(E) \wedge \text{maus}(M) \Rightarrow \text{angst}(E, M)$
7. $\forall T, \exists D: \text{topf}(T) \wedge \text{deckel}(D) \Rightarrow \text{passt}(T, D)$

Lösung (2)

- Um zu verstehen, weshalb in den Lösungen eine Implikation eingesetzt wird, hilft es zu überprüfen, ob der Satz in eine Wenn/Dann-Form gebracht werden kann. Also z. B.
„Wenn E ein Elephant ist und M eine Maus, dann hat E vor M Angst.“
- Einen Zeitraum zu modellieren, ist sicherlich ungewohnt und für die meisten nicht naheliegend. Machen Sie sich nichts daraus, wenn Sie darauf nicht gekommen sind. Sie können hier sehen, wie man Aspekte der Zeit, wie z.B. in dem Wort „manchmal“, in der Prädikatenlogik modellieren kann.

Historie

- Als Begründer der modernen Logik gilt Gottlob Frege (1848 – 1925).
- Grundlage für den Prädikatenkalkül 1. Stufe in der Begriffsschrift von 1878:
 - Eine auf drei Operationen basierende Logik: **Implikation**, **Negation**, **All-Quantifizierung**
 - Beispiel:



Zu interpretieren als:

Für alle x gilt: Wenn x ein Mensch ist, dann ist x nicht unsterblich.

In Prädikatenlogik:

$$\forall x : \text{mensch}(x) \Rightarrow \neg \text{unsterblich}(x)$$

- **Versuchen Sie, die grafische Darstellung des Beispiels zu verstehen.**

Lösung

- Der Halbkreis unter dem x entspricht dem Allquantor.
- Die Verzweigung rechts davon entspricht einer Implikation, wobei das Antezedens unten steht und das Sukzedens oben.
 - Antezedens steht links vom Implikationszeichen (Latein: antecedens = vorausgehend)
 - Sukzedens steht rechts vom Implikationszeichen (Latein: succedens = nachfolgend)
- Das kleine Häkchen auf der Linie, die zu $\text{unsterblich}(x)$ führt, ist eine Negation.
- (Diese Notation zu beherrschen, ist nicht prüfungsrelevant.)

Lernziele L2

- V1: Prädikatenlogische Sätze / Formeln erklären können
- A1: Prädikatenlogische Sätze / Formeln bilden können
 - Es muss nicht eine saubere Interpretation natürlicher Sprache trainiert werden!

P1: Umsetzung prädikatenlogischer Formeln durch Prolog

Lernziele P1

- V1: Die Beziehung zwischen einem Prolog-Programm und prädikatenlogischen Formeln erläutern können
- V2: Die Unterscheidung von Fakten, Regeln und Anfragen in einem Prolog-Programm erläutern können
- V3: Die Verwendung von Konstanten, Variablen und Strukturen in Prolog erläutern können
- A1: Prolog-Programme in prädikatenlogische Formeln überführen können

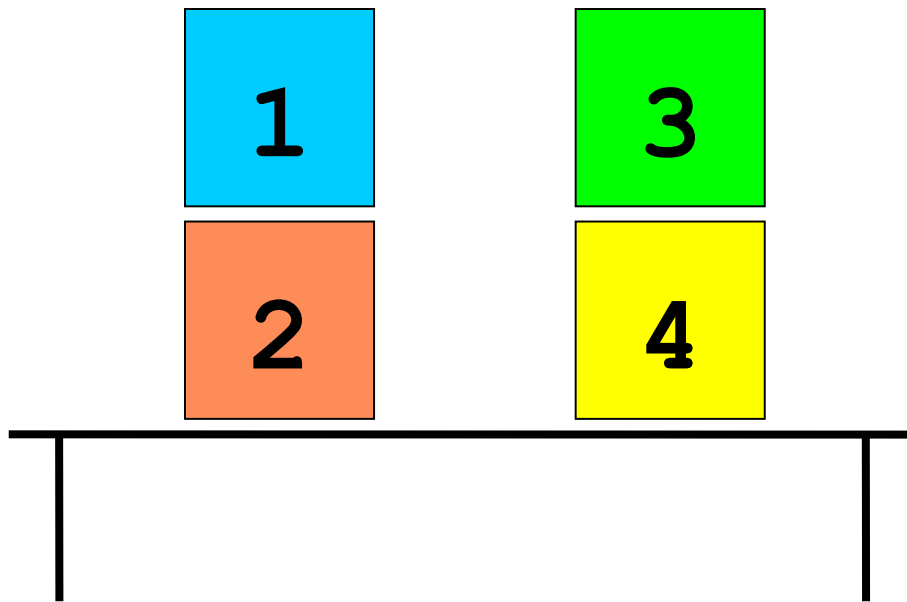
Prolog: PROgramming in LOGic

- Ideal: **Algorithmus = Logik + Kontrolle/Steuerung**
- Ein PROLOG-Programm beschreibt die Logik.
 - In Gestalt von **Horn-Klauseln**
 - Drei Arten von Horn-Klauseln:
 - **Fakten**
 - **Regeln**
 - **Anfragen**
- Das PROLOG-System setzt die Kontrolle um.
 - **SLD-Resolution**

Erläuterungen

- Es wurde in Prolog versucht, die Idealvorstellung der Künstlichen Intelligenz für eine Umsetzung von Algorithmen zu realisieren, nämlich eine Trennung von Anwendungs-abhängiger Wissensbasis und Anwendungs-unabhängigem Inferenzsystem (= Schlussfolgerungssystem). In Prolog wird eine Wissensbasis in prädikatenlogischen Formeln ausgedrückt.
- Was Horn-Klauseln sind, wird weiter hinten erklärt.
- Die SLD-Resolution wird ebenfalls weiter hinten erklärt.

Prolog-Programm: Fakten und Regeln (1)



≡ **Fakten**

`block(block1) .`

`block(block2) .`

`block(block3) .`

`block(block4) .`

`table(table1) .`

`on(block1,block2) .`

`on(block2,table1) .`

`on(block3,block4) .`

`on(block4,table1) .`

Erläuterungen

- Die Prädikate **block** und **table** sind einstellig. Einstellige Prädikate beschreiben Eigenschaften.
- Das Prädikat **on** ist zweistellig. Mehrstellige Prädikate beschreiben Beziehungen.

Prolog-Programm: Fakten und Regeln (2)

≡ Regeln

`above (X,Y) :- block (X) , block (Y) , on (X,Y) .`

`above (X,Y) :- block (X) , table (Y) , on (X,Y) .`

`above (X,Y) :- block (X) , block (Z) , on (X,Z) ,
 above (Z,Y) .`

≡ Und-Verknüpfung durch ","

≡ Oder-Verknüpfung durch ";"

■ entspricht mehreren alternativen Regeln

Anfragen rufen Prolog-Programm auf

```
?- block(block1) .
```

```
yes
```

```
?- block(table1) .
```

```
no
```

```
?- block(X) .
```

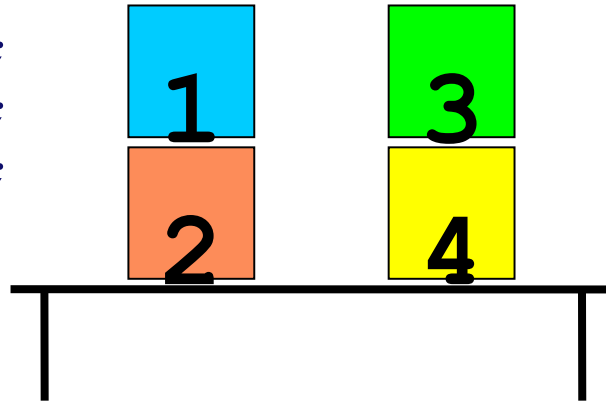
```
X=block1;
```

```
X=block2;
```

```
X=block3;
```

```
X=block4;
```

```
no
```



```
?- on(block1,block2) .
```

```
yes
```

```
?- on(block3,Table) .
```

```
Table=block4;
```

```
no
```

```
?- above(block3,table1) .
```

```
yes
```

```
?- above(block3,X) .
```

```
X=block4;
```

```
X=table1;
```

```
no
```

Überführung in prädikatenlogische Notation

- Variablen sind in Fakten und Regeln allquantifiziert.

`above(X,Y) :- block(X), block(Y), on(X,Y).`

entspricht

$\forall X,Y: \text{block}(X) \wedge \text{block}(Y) \wedge \text{on}(X,Y) \Rightarrow \text{above}(X,Y)$

- Variablen sind in Anfragen existenzquantifiziert.

`?- on(X,block3)` entspricht $\exists X: \text{on}(X,\text{block3})$

- Um zu prüfen, ob diese Formel logisch aus den Formeln des Prolog-Programms folgt, wird sie negiert und der Formelmenge hinzugefügt.
`¬∃ X: on(X,block3)`

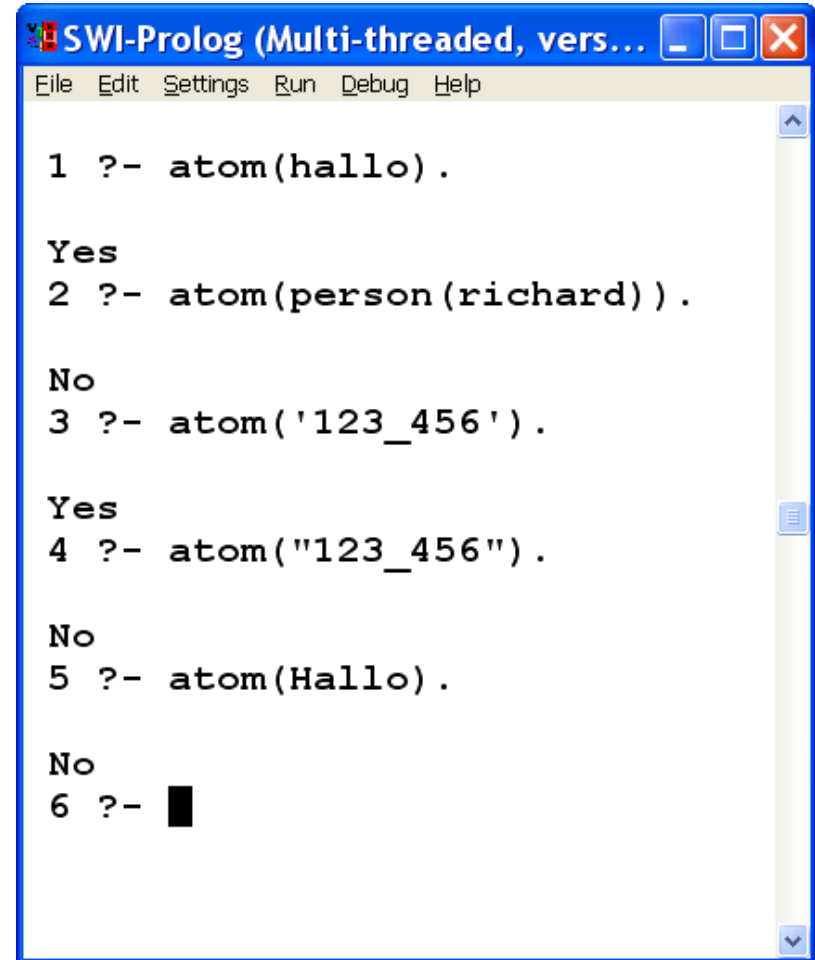
- Der Gültigkeitsbereich von Variablen erstreckt sich nur auf ein Faktum, eine Regel oder eine Anfrage.

Konstanten in Prolog

- Atome
- Zahlen
- Zeichenketten

Atome

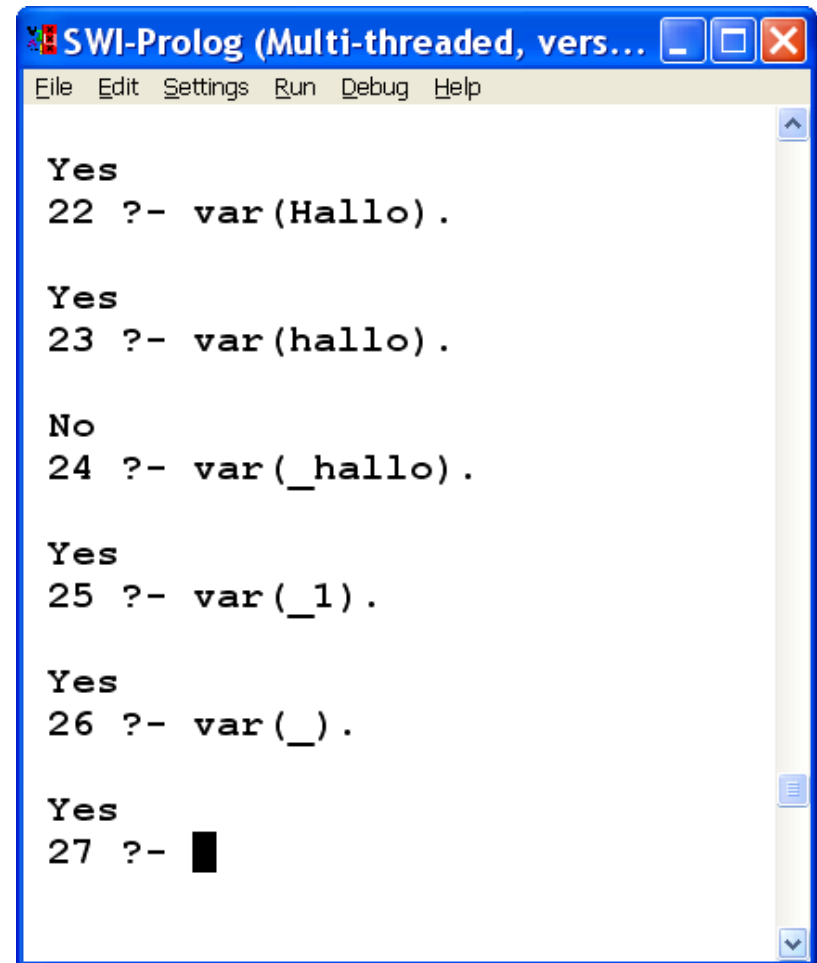
- Zeichenketten, die mit einem Kleinbuchstaben beginnen
- Zeichenketten in einfachen Anführungsstrichen
- Besondere Zeichen
 - Z. B.: `'+', '-',`
 - auch Zeichenketten aus diesen Zeichen



```
SWI-Prolog (Multi-threaded, vers...  
File Edit Settings Run Debug Help  
1 ?- atom(hallo) .  
Yes  
2 ?- atom(person(richard)) .  
No  
3 ?- atom('123_456') .  
Yes  
4 ?- atom("123_456") .  
No  
5 ?- atom(Hallo) .  
No  
6 ?- █
```

Variablen in Prolog

- Beginnen mit einem Großbuchstaben oder einem Unterstrich
- Anonyme Variable: '_'



```
SWI-Prolog (Multi-threaded, vers...  
File Edit Settings Run Debug Help  
  
Yes  
22 ?- var(Hallo).  
  
Yes  
23 ?- var(hallo).  
  
No  
24 ?- var(_hallo).  
  
Yes  
25 ?- var(_1).  
  
Yes  
26 ?- var(_).  
  
Yes  
27 ?- █
```

Erläuterungen

- Die anonyme Variable hat keinen Namen. Sie wird eingesetzt, wenn ihr Wert nicht an anderer Stelle innerhalb einer Prolog-Klausel benötigt wird. Wenn die anonyme Variable an mehreren Stellen innerhalb einer Klausel auftritt, so kann sie an jeder Position mit unterschiedlichen Konstrukten unifiziert werden.

Funktionen in Prolog

- Prolog hat kein funktionales Verhalten
 - Funktionale Ausdrücke werden nicht ausgewertet
- Strukturen bestehen aus einem Funktor und seinen Komponenten
`schachzug(von(a,4), nach(b,6))`.
- Komponenten dürfen auch Strukturen sein.

```
besitzt(frank,  
        buch(autor(lloyd),  
              titel(fundamentals_of_logic_programming)))
```

- Eine Variable darf nicht als Funktor eingesetzt werden!

Erläuterungen

- Ein Funktor ist alles, was vor einer Klammer steht.
- Strukturen sind ein Funktor mit nachfolgendem Klammersausdruck.
- Innerhalb eines Prolog-Programms ist die Menge aller Fakten mit demselben Funktor (und derselben Stelligkeit) das Prädikat.

Prolog-Übung, Teil 1

- Erstellen Sie eine Wissensbasis mit folgenden Inhalten:
 - Frank isst gern:
 - Italienische Gerichte
 - Indische Gerichte, wenn sie mild sind.
 - Pizza ist ein italienisches Gericht.
 - Spaghetti ist ein italienisches Gericht.
 - Dahl ist ein indisches Gericht.
 - Dahl ist mild.
 - Curry ist ein indisches Gericht.
 - Curry ist scharf.
- Stellen Sie Anfragen.

Eine mögliche Lösung

```
likes(frank, Food):-  
    italian(Food).
```

```
likes(frank, Food):-  
    indian(Food), mild(Food).
```

```
italian(pizza).  
italian(spaghetti).  
indian(dahl).  
indian(curry).  
mild(dahl).  
hot(curry).
```

Prolog-Übung, Teil 2

- Erweitern Sie die Wissensbasis um folgende Inhalte:
 - Anna ist gern:
 - Indische Gerichte (egal ob mild oder scharf).
 - Hamburger
- Stellen Sie Anfragen

Lösung

likes(frank, Food) :-
 italian(Food).

likes(frank, Food) :-
 indian(Food), mild(Food).

likes(anna, Food) :-
 indian(Food).

likes(anna, hamburger).

italian(pizza).
italian(spaghetti).
indian(dahl).
indian(curry).
mild(dahl).
hot(curry).

Prolog-Übung, Teil 3

- Erweitern Sie die Wissensbasis um folgende Inhalte:
 - Kurt mag alles, was Anna gern isst, und außerdem noch Pizza.
- Stellen Sie die Anfrage,
 - wer gern Pizza mag.
 - wer gern Dahl mag.
 - wer gern Hamburger und Pizza mag.

Lösung

likes(frank, Food) :-
 italian(Food).

likes(frank, Food) :-
 indian(Food), mild(Food).

likes(anna, Food) :-
 indian(Food).

likes(anna, hamburger).

likes(kurt, pizza).

likes(kurt, Food) :-
 likes(anna, Food).

italian(pizza).

italian(spaghetti).

indian(dahl).

indian(curry).

mild(dahl).

hot(curry).

Anfragen:

?- likes(X, pizza).

?- likes(X, dahl).

?- likes(X, hamburger),
likes(X, pizza).

Lernziele P1

- V1: Die Beziehung zwischen einem Prolog-Programm und prädikatenlogischen Formeln erläutern können
- V2: Die Unterscheidung von Fakten, Regeln und Anfragen in einem Prolog-Programm erläutern können
- V3: Die Verwendung von Konstanten, Variablen und Strukturen in Prolog erläutern können
- A1: Prolog-Programme in prädikatenlogische Formeln überführen können

L3: Konjunktive Normalform

Lernziele L3

- V1: Den Aufbau von Formeln in konjunktiver Normalform erläutern können
- V2: Die Begriffe Literal und Klausel beherrschen
- V3: Die Besonderheiten von Horn-Klauseln erläutern können

- A1: Aussagenlogische und prädikatenlogische Sätze in die KNF überführen können, inklusive Skolemisierung (--> Teil 2 der Klausur)

Was ist eine Normalform, wozu ist sie gut?

- Normalform: Standardisierte Form
- Vorteil: Leichter von einem Computer zu verarbeiten

Konjunktive Normalform

- Ein **Literal** ist ein atomarer Satz (**positives Literal**) oder ein negierter atomarer Satz (**negatives Literal**).

- Hat ein Satz die Struktur

$$(L_{1,1} \vee \dots \vee L_{1,j_1}) \wedge (L_{2,2} \vee \dots \vee L_{2,j_2}) \wedge \dots \wedge (L_{n,1} \vee \dots \vee L_{n,j_n})$$

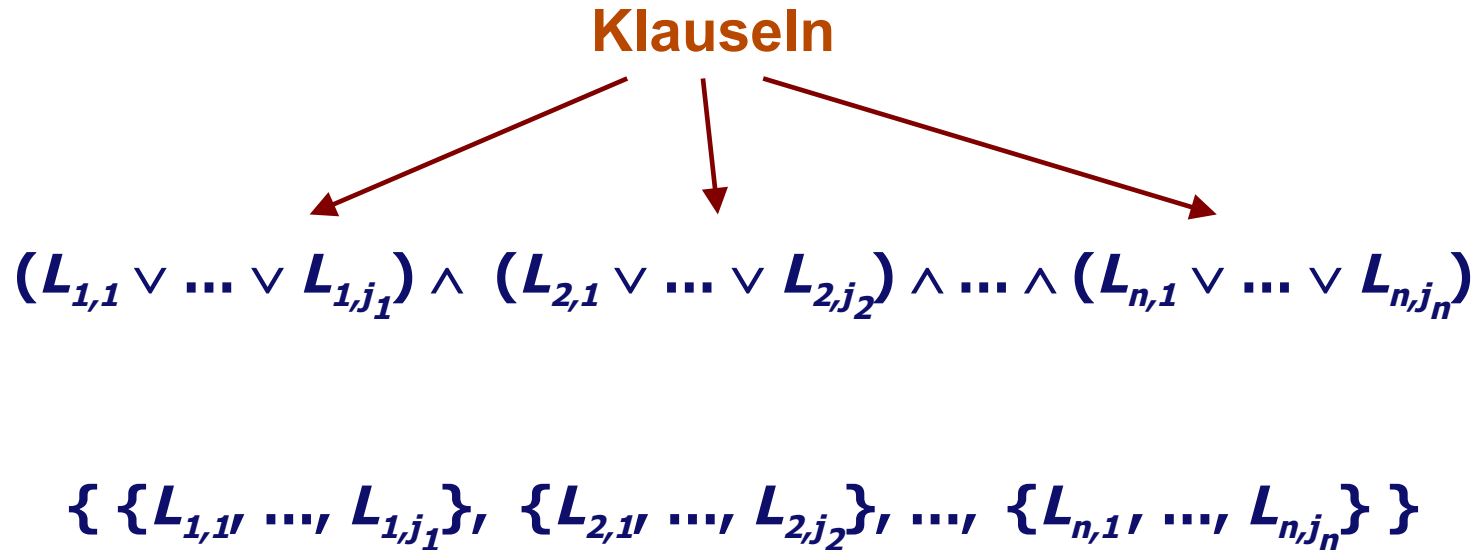
und sind $L_{i,j}$ Literale,

so ist der Satz in **konjunktiver Normalform (KNF)**.

Erläuterungen

- Wenn Sie sich die Definitionen für die Syntax aussagenlogischer und prädikatenlogischer Sätze ansehen, finden Sie auch eine Definition für **atomare Sätze**. Damit ist dann klar, was aussagenlogische bzw. prädikatenlogische **Literale** sind.

KNF: Alternative Schreibweisen



Erläuterungen

- Klauseln sind also disjunktiv verknüpfte Literale.
- Und ein Satz in konjunktiver Normalform besteht aus konjunktiv verknüpften Klauseln.
- Die Mengenschreibweise (die untere Schreibweise) hat den Vorteil, dass keine Sonderzeichen benötigt werden.

Überführung in KNF

- Jeder aussagenlogische oder prädikatenlogische Satz hat mindestens eine äquivalente KNF
- Jeder aussagenlogische bzw. prädikatenlogische Satz ist also in die KNF überführbar!
- **Wie könnte ein standardisierter Ablauf zur Überführung in die KNF für die Aussagenlogik aussehen?**

Lösung

1. Ersetze alle Äquivalenzen $A \Leftrightarrow B$ durch Implikationen $(A \Rightarrow B) \wedge (B \Rightarrow A)$
2. Ersetze alle Implikationen $A \Rightarrow B$ durch $\neg A \vee B$
3. Verschiebe die Negation bis zu den Variablen entsprechend der de Morganschen Gesetze
4. Entferne doppelte Negation
5. Überführe in KNF durch Distributivgesetze

Übung

■ Überführen Sie in die KNF:

$$\neg A \Leftrightarrow (B \wedge C)$$

Lösung

$$\neg A \Leftrightarrow (B \wedge C) \equiv$$

$$(\neg A \Rightarrow (B \wedge C)) \wedge ((B \wedge C) \Rightarrow \neg A) \equiv$$

$$(\neg(\neg A) \vee (B \wedge C)) \wedge (\neg(B \wedge C) \vee \neg A) \equiv$$

$$(\neg(\neg A) \vee (B \wedge C)) \wedge ((\neg B \vee \neg C) \vee \neg A) \equiv$$

$$(A \vee (B \wedge C)) \wedge ((\neg B \vee \neg C) \vee \neg A) \equiv$$

$$(A \vee B) \wedge (A \vee C) \wedge (\neg B \vee \neg C \vee \neg A)$$

Übung

- Führen Sie jetzt die Schritte 1 bis 4 für den folgenden prädikatenlogischen Satz durch:

$$\begin{aligned} &\forall X: (a(X) \wedge b(X) \Rightarrow \\ &\quad c(X, k) \wedge \exists Y \exists Z: (c(Y, Z) \Rightarrow d(X, Y))) \\ &\vee \neg \exists X: e(X) \end{aligned}$$

- Sie müssen folgende allgemeine Zusammenhänge kennen und beim Schritt 3 zuerst die negierten Quantoren umwandeln:

$$\neg \exists X: \alpha \equiv \forall X: \neg \alpha \qquad \neg \forall X: \alpha \equiv \exists X: \neg \alpha$$

Lösung (1)

1. Äquivalenzen in Implikationen: nicht nötig

2. Implikationen in Disjunktionen:

$$\begin{aligned} \forall \mathbf{X}: & \quad (\neg (\mathbf{a}(\mathbf{X}) \wedge \mathbf{b}(\mathbf{X})) \vee \\ & \quad \mathbf{c}(\mathbf{X}, \mathbf{k}) \wedge \exists \mathbf{Y} \exists \mathbf{Z}: (\neg \mathbf{c}(\mathbf{Y}, \mathbf{Z}) \vee \mathbf{d}(\mathbf{X}, \mathbf{Y}))) \\ \vee & \quad \neg \exists \mathbf{X}: \mathbf{e}(\mathbf{X}) \end{aligned}$$

Lösung (2)

3. Negation nach innen:

$$\neg \exists X: \alpha \equiv \forall X: \neg \alpha \qquad \neg \forall X: \alpha \equiv \exists X: \neg \alpha$$

Also erst Umwandlung der negierten Quantoren:

$$\begin{aligned} \forall X: & \left(\neg (a(X) \wedge b(X)) \vee \right. \\ & \left. c(X, k) \wedge \exists Y \exists Z: (\neg c(Y, Z) \vee d(X, Y)) \right) \\ & \vee \forall X: \neg e(X) \end{aligned}$$

Jetzt nach innen:

$$\begin{aligned} \forall X: & \left(\neg a(X) \vee \neg b(X) \vee \right. \\ & \left. c(X, k) \wedge \exists Y \exists Z: (\neg c(Y, Z) \vee d(X, Y)) \right) \\ & \vee \forall X: \neg e(X) \end{aligned}$$

4. Doppelte Negation entfernen: nicht nötig

Weitere Schritte für die Prädikatenlogik

5. Alle Quantoren nach vorn, dabei Umbenennung von Variablen, wenn nötig (*pränexe Normalform*)

$$\forall X \exists Y \exists Z \forall W: (\neg a(X) \vee \neg b(X) \vee \\ c(X, k) \wedge (\neg c(Y, Z) \vee d(X, Y)) \\ \vee \neg e(W))$$

6. Skolemisierung (siehe übernächste Seite)



$$\forall X \forall W: (\neg a(X) \vee \neg b(X) \vee \\ c(X, k) \wedge (\neg c(f(X), g(X)) \vee d(X, f(X))) \\ \vee \neg e(W))$$



Erläuterungen

- Wenn Quantoren nach vorn verschoben werden, erweitert sich ja ihr Skopus und könnte sich damit auf Variablen beziehen, die nur zufällig gleich benannt sind. Damit sich die Bedeutung der Formeln nicht ändert, müssen also ggf. Variablen umbenannt werden. Denken Sie z.B. an folgende Formel, die ausdrücken soll, dass entweder alle Individuen schwarz oder alle Individuen weiß sein sollen. Ohne Umbenennung wären auch gemischte Mengen erlaubt:

$$\forall X: \text{schwarz}(X) \vee \forall X: \text{weiß}(X)$$

Skolemisierung (1)

- Zweck: Beseitigung von Existenzquantoren in prädikatenlogischen Ausdrücken
- Entwickelt durch den norwegischen Mathematiker A. T. Skolem
- Ein Existenzquantor: Ersetzung durch Skolemkonstante

$$\exists x: p(x) \qquad p(a)$$

- Mehrere Existenzquantoren: Mehrere Skolemkonstanten

$$\exists x \exists y: p(x, y) \qquad p(a, b)$$

- Existenzquantor im Gültigkeitsbereich von Allquantoren: Skolemfunktion

$$\forall x \forall y \exists z: p(x, y, z) \qquad \forall x \forall y: p(x, y, f(x, y))$$

Erläuterungen

- Wenn ein **X** existiert, dann kann man diesem **X** auch einen konkreten Namen geben, z. B. **a**
- Wenn sich der Existenzquantor im Skopus von einem oder mehreren Allquantoren befindet, ist die Wahl für die existenzquantifizierte Variable ja von der Wahl der allquantifizierten Variablen abhängig. Denken Sie an: „Für jeden Topf gibt es einen passenden Deckel.“ Die Wahl des Deckels ist von der Wahl des Topfes abhängig. Deshalb müssen in diesen Fällen Skolem-Funktionen und nicht Skolem-Konstanten genommen werden.

Skolemisierung (2)

- Allquantoren im Gültigkeitsbereich von Existenzquantoren:

$$\exists x \forall y \forall z: p(x, y, z)$$

$$\forall y \forall z: p(a, y, z)$$

- Allgemeiner Fall:

$$\forall v \forall w \exists x \forall y \forall z: p(v, w, x, y, z)$$

$$\forall v \forall w \forall y \forall z :$$

$$p(v, w, f(v, w), y, z)$$

- Wichtig: Namen der Skolem-Konstanten und –Funktionen alle verschieden und noch nicht verwendet



Letzte Schritte

7. Entfernung aller Allquantoren:

$$\neg a(X) \vee \neg b(X) \vee \\ (c(X, k) \wedge (\neg c(f(X), g(X)) \vee d(X, f(X)))) \\ \vee \neg e(W)$$

8. Konjunktion von Disjunktionen durch Distributivgesetz:

$$(\neg a(X) \vee \neg b(X) \vee c(X, k) \vee \neg e(W)) \wedge \\ (\neg a(X) \vee \neg b(X) \vee \neg c(f(X), g(X)) \vee \\ d(X, f(X)) \vee \neg e(W))$$

Erläuterungen

- Wenn sowieso alle Formeln allquantifiziert sind, kann man die Quantoren auch ebenso gut weglassen.

Übung

- Überführen Sie folgende Formel in die KNF:

$\neg \exists R: (\exists S \forall T:$

$(p(f(S), g(T)) \Rightarrow q(d, S) \wedge q(g(T), V))$

$\vee \exists T \forall U: \neg q(h(T, U), V))$

$\wedge \neg \exists V: (p(U, g(V)) \vee q(f(V), c))$

Lösung (1)

- Implikation beseitigen:

$\neg \exists R: (\exists S \forall T:$

$(\neg p(f(S), g(T)) \vee q(d, S) \wedge q(g(T), V))$
 $\vee \exists T \forall U: \neg q(h(T, U), V))$

$\wedge \neg \exists V: (p(U, g(V)) \vee q(f(V), c))$

- Negation nach innen:

$\forall R: \neg (\exists S \forall T:$

$(\neg p(f(S), g(T)) \vee q(d, S) \wedge q(g(T), V))$
 $\vee \exists T \forall U: \neg q(h(T, U), V))$

$\wedge \forall V: \neg (p(U, g(V)) \vee q(f(V), c))$

Lösung (2)

- Kopie der vorigen Seite:

$$\begin{aligned} \forall R: & \neg (\exists S \forall T: \\ & (\neg p(f(S), g(T)) \vee q(d, S) \wedge q(g(T), V)) \\ & \vee \exists T \forall U: \neg q(h(T, U), V)) \\ & \wedge \forall V: \neg (p(U, g(V)) \vee q(f(V), c)) \end{aligned}$$

- Negation weiter nach innen:

$$\begin{aligned} \forall R: & (\neg \exists S \forall T: \\ & (\neg p(f(S), g(T)) \vee q(d, S) \wedge q(g(T), V)) \\ & \wedge \neg \exists T \forall U: \neg q(h(T, U), V)) \\ & \wedge \forall V: (\neg p(U, g(V)) \wedge \neg q(f(V), c)) \end{aligned}$$

Lösung (3)

- Kopie der vorigen Seite:

$$\begin{aligned} \forall R: & \quad (\neg \exists S \quad \forall T: \\ & \quad (\neg p(f(S), g(T)) \vee q(d, S) \wedge q(g(T), V)) \\ & \quad \wedge \neg \exists T \quad \forall U: \neg q(h(T, U), V)) \\ \wedge \forall V: & \quad (\neg p(U, g(V)) \wedge \neg q(f(V), c)) \end{aligned}$$

- Negation noch weiter nach innen:

$$\begin{aligned} \forall R: & \quad (\forall S \neg \forall T: \\ & \quad (\neg p(f(S), g(T)) \vee q(d, S) \wedge q(g(T), V)) \\ & \quad \wedge \forall T \neg \forall U: \neg q(h(T, U), V)) \\ \wedge \forall V: & \quad (\neg p(U, g(V)) \wedge \neg q(f(V), c)) \end{aligned}$$

Lösung (4)

- Kopie der vorigen Seite:

$$\begin{aligned} \forall R: & \quad (\neg \exists S \quad \forall T: \\ & \quad (\neg p(f(S), g(T)) \vee q(d, S) \wedge q(g(T), V)) \\ & \quad \wedge \forall T \neg \forall U: \neg q(h(T, U), V)) \\ \wedge \forall V: & \quad (\neg p(U, g(V)) \wedge \neg q(f(V), c)) \end{aligned}$$

- Negation noch weiter nach innen:

$$\begin{aligned} \forall R: & \quad (\forall S \neg \forall T: \\ & \quad (\neg p(f(S), g(T)) \vee q(d, S) \wedge q(g(T), V)) \\ & \quad \wedge \forall T \exists U: q(h(T, U), V)) \\ \wedge \forall V: & \quad (\neg p(U, g(V)) \wedge \neg q(f(V), c)) \end{aligned}$$

Lösung (5)

- Kopie der vorigen Seite:

$$\begin{aligned} \forall R: & \quad (\forall S \neg \forall T: \\ & \quad (\neg p(f(S), g(T)) \vee q(d, S) \wedge q(g(T), V)) \\ & \quad \wedge \forall T \exists U: q(h(T, U), V)) \\ & \wedge \forall V: (\neg p(U, g(V)) \wedge \neg q(f(V), c)) \end{aligned}$$

- Negation noch weiter nach innen:

$$\begin{aligned} \forall R: & \quad (\forall S \exists T: \\ & \quad \neg (\neg p(f(S), g(T)) \vee q(d, S) \wedge q(g(T), V)) \\ & \quad \wedge \forall T \exists U: q(h(T, U), V)) \\ & \wedge \forall V: (\neg p(U, g(V)) \wedge \neg q(f(V), c)) \end{aligned}$$

Lösung (6)

- Kopie der vorigen Seite:

$$\begin{aligned} \forall R: & \quad (\forall S \exists T: \\ & \quad \neg (\neg p(f(S), g(T)) \vee q(d, S) \wedge q(g(T), V)) \\ & \quad \wedge \forall T \exists U: q(h(T, U), V)) \\ & \wedge \forall V: (\neg p(U, g(V)) \wedge \neg q(f(V), c)) \end{aligned}$$

- Negation noch weiter nach innen:

$$\begin{aligned} \forall R: & \quad (\forall S \exists T: \\ & \quad (p(f(S), g(T)) \wedge \neg (q(d, S) \wedge q(g(T), V))) \\ & \quad \wedge \forall T \exists U: q(h(T, U), V)) \\ & \wedge \forall V: (\neg p(U, g(V)) \wedge \neg q(f(V), c)) \end{aligned}$$

Lösung (7)

- Kopie der vorigen Seite:

$$\begin{aligned} \forall R: & \quad (\forall S \exists T: \\ & \quad (p(f(S), g(T)) \wedge \neg (q(d, S) \wedge q(g(T), V))) \\ & \quad \wedge \forall T \exists U: q(h(T, U), V)) \\ & \wedge \forall V: (\neg p(U, g(V)) \wedge \neg q(f(V), c)) \end{aligned}$$

- Negation noch weiter nach innen:

$$\begin{aligned} \forall R: & \quad (\forall S \exists T: \\ & \quad (p(f(S), g(T)) \wedge (\neg q(d, S) \vee \neg q(g(T), V))) \\ & \quad \wedge \forall T \exists U: q(h(T, U), V)) \\ & \wedge \forall V: (\neg p(U, g(V)) \wedge \neg q(f(V), c)) \end{aligned}$$

Lösung (8)

- Kopie der vorigen Seite:

$$\begin{aligned} \forall R: & \quad (\forall S \exists T: \\ & \quad (p(f(S), g(T)) \wedge (\neg q(d, S) \vee \neg q(g(T), V))) \\ & \quad \wedge \forall T \exists U: q(h(T, U), V)) \\ & \wedge \forall V: (\neg p(U, g(V)) \wedge \neg q(f(V), c)) \end{aligned}$$

- Quantoren nach vorn mit Umbenennung:

$$\begin{aligned} \forall R \forall S \exists T \forall W \exists X \forall Y : \\ & \quad (p(f(S), g(T)) \wedge (\neg q(d, S) \vee \neg q(g(T), V))) \\ & \quad \wedge q(h(W, X), V) \\ & \wedge \neg p(U, g(Y)) \wedge \neg q(f(Y), c) \end{aligned}$$

Lösung (9)

- Kopie der vorigen Seite:

$\forall R \ \forall S \ \exists T \ \forall W \ \exists X \ \forall Y :$

$$\begin{aligned} & (p(f(S), g(T)) \wedge (\neg q(d, S) \vee \neg q(g(T), V)) \\ & \quad \wedge q(h(W, X), V) \\ & \wedge \neg p(U, g(Y)) \wedge \neg q(f(Y), c)) \end{aligned}$$

- Skolemisieren:

$\forall R \ \forall S \ \forall W \ \forall Y :$

$$\begin{aligned} & (p(f(S), g(j(R, S))) \\ & \wedge (\neg q(d, S) \vee \neg q(g(j(R, S)), V)) \\ & \wedge q(h(W, k(R, S, W)), V) \\ & \wedge \neg p(U, g(Y)) \wedge \neg q(f(Y), c)) \end{aligned}$$

Lösung (10)

- Kopie der vorigen Seite:

$\forall R \ \forall S \ \forall W \ \forall Y :$

$(p(f(S), g(j(R, S)))$

$\wedge (\neg q(d, S) \vee \neg q(g(j(R, S)), V))$

$\wedge q(h(W, k(R, S, W)), V)$

$\wedge \neg p(U, g(Y)) \wedge \neg q(f(Y), c)$

- Entfernung der Allquantoren:

$p(f(S), g(j(R, S)))$

$\wedge (\neg q(d, S) \vee \neg q(g(j(R, S)), V))$

$\wedge q(h(W, k(R, S, W)), V)$

$\wedge \neg p(U, g(Y)) \wedge \neg q(f(Y), c)$

Horn-Klauseln

- **Horn-Klauseln** sind Klauseln mit maximal einem positiven Literal.

- Der allgemeine Fall: **Regeln**

- $\{\neg B_1, \neg B_2, \dots, \neg B_n, K\}$

- Äquivalent: $B_1 \wedge B_2 \wedge \dots \wedge B_n \Rightarrow K$

- Keine negativen Literale: **Fakten**

- $\{F\}$

- Kein positives Literal: **Anfragen** (das, was es zu beweisen gilt)

- $\{\neg Z_1, \neg Z_2, \dots, \neg Z_m\}$

- äquivalent: $Z_1 \wedge Z_2 \wedge \dots \wedge Z_m \Rightarrow \text{falsch}$

- Zum Namen: Erste Untersuchungen von Alfred Horn, ca. 1950

Erläuterungen

- Regeln haben also ein positives und mindestens ein negatives Literal.
- Fakten haben ein positives und kein negatives Literal.
- Anfragen haben nur negative Literale, mindestens eines.
- Horn-Klauseln unterliegen also einer Einschränkung. Da zwei positive Literale nicht erlaubt sind, kann z. B. $a \vee b$ nicht ausgedrückt werden. Es gibt also Formeln, die nicht in Horn-Klauseln überführbar sind.
- Die Notation mit den geschweiften Klammern enthält Literale, die disjunktiv verknüpft sind, also durch \vee . Durch Umformung erhält man die Darstellung mit einer Implikation. Wenn man bei den Implikationen die rechte und die linke Seite vertauscht und auch das Implikationssymbol umdreht, erhält man eine Prolog-Klausel.

Horn-Klauseln und Prolog

- Ein Prolog-Programm besteht nur aus Horn-Klauseln, und zwar Fakten und Regeln.

- Prolog-Fakt: `on(block2, table1) .`

- Prolog-Regel:

- `above(X, Y) :- block(X), table(Y), on(X, Y) .`

Entspricht in Prädikatenlogik:

`block(X) ∧ table(Y) ∧ on(X, Y) ⇒ above(X, Y)`

- Der Programm-Aufruf ist eine Anfrage.

- Prolog-Anfrage: `?- on(block1, block2) .`

Entspricht in Prädikatenlogik:

`on(block1, block2) ⇒ falsch`

Lernziele L3

- V1: Den Aufbau von Formeln in konjunktiver Normalform erläutern können
 - V2: Die Begriffe Literal und Klausel beherrschen
 - V3: Die Besonderheiten von Horn-Klauseln erläutern können
 - V4: Prolog-Programme und Anfragen als Horn-Klauseln begreifen können.
-
- A1: Aussagenlogische und prädikatenlogische Sätze in die KNF überführen können, inklusive Skolemisierung (--> Teil 2 der Klausur)

L4: Schlussfolgerungssystem und reale Welt

Lernziele L4

- V1: Den Zusammenhang zwischen einem Schlussfolgerungssystem und dem modellierten Ausschnitt der realen Welt erläutern können

Zwischenstand

- Sie wissen jetzt, wie man **syntaktisch korrekte Formeln** in der Aussagenlogik und in der Prädikatenlogik erstellt.
- Sie können auch konkrete Sachverhalte, die in natürlicher Sprache beschrieben sind, in Aussagenlogik bzw. Prädikatenlogik **modellieren**.
- Sie kennen auch die **Prolog-Notation** für Horn-Klauseln als eine konkrete Umsetzung prädikatenlogischer Formeln, die dann auf einem Computer verarbeitet werden können.

Zusammenhang zwischen Modellwelt und Realität

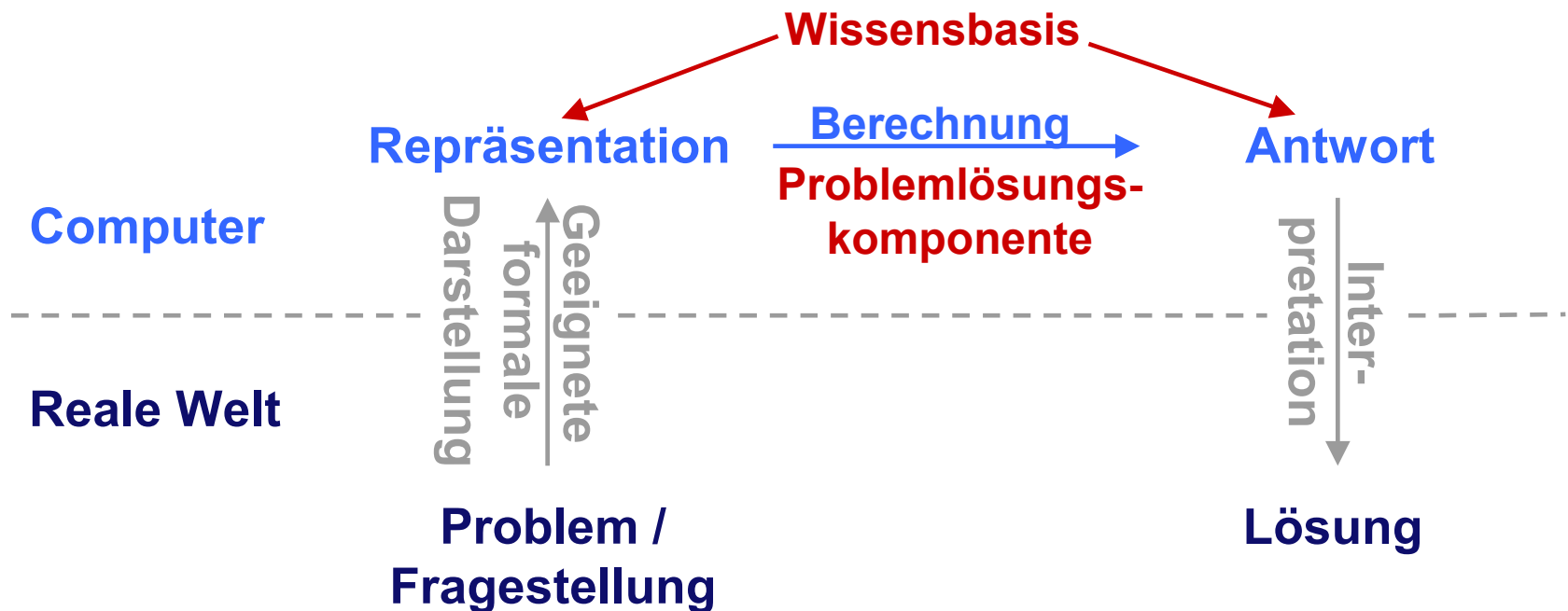
- Verarbeitung auf dem Computer: Schlussfolgerungen
- Schlussfolgerungen haben Bedeutung für reale Welt
- Müssen interpretiert werden

Erläuterungen

- Das heißt, auf dem Computer wird Ihre Modellierung benutzt, um Berechnungen durchzuführen. Diese Berechnungen sind - in der Sprache der Logik - Schlussfolgerungen. Und das, was geschlussfolgert wird, also z. B. die Antwort oder das Ergebnis eines Prolog-Programms, bedeutet ja etwas für die von Ihnen modellierte Aufgabenstellung.
- Also:
 - Sie modellieren eine Anwendung der realen Welt, erstellen also eine Repräsentation eines Teils der realen Welt
 - Diese Repräsentation wird für Schlussfolgerungen benutzt
 - Die Schlussfolgerungen bedeuten etwas für ihre reale Anwendung
 - Die Schlussfolgerungen müssen also interpretiert werden, um die Bedeutung zu erfahren

„Wissensverarbeitung“ in der klassischen KI

- Forderung in der klassischen KI:
Repräsentation von Wissen und Verarbeitung von Wissen trennen



Erläuterungen

- Eine Problemstellung der realen Welt wird durch eine geeignete formale Darstellung repräsentiert. Dies kann z. B. durch aussagenlogische oder prädikatenlogische Sätze geschehen.
- Aus der formalen Beschreibung werden mit Hilfe von Schlussfolgerungsregeln, so genannten Inferenzregeln, neue formale Beschreibungen abgeleitet. Im Falle von logischen Beschreibungen sind dies logische Sätze.
- Alle abgeleiteten Sätze können interpretiert werden, so dass sich für die reale Welt damit ihre Bedeutung ergibt. Abhängig von den Eigenschaften des verwendeten Schlussfolgerungssystems kann die Antwort auf die Fragestellung aus der realen Welt gezielt gesucht oder, wenn möglich, ungezielt neben vielen anderen Schlussfolgerungen erzeugt werden.
- Es ist eine gewünschte Eigenschaft intelligenter Systeme, dass die formalen Beschreibungen der Fragestellung und ihre Antwort von den allgemeinen Schlussfolgerungsregeln getrennt werden. Die formalen Beschreibungen werden auch als Wissensbasis und das Schlussfolgerungssystem als Problemlösungskomponente bezeichnet. Die Problemlösungskomponente ist typischerweise anwendungsunabhängig, d. h. sie kann auf verschiedenen Wissensbasen arbeiten.

Die nächsten Schritte...

1. Bedeutung der Formeln: Semantik

- Wie kann man die **exakte Bedeutung eines Berechnungs-Ergebnisses**, also von Schlussfolgerungen, für die reale Welt erhalten?
- Und was ist überhaupt die **exakte Bedeutung Ihrer logischen Modellierung**?
- Es ist eine Besonderheit und ein großer Vorteil der Logik gegenüber anderen Formen der Wissensrepräsentation, dass die **Bedeutung der Formeln klar definiert** ist.

2. Eigenschaften von Schlussfolgerungssystemen

- Das Schlussfolgerungssystem **sollte** natürlich **korrekt arbeiten...**
- Aber **wann** arbeitet ein Schlussfolgerungssystem **korrekt**?
- **Schlussfolgerungen** betreffen die **syntaktische** Seite
- **Korrektheit** bezieht sich auf die **semantische** Seite
- Eine andere wichtige Eigenschaft ist die **Vollständigkeit**

Lernziele L4

- V1: Den Zusammenhang zwischen einem Schlussfolgerungssystem und dem modellierten Ausschnitt der realen Welt erläutern können

L5: Semantik

- Semantik der Aussagenlogik und Prädikatenlogik
- Interpretation, Modell
- Kontradiktion und Tautologie
- Semantische / logische Folgerung
- Semantische / logische Äquivalenz
- Beweise mittels Wahrheitstabelle

Lernziele L5

- V1: Erklären können, was Semantik für Logiken bedeutet
- V2: Begriffe Interpretation und Modell erläutern können, insbesondere für Aussagenlogik und Prädikatenlogik, auch die Details für die Prädikatenlogik.
- V3: Logische Folgerbarkeit erklären können
- V4: Begriffe Tautologie und Kontradiktion erklären können
- V5: Logische Äquivalenz erklären können
- A1: Beweise durch Wahrheitstabelle durchführen können (--> Teil 2 der Klausur)

Semantik in logischen Sprachen

- Es geht nicht darum, durch menschliche Überlegungen die Bedeutung von Formeln zu erforschen...
- Nein, in einer logischen Modellierung geht es viel mathematischer zu.
- In zweiwertigen Logiken legt die Semantik exakt fest, wie einer Formel ein Wahrheitswert zugewiesen werden kann.
 - Als Ergebnis kommt also letztendlich ein Wahrheitswert heraus
- Konkret: Formeln werden auf Wahrheitswerte abgebildet
- Die Abbildung heißt **Interpretation**

Semantik der Aussagenlogik

- Interpretation:

- Aussagenlogische Sätze erhalten einen Wahrheitswert

- $I: \Phi \rightarrow \{W, F\}$ (Φ : Menge aussagenlogischer Sätze)

- Beispiel

- **R** stehe für "Es regnet.", **S** stehe für "Die Straße ist nass."

- Je zwei mögliche Interpretationen für **R** und **S**, nämlich **W** und **F**

- Also vier mögliche Interpretationen in Kombination

- Überlegen Sie sich, wie man den Wahrheitswert zusammengesetzter Aussagen ermitteln könnte?
 - *Es regnet und die Straße ist nass.*
 - *Wenn es regnet, ist die Straße nass.*

Lösung

- Man benötigt Regeln, wie man aus den Wahrheitswerten der atomaren Sätze den Wahrheitswert zusammengesetzter Sätze bestimmen kann. So müsste der Wahrheitswert zweier UND-verknüpfter Sätze genau dann **W** lauten, wenn die beiden einzelnen Sätze den Wert **W** haben.

Regeln zur Bestimmung des Wahrheitswertes

- φ und ξ seien beliebige aussagenlogische Sätze:
- $I(\text{wahr}) = W$
- $I(\text{falsch}) = F$
- $I(\neg \varphi) = W$, wenn $I(\varphi) = F$, sonst $I(\neg \varphi) = F$
- $I(\varphi \wedge \xi) = W$, wenn $I(\varphi) = W$ und $I(\xi) = W$, sonst $I(\varphi \wedge \xi) = F$
- $I(\varphi \vee \xi) = F$, wenn $I(\varphi) = F$ und $I(\xi) = F$, sonst $I(\varphi \vee \xi) = W$
- $I(\varphi \Rightarrow \xi) = F$, wenn $I(\varphi) = W$ und $I(\xi) = F$, sonst $I(\varphi \Rightarrow \xi) = W$
- $I(\varphi \Leftrightarrow \xi) = W$, wenn $I(\varphi) = I(\xi)$, sonst $I(\varphi \Leftrightarrow \xi) = F$

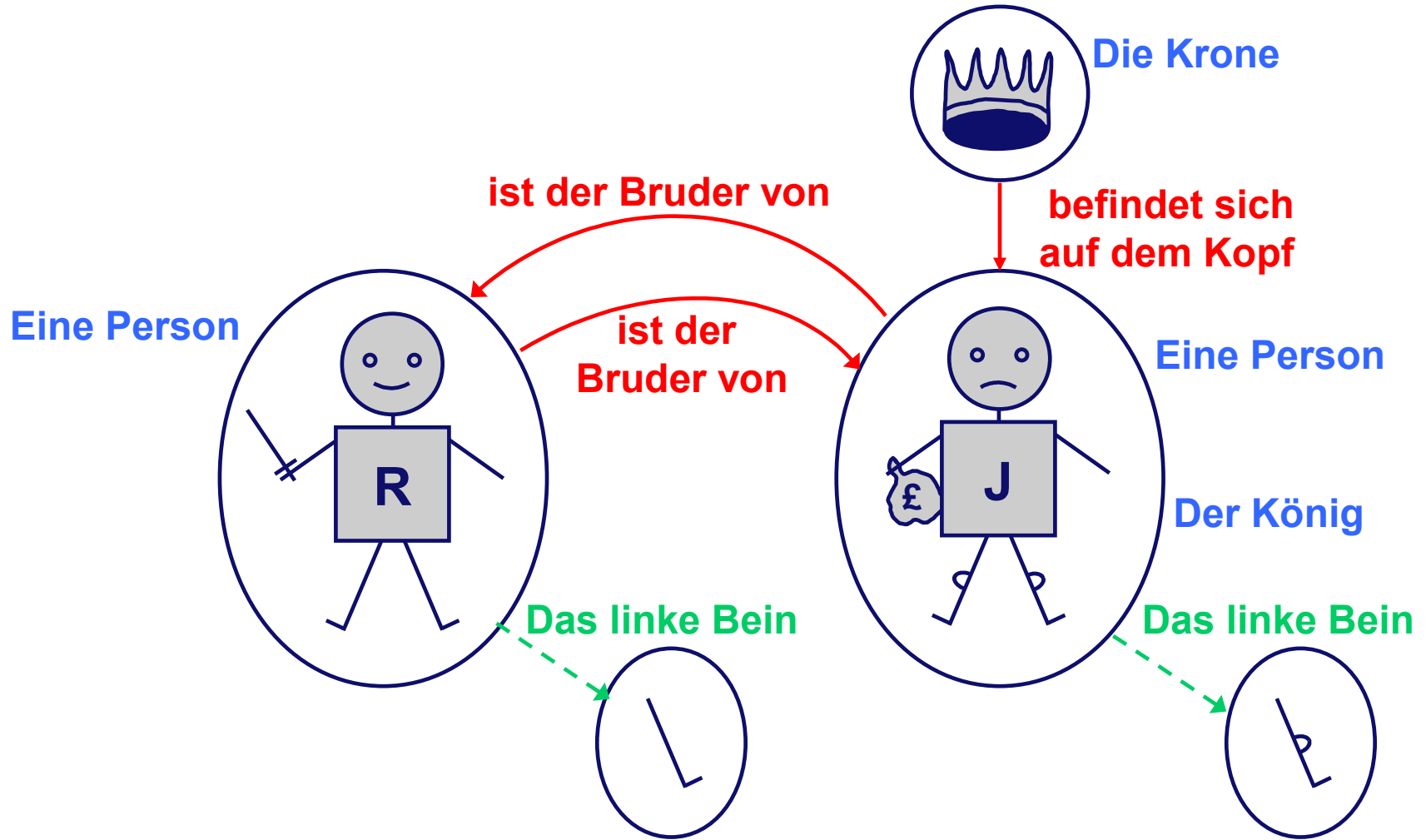
Semantik der Prädikatenlogik

- Prädikatenlogik hat unterschiedliche Symbole:
 - Konstanten
 - Variablen
 - Prädikate
 - Funktionssymbole
- Die Interpretation muss sich auf diese Symbole und Ausdrücke auf Basis dieser Symbole beziehen
 - Prädikatenlogische Ausdrücke werden auf Wahrheitswerte abgebildet
 - Isolierte Symbole werden auf Konstrukte aus der **Domäne** abgebildet

Erläuterungen

- Die Interpretation hat im Falle der Prädikatenlogik eine wesentlich komplexere Abbildung zu leisten als im Fall der einfacheren Aussagenlogik.
- In der Aussagenlogik bestehen Sätze nur aus logisch verknüpften Aussagensymbolen.
- In der Prädikatenlogik gibt es aber Konstanten, Variablen, Funktionen, Prädikate, Quantoren.
- Die Interpretation legt letztendlich auch für die Prädikatenlogik fest, wie einer Formel ein Wahrheitswert zugeordnet wird. Und dazu muss die Interpretation auch festlegen, wie einzelne Bestandteile aus einer Formel abgebildet werden. Die Bestandteile werden auf Objekte oder Konstrukte (z.B. Relationen über Objekten) aus der modellierten Welt, der so genannten Domäne abgebildet.

Beispiel: Bezugswelt (Domäne) mit fünf Objekten



Erläuterungen

- Die hier betrachtete Domäne besteht nur aus fünf Objekten.
- Diese Objekte haben Eigenschaften (blau), Beziehungen zueinander (rot), und funktionale Abhängigkeiten (grün).
 - Beziehungen gelten oder gelten nicht. Funktionale Abhängigkeiten schließen von einem Objekt (z.B. Richard) oder mehreren Objekten auf ein anderes Objekt (z.B. Richards linkes Bein).

Abbildung von Konstanten und Variablen...

- ...auf Domänenobjekte

$$D = \{ \text{Richard}, \text{linkes_bein}, \text{John}, \text{Krone}, \text{rechter_arm} \}$$

- Konstanten und Variablen werden durch eine Interpretation **I** auf Domänenobjekte abgebildet. (Variablen benötigen zunächst einen Wert.)

$$I(\text{richard}) = \text{Richard}$$

$$I(\text{richards_linkes_bein}) = \text{linkes_bein}$$

Erläuterungen

- Konstanten und Variablen werden also nicht auf Wahrheitswerte abgebildet, sondern auf Objekte der Domäne.
- Für eine Variable kann diese Abbildung nur geleistet werden, wenn die Variable instantiiert ist, also eine Konstante als Wert hat.

Interpretation von Prädikaten

- Prädikate werden auf Relationen über den Domänenobjekten abgebildet:





$$I(\text{bruder}) = \{ (\text{Richard}, \text{John}), (\text{John}, \text{Richard}) \}$$

$$I(\text{person}) = \{ (\text{Richard}), (\text{John}) \}$$

- Atomare Sätze werden auf Wahrheitswerte abgebildet,
 - weil jedem n-stelligen Prädikat eine Abbildung $D^n \rightarrow \{W, F\}$ zugeordnet wird.

$$I(\text{bruder}(\text{richard}, \text{john})) = W$$

Erläuterungen

- Prädikatensymbole (oder kurz Prädikate) gehören ja zur Modellierungssprache, nämlich der Prädikatenlogik. Sie stehen für Eigenschaften oder Beziehungen, also einstellige oder mehrstellige Relationen, aus der Domäne.
- Prädikate werden also durch die Interpretation auf Relationen über den Domänenobjekten abgebildet. So wird z. B. das Prädikat **bruder** auf eine Bruderbeziehung zwischen den Domänenobjekten abgebildet. Die Domänenobjekte  und  sind Elemente dieser Beziehung (Relation), die Domänenobjekte  und  hingegen nicht.
- Ein kompletter atomarer Satz wird wie in der Aussagenlogik auch auf einen Wahrheitswert abgebildet.

Interpretation von Funktionssymbolen

- Eine Interpretation ordnet jedem n-stelligen Funktionssymbol eine Abbildung $D^n \rightarrow D$ zu.

$$I(\text{linkes_bein}(\text{john})) = \textcircled{L}$$

wobei: $\text{linkes_bein}(\text{john}) = \text{johns_linkes_bein}$

Erläuterungen

- Funktionssymbole stehen ja für Funktionen. Und diese Funktionen existieren in der Domäne.

Modelle

- Von besonderem Interesse:

Interpretationen, unter denen ein Satz oder eine Menge von Sätzen wahr sind.

- Wenn φ unter I wahr ist, bezeichnet man I als **Modell** von φ .
- Ein Modell für eine Menge von Formeln Φ ist analog definiert.

Erläuterungen

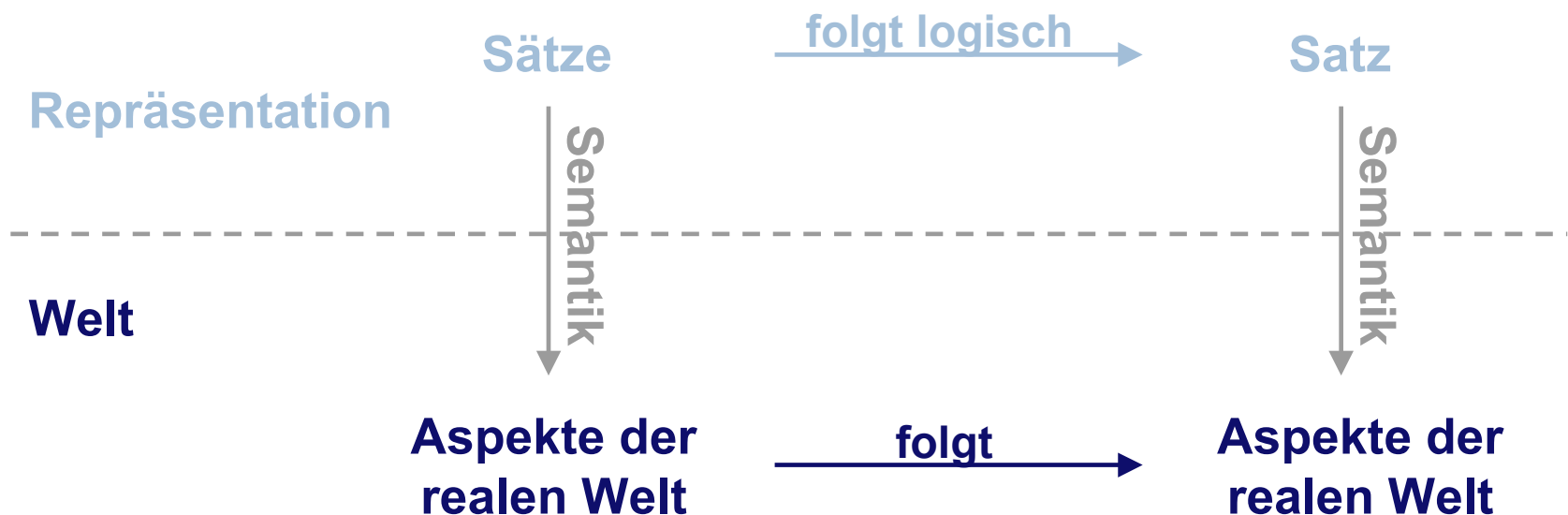
- Modelle sind also genau die Interpretationen von aussagenlogischen oder prädikatenlogischen Sätzen, die als Funktionswert **W** liefern.
- Anschaulich ist ein Modell also eine „sinnvolle“ Zuordnung von Wahrheitswerten. Eine Zuordnung, die die Zusammenhänge der Domäne passend wiedergibt.
- Die Formel in einer Formelmenge sind ja UND-verknüpft. Insofern kann eine Menge von Formeln auch als zusammengesetzte Formel betrachtet werden.

Modelle in der Prädikatenlogik: Höhere Komplexität

- Quantoren und Variablen sind zu beachten:
 - Offene Formel: Wahrheitswert hängt von konkreter Variablenbelegung ab
 - Eine Formel ist offen, wenn mindestens eine Variable frei ist.
 - Geschlossene Formel: Wahrheitswert hängt nicht von Variablenbelegungen ab
 - Eine Formel ist geschlossen, wenn alle Variablen durch Quantoren gebunden sind.
- Existenzquantor ist erfüllt, wenn für mind. eine Variablenbelegung wahr
- Allquantor ist erfüllt, wenn für alle Variablenbelegungen wahr

Logische / semantische Folgerung: Motivation

- Welche weiteren Aussagen folgen logisch aus einer Menge gegebener Aussagen?
 - Interessant für Rückschlüsse auf die reale Welt bzw. die Bezugswelt



(Darstellung aus Russel, Norvig: „Künstliche Intelligenz – Ein moderner Ansatz“)

Erläuterungen

- Bisher haben wir uns mit der Abbildung zwischen der Modellwelt (Repräsentation) und der modellierten Welt (Welt, Domäne) beschäftigt. Letztendlich wollen wir ja eine Problemstellung aus der Domäne in der Modellwelt lösen um dann durch Interpretation der Lösung aus der Modellwelt auf die Lösung in der Domäne zu kommen.
- Dazu müssen wir wissen, welche Zusammenhänge aus der Problemstellung logisch folgen.

Aber...

...was bedeutet es, dass eine Aussage aus einer Menge von Aussagen logisch folgt?

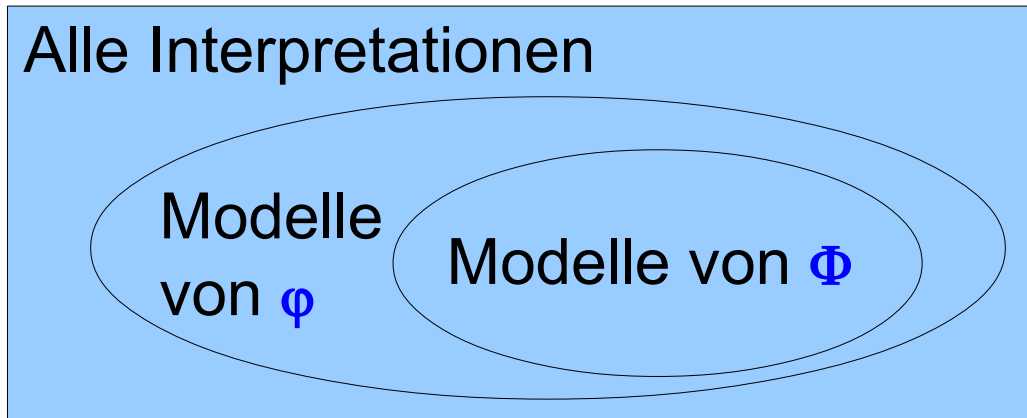
Logische / semantische Folgerung


- Ein Satz φ **folgt logisch** (oder **folgt semantisch** oder einfach nur **folgt**) aus einer Menge von Sätzen Φ , falls jedes Modell von Φ auch Modell von φ ist.

- Man schreibt dafür

$$\Phi \models \varphi$$

- Mengendarstellung:



-  Dass φ aus Φ logisch folgt, heißt nicht,
 - dass φ tatsächlich durch ein bestimmtes Schlussfolgerungsverfahren abgeleitet wird.
 - dass φ aus Φ durch ein bestimmtes Schlussfolgerungsverfahren ableitbar ist.

Erläuterungen

- Die Begriffe „Ableitung“ und „Schlussfolgerungsverfahren“ werden erst weiter hinten erklärt. Aber auf die genannte wichtige Abgrenzung sei bereits hier hingewiesen.
- Der Begriff der logischen Folgerung ist also wieder rein mathematisch definiert, also unabhängig von einer konkreten Umsetzung (durch ein Schlussfolgerungsverfahren). Wenn eine Teilmengen-Beziehung zwischen den Modellen einer Menge von Formeln Φ und den Modellen einer einzelnen Formel ϕ besteht, so folgt ϕ logisch aus Φ .
- ϕ hat also einen allgemeineren Charakter als Φ , weil die Modelle von ϕ eine Obermenge der Modelle von Φ darstellen. So folgt z. B. der Satz $\alpha \vee \beta$ aus dem Satz $\alpha \wedge \beta$. Denn unter den vier möglichen Interpretationen ist nur ein Modell von $\alpha \wedge \beta$, aber gleich drei Modelle von $\alpha \vee \beta$, darunter auch das Modell von $\alpha \wedge \beta$.

Übung

- Wie können Sie beweisen, dass

$$(P \Rightarrow Q) \wedge (Q \Rightarrow R) \models (P \Rightarrow R)$$

Lösung

- Machen Sie sich zunächst klar, was hier ausgedrückt wurde:
 - Immer wenn $(P \Rightarrow Q) \wedge (Q \Rightarrow R)$ gilt, soll auch $(P \Rightarrow R)$ gelten.
 - Jedes Modell von $(P \Rightarrow Q) \wedge (Q \Rightarrow R)$ muss also auch Modell von $(P \Rightarrow R)$ sein.
 - $(P \Rightarrow R)$ darf durchaus mehr Modelle als $(P \Rightarrow Q) \wedge (Q \Rightarrow R)$ haben. Aber nicht umgekehrt.
 - Modelle sind ja die Interpretationen, die **w** liefern.
- **Wie viele verschiedene Interpretationen gibt es überhaupt?**

Lösung

- Insgesamt haben wir es mit drei Symbolen zu tun, nämlich P , Q und R . Jedes Symbol kann **W** oder **F** sein. Also gibt es insgesamt $2^3 = 8$ verschiedene Interpretationen.
- Nun müssen Sie feststellen, für welche dieser 8 Interpretationen $(P \Rightarrow Q) \wedge (Q \Rightarrow R)$ den Wert **W** ergibt und für welche $(P \Rightarrow R)$ den Wert **W** aufweist. Bei einer komplexeren Formel bietet es sich an, zunächst festzustellen, wann Teile der Formel **W** sind, also z. B. $(P \Rightarrow Q)$.
- Diese Art der Beweisführung bezeichnet man als **Wahrheitstabelle**.
- **Führen Sie also als Nächstes den Beweis durch eine Wahrheitstabelle!**

Wahrheitstabelle: $((P \Rightarrow Q) \wedge (Q \Rightarrow R)) \models (P \Rightarrow R)$

- Beweisverfahren für logische Abhängigkeiten auf semantischer Ebene

P	Q	R	$(P \Rightarrow Q)$	$(Q \Rightarrow R)$	$(P \Rightarrow Q) \wedge (Q \Rightarrow R)$	$(P \Rightarrow R)$	$((P \Rightarrow Q) \wedge (Q \Rightarrow R)) \Rightarrow (P \Rightarrow R)$
W	W	W					
W	W	F					
W	F	W					
W	F	F					
F	W	W					
F	W	F					
F	F	W					
F	F	F					

Erläuterungen

- 'Wahr' und 'falsch' kann auch durch '1' und '0' ausgedrückt werden.
- Evt. wundern Sie sich, weshalb in der letzten Spalte das Symbol \Rightarrow zwischen den beiden Sätzen steht.
- Immer wenn $(P \Rightarrow Q) \wedge (Q \Rightarrow R)$ wahr ist, muss ja auch $(P \Rightarrow R)$ wahr sein.
- Anders ausgedrückt:
$$((P \Rightarrow Q) \wedge (Q \Rightarrow R)) \Rightarrow (P \Rightarrow R)$$
muss immer wahr sein.
 - Ein Satz, der immer wahr ist, heißt **Tautologie**.

Lösung

P	Q	R	$(P \Rightarrow Q)$	$(Q \Rightarrow R)$	$(P \Rightarrow Q) \wedge (Q \Rightarrow R)$	$(P \Rightarrow R)$	$((P \Rightarrow Q) \wedge (Q \Rightarrow R)) \Rightarrow (P \Rightarrow R)$
W	W	W	W	W	W	W	W
W	W	F	W	F	F	F	W
W	F	W	F	W	F	W	W
W	F	F	F	W	F	F	W
F	W	W	W	W	W	W	W
F	W	F	W	F	F	W	W
F	F	W	W	W	W	W	W
F	F	F	W	W	W	W	W

Tautologie und Kontradiktion

- **Tautologie:** Satz, der immer wahr ist

- Beispiel:

$$H \Rightarrow W \vee \neg W$$

- **Kontradiktion:** Satz, der nie wahr ist

- Beispiel:

$$\neg A \wedge A$$

Semantische / logische Äquivalenz

- Zwei Sätze/Formeln φ und ξ sind logisch äquivalent, wenn sie dieselben Modelle aufweisen.
 - D. h., eine Interpretation ist Modell für φ genau dann, wenn sie auch Modell für ξ ist.
- Man schreibt: $\varphi \equiv \xi$
- Formale Definition:
 $\varphi \equiv \xi$ genau dann, wenn $\varphi \models \xi$ und $\xi \models \varphi$.

Erläuterungen

- Wenn φ aus ξ logisch folgt und umgekehrt auch, so sind beide Sätze logisch äquivalent.
 - Das heißt: Beide Sätze haben dieselbe Menge an Modellen.
-
- Und jetzt können wir mit einem tieferen Verständnis noch einmal die Seite mit den logischen Äquivalenzen betrachten:

Semantische / logische Äquivalenzen

Kommutativität von \wedge	$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$
Kommutativität von \vee	$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$
Assoziativität von \wedge	$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$
Assoziativität von \vee	$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$
Eliminierung der doppelten Negation	$\neg(\neg\alpha) \equiv \alpha$
Kontraposition	$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$
Implikationseliminierung	$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$
Bikonditionaleeliminierung	$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$
de Morgan	$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$
de Morgan	$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$
Distributivität von \wedge über \vee	$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$
Distributivität von \vee über \wedge	$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$

- **Beweisen Sie unter Verwendung einer Wahrheitstabelle:**

$$(P \vee Q) \models (\neg P \Rightarrow Q)$$

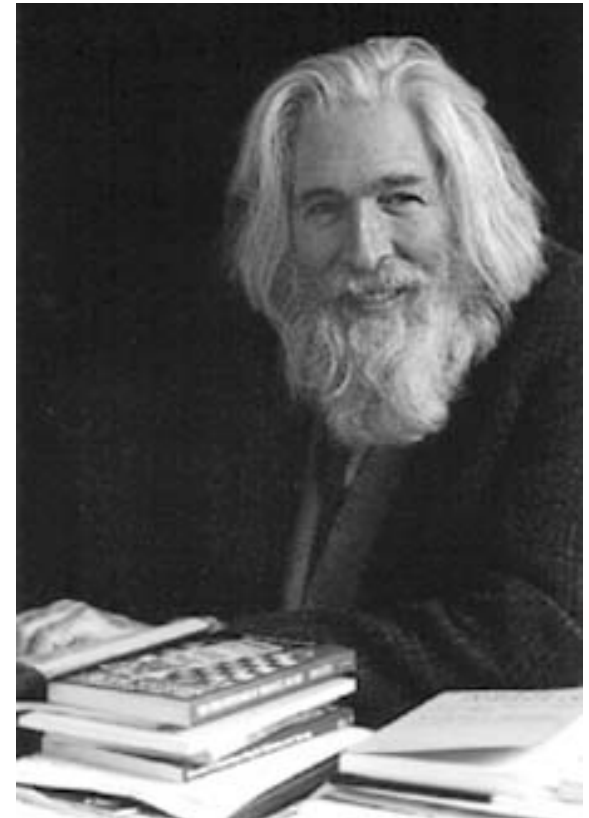
$$(\neg P \Rightarrow Q) \models (P \vee Q)$$

Lösung

P	Q	$(P \vee Q)$	$(\neg P \Rightarrow Q)$	$(P \vee Q) \Leftrightarrow (\neg P \Rightarrow Q)$
W	W	W	W	W
W	F	W	W	W
F	W	W	W	W
F	F	F	F	W

Erläuterungen

- Auf der nächsten Seite finden Sie ein Logikrätsel aus einem der Bücher von Raymond Smullyan.
- Auf diese Rätsel können Sie die hier gelernten Methoden anwenden. So kommen Sie auf systematische und aufgrund der formalen Beschreibungen sehr gut nachvollziehbare Art und Weise zu einer korrekten Lösung.



Optionale Übung: „Ritter und Schurken“

Mit einem so flauen Gefühl, wie er es noch nie zuvor verspürt hatte, betrat der Anthropologe Abercrombie die Insel der Ritter und Schurken. Er wusste, dass diese Insel von höchst erstaunlichen Menschen bevölkert wurde: Die Ritter machten immer nur wahre Aussagen, die Schurken stets falsche. "Wie", fragte sich Abercrombie, "kann ich jemals etwas über diese Insel erfahren, wenn ich nicht weiß, wer lügt und wer die Wahrheit sagt?"

Abercrombie wusste, dass er, bevor er überhaupt irgend etwas in Erfahrung bringen konnte, einen Freund finden musste, jemanden, dessen Aussagen er immer vertrauen konnte. Deshalb dachte er sich, als er die ersten Inselbewohner traf, drei Leute, die Arthur, Bernard und Charles hießen: "Das ist die Chance, einen Ritter für mich zu finden." Abercrombie fragte zunächst Arthur: "Sind Bernard und Charles beide Ritter?" Arthur antwortete mit "Ja". Abercrombie fragte dann: "Ist Bernard ein Ritter?" Zu seiner großen Überraschung antwortete Arthur nun mit "Nein". Ist Charles ein Ritter oder ein Schurke?

≡ **Schritt 1: Drücken Sie die wesentlichen Zusammenhänge in aussagenlogischen Formeln aus**

≡ **Schritt 2: Zeigen Sie anhand einer Wahrheitstabelle, ob Charles Ritter oder Schurke ist.**

Lösung „Ritter und Schurken“

A	B	C	$B \wedge C$	$A \Rightarrow (B \wedge C)$	$A \Rightarrow \neg B$	$\neg A \Rightarrow \neg (B \wedge C)$	$\neg A \Rightarrow B$
W	W	W	W	W	F	W	W
W	W	F	F	F	F	W	W
W	F	W	F	F	W	W	W
W	F	F	F	F	W	W	W
F	W	W	W	W	W	F	W
F	W	F	F	W	W	W	W
F	F	W	F	W	W	W	F
F	F	F	F	W	W	W	F

Erläuterungen

- **A** steht für „Arthur ist ein Ritter“. Daher steht $\neg \mathbf{A}$ dann für „Arthur ist ein Schurke“.
- Arthur trifft ja zwei Aussagen. Wenn Arthur ein Ritter ist, so sind seine beiden Aussagen richtig. Wenn er ein Schurke ist, dann sind sie nicht richtig. Diese vier Zusammenhänge kann man dem Text entnehmen. Sie sind in den vier rechten Spalten der Tabelle wiedergegeben.
- Nur unter einer der acht möglichen Interpretationen sind alle vier Aussagen der Aufgabenstellung wahr. Oder anders ausgedrückt: Nur eine der acht möglichen Interpretationen ist Modell für die in der Aufgabe beschriebene Situation.

Lernziele L5

- V1: Erklären können, was Semantik für Logiken bedeutet
- V2: Interpretation und Modell erläutern können, insbesondere für Aussagenlogik und Prädikatenlogik, auch die Details für die Prädikatenlogik.
- V3: Logische Folgerbarkeit erklären können
- V4: Begriffe Tautologie und Kontradiktion erklären können
- V5: Logische Äquivalenz erklären können
- A1: Beweise durch Wahrheitstabelle durchführen können (--> Teil 2 der Klausur)

L6: Inferenzverfahren

- Inferenzverfahren
- Modus Ponens, Modus Tollens
- Ableitbarkeit
- Vollständigkeit, Korrektheit eines Inferenzverfahrens
- Resolution

Lernziele L6

- V1: Erklären können, was ein Inferenzverfahren und speziell ein Beweisverfahren ist
- V2: Modus Ponens und Modus Tollens erklären können
- V3: Ableitbarkeit erklären können
- V4: Korrektheit und Vollständigkeit erklären können
- V5: Resolution und ihre Eigenschaften erklären können

- A1: Modus Ponens und Modus Tollens anwenden können (--> Teil 2 der Klausur)
- A2: Resolution in der Aussagenlogik anwenden können (--> Teil 2 der Klausur)

Erläuterungen

- Durch den Begriff der Semantik wissen wir jetzt ganz exakt, was eine logische Formel bezogen auf unsere Domäne bedeutet.
- Und wir wissen auch, dass die Lösung für unsere Problemstellung aus der Problemstellung logisch folgen muss. Dabei wissen wir auch ganz exakt, was es bedeutet, dass eine Lösung logisch aus der Problemstellung folgt.
- Wir können sogar mit Hilfe einer Wahrheitstabelle beweisen, dass eine Lösung logisch aus einer Problemstellung folgt.
 - Aber das ist sehr aufwändig und funktioniert daher für komplexere Aufgabenstellung nicht.
- Wir brauchen also ein Verfahren, dass uns die Lösung berechnet. Also ein Schlussfolgerungsverfahren, auch Inferenzverfahren genannt.
- In diesem Abschnitt geht es primär ein spezielles Inferenzverfahren, nämlich die Resolution.
- Nur zum Vergleich wird noch eine andere, sehr einfache Form des Schlussfolgerns vorgestellt, nämlich Modus Ponens und Modus Tollens.
- Das Berechnen von Sätzen, die logisch aus einer Menge von Sätzen folgen, nennt man Ableiten. Es ist nun sehr wichtig, einen bedeutsamen Unterschied zu verstehen: Ableiten ist ein rein syntaktischer Vorgang. Die „Folgt logisch“-Beziehung ist hingegen eine rein semantische Betrachtung

Was ist ein Inferenzverfahren?

- Ein Schlussfolgerungsverfahren
- Verfahren, um auf Basis syntaktischer Manipulationen neue Sätze aus gegebenen Sätzen zu erzeugen, die vernünftigerweise aus den gegebenen Sätzen **logisch folgen**
- Oder ein Verfahren, dass durch syntaktische Manipulationen gezielt überprüft, ob ein gegebener Satz aus einer Menge gegebener Sätze logische folgt.
- Es gibt also zwei Typen von Inferenzregeln
 - Neue Sätze „blind“ ableiten
 - Alles ableiten, was ableitbar ist, bis das gewünschte Resultat gefunden wird
 - Beweisverfahren
 - Vorgegebene Sätze gezielt überprüfen
- Manchmal besteht ein Inferenzverfahren nur aus einer Inferenzregel

Erläuterungen

- Ein Inferenzverfahren ist ein Schlussfolgerungsverfahren, dass aus einer Menge gegebener Sätze neue Sätze ableitet. Der Ableitungsvorgang ist eine rein syntaktische Manipulation und kann daher leicht von einem Computer ausgeführt werden. Ein Inferenzverfahren besteht aus einer oder mehreren Inferenzregeln. Eine Inferenzregel gibt exakt vor, wie die Gestalt eines oder mehrerer gegebener Sätze aussehen muss, damit die Regel anwendbar ist. Die Anwendung einer Inferenzregel führt dann dazu, dass ein neuer Satz erzeugt wird und der Gesamtmenge der Sätze hinzugefügt wird.
- Wir lernen in diesem Kapitel die beiden Inferenzregeln Modus Ponens und Modus Tollens kennen und das Inferenzverfahren der Resolution, das nur eine Inferenzregel umfasst, nämlich die Resolutionsregel.
- Es gibt Inferenzverfahren, die aus einer Menge vorgegebener Sätze blind alles ableiten, was ableitbar ist. Und es gibt Inferenzverfahren, die gezielt einen bestimmten Satz prüfen.

Inferenzregeln Modus Ponens, Modus Tollens

- Gegeben eine Implikation: *Wenn es regnet, ist die Straße nass.*

$$R \Rightarrow N$$

- Gegeben sei außerdem eine der vier folgenden Aussagen. **In welchen der vier Fälle ist auf Basis der Aussage und der oben stehenden Implikation eine Schlussfolgerung möglich?**

- | | |
|-------------------------------------|----------|
| ■ <i>Die Straße ist nass.</i> | N |
| ■ <i>Die Straße ist nicht nass.</i> | $\neg N$ |
| ■ <i>Es regnet.</i> | R |
| ■ <i>Es regnet nicht.</i> | $\neg R$ |

Modus Ponens, Modus Tollens i. d. Aussagenlogik

■ Modus Ponens:

$$\begin{array}{c} R \\ R \Rightarrow N \\ \hline N \end{array}$$

■ Modus Tollens:

$$\begin{array}{c} \neg N \\ R \Rightarrow N \\ \hline \neg R \end{array}$$

Erklärungen

- Modus Ponens und Modus Tollens sind Inferenzregeln, die blind entsprechend ihrer Vorbedingungen das ableiten, was ihren Nachbedingungen entspricht. Es liegt also kein zielgerichtetes Vorgehen vor.
- Über der horizontalen Linie stehen die Vorbedingungen, unter der Linie die Nachbedingung der Regeln.
- Zu den Beispielen:
 - Modus Ponens: Wenn ich weiß, dass R gilt („Es regnet.“) und ich weiß, dass R N impliziert („Wenn es regnet, ist die Straße nass.“), dann kann ich schlussfolgern, dass N gilt („Die Straße ist nass.“).
 - Modus Tollens: Wenn ich weiß, dass N nicht gilt („Die Straße ist nicht nass.“) und ich weiß, dass R N impliziert („Wenn es regnet, ist die Straße nass.“), dann kann ich schlussfolgern, dass R nicht gelten kann („Es regnet nicht.“).

Opt. Übung: Gerichtsprozess (Ritter u. Schurken)

Diese Übung ist optional, geringe Prüfungsrelevanz!

... "Im Prozess geht es um ein gestohlenes Pferd. Es gibt vier Verdächtige – Andrew, Bruce, Clayton und Edward. Dem Gericht ist unzweifelhaft bekannt, dass genau einer dieser vier der Dieb ist. Die ersten drei sind bereits verhaftet worden, doch Edward konnte nirgendwo aufgestöbert werden. Der Prozess wird deshalb ohne ihn geführt werden müssen". ...

... Als erstes ließ der Richter seinen Hammer fallen und stellte die relevante Frage: "Wer hat das Pferd gestohlen?" Er erhielt darauf von den Angeklagten die folgenden Antworten:

Andrew: Bruce hat das Pferd gestohlen.

Bruce: Clayton hat das Pferd gestohlen.

Clayton: Edward ist es, der das Pferd gestohlen hat.

Doch dann, sehr unerwartet, sagte einer der drei Angeklagten: "Die beiden anderen lügen."

Der Richter überlegte eine Weile, wies dann auf einen der drei und sprach: "Es ist offensichtlich, dass du das Pferd nicht gestohlen hast; du darfst das Gericht deshalb verlassen." ... **Wer darf gehen? Tipp: Ritter sagen immer die Wahrheit, Schurken lügen immer. Beides muss in Betracht gezogen werden. (Anleitung auf der folgenden Seite, nur im Notfall)**

Lösung: Vorgehensweise

- 1) Zunächst geht es wieder darum, Aussagen der Aufgabenstellung zu modellieren. Alle drei anwesenden Verdächtigen machen ja eine Aussage bezüglich des jeweils nächsten Verdächtigen. Abhängig davon, ob sie Ritter oder Schurke sind, ergeben sich insgesamt sechs aussagenlogische Sätze.
- 2) Abhängig davon, wer von den dreien dann den anderen beiden das Lügen vorwirft, und abhängig davon ob er Ritter oder Schurke ist, ergeben sich weitere sechs aussagenlogische Sätze.
- 3) Außerdem muss man noch ausdrücken, dass nur einer der Dieb sein kann. Wenn also Andrew der Dieb ist, können die anderen drei dies nicht sein, usw. Daraus ergeben sich insgesamt vier aussagenlogische Sätze.
- 4) Jetzt kann man den Modus Ponens für die sechs unterschiedlichen Annahmen, nämlich dass entweder Andrew oder Bruce oder Clayton die unerwartete Aussage machte und dabei entweder die Wahrheit sagte oder log, anwenden. Welche Annahme zutrifft, weiß man natürlich nicht. Es fällt aber auf, dass in allen Fällen Andrew unschuldig wäre.

Lösung: Schritt 1)

Zunächst geht es wieder darum, Aussagen der Aufgabenstellung zu modellieren. Alle drei anwesenden Verdächtigen machen ja eine Aussage bezüglich des jeweils nächsten Verdächtigen. Abhängig davon, ob sie Ritter oder Schurke sind, ergeben sich insgesamt sechs aussagenlogische Sätze.

Wir setzen Aussagensymbole:

- **A** für Andrew ist Ritter (**B**, **C** entsprechend)
- Also $\neg A$ für Andrew ist Schurke
- **Ad** für Andrew ist Dieb, $\neg Ad$ für Andrew ist nicht Dieb

Und jetzt die Umsetzung der drei Aussagen, wer der Dieb ist:

$A \Rightarrow Bd$ **Wenn Andrew Ritter ist, dann ist Bruce der Dieb.**

$\neg A \Rightarrow \neg Bd$ **Wenn Andrew Schurke ist, dann ist Bruce nicht der Dieb.**

$B \Rightarrow Cd$

$\neg B \Rightarrow \neg Cd$

$C \Rightarrow Ed$

$\neg C \Rightarrow \neg Ed$

Lösung: Schritt 2)

Abhängig davon, wer von den dreien dann den anderen beiden das Lügen vorwirft, und abhängig davon ob er Ritter oder Schurke ist, ergeben sich weitere sechs aussagenlogische Sätze.

Wir setzen Aussagensymbole:

Aa für Andrew hat die Aussage über die beiden anderen gemacht (**Ba**, **Ca**, entsprechend)

Und jetzt die Umsetzung der drei Möglichkeiten dieser Aussage. Grundsätzlich können ja A, B oder C die Aussage gemacht haben. Und alle drei können dabei die Wahrheit sagen oder lügen, also Ritter oder Schurke sein. Es ergeben sich also sechs mögliche Interpretationen. Dafür stehen die Farben. (Interpretationen, die von vornherein mit der Aufgabenstellung im Konflikt stehen, werden hier nicht betrachtet):

$$\mathbf{A \wedge Aa \Rightarrow \neg B \wedge \neg C}$$

Wenn A Ritter ist und die Aussage gemacht hat, sind B und C Schurken

$$\neg \mathbf{A \wedge Aa \Rightarrow B \vee C}$$

Wenn A Schurke ist und die Aussage gemacht, stimmt es nicht, dass B und C Schurken sind. Also muss B oder C Ritter sein (oder beide)

$$\mathbf{B \wedge Ba \Rightarrow \neg A \wedge \neg C}$$

Blau für B ist Ritter und hat die Aussage gemacht

$$\neg \mathbf{B \wedge Ba \Rightarrow A \vee C}$$

Grün für B ist Schurke und hat die Aussage gemacht

$$\mathbf{C \wedge Ca \Rightarrow \neg A \wedge \neg B}$$

$$\neg \mathbf{C \wedge Ca \Rightarrow A \vee B}$$

Lösung: Schritt 3)

Außerdem muss man noch ausdrücken, dass nur einer der Dieb sein kann. Wenn also Andrew der Dieb ist, können die anderen drei dies nicht sein, usw. Daraus ergeben sich insgesamt vier aussagenlogische Sätze.

$Ad \Rightarrow \neg Bd \wedge \neg Cd \wedge \neg Ed$

Wenn A der Dieb ist, sind B, C und E nicht Dieb.

$Bd \Rightarrow \neg Ad \wedge \neg Cd \wedge \neg Ed$

Wenn B der Dieb ist...

B ist Dieb, wenn A Ritter ist. Deshalb die Farbe rot.

$Cd \Rightarrow \neg Ad \wedge \neg Bd \wedge \neg Ed$

$Ed \Rightarrow \neg Ad \wedge \neg Bd \wedge \neg Cd$

Lösung: Schritt 4)

Jetzt kann man den Modus Ponens für die sechs unterschiedlichen Annahmen, nämlich dass entweder Andrew oder Bruce oder Clayton die unerwartete Aussage machte und dabei entweder die Wahrheit sagte oder log, anwenden. Welche Annahme zutrifft, weiß man natürlich nicht. Es fällt aber auf, dass in allen Fällen Andrew unschuldig wäre.

$Bd \wedge \neg Cd \wedge \neg Ed \wedge \neg Ad$ **Wenn A Ritter ist und die Implikation aus Schritt 1 gilt, dann wird durch Modus Ponens abgeleitet, dass B der Dieb ist. Wenn B Dieb ist und die Implikation aus Schritt 3 gilt, dann sind (durch Modus Ponens abgeleitet) C, E und A nicht der Dieb.**

$\neg Bd \wedge (Cd \vee Ed) \wedge \neg Ad$ **Wenn A Schurke ist und die Implikation aus Schritt 1 gilt, dann ist B nicht der Dieb (Modus Ponens). Wenn A Schurke ist und die Aussage gemacht hat und die Implikation aus Schritt 2 gilt, dann ist B oder C Ritter (oder beide) (durch Modus Ponens abgeleitet). Wenn B Ritter ist und die Implikation aus Schritt 1 gilt, dann ist C der Dieb (Modus Ponens). Entsprechend ist E der Dieb, wenn C Ritter ist. Also ist C oder E der Dieb. In beiden Fällen ist A nicht der Dieb (auch durch Modus Ponens).**

$Cd \wedge \neg Ad \wedge \neg Bd \wedge \neg Ed$

$\neg Cd \wedge (Bd \vee Ed) \wedge \neg Ad$

$Ed \wedge \neg Ad \wedge \neg Bd \wedge \neg Cd$

$\neg Ed \wedge (Bd \vee Cd) \wedge \neg Ad$

In den möglichen sechs Interpretationen ist A nicht der Dieb.

Lösung Überblick

$$A \Rightarrow Bd$$

$$\neg A \Rightarrow \neg Bd$$

$$A \wedge Aa \Rightarrow \neg B \wedge \neg C$$

$$\neg A \wedge Aa \Rightarrow B \vee C$$

$$Bd \wedge \neg Cd \wedge \neg Ed \wedge \neg Ad$$

$$\neg Bd \wedge (Cd \vee Ed) \wedge \neg Ad$$

$$B \Rightarrow Cd$$

$$\neg B \Rightarrow \neg Cd$$

$$B \wedge Ba \Rightarrow \neg A \wedge \neg C$$

$$\neg B \wedge Ba \Rightarrow A \vee C$$

$$Cd \wedge \neg Ad \wedge \neg Bd \wedge \neg Ed$$

$$\neg Cd \wedge (Bd \vee Ed) \wedge \neg Ad$$

$$C \Rightarrow Ed$$

$$\neg C \Rightarrow \neg Ed$$

$$C \wedge Ca \Rightarrow \neg A \wedge \neg B$$

$$\neg C \wedge Ca \Rightarrow A \vee B$$

$$Ed \wedge \neg Ad \wedge \neg Bd \wedge \neg Cd$$

$$\neg Ed \wedge (Bd \vee Cd) \wedge \neg Ad$$

$$Ad \Rightarrow \neg Bd \wedge \neg Cd \wedge \neg Ed$$

$$Bd \Rightarrow \neg Ad \wedge \neg Cd \wedge \neg Ed$$

$$Cd \Rightarrow \neg Ad \wedge \neg Bd \wedge \neg Ed$$

$$Ed \Rightarrow \neg Ad \wedge \neg Bd \wedge \neg Cd$$

Modus Ponens in der Prädikatenlogik

- Wie kann man aus

$\forall X: (\text{mensch}(X) \Rightarrow \text{sterblich}(X)) \wedge \text{mensch}(\text{sokrates})$

auf $\text{sterblich}(\text{sokrates})$ schließen?

- Vorgeschalteter Schritt:

All-Instanziierung / Universelle Instanziierung

$\forall X: p(X)$

$p(a)$

a sei eine beliebige Konstante

Zusatzinformation

Erläuterungen

- Modus Ponens und Modus Tollens sind auch in der Prädikatenlogik anwendbar. Hierfür muss eine allquantifizierte Implikation mit einem konkreten Individuum instantiiert werden.
- Diese Folie ist nicht prüfungsrelevant.

Ableitbarkeit

- Φ : Menge aussagenlogischer Formeln
- φ : Weitere aussagenlogische Formel
- φ ist aus Φ **ableitbar**,
wenn es eine endliche Folge von Inferenzschritten gibt,
so dass man von Φ zu φ gelangt.
- Man schreibt dafür $\Phi \vdash \varphi$

Erläuterungen

- Wie man sieht, ist die Ableitbarkeit nur durch die mechanische Anwendbarkeit von Inferenzregeln gegeben, also rein syntaktischer Natur.

Korrektheit, Vollständigkeit

- Ein Inferenzverfahren heißt **korrekt**, wenn es
 - nur Formeln aus einer Menge von Formeln Φ ableitet, die auch wirklich logisch aus Φ folgen
 - Formal: Falls $\Phi \vdash \varphi$ gilt, dann auch $\Phi \models \varphi$.
 - Natürlichsprachlich: Wenn eine Formel abgeleitet wird, dann folgt sie auch logisch aus der Menge der Formeln.
- Ein Inferenzverfahren heißt **vollständig**, wenn es
 - alle Formeln aus Φ ableiten kann, die logisch aus Φ folgen.
 - Formal: Falls $\Phi \models \varphi$ gilt, dann auch $\Phi \vdash \varphi$.
 - Natürlichsprachlich: Wenn eine Formel logisch aus einer Formelmenge folgt, dann kann sie auch vom Inferenzverfahren abgeleitet werden.

Erläuterungen

- Ein Inferenzverfahren bzw. eine Inferenzregel sollte nur solche Sätze ableiten, die aus der Menge der vorgegebenen Sätze logisch folgen. Oder anders ausgedrückt: Aus wahren Sätzen sollten nur wahre und keine falschen Sätze abgeleitet werden. Diese Eigenschaft eines Inferenzverfahrens oder einer Inferenzregel wird als Korrektheit bezeichnet.
- Außerdem wäre es gut, wenn ein Inferenzverfahren **alle** Sätze ableiten könnte, die logisch aus der Menge der vorgegebenen Sätze logisch folgen. Diese Eigenschaft wird als Vollständigkeit bezeichnet.
- Ob ein Satz aus einer Menge von Sätzen logisch folgt oder nicht, ergibt sich aus den Modellen der Sätze, also daraus, unter welchen Interpretationen die Sätze wahr sind und ob hierbei eine Teilmengenbeziehung besteht. (Näheres hierzu haben wir im Abschnitt über Semantik behandelt.) Dies ist also eine rein semantische Betrachtung.
- Die Definitionen für Korrektheit und Vollständigkeit stellen also einen Zusammenhang her zwischen der syntaktischen Eigenschaft der Ableitbarkeit und der semantischen Eigenschaft der logischen Folgerung.

Qualität eines Inferenzverfahrens

■ Inferenzverfahren:

- Korrekt
- Vollständig
- Möglichst wenige verschiedene Inferenzregeln
- Klare Abarbeitungsreihenfolge

■ Wissensbasis:

- Einheitliche Gestalt der Formeln
 - Ggf. Äquivalenzumformungen, um Einheitlichkeit zu erreichen

Erläuterungen

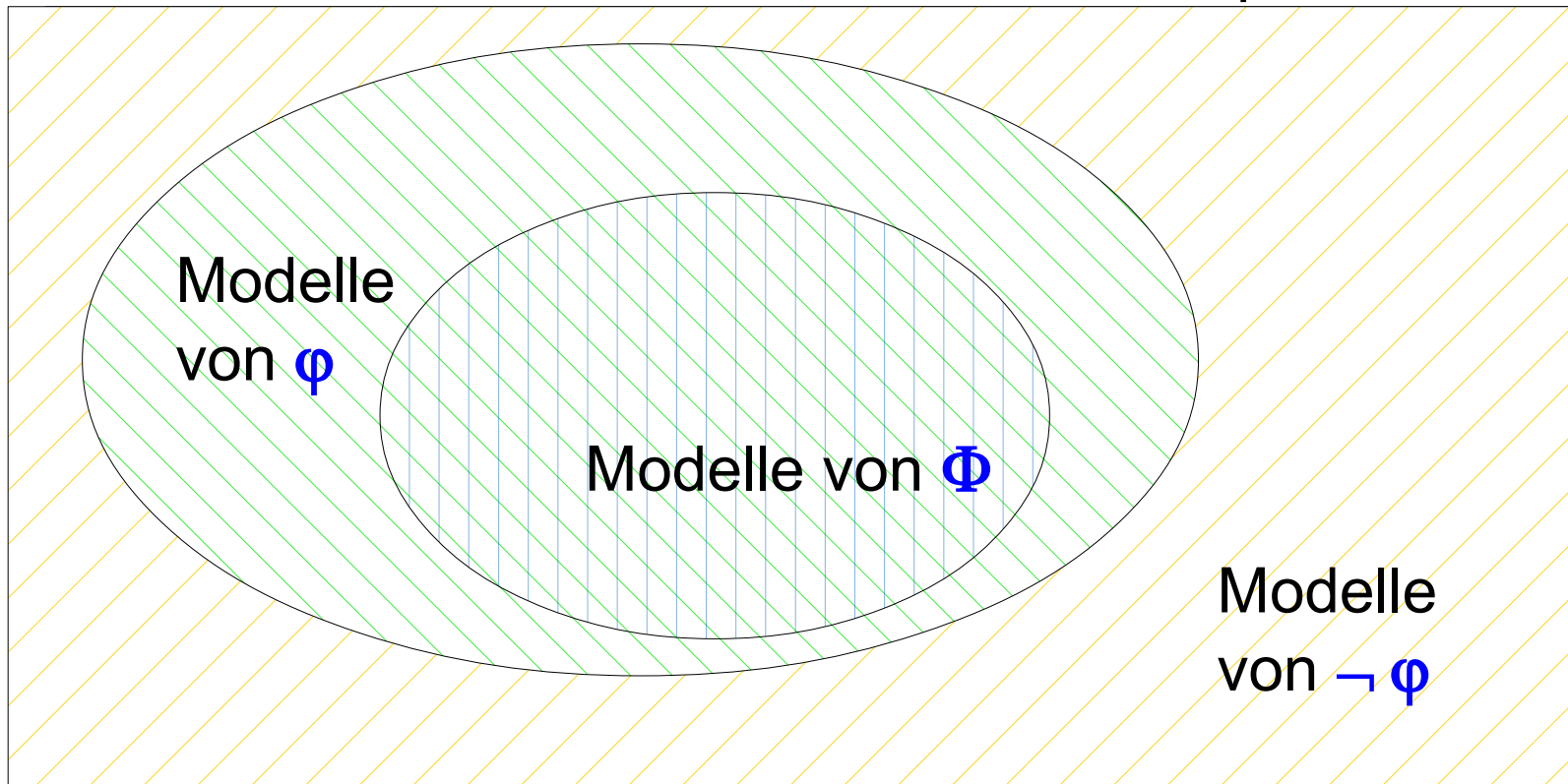
- Neben der Korrektheit und der Vollständigkeit gibt es weitere wünschenswerte Eigenschaften eines Inferenzverfahrens. Einfachheit ist von Vorteil, weil damit eine starre, leicht zu steuernde Abarbeitung realisiert werden kann. Wenige verschiedene Inferenzregeln tragen dazu bei. Außerdem ist es gut, wenn klar ist, welche Formeln der Wissensbasis für den nächsten Ableitungsschritt genommen werden sollten. Eine einheitliche Gestalt aller Formeln der Wissensbasis trägt zu einem einfachen Verfahren bei.
- Das Resolutionsverfahren erfüllt diese Bedingungen weitgehend.

Resolution

- Das Resolutionsverfahren liefert einen Widerspruchsbeweis.
- Es basiert auf der Äquivalenz der beiden folgenden Aussagen:
 $\Phi \models \varphi$
 $\Phi \wedge \neg \varphi$ ist widersprüchlich
- Es geht also darum, aus $\Phi \wedge \neg \varphi$ einen Widerspruch abzuleiten.
- Voraussetzung für Resolution: Alle Sätze befinden sich in KNF.

- Machen Sie sich den Zusammenhang zwischen $\Phi \models \varphi$ und $\Phi \wedge \neg \varphi$ anhand eines Mengendiagramms für die Modellmengen klar. Die Modelle von φ sind ja eine Obermenge der Modelle von Φ . Das Komplement von φ kann daher keine Schnittmenge mit den Modellen von Φ haben. Daher ist die Widersprüchlichkeit von $\Phi \wedge \neg \varphi$ offensichtlich.

Alle Interpretationen



Vorbereitungen zum Resolutionsverfahren

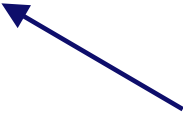
- Alle Formeln der Formelmenge Φ in die KNF
- Die zu beweisende Formel ϕ negieren, ebenfalls in die KNF und zur Formelmenge hinzufügen.

Erläuterungen

- Eines vorab: Sie sollten aus der Literatur ein Beispiel entnehmen, an dem die Anwendung des Resolutionsverfahrens demonstriert wird.
- Die Resolutionsregel verlangt, dass sich alle Formeln in der KNF befinden.
- Die zu beweisende Formel wird ja in negierter Form zur Formelmenge hinzugefügt. Das entspricht dann $\Phi \wedge \neg \phi$. Diese Formelmenge muss einen Widerspruch enthalten, wenn ϕ aus Φ logisch folgen sollte.

Die Resolutionsregel

$$\begin{array}{l} (L_{1,1} \vee \dots \vee L_{1,i-1} \vee L \vee L_{1,i+1} \vee \dots \vee L_{1,m}) \\ (L_{2,2} \vee \dots \vee L_{2,j-1} \vee \neg L \vee L_{2,j+1} \vee \dots \vee L_{2,n}) \\ \hline (L_{1,1} \vee \dots \vee L_{1,i-1} \vee L_{1,i+1} \vee \dots \vee L_{1,m} \vee L_{2,2} \vee \dots \vee L_{2,j-1} \vee L_{2,j+1} \vee \dots \vee L_{2,n}) \end{array}$$

 **Resolvente**

- Es werden zwei geeignete Klauseln der Klauselmenge gewählt.
- Die Resolvente wird zur Klauselmenge hinzugefügt.
- Durch wiederholtes Anwenden der Resolutionsregel versucht man, die leere Klausel abzuleiten. (Die leere Klausel ist Resolvente zweier Klauseln (L) und $(\neg L)$).

Erläuterungen

- Man muss für jeden Resolutionsschritt zwei Formeln suchen, die über dasselbe Literal verfügen, einmal negiert und einmal nicht negiert.
- Aus diesen beiden Formeln wird eine neue, dritte Formel erzeugt. Diese neue Formel setzt sich aus der Vereinigungsmenge aller Literale der beiden Ursprungsformeln zusammen. Nur das eine Literal, das einmal negiert und einmal nicht negiert vorlag, ist nicht in der neuen Formel enthalten.
- Die neu erzeugte Formel wird zur Formelmenge hinzugefügt. Mit jeder Anwendung der Resolutionsregeln wächst die Formelmenge daher um eine neue Formel.
- Letztendlich muss man versuchen, die leere Klausel abzuleiten. Dies passiert dadurch, dass man die Resolutionsregel auf zwei Klauseln **(L)** und **(¬ L)** anwendet. Durch geschickte Wahl kann man mit wenigen Schritten zum Widerspruch kommen, bei ungeschickter Wahl kann es hingegen sehr lange dauern. (Im Fall der Prädikatenlogik sogar unendlich lange.)
- Es ist übrigens nicht so, dass die Resolvente immer länger ist als die Ursprungsklauseln. Denn gleiche Literale werden in der Resolvente ja nicht doppelt aufgeführt.

- Beweisen Sie, dass die Resolutionsregel korrekt ist.

Lösung

- Dazu muss man zeigen, dass jedes Modell der Ursprungsklauseln auch Modell der Resolvente ist.
- Die beiden Ursprungsklauseln lauten $\{A_1, A_2, \dots, A_n\}$ und $\{B_1, B_2, \dots, B_m, \neg A_1\}$
- Dann wäre die Resolvente $\{A_2, A_3, \dots, A_n, B_1, B_2, \dots, B_m\}$
- Die Literale einer Klausel sind ja ODER-verknüpft und die Klauseln UND-verknüpft. In einem Modell der Klauselmenge sind also aufgrund der UND-Verknüpfung alle Klauseln wahr. Für jedes Modell der Ursprungsklauseln gilt: Wenn A_1 wahr ist, muss mindestens ein Literal aus $B_1 \dots B_m$ wahr sein. Wenn hingegen A_1 falsch ist, muss mindestens ein Literal aus $A_2 \dots A_n$ wahr sein. In jedem Fall ist dann aber mindestens ein Literal der Resolvente wahr. Das Modell der Ursprungsklauseln ist also auch Modell der Resolvente.

Resolution: Eigenschaften

- Resolution ist korrekt
- Resolution ist **widerlegungsvollständig**
 - Wenn eine Klauselmengue widersprüchlich ist, so wird dieser Widerspruch gefunden
 - Irgendwann wird also die leere Klausel abgeleitet
 - Resolution ist also ein Beweisverfahren
- Aber: Resolution kann nicht aktiv alles ableiten, was logisch folgt
 - Beispiel: Aus $((L_1) \wedge (L_2))$ lässt sich nicht $(L_1 \vee L_2)$ ableiten
- Resolution terminiert nur mit Sicherheit für die Aussagenlogik
 - Prädikatenlogik ist nicht entscheidbar

Erläuterungen

- **Korrektheit:** Wenn die Resolution einen Widerspruch findet, so ist es sicher, dass die untersuchte Klausel logisch aus der gegebenen Formelmenge folgt.
- **Vollständigkeit:** Die Resolution ist insofern vollständig, als für jeden Satz, der aus der gegebenen Formelmenge logisch folgt, ein Widerspruch gefunden wird. Nur muss dieser Satz explizit vorgegeben werden. So kann man zwar mit dem Resolutionsverfahren beweisen, dass $(L_1 \vee L_2)$ aus $((L_1) \wedge (L_2))$ logisch folgt, wenn man gezielt $(L_1 \vee L_2)$ untersucht, aber die Formel $(L_1 \vee L_2)$ wird nicht aus $((L_1) \wedge (L_2))$ abgeleitet.
- Nun kann man sich noch fragen, ob die Resolution denn immer terminiert. Für die Aussagenlogik kann man diese Frage mit „ja“ beantworten, denn das Vokabular ist endlich und daher kann man nur endlich viele Klauseln mit Hilfe des Vokabulars erzeugen. Im Falle der Prädikatenlogik kann man aufgrund funktionaler Schachtelungen aber beliebig viele Klauseln für ein gegebenes Vokabular erzeugen. Daher ist es für die Prädikatenlogik nicht sicher, dass die Resolution terminiert. Wenn man also feststellen will, ob ein Satz aus einer Formelmenge folgt, so weiß man, während das Verfahren rechnet, nicht, ob es nicht terminieren wird, weil der Satz nicht logisch folgt, oder ob noch eine Antwort kommen wird, weil ein Widerspruch gefunden wird. Die Prädikatenlogik ist nicht entscheidbar, d. h., es gibt kein Verfahren, das die Frage beantwortet, ob ein bestimmter Satz aus einer Formelmenge logisch folgt, und immer terminiert.

Entscheidungsprobleme der Prädikatenlogik

■ Unerfüllbarkeitsproblem

- Semi-entscheidbar
- Resolution prüft Unerfüllbarkeit

■ Folgerbarkeitsproblem

- Semi-entscheidbar („PL1 ist semi-entscheidbar“, bezieht sich hierauf.)
- $\Phi \models \phi$ ist äquivalent zu $\Phi \cup \neg\phi$ (also Unerfüllbarkeit)
- Verdeutlichung anhand des Mengenmodells

■ Gültigkeitsproblem

- Gemeint ist Allgemeingültigkeit, also ob ein Satz eine Tautologie ist
- ϕ ist erfüllt, wenn $\neg\phi$ unerfüllbar ist

■ Erfüllbarkeitsproblem

- Nicht entscheidbar, also auch nicht semi-entscheidbar
- Sonst wäre ja auch Unerfüllbarkeit entscheidbar, denn jeder Satz könnte dann beiden Prüfungen unterzogen werden

Gödelsche Sätze zur Prädikatenlogik

■ Gödelscher Vollständigkeitssatz

- Sagt aus, dass die PL1 vollständig ist
- Genauer, dass es einen Kalkül gibt, der vollständig und korrekt ist
- Bezieht sich auf Beweise, also vollständig im Sinne von widerlegungsvollständig

■ Gödelscher Unvollständigkeitssatz

- Bezieht sich auf eine andere Art Vollständigkeit
- Es kann mit Modellierungsmitteln der Theorie ein Satz gebildet werden, der weder bewiesen noch widerlegt werden kann
- Solche Sätze haben einen Selbstbezug, wie z. B. „Ich lüge“

Zusatzinformation

Beispiel

1. $\{P, \neg Y, X\}$

2. $\{\neg X, Z\}$

3. $\{\neg Z, \neg Q\}$

4. $\{Q\}$

5. $\{X, P\}$

Folgt P aus der
Formelmenge?

Also:

6. $\{\neg P\}$

Eine mögliche Lösung:

3,4: $\{\neg Z\}$ $--> 7$

7,2: $\{\neg X\}$ $--> 8$

6,5: $\{X\}$ $--> 9$

8,9: $\{\}$

Übung für Aussagenlogik

- ≡ Inspektor Craig hat einen Fall zu lösen. Er hat drei Personen im Verdacht, die Tat begangen zu haben. Als Täter kommen nur A, B und C in Frage. Inspektor Craig hat die Informationen:
 1. Wenn A schuldig und B unschuldig ist, so ist C schuldig.
 2. C arbeitet niemals allein.
 3. A arbeitet niemals mit C.
 4. Nur A, B oder C kommen als Täter in Frage.
- ≡ Drücken Sie die Informationen durch aussagenlogische Sätze aus.
- ≡ Überführen Sie die Sätze in die KNF.
- ≡ Prüfen Sie Ihren Verdacht mittels Resolution.

Lösung (1)

1. $A \wedge \neg B \Rightarrow C$

2. $C \Rightarrow A \vee B$

3. $A \Rightarrow \neg C$

4. $A \vee B \vee C$

Als Klauseln:

1. $\{\neg A, B, C\}$

2. $\{\neg C, A, B\}$

3. $\{\neg A, \neg C\}$

4. $\{A, B, C\}$

Lösung (2)

- Eigentlich müsste man jetzt noch einen Verdacht äußern, z. B. dass A der Täter ist. Und dann müsste man die entsprechende Formel negieren, in die KNF überführen und der Klauselmenge hinzufügen. Für die Annahme, dass A der Täter ist, würde man also die (sehr einfache) Klausel $\{\neg A\}$ hinzufügen.
- Wir können allerdings auch zunächst nur mit Hilfe der Klauseln 1 bis 4 Resolventen erzeugen und hoffen, dass dabei eine Klausel $\{A\}$ oder $\{B\}$ oder $\{C\}$ entsteht. Denn dann wäre es ja klar, welchen Verdacht man äußern müsste.
- Also:
1,4: $\{B, C\}$
2,3: $\{B, \neg C\}$
Diese beiden Resolventen ergeben dann $\{B\}$

Unifikation in der Prädikatenlogik

- Aussagenlogik: Die zu resolvierenden Literale unterscheiden sich nur im Vorzeichen
- Prädikatenlogik: Es können Variablen auftreten
 - Die Literale müssen durch Variableninstantiierung gleich gemacht werden
- Gleich machen = **unifizieren** (unus: einer, facere: machen)
- **Unifikation** erfolgt auch in Prolog

- Den Vorgang der Unifikation wollen wir im Umgang mit Prolog lernen. Daher wird dieser letzte Aspekt des Resolutionsverfahrens auf einen späteren Abschnitt verschoben.

Lernziele L6

- V1: Erklären können, was ein Inferenzverfahren und speziell ein Beweisverfahren ist
- V2: Modus Ponens und Modus Tollens erklären können
- V3: Ableitbarkeit erklären können
- V4: Korrektheit und Vollständigkeit erklären können
- V5: Resolution und ihre Eigenschaften erklären können

- A1: Modus Ponens und Modus Tollens anwenden können (--> Teil 2 der Klausur)
- A2: Resolution in der Aussagenlogik anwenden können (--> Teil 2 der Klausur)

P2: Unifikation in Prolog

- Unifikation
- Unifikationsalgorithmus
- Occur Check
- Vergleich mit Identität und Zuweisung

P2: Lernziele

- V1: Unifikations-Algorithmus erklären können
- V2: Occurs-Check verstehen
- A1: Unifikation in Prolog anwenden können, inklusive Occurs-Check

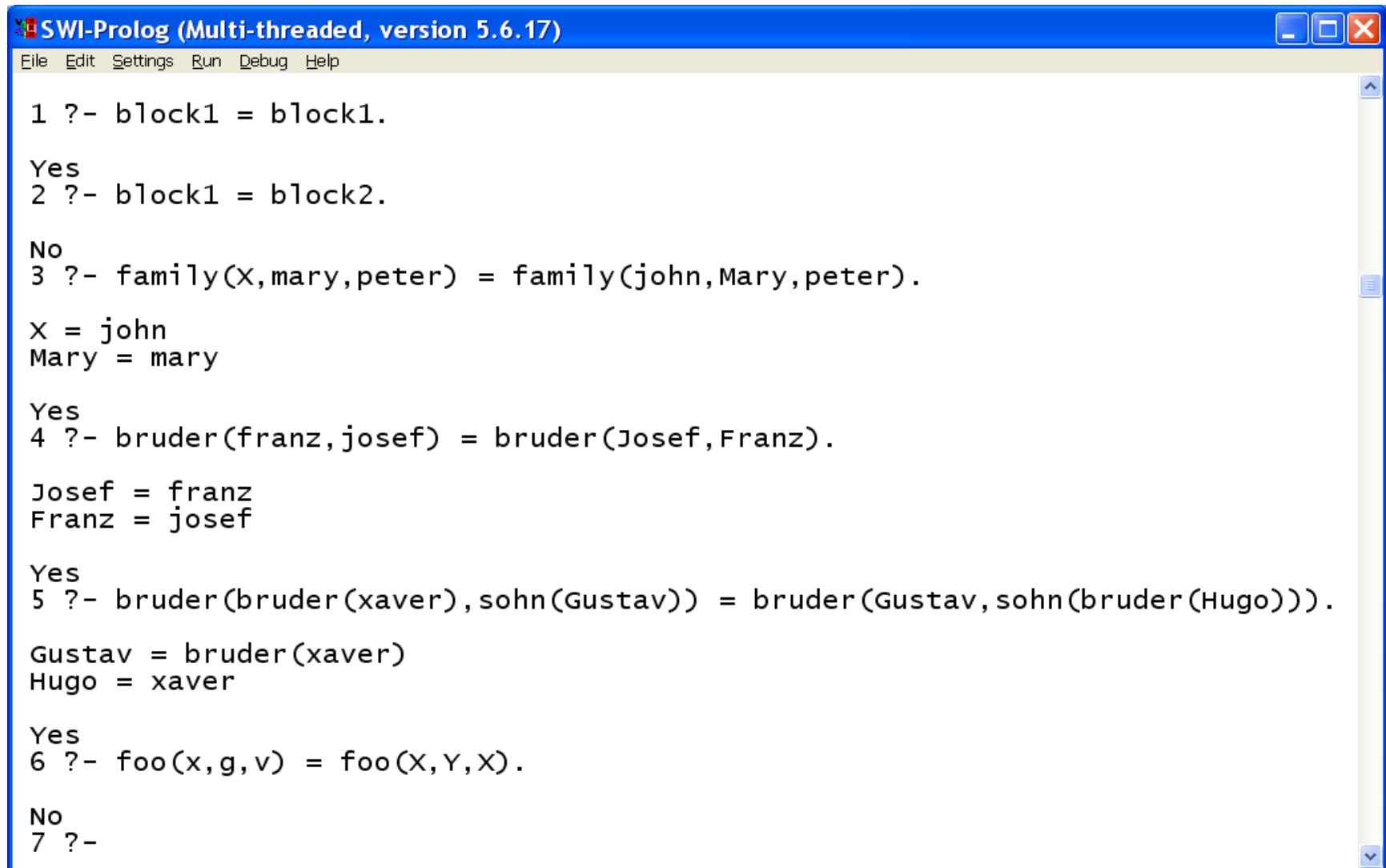
Unifikation in Prolog

- Unifikation passiert im Rahmen eines Resolutionsschrittes.
- Dabei finden ggf. Variablensubstitutionen statt.
- In PROLOG ist es möglich, eine Unifikation im Rahmen einer Anfrage durchzuführen.
 - Symbol für Unifikation: `' = '`

Erläuterungen

- Sie können in der Konsole alle Aspekte der Unifikation testen und so ein tiefes Verständnis dafür erlangen.

Unifikation: Beispiele



```
SWI-Prolog (Multi-threaded, version 5.6.17)
File Edit Settings Run Debug Help

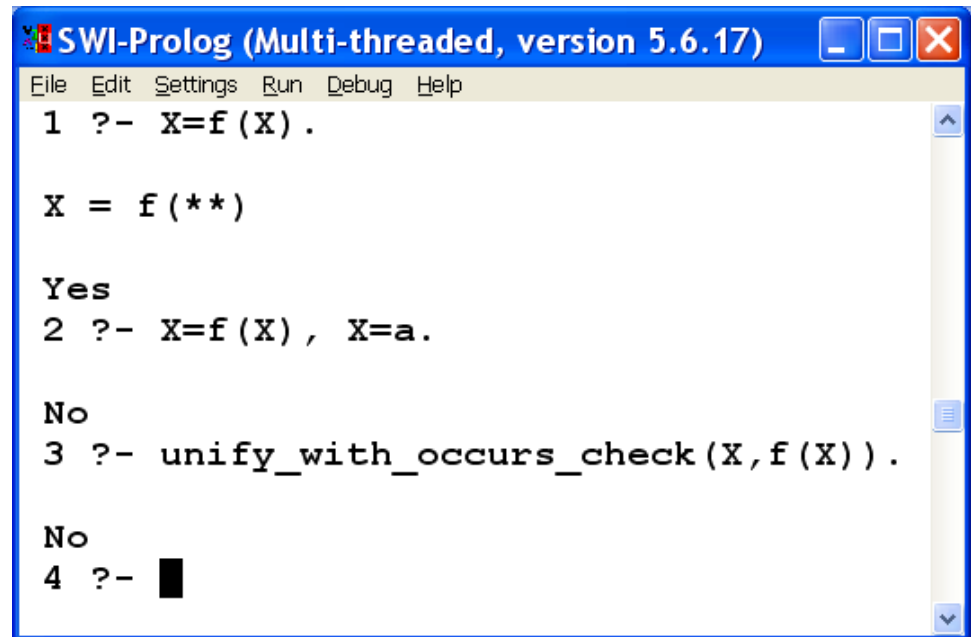
1 ?- block1 = block1.
Yes
2 ?- block1 = block2.
No
3 ?- family(X,mary,peter) = family(john,Mary,peter).
X = john
Mary = mary
Yes
4 ?- bruder(franz,josef) = bruder(Josef,Franz).
Josef = franz
Franz = josef
Yes
5 ?- bruder(bruder(xaver),sohn(Gustav)) = bruder(Gustav,sohn(bruder(Hugo))).
Gustav = bruder(xaver)
Hugo = xaver
Yes
6 ?- foo(x,g,v) = foo(X,Y,X).
No
7 ?-
```

Regeln für die Unifikation

- t_1 und t_2 seien Terme.
- Wenn t_1 und t_2 beide Konstanten sind, müssen sie identisch sein.
- Wenn t_1 eine Variable ist, wird sie mit t_2 belegt. Wenn t_2 eine Variable ist, wird sie mit t_1 belegt.
- Wenn t_1 und t_2 Strukturen sind, also $f_{t_1}(t_{t_1,1}, t_{t_1,2}, \dots, t_{t_1,n})$ und $f_{t_2}(t_{t_2,1}, t_{t_2,2}, \dots, t_{t_2,m})$,
 - müssen f_{t_1} und f_{t_2} derselbe Funktor sein,
 - muss $n = m$ gelten
 - und müssen alle $t_{t_1,i}$ und $t_{t_2,i}$ unifizierbar sein.

Occurs Check

- Wie lautet die Antwort auf folgende Frage? `?- X=f(X) .`
- Korrekt wäre: `X=f(f(f(...f(f(X))...)))`
- Innerhalb endlicher Ausdrücke gibt es keine Lösung.
 - Vorab-Prüfung notwendig:
Occurs-Check.
 - Aus Effizienzgründen in den meisten PROLOG-Implementierungen nicht enthalten.



```
SWI-Prolog (Multi-threaded, version 5.6.17)
File Edit Settings Run Debug Help
1 ?- X=f(X) .

X = f(**)

Yes
2 ?- X=f(X) , X=a.

No
3 ?- unify_with_occurs_check(X,f(X)) .

No
4 ?- 
```

Erläuterungen

- Die beiden Ausdrücke X und $f(X)$ sind nicht unifizierbar, jedenfalls nicht, wenn das Ergebnis von endlicher Länge sein soll.
- Eine Vorab-Überprüfung, ob eine Variable mit einem Term unifiziert werden soll, in dem sie selber vorkommt, nennt man Occurs-Check. Diese Überprüfung verhindert das Erzeugen unendlich großer Datenstrukturen. Die Vorab-Überprüfung ist aus Effizienzgründen in den meisten Prolog-Implementationen abgeschaltet.

Unterschied zwischen '=' und '=='

Unifikation: =

Identität: ==

nicht unifizierbar: \=

nicht identisch: \==

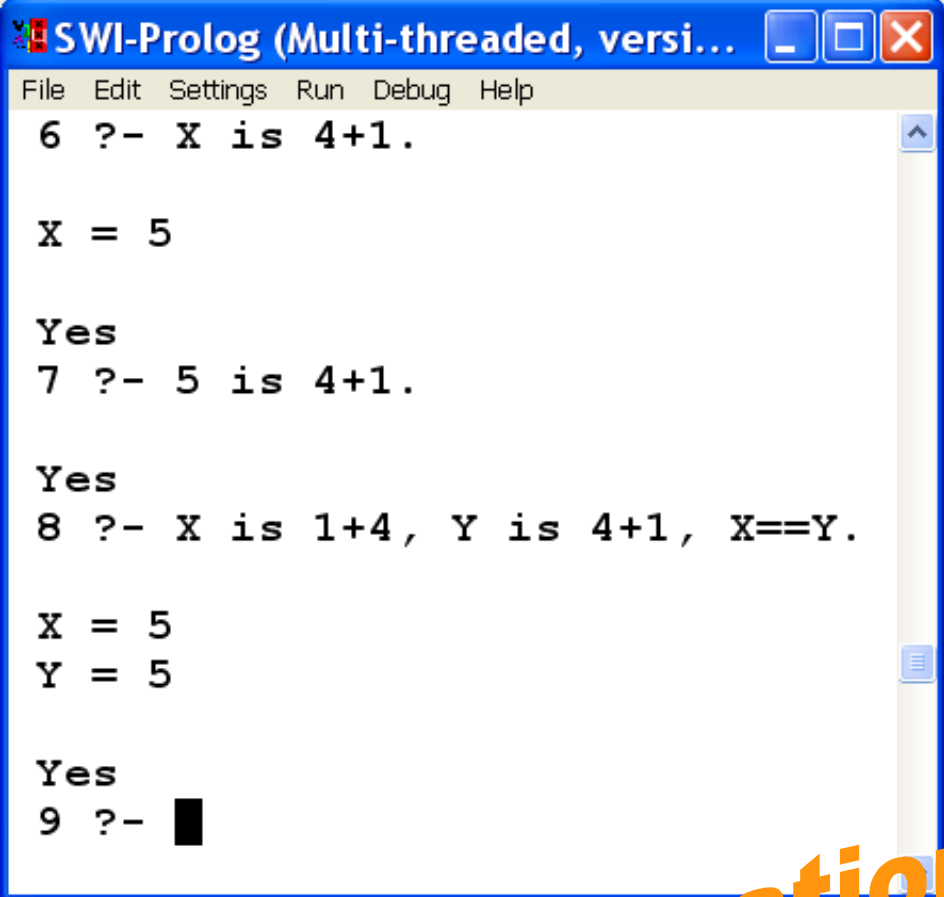
```
SWI-Prolog (Multi-thre...
File Edit Settings Run Debug Help
1 ?- X=4 .
X = 4
Yes
2 ?- X==4 .
No
3 ?- X=4 , X==4 .
X = 4
Yes
4 ?- X=4+1 .
X = 4+1
Yes
5 ?- 4+1 = 1+4 .
No
6 ?- ■
```

Erläuterungen

- Mit dem Symbol '==' wird in Prolog überprüft, ob zwei Ausdrücke auf dasselbe Objekt verweisen.

Zuweisungen in Prolog

- Unifikation ist die einzige Möglichkeit, einer Variablen einen Wert zuzuweisen.
- Infix-Operator '**is**' für arithmetische Ausdrücke
 - Z. B. **X is 4+1.**
 - Rechte Seite wird ausgewertet und dann mit linker Seite unifiziert.
 - Es gibt keine Möglichkeit, den Wert einer Variablen zu verändern, z. B. im Sinne von $x := x + 1$.



```
SWI-Prolog (Multi-threaded, versi...
File Edit Settings Run Debug Help
6 ?- X is 4+1.

X = 5

Yes
7 ?- 5 is 4+1.

Yes
8 ?- X is 1+4, Y is 4+1, X==Y.

X = 5
Y = 5

Yes
9 ?- █
```

- Da das Symbol '=' in vielen Programmiersprachen für Zuweisungen verwendet wird, kann man sich an dieser Stelle fragen, wie Zuweisungen in Prolog funktionieren.

Übung: Wie antwortet Prolog?

1. `?- hallo = 'hallo'.`
2. `?- hallo = "hallo".`
3. `?- hallo == 'hallo'.`
4. `?- 5 = '5'.`
5. `?- Hallo = 'hallo'.`
6. `?- 'Hallo' = 'hallo'`
7. `?- X = 'hallo', X= 'Hallo'.`
8. `?- X == Y.`
9. `?- X = Y.`
10. `?- foo(X,f(g)) = foo(f(g),Y) .`
11. `?- foo(X,f(g)) = foo(f(f(g)),f(Y)) .`
12. `?- foo(X,f(g)) = foo(f(f(g)),f(X)) .`
13. `?- foo(X,f(g)) = foo(f(f(g)),f(_X)) .`
14. `?- foo(f(f(g)),X) = foo(f(X),f(g)) .`
15. `?- foo(f(f(X)),_X) = foo(f(X),f(g)) .`

Erläuterungen

Benutzen Sie Prolog, um die Antworten zu sehen

1. Zweimal dasselbe Atom (ist unifizierbar)
2. Atom und Zeichenkette
3. Zweimal dasselbe Atom (ist identisch)
4. Zahl und Atom
5. Variable und Atom
6. Zwei unterschiedliche Atome
7. X wird an Atom gebunden, ist dann nicht mit anderem Atom unifizierbar
8. Zwei ungebundene Variablen zeigen nicht von vornherein auf dasselbe Objekt
9. Zwei ungebundene Variablen sind unifizierbar
10. - 15. müssten durch Antwort von Prolog klar sein.

Lernziele P2

- V1: Unifikations-Algorithmus erklären können
- V2: Occurs-Check verstehen
- A1: Unifikation in Prolog anwenden können, inklusive Occurs-Check

L7: Resolution in der Prädikatenlogik

- Allgemeine Resolution
- SLD-Resolution

Erläuterungen

- Sie haben jetzt gelernt, wie man unifiziert. Zuvor waren Sie bereits in der Lage, das Resolutionsverfahren für die Aussagenlogik anzuwenden. Jetzt geht es zunächst darum, die (allgemeine) Resolution für eine Aufgabenstellung aus der Prädikatenlogik zu erproben. Dabei müssen Sie auch skolemisieren!
- Dann werden Sie ein effizienteres Verfahren kennen lernen, nämlich die SLD-Resolution. Prolog arbeitet mit SLD-Resolution.

Lernziele L7

- V1: Horn-Klauseln mit ihren speziellen Ausprägungen erklären können
- V2: SLD-Resolution mit ihren Eigenschaften auch im Vergleich mit der allgemeinen Resolution erklären können
- A1: Allgemeine Resolution in der Prädikatenlogik anwenden können (--> Teil 2 der Klausur)
- A2: SLD-Resolution anwenden können (--> Teil 2 der Klausur)

Übung für Prädikatenlogik

≡ Gegeben seien die Aussagen

1. Pferde sind schneller als Hunde.
2. Es gibt einen Windhund, der schneller als jeder Hase ist.
3. Fury ist ein Pferd.
4. Bunny ist ein Hase.

≡ **Beweisen Sie mittels Resolution, dass Fury schneller als Bunny ist.**

- ≡ **Achtung: Es gibt noch einen allgemeinen Zusammenhang, der mit modelliert werden muss. Wenn Sie die ersten Resolutionsschritte gemacht haben, werden Sie wahrscheinlich merken, was fehlt.**
- ≡ **Wenn Sie nicht wissen, wie Sie grundsätzlich vorgehen müssen: Auf der nächsten Seite finden Sie eine Anleitung.**

- 1. Tipp: Hund statt Windhund modellieren
- 2. Tipp: Es gibt einen Hund und der ist ...

Lösung: Grundsätzliches Vorgehen

- Im ersten Schritt müssen Sie die vier Aussagen in prädikatenlogische Formeln überführen.
- Dann müssen Sie diese Formeln in die KNF übertragen. Grundsätzlich ist es möglich, dass aus einer Ursprungsformel mehrere Klauseln entstehen.
- Im dritten Schritt drücken Sie den zu beweisenden Satz auch in der Prädikatenlogik aus, negieren ihn, übertragen ihn in die KNF und fügen ihn der Klauselmenge hinzu.
- Jetzt können Sie das Resolutionsverfahren anwenden. Dabei ist Unifikation notwendig. Sie werden feststellen, dass Ihnen kein Widerspruch gelingt. Dies liegt daran, dass noch ein wichtiger Zusammenhang modelliert werden muss: Die Transitivität des Prädikats 'schneller'. Auch dieser Zusammenhang muss in KNF der Formelmenge hinzugefügt werden.
- Probieren Sie diese Vorgehensweise, bevor Sie Details der Lösung ansehen.

Lösung: Modellierung der Aufgabe

1. $\forall P, \forall H: \text{pferd}(P) \wedge \text{hund}(H) \Rightarrow \text{schneller}(P,H)$
2. $\exists H, \forall A: \text{hund}(H) \wedge (\text{hase}(A) \Rightarrow \text{schneller}(H,A))$
3. $\text{pferd}(\text{fury})$
4. $\text{hase}(\text{bunny})$
5. $\forall X, \forall Y, \forall Z: \text{schneller}(X,Y) \wedge \text{schneller}(Y,Z) \Rightarrow \text{schneller}(X,Z)$

■ Anfrage negiert:

6. $\neg \text{schneller}(\text{fury}, \text{bunny})$

Erläuterungen

- Die Modellierung der zweiten Aussage ist schwierig. Vielleicht hilft Ihnen folgende Ausdrucksweise: „Es gibt einen Hund und für jeden Hasen gilt, dass dieser Hund schneller ist.“
- Ein Windhund ist auch ein Hund. Diese Teilmengen-Beziehung zu modellieren, trägt nicht zur Lösung bei. Es reicht also zu sagen: „Es existiert ein Hund...“

Lösung: KNF

1. $\{ \neg \text{pferd}(P), \neg \text{hund}(H), \text{schneller}(P,H) \}$
2.
 1. $\{ \text{hund}(w) \}$
 2. $\{ \neg \text{hase}(A), \text{schneller}(w,A) \}$
3. $\{ \text{pferd}(\text{fury}) \}$
4. $\{ \text{hase}(\text{bunny}) \}$
5. $\{ \neg \text{schneller}(X,Y), \neg \text{schneller}(Y,Z), \text{schneller}(X,Z) \}$

■ Anfrage negiert:

6. $\{ \neg \text{schneller}(\text{fury}, \text{bunny}) \}$

Lösung: Resolution

1, 3: $\{ \neg \text{hund}(H), \text{schneller}(\text{fury}, H) \} \rightarrow 7$

2.1, 7: $\{ \text{schneller}(\text{fury}, w) \} \rightarrow 8$

2.2, 4: $\{ \text{schneller}(w, \text{bunny}) \} \rightarrow 9$

5, 8: $\{ \neg \text{schneller}(w, Z), \text{schneller}(\text{fury}, Z) \} \rightarrow 10$

9, 10: $\{ \text{schneller}(\text{fury}, \text{bunny}) \} \rightarrow 11$

6, 11: $\{ \}$

Nachteile des allgemeinen Resolutionsverfahrens

- Hoher Rechenaufwand
 - Ursache: Viele alternative nächste Schritte
- Lösungsmöglichkeit: Einschränkung auf spezielle Klauseln und ein spezielles Ableitungsverfahren
 - Horn-Klauseln
 - SLD-Resolution

Erläuterungen

- Sie haben es möglicherweise selbst bei den Übungen erlebt, dass man viele alternative Möglichkeiten hat, zwei Klauseln zu einer neuen Resolvente zu kombinieren. Die SLD-Resolution ist ein Verfahren, das mit Horn-Klauseln arbeitet und eine klare Vorschrift hat, welche Klauseln als nächstes kombiniert werden müssen.

Horn-Klauseln

- **Horn-Klauseln** sind Klauseln mit maximal einem positiven Literal.

- Der allgemeine Fall: **Regeln**

- $\{\neg B_1, \neg B_2, \dots, \neg B_n, K\}$

- Äquivalent: $B_1 \wedge B_2 \wedge \dots \wedge B_n \Rightarrow K$

- Keine negativen Literale: **Fakten**

- $\{F\}$

- Kein positives Literal: **Anfragen** (das, was es zu beweisen gilt)

- $\{\neg Z_1, \neg Z_2, \dots, \neg Z_m\}$

- äquivalent: $Z_1 \wedge Z_2 \wedge \dots \wedge Z_m \Rightarrow \text{falsch}$

- Zum Namen: Erste Untersuchungen von Alfred Horn, ca. 1950

Wiederholung

Erläuterungen

- Horn-Klauseln kennen Sie bereits. Und wie Sie auch wissen, besteht ein Prolog-Programm aus Fakten und Regeln. Der Aufruf eines Prolog-Programms ist hingegen eine Anfrage. Damit ist die Verwendung der drei Typen von Horn-Klauseln in Prolog klar.
- Die Anfrage ist der Ausgangspunkt des Resolutionsverfahrens in Prolog. Wenn Sie sich mit der Arbeitsweise von Prolog, die letztendlich eine Tiefensuche darstellt, beschäftigen, werden Sie ein Verfahren kennenlernen, das SLD-Resolution heißt.

SLD-Resolution

- **SLD Resolution:** „*Selective Linear Definite clause Resolution*“
- Die Zielklausel wird im ersten Resolutionsschritt benutzt.
- Die noch zu beweisenden Literale werden immer in der Reihenfolge ihres Auftretens innerhalb der Klausel abgearbeitet.
- Die jeweils zuletzt erzeugte Resolvente wird für den nächsten Resolutionsschritt benutzt.
- Die Programmiersprache Prolog basiert auf SLD-Resolution.

- 'Definite Clauses' sind Horn-Klauseln
- Machen Sie sich klar, dass SLD-Resolution eine Tiefensuche beschreibt. Das heißt, wenn eine Anfrage aus mehreren Literalen besteht, dann wird das erste Literal komplett behandelt, bevor das zweite an die Reihe kommt. Konkret bedeutet dies, dass für das erste Literal eine passende Klausel gesucht wird. Gibt es mehrere passende, so wird die erste genommen. Aus diesem Resolutionsschritt ergibt sich eine Resolvente. Da hier Horn-Klauseln vorliegen, besteht die Resolvente aus den negierten Literalen der eben erwähnten ersten passenden Klausel. Die negierten Literale stellen ja wiederum eine Anfrage dar und werden nun von vorn nach hinten abgearbeitet. D.h., für das erste Literal wird eine passende Klausel gesucht. Und so weiter. So ergibt sich eine Tiefensuche.

Eigenschaften der SLD-Resolution

- Korrekt
- Widerlegungsvollständig
- Eingeschränkte Ausdrucksmöglichkeiten

Erläuterungen

- Auch die SLD-Resolution ist korrekt und widerlegungsvollständig. Allerdings besitzen Horn-Klauseln geringere Ausdrucksmöglichkeiten, da pro Klausel nur maximal ein positives Literal erlaubt ist.

Übung

- Beweisen Sie mittels SLD-Resolution, dass Fury schneller als Bunny ist!
- Tun Sie dasselbe mit Hilfe eines Prolog-Programms!

Lösung (1)

- Glücklicherweise erhält man bei der Modellierung dieser Aufgabenstellung Horn-Klauseln. Sonst könnte man die SLD-Resolution gar nicht anwenden. Der Unterschied zu der vorigen Übung besteht also lediglich darin, die Abarbeitungsreihenfolge der SLD-Resolution einzuhalten.

Lösung (2)

1. $\{ \neg \text{pferd}(P), \neg \text{hund}(H), \text{schneller}(P,H) \}$
2.
 1. $\{ \text{hund}(w) \}$
 2. $\{ \neg \text{hase}(A), \text{schneller}(w,A) \}$
3. $\{ \text{pferd}(\text{fury}) \}$
4. $\{ \text{hase}(\text{bunny}) \}$
5. $\{ \neg \text{schneller}(X,Y), \neg \text{schneller}(Y,Z), \text{schneller}(X,Z) \}$

■ Anfrage negiert:

6. $\{ \neg \text{schneller}(\text{fury},\text{bunny}) \}$

Lösung (3)

6, 1: $\{ \neg \text{pferd}(\text{fury}), \neg \text{hund}(\text{bunny}) \} \rightarrow 7$

7, 3: $\{ \neg \text{hund}(\text{bunny}) \} \rightarrow 8$ (Backtracking)

6, 5: $\{ \neg \text{schneller}(\text{fury}, Y), \neg \text{schneller}(Y, \text{bunny}) \} \rightarrow 9$

9, 1: $\{ \neg \text{pferd}(\text{fury}), \neg \text{hund}(H), \neg \text{schneller}(H, \text{bunny}) \} \rightarrow 10$

10, 3: $\{ \neg \text{hund}(H), \neg \text{schneller}(H, \text{bunny}) \} \rightarrow 11$

11, 2.1: $\{ \neg \text{schneller}(w, \text{bunny}) \} \rightarrow 12$

12, 1: $\{ \neg \text{pferd}(w), \neg \text{hund}(\text{bunny}) \} \rightarrow 13$ (Backtracking)

12, 2.2: $\{ \neg \text{hase}(\text{bunny}) \} \rightarrow 14$

14, 4: $\{ \}$

Erläuterungen ergänzen

- Die Sackgassen bei der Herleitung der Klauseln 7 und 13 erscheinen Ihnen vielleicht sehr unsinnig und evt. sind Sie nicht überzeugt davon, dass die SLD-Resolution effizienter ist als die allgemeine Resolution. Sie müssen aber bedenken, dass die Lösung zur allgemeinen Resolution mit menschlicher Intelligenz erstellt wurde und dass es sich um eine sehr einfache, übersichtliche Aufgabenstellung handelt.
- Erläuterungen zum Backtracking:
 - Für Klausel 8 gibt es kein Gegenstück in der Wissensbasis. Also endet hier der Versuch, die leere Klausel abzuleiten. Auf Prolog übertragen bedeutet dies, dass versucht wurde, `hund(bunny)` zu beweisen. Dafür gibt es aber keine passenden Fakten oder Regeln. Die Klausel 8 ist aus dem Versuch entstanden, für $\neg \text{pferd}(\text{fury})$ eine passende Klausel zu finden. Die Entscheidung für Klausel 3 wird nun zurückgezogen. Es gibt allerdings keine Alternative. Dies bedeutet, dass die Klausel 7 insgesamt nicht zu einer Lösung führt, d.h. nicht bewiesen werden kann. Also wird für die Ausgangsklausel 6 jetzt eine Alternative gesucht. Die Klausel 2.2 passt nicht, weil hier eine Konstante `w` im Konflikt mit `fury` steht. Aber die Klausel 5 passt.
 - Weiter unten wird für Klausel 13, konkret für $\neg \text{pferd}(w)$ keine passende Klausel gefunden. Deshalb wird für den vorherigen Schritt, nämlich für $\neg \text{schneller}(w, \text{bunny})$, eine Alternative gesucht. Außer der Klausel 1 passt diesmal auch die Klausel 2.2 (und außerdem auch die Klausel 5 als dritte Alternative).

Lösung: Prolog-Programm

schneller(P,H) :- pferd(P) , hund(H) .

schneller(w,A) :- hase(A) .

schneller(X,Z) :- schneller(X,Y) , schneller(Y,Z) .

hund(w) .

pferd(fury) .

hase(bunny) .

Erläuterungen

- In Prolog ist es guter Stil, alle Regeln mit demselben Regelkopf untereinander aufzuführen. Es ergibt sich hierdurch aber keine geänderte Abarbeitungsreihenfolge.
- **Fragen Sie Prolog auch einmal, wer alles schneller ist als Bunny! (Eingabe eines Semikolons für weitere Lösungen)**

Übung (1)

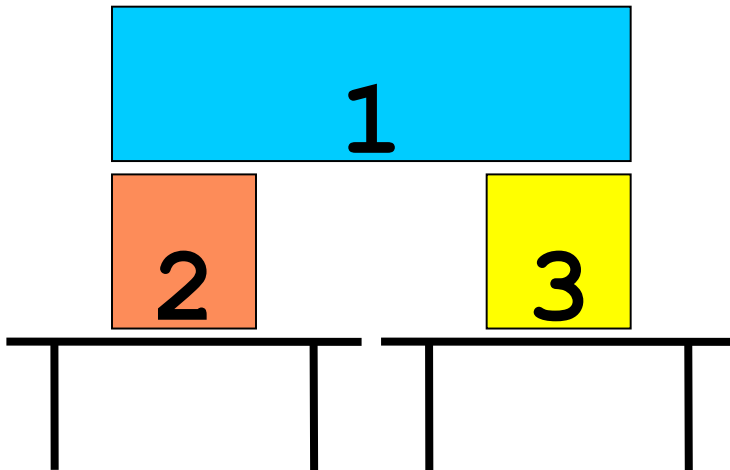
- Machen Sie sich den Ablauf der SLD-Resolution anhand des folgenden Beispiels aus Prolog klar.

Übung (2)

```
on(block1,block2) .  
on(block1,block3) .  
on(block2,table1) .  
on(block3,table2) .
```

```
above(X,Y) :-  
    on(X,Y) .
```

```
above(X,Y) :-  
    on(X,Z) , above(Z,Y) .
```



```
SWI-Prolog (Multi-threaded, version 5.6.17)
File Edit Settings Run Debug Help

% Spy point on above/2
1 ?- above(block1,table2).
    Call: (7) above(block1, table2) ? creep
    Call: (8) on(block1, table2) ? creep
    Fail: (8) on(block1, table2) ? creep
    Redo: (7) above(block1, table2) ? creep
    Call: (8) on(block1, _L169) ? creep
    Exit: (8) on(block1, block2) ? creep
    Call: (8) above(block2, table2) ? creep
    Call: (9) on(block2, table2) ? creep
    Fail: (9) on(block2, table2) ? creep
    Redo: (8) above(block2, table2) ? creep
    Call: (9) on(block2, _L197) ? creep
    Exit: (9) on(block2, table1) ? creep
    Call: (9) above(table1, table2) ? creep
    Call: (10) on(table1, table2) ? creep
    Fail: (10) on(table1, table2) ? creep
    Redo: (9) above(table1, table2) ? creep
    Call: (10) on(table1, _L208) ? creep
    Fail: (10) on(table1, _L208) ? creep
    Redo: (8) on(block1, _L169) ? creep
    Exit: (8) on(block1, block3) ? creep
    Call: (8) above(block3, table2) ? creep
    Call: (9) on(block3, table2) ? creep
    Exit: (9) on(block3, table2) ? creep
    Exit: (8) above(block3, table2) ? creep
    Exit: (7) above(block1, table2) ? creep

Yes
[debug] 2 ?- █
```

- Eine weitere Möglichkeit zum Üben der SLD-Resolution ist die Aufgabe 9 der IS-Klausur aus dem SS 2012.

Lernziele L7

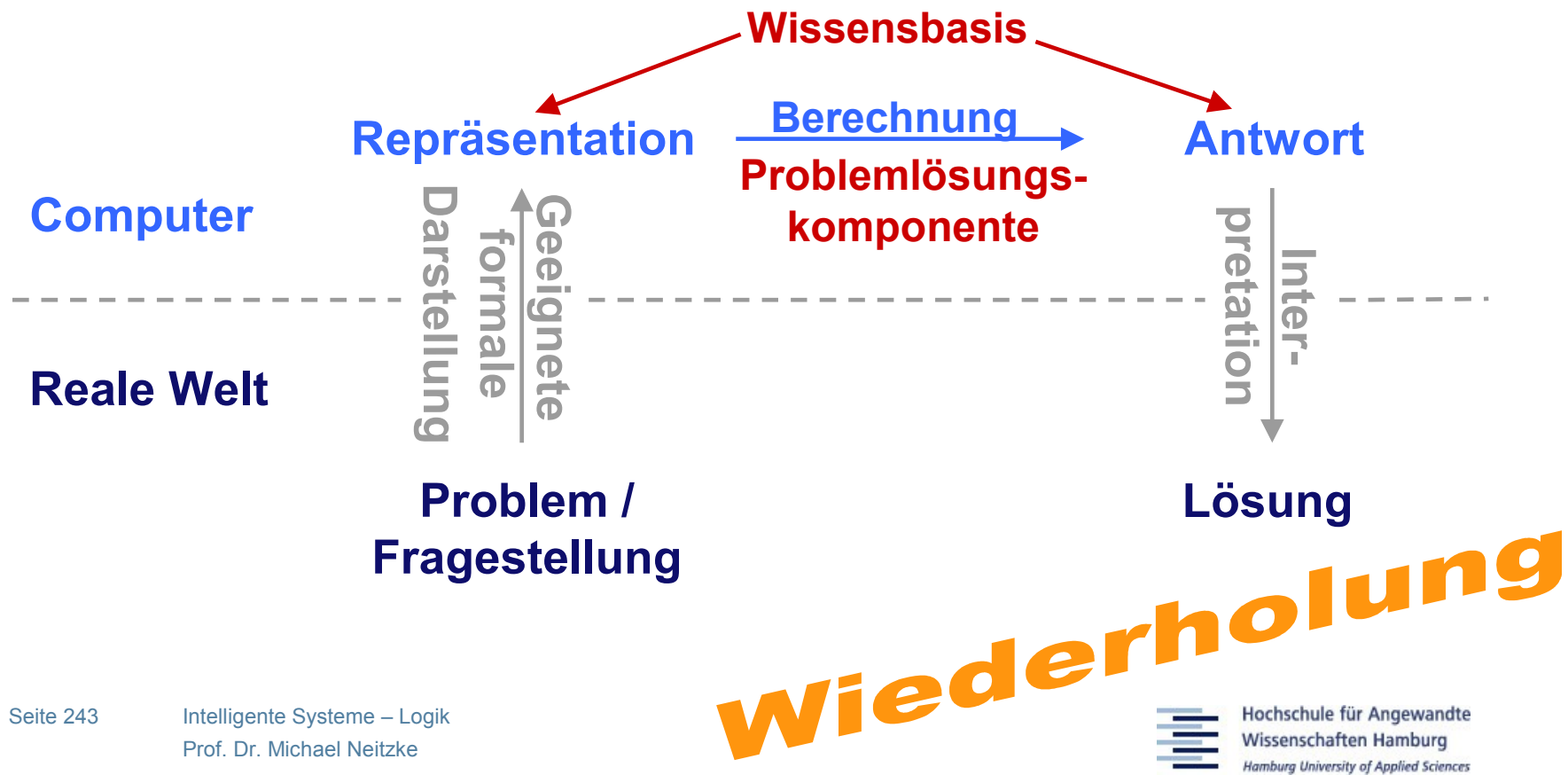
- V1: Horn-Klauseln mit ihren speziellen Ausprägungen erklären können
- V2: SLD-Resolution mit ihren Eigenschaften auch im Vergleich mit der allgemeinen Resolution erklären können
- A1: Allgemeine Resolution in der Prädikatenlogik anwenden können (--> Teil 2 der Klausur)
- A2: SLD-Resolution anwenden können (--> Teil 2 der Klausur)

P3: Logik und Steuerung in Prolog

Lernziele P3

- V1: Die Trennung von Logik und Steuerung in Prolog erläutern können
- V2: Closed World Assumption und Open World Assumption erläutern können
- V3: Arbeitsweise von Prolog durch Entwicklung eines Tiefensuch-Baums inklusive Backtracking erläutern können (Dies wird auch im Praktikum geprüft.) (--> Teil 2 der Klausur)

Zur Erinnerung



Trennung von Logik und Steuerung

- Algorithm = Logic + Control
 - Logic: Wissensbasis
 - Control: Problemlösungskomponente / Schlussfolgerungsmechanismus / Inferenzkomponente
- Wissensbasis:
 - Aussagenlogik: Aussagenlogische Formeln
 - Prädikatenlogik: Prädikatenlogische Formeln
- Problemlösungskomponente / Schlussfolgerungsmechanismus / **Inferenz**mechanismus in Aussagenlogik bzw. Prädikatenlogik:
 - Resolution (Resolutionskalkül)
 - Tableaukalkül

Erläuterungen

- 1979 hatte Robert Kowalski in seiner ACM-Veröffentlichung „Algorithm = Logic + Control“ diese Trennung beschrieben.
- Neben dem Resolutionskalkül gibt es noch andere Schlussfolgerungssysteme, die mit logischen Formeln umgehen können, wie z. B. der Tableauekalkül. (Ein Kalkül ist ein aus Regeln bestehendes Schlussfolgerungssystem.)

Prolog: PROgramming in LOGic

- Ideal: **Algorithmus = Logik + Kontrolle/Steuerung**
- Ein PROLOG-Programm beschreibt die Logik.
 - In Gestalt von **Horn-Klauseln**
 - Drei Arten von Horn-Klauseln:
 - **Fakten**
 - **Regeln**
 - **Anfragen**
- Das PROLOG-System setzt die Kontrolle um.
 - **SLD-Resolution**

Wiederholung

Klassische Logiken = zweiwertige Logiken

- Zweiwertig: „wahr“, „falsch“
- Zweiwertige Logiken („klassische“):
 - Aussagenlogik
 - Prädikatenlogik 1. Stufe (PL1)
 - Prädikatenlogik höherer Stufe / höherer Ordnung
 - Beschreibungslogiken (meist entscheidbare Teilmengen der PL1)
 - Hornklausel-Logik (Teilmenge der Aussagenlogik oder PL)
 - Grundlage von PROLOG

Erläuterungen

- Die so genannten klassischen Logiken sind zweiwertig, arbeiten nur mit den beiden Werten „wahr“ und „falsch“.
- Die Prädikatenlogik zweiter Stufe formuliert Prädikate über Prädikate. So kann z. B. direkt ausgedrückt werden, dass das Prädikat **schneller** transitiv ist: **transitiv(schneller)**. Allerdings gibt es für die PL2 keinen vollständigen und korrekten Kalkül.
- Beschreibungslogiken sind Teilmengen der PL1 mit der für die Praxis so wichtigen Eigenschaft der Entscheidbarkeit. D. h., das für jede gegebene Formel festgestellt werden kann, ob sie aus einer Formelmenge logisch folgt oder nicht. Im Gegensatz zur PL1 terminiert diese Überprüfung also immer.
- Neben zweiwertigen Logiken gibt es z. B. mehrwertige und nichtmonotone Logiken
 - Mehrwertige Logiken
 - Dreiwertig, z. B. „wahr“, „falsch“, „weiß nicht“
 - Fuzzy-Logik: Grad der Zugehörigkeit wird repräsentiert (**schnell(bunny)** gilt zu 70%)
 - Nichtmonotone Logiken
 - Durch neues Wissen können bisherige Aussagen wieder zurückgezogen werden. Tweety ist ein Vogel. Also kann er fliegen. Neue Information: Tweety ist ein Pinguin.

CWA vs. OWA

- Wie wird mit einer Aussage umgegangen, die sich innerhalb der zur Verfügung stehenden Wissensbasis nicht beweisen lässt?
- Closed World Assumption: Aussage ist falsch
 - Beispiel Blocks World mit vier Blöcken auf einem Tisch: Keine explizite Information darüber, dass sich kein Objekt auf „block3“ befindet.
 - `?-on(X,block3)` liefert **no**.
 - Diese Art der Schlussfolgerung wird auch als „**negation as failure**“ bezeichnet (weil die Negation eines Prädikats gilt, wenn das Prädikat nicht bewiesen werden konnte (Failure)).
 - Gilt in Prolog, ist auch sinnvoll in Datenbanksystemen
- Open World Assumption: Man kann weder davon ausgehen, dass die Aussage wahr ist, noch dass sie falsch ist. Keine weiteren Schlussfolgerungen erlaubt, so lange man nichts weiß.
 - Sinnvoll z. B. für Internet-Recherchen

Erläuterungen

- Negation as failure: Ein negiertes Ziel $\neg p$ wird dann als bewiesen betrachtet, wenn p nicht bewiesen werden kann.
 - `clean_top(block3) :- not(on(_,block3)) .`
- Achtung: Folgendes funktioniert nicht:
 - `clean_top(X) :- not(on(_,X)) .`
 - Prolog versucht zunächst `(on(_,X))` zu beweisen. Das wird für alle Blöcke gelingen, auf denen etwas drauf liegt. Wenn diese erfolgreiche Instantiierung dann durch `not` negiert wird, ist die gesamte Regel gescheitert. Soweit ist das auch in Ordnung. Nur wird Prolog auf diese Weise nie `x` mit einem Block instantiieren, für den die `on`-Beziehung nicht erfüllt werden kann. `x` muss daher vorab instantiiert werden:
 - `clean_top(X) :- block(X) , not(on(_,X)) .`
- Die CWA ist immer dann sinnvoll, wenn man vollständige Kontrolle über die Wissensbasis hat, also z. B. in vielen Datenbankanwendungen.
- Die OWA ist dann sinnvoll, wenn die Wissensbasis ohne eigene Kontrolle verändert werden kann, wie z. B. das Internet. Die Semantic Web Sprache OWL arbeitet mit der OWA.

Übung

- Gegeben sei folgende Wissensbasis:

`word(abalone,a,b,a,l,o,n,e) .`

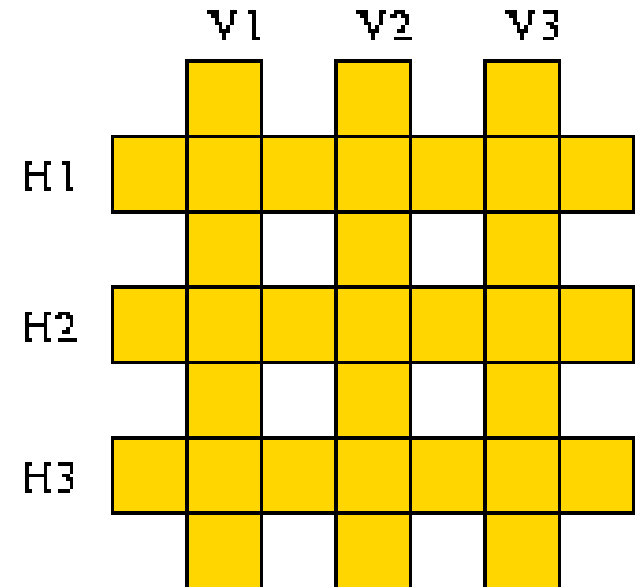
`word(abandon,a,b,a,n,d,o,n) .`

`word(enhance,e,n,h,a,n,c,e) .`

`word(anagram,a,n,a,g,r,a,m) .`

`word(connect,c,o,n,n,e,c,t) .`

`word(elegant,e,l,e,g,a,n,t) .`



- Fügen Sie ein Prädikat `crosswd` hinzu, das eine Lösung für die vertikalen und horizontalen Wörter liefert.

Lösung

```
% Idee: Learn Prolog now!
```

```
word(abalone,a,b,a,l,o,n,e) .  
word(abandon,a,b,a,n,d,o,n) .  
word(enhance,e,n,h,a,n,c,e) .  
word(anagram,a,n,a,g,r,a,m) .  
word(connect,c,o,n,n,e,c,t) .  
word(elegant,e,l,e,g,a,n,t) .
```

```
%Fxy für Feld mit Koordinaten x,y
```

```
crosswd(V1,V2,V3,H1,H2,H3) :-  
    word(V1,_,F22,_,F24,_,F26,_) ,  
    word(H1,_,F22,_,F42,_,F62,_) ,  
    word(V2,_,F42,_,F44,_,F46,_) ,  
    word(H2,_,F24,_,F44,_,F64,_) ,  
    word(V3,_,F62,_,F64,_,F66,_) ,  
    word(H3,_,F26,_,F46,_,F66,_) .
```

Übung: Successor

- Entwickeln Sie ein PROLOG-Programm, das folgendes Ein-/Ausgabe-Verhalten erzeugt:

```
?- zahl(X) .
```

```
X = 0 ;
```

```
X = succ(0) ;
```

```
X = succ(succ(0)) ;
```

```
X = succ(succ(succ(0))) ;
```

USW.

Lösung

`zahl (0) .`

`zahl (succ (X)) :- zahl (X) .`

Steuerung / Kontrolle in der Wissensbasis

- Die ideale Trennung von Logik und Steuerung wird in Prolog manchmal verletzt:
 - Reihenfolge der Klauseln ist von Bedeutung
 - **repeat** (siehe folgende Seiten)
 - **cut** (siehe folgende Seiten)
 - **call**, **assert**, **retract**, **findall** (siehe folgende Seiten)

Repeat

- Ist immer wahr.
- Bewirkt im Zusammenhang mit Backtracking das wiederholte Testen nachfolgender Prädikate.
- Vorhergehende Prädikate sind nicht mehr erreichbar.

```
repeat-demo:-  
    repeat,  
    write('Term eingeben ("q." zum Beenden): '),  
    read(X),  
    write(X), nl,  
    X = q.
```


Erläuterungen

- Da **repeat** immer wieder „wahr“ liefert, werden nachfolgende Prädikate im Rahmen des Backtracking immer wieder getestet.

Cut (1)

- Unnötige Berechnungen zur Feststellung einer Ermäßigung:

```
ermaessigung(X) :- rentner(X), einkommen(X,E), E =< 1000.
```

```
ermaessigung(X) :- student(X), semester(X,S), S =< 12.
```

```
ermaessigung(X) :- alter(X,A), A =< 12.
```

- Lösung: Verhinderung weiteren Backtrackings

```
ermaessigung(X) :- rentner(X), !, einkommen(X,E), E =< 1000.
```

```
ermaessigung(X) :- student(X), !, semester(X,S), S =< 12.
```

```
ermaessigung(X) :- alter(X,A), A =< 12.
```

- **Cut-Fail**

```
ermaessigung(X) :- einkommen(X,E), E > 1000, !, fail.
```

Cut(2)

- Allgemein:

`p:- a1, a2, a3, !, b1, b2, b3, b4.`

`p:- c1, c2.`

- Wenn **a1**, **a2**, **a3** einmal erfüllt sind,

- werden keine weiteren Alternativen für **a1**, **a2**, **a3** getestet
- wird keine weitere Regel ausprobiert
- findet Backtracking nur noch innerhalb **b1**, **b2**, **b3**, **b4** statt

Erläuterungen

- Wenn es klar ist, dass jemand Rentner ist und er nur deswegen keine Ermäßigung bekommt, weil sein Einkommen zu hoch ist, so macht es keinen Sinn, die Bedingungen auch noch für Studenten und Kinder zu überprüfen. Das Cut sorgt dafür, dass im Tiefensuchbaum weiter rechts liegende Äste nicht mehr untersucht werden. Backtracking ist also nur noch unterhalb des Cut möglich. (Weiter links liegende Äste wurden ja bereits vollständig untersucht.)

Übung: Wie antwortet Prolog?

■ Wissensbasis:

`huhu(b) .`

`huhu(p) .`

`hihi(i) .`

`hihi(j) .`

`hihi(k) .`

`hallo(o) .`

`hallo(p) .`

`hallo(q) .`

`foo(a,a) .`

`foo(X,Y) :- huhu(b),hihi(X),!,hallo(Y) .`

`foo(X,c) :- hallo(X),!,huhu(X) .`

■ Anfragen

`?- foo(X,Y) .`

`?- foo(j,Y) .`

`?- foo(p,Y) .`

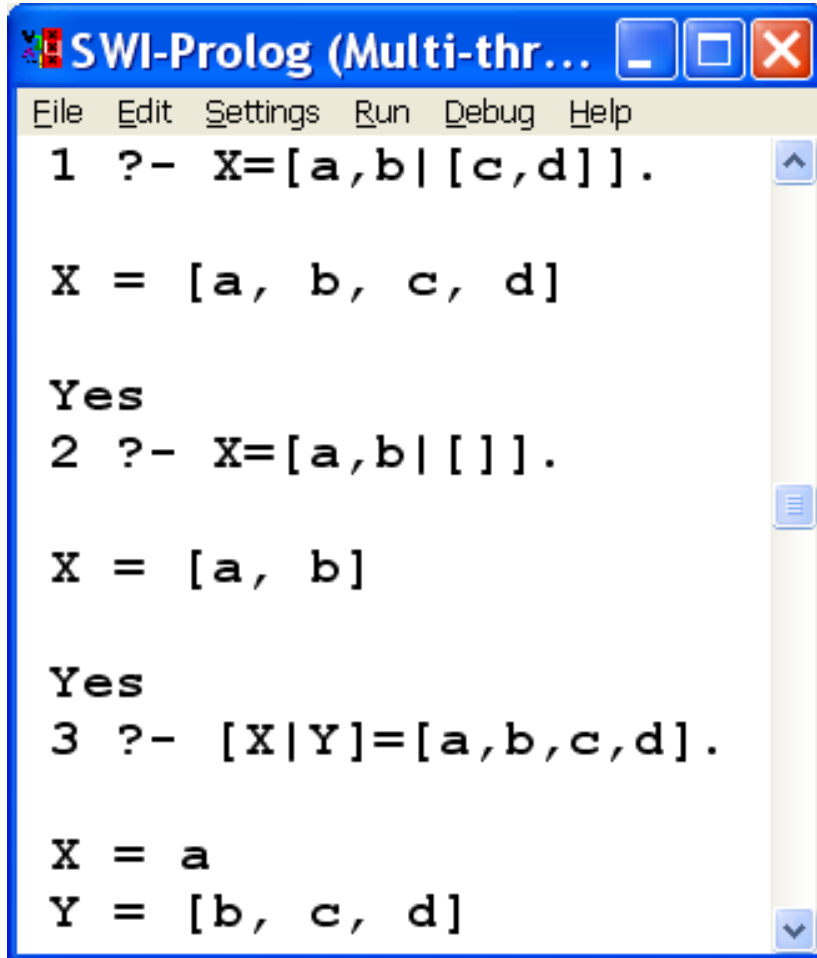
`?- foo(o,Y) .`

`?- foo(X,c) .`

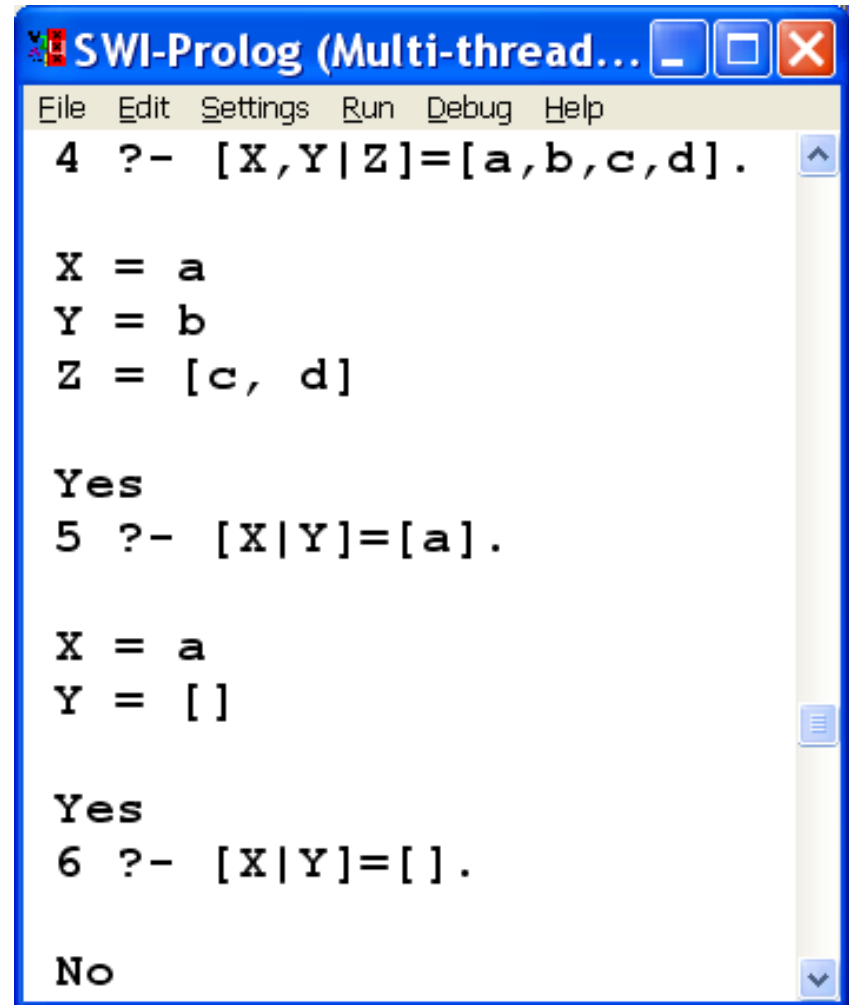
Listen

- Listen bestehen aus einer beliebigen Anzahl von Elementen.
- Elemente können beliebige PROLOG Objekte sein.
 - auch Listen
- Beispiele:
 - `[a,b,c]`
 - `[1,2,3,4]`
 - `[on(block1,block2),on(block3,block4)]`
 - `[a,b,on(block1,block2)]`
 - `[a,b,[x,y,z],c,d]`
 - `[a,3,a,vater(X),Kurt,[],[a,b,Kurt]]`
 - `[]`

Restlistenoperator '|': Zerlegen und Konstruieren



```
SWI-Prolog (Multi-threaded)
File Edit Settings Run Debug Help
1 ?- X=[a,b|[c,d]].
X = [a, b, c, d]
Yes
2 ?- X=[a,b|[]].
X = [a, b]
Yes
3 ?- [X|Y]=[a,b,c,d].
X = a
Y = [b, c, d]
```

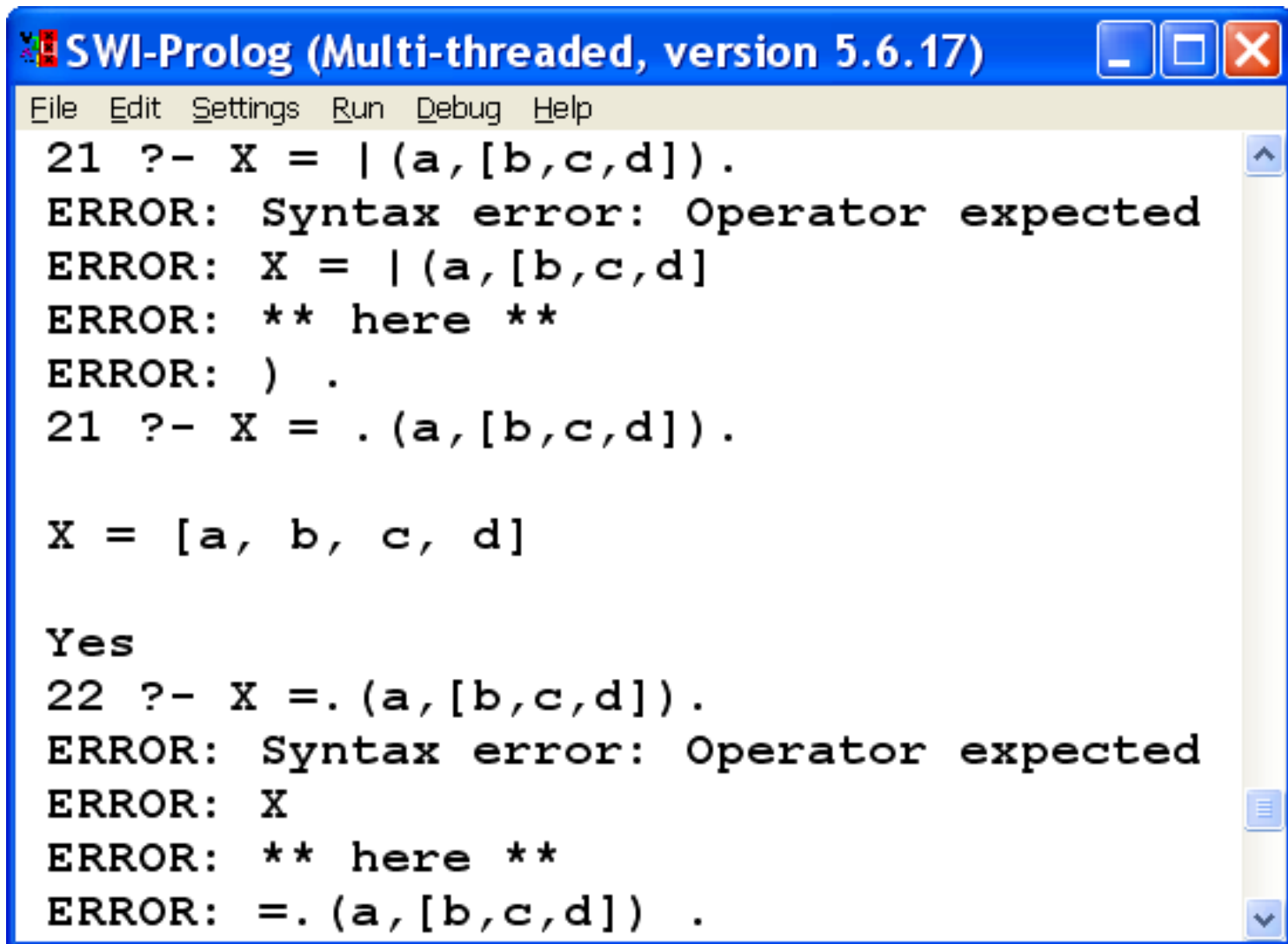


```
SWI-Prolog (Multi-threaded)
File Edit Settings Run Debug Help
4 ?- [X,Y|Z]=[a,b,c,d].
X = a
Y = b
Z = [c, d]
Yes
5 ?- [X|Y]=[a].
X = a
Y = []
Yes
6 ?- [X|Y]=[].
No
```

Übung: Wie antwortet Prolog?

1. ?- [a,b,c] = [a|X] .
2. ?- [a,b,c] = [a,X] .
3. ?- [a,b,c] = [a,b,X] .
4. ?- [a,b,c] = [_,_|X] .
5. ?- [a,b,c] = [_|[X]] .
6. ?- [] = _.
7. ?- [] = [_] .
8. ?- [a,b,c] = [X] .

Restlistenoperator: Präfix-Schreibweise



```
SWI-Prolog (Multi-threaded, version 5.6.17)
File Edit Settings Run Debug Help
21 ?- X = | (a, [b,c,d]) .
ERROR: Syntax error: Operator expected
ERROR: X = | (a, [b,c,d]
ERROR: ** here **
ERROR: ) .
21 ?- X = . (a, [b,c,d]) .

X = [a, b, c, d]

Yes
22 ?- X =. (a, [b,c,d]) .
ERROR: Syntax error: Operator expected
ERROR: X
ERROR: ** here **
ERROR: =. (a, [b,c,d]) .
```

Append

Aufrufbeschreibung von append:

append(?List1, ?List2, ?List3)

Succeeds when **List3** unifies with the concatenation of **List1** and **List2**. The predicate can be used with any instantiation pattern (even three variables).

≡ Definition:

append([], L, L) .

append([H|T], L2, [H|L3]) :- append(T, L2, L3) .

- In der Aufrufbeschreibung von Prolog-Prädikaten wird für alle Komponenten festgelegt, ob sie vorgegeben sein müssen oder ob sie auch berechnet werden dürfen. Das Fragezeichen vor einem Argument deutet darauf hin, dass dieses Argument nicht vorgegeben werden muss, dass hier also eine nicht instantiierte Variable stehen darf. Oder anders ausgedrückt, dass dieses Argument als Ausgabe des Unifikationsprozesses betrachtet werden kann. Allerdings macht es keinen Sinn, alle drei Parameter mit nicht instantiierten Variablen zu belegen.

Append-Beispiele

SWI-Prolog (Multi-threaded, version 5.6.17)

File Edit Settings Run Debug Help

```
1 ?- append([a,b,c],[1,2,3],L3) .
```

```
L3 = [a, b, c, 1, 2, 3]
```

Yes

```
2 ?- append([], [1,2,3], L3) .
```

```
L3 = [1, 2, 3]
```

Yes

```
3 ?- append(L1, [1,2,3], [4,5,6,1,2,3]) .
```

```
L1 = [4, 5, 6]
```

Yes

SWI-Prolog (Multi-threaded, ve... _ □ ×

File Edit Settings Run Debug Help

```
4 ?- append(L1,L2,[a,b,c]) .
```

```
L1 = []
```

```
L2 = [a, b, c] ;
```

```
L1 = [a]
```

```
L2 = [b, c] ;
```

```
L1 = [a, b]
```

```
L2 = [c] ;
```

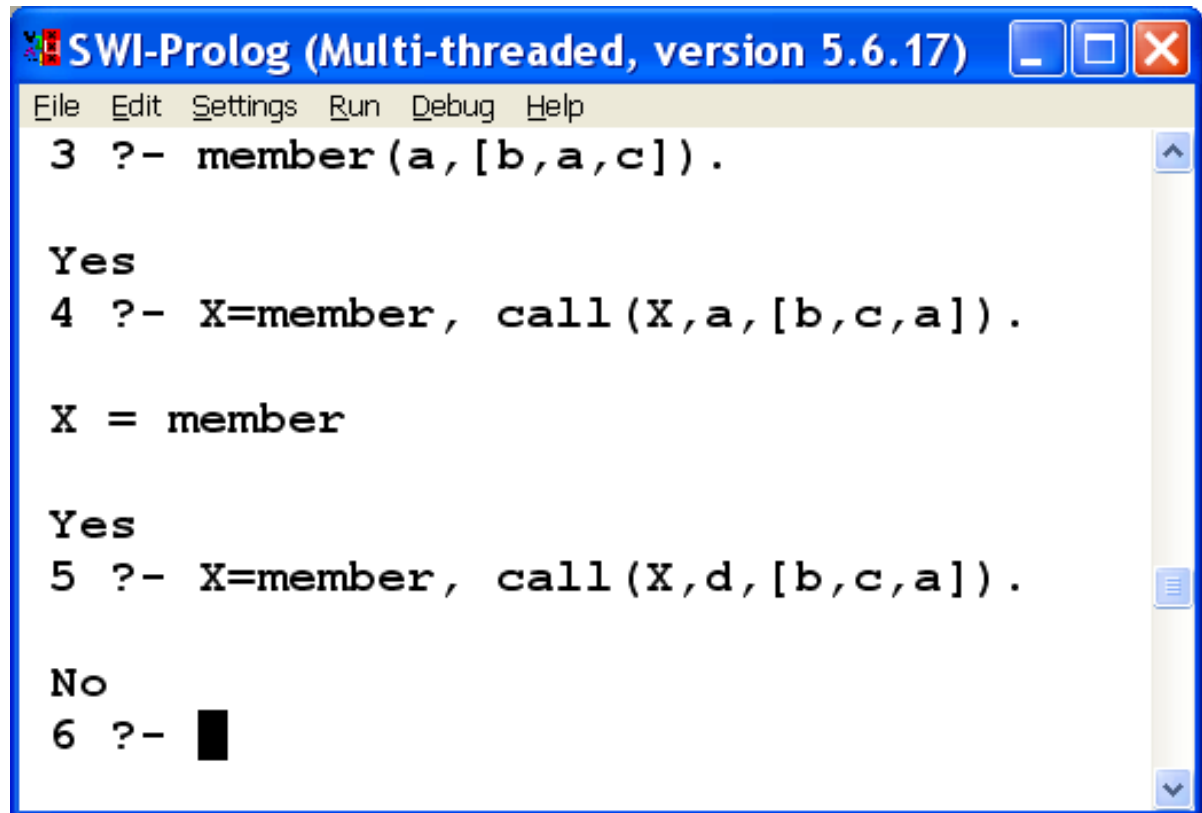
```
L1 = [a, b, c]
```

```
L2 = [] ;
```

No

Call

- Ruft eine Anfrage auf.



```
SWI-Prolog (Multi-threaded, version 5.6.17)
File Edit Settings Run Debug Help
3 ?- member(a,[b,a,c]).

Yes
4 ?- X=member, call(X,a,[b,c,a]).

X = member

Yes
5 ?- X=member, call(X,d,[b,c,a]).

No
6 ?- 
```

Manipulation der Wissensbasis

- Hinzufügen von Fakten und Regeln:

```
assert(ledig(xaver)) .
```

```
assert( (ermaessigung(X) :- alter(X,A) , A =< 12) ) .
```

- Hinzufügen an den Anfang gleichnamiger Prädikate: **asserta**

- Hinzufügen an das Ende gleichnamiger Prädikate: **assertz**

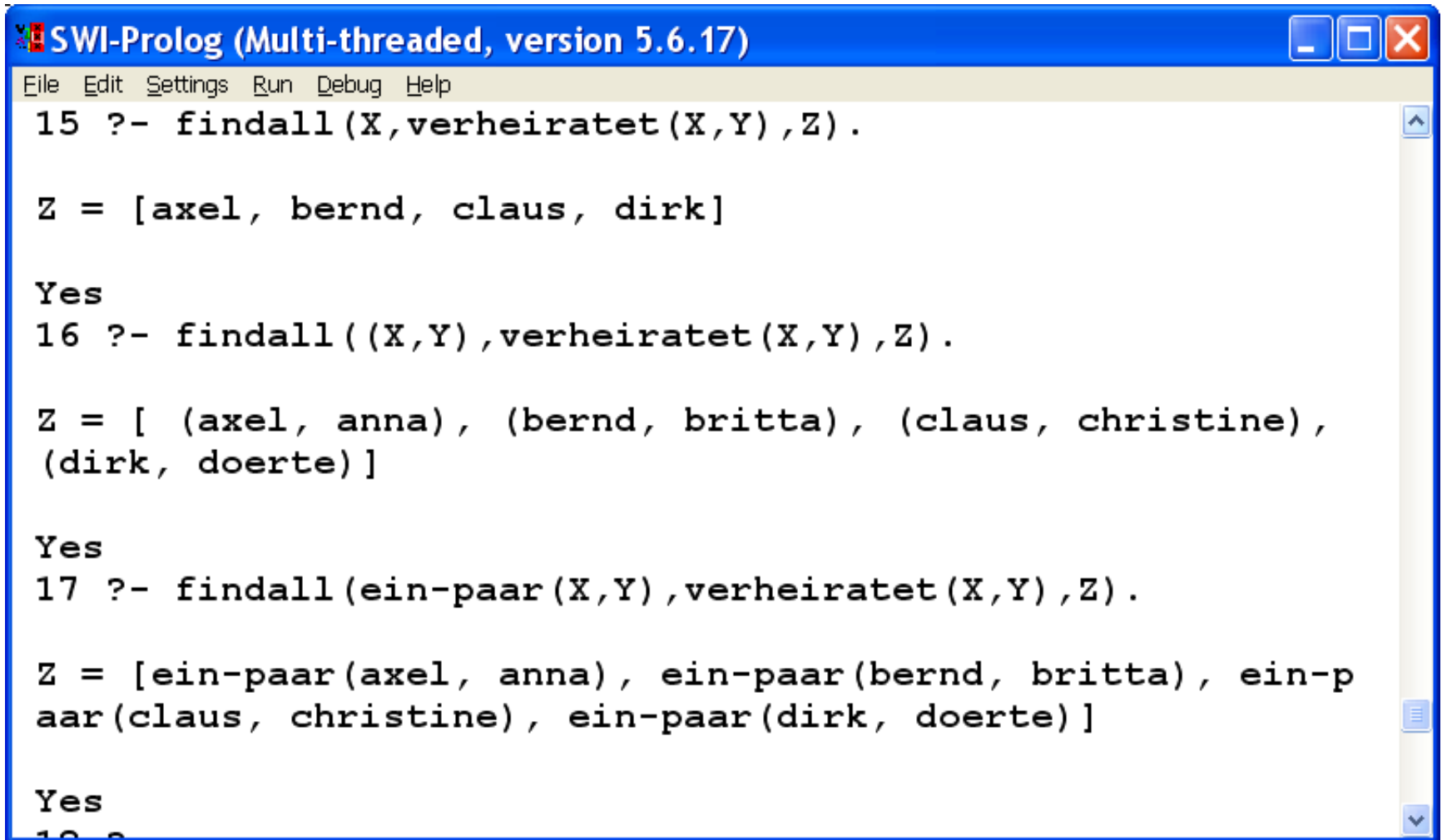
- Löschen: **retract**

- Ausgabe des gesamten Programms: **listing**

- Ausgabe der Klauseln für ein vorgegebenes Prädikat, z. B.

```
listing(ledig) .
```

Alle Lösungen finden: findall



```
SWI-Prolog (Multi-threaded, version 5.6.17)
File Edit Settings Run Debug Help
15 ?- findall(X,verheiratet(X,Y),Z).

Z = [axel, bernd, claus, dirk]

Yes
16 ?- findall((X,Y),verheiratet(X,Y),Z).

Z = [ (axel, anna), (bernd, britta), (claus, christine),
      (dirk, doerte) ]

Yes
17 ?- findall(ein-paar(X,Y),verheiratet(X,Y),Z).

Z = [ein-paar(axel, anna), ein-paar(bernd, britta), ein-paar(claus, christine), ein-paar(dirk, doerte)]

Yes
```

Lernziele P3

- V1: Die Trennung von Logik und Steuerung in Prolog erläutern können
- V2: Closed World Assumption und Open World Assumption erläutern können
- V3: Arbeitsweise von Prolog durch Entwicklung eines Tiefensuch-Baums inklusive Backtracking erläutern können (Dies wird auch im Praktikum geprüft.) (--> Teil 2 der Klausur)