DAI-Labor
TU Berlin

Grundlagen der Künstlichen Intelligenz

Logikbasierte Agenten 2

Prädikatenlogik

24.11.2005

Stefan Fricke
stefan.fricke@dai-labor.de

A|O|T
Agententechnologien in betrieblichen Anwendungen und der Telekommunikation

Gliederung

Aussagenlogik revisited

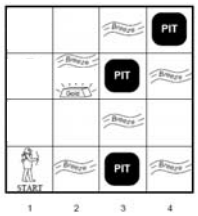
- ⇨ **Repräsentation der Wissensbasis durch Formeln der Aussagenlogik**
  - → Formeln bestehen aus mit Junktoren verknüpften Aussagen
    - → inklusive $\top$ und $\bot$
- ⇨ **Interpretation I weist jeder Aussage einen Wahrheitswert zu**
- ⇨ **Modell ist I mit für $\top$ für jede Formel in Wissensbasis**
- ⇨ **Logische Folgerbarkeit ist durch Wahrheitstabellen beweisbar**

Aussagenlogik revisited

- ⇨ **WB $\models \phi$ durch Widerspruch beweisen: WB $\cup \neg \phi \models \bot$**
- ⇨ **Resolutionsmethode ist ein Inferenzmechanismus basierend auf Formeln konjunktiver Normalformen**
  - → Aus 2 Formeln wird eine neue Formel generiert und der Klauselmenge hinzugefügt
- ⇨ **Resolutionsmethode ist konstruktiv, vollständig und korrekt**

Schwächen der Aussagenlogik

- ⇨ **Modellierung von Umgebungen mit sehr vielen Objekten ist schwierig**
  - → ϕ_1: b[1,1] $\Leftrightarrow$ (p[1,2] $\vee$ p[2,1])
  - → …
  - → ϕ_{16}: b[4,4] $\Leftrightarrow$ (p[4,3] $\vee$ p[3,4])
- ⇨ **Gesucht ist eine kompaktere Form, wie „Nachbarfelder von Falltüren sind windig“**
  - → analog zur menschlichen Sprache, die für die Beschreibung von Situationen Objekte und Relationen / Eigenschaften verwendet.



Von der Aussagenlogik zur Prädikatenlogik

- ⇨ **Aussagenlogik beschreibt die Welt als eine Menge von Fakten**
  - → Symbole mit Verknüpfungen: $\neg, \Rightarrow, \wedge, \vee, \Leftrightarrow$
- ⇨ **Prädikatenlogik beschreibt die Welt durch**
  - → Objekte: Orte, Personen, …
  - → Relationen: links_von/2, windig, …
  - → Funktionen: links_von/1, successor/1

Gliederung

- Einleitung
- **Prädikatenlogik**
  - Formalismus
  - Beispiele
- Inferenzen in der Prädikatenlogik
  - Propositionale Inferenz
  - Unifikation
  - Resolution
- Zusammenfassung

Prädikatenlogik (First Order Logic, FOL) — Terme

- **Konstanten und Variablen sind Terme.**
  - Konstanten beginnen mit Großbuchstaben, z.B. A, B, Wert
  - Variablen beginnen mit Kleinbuchstaben, z.B. x, y, variable
- **Falls *f* ein Funktionssymbol der Stelligkeit *k* ist und falls t_1, ... t_k Terme sind, so ist auch $f(t_1,...,t_k)$ ein Term.**

Prädikatenlogik (First Order Logic, FOL) — Formeln

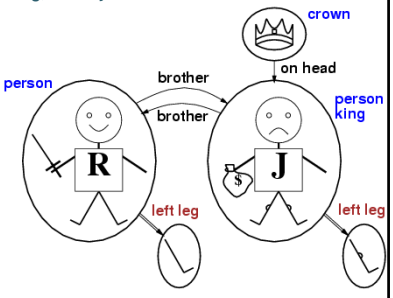
- **Falls *P* ein Prädikatsymbol der Stelligkeit *k* ist, und falls t_1, ... t_k Terme sind, dann ist $P(t_1,...,t_k)$ eine (atomare) Formel.**
- **Für alle Formeln F und G sind auch $\neg F$, $F \wedge G$, $F \vee G$, $F \Rightarrow G$, $F \Leftrightarrow G$ Formeln**
- **Für alle Formeln F und G ist auch $F = G$ eine Formel.**
- **Falls x eine Variable ist und F eine Formel, so sind auch $\exists x\ F$ und $\forall x\ F$ Formeln.**

FOL — Wahrheit in FOL

- **Formeln sind wahr hinsichtlich einer Interpretation**
- **Interpretation bildet ab:**
  - Konstanten → Objekte
  - Prädikatensymbole → Relationen
  - Funktionssymbole → Funktionen
- **Eine Formel $P(t_1,...,t_k)$ ist wahr, genau dann wenn die referenzierten Objekte $t_1,...,t_k$ die Relation des Prädikats P darstellen.**

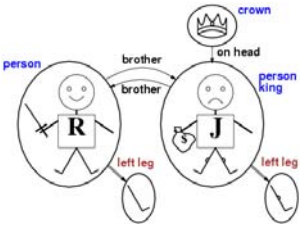
FOL — Beispiel

- **5 Objekte {R,J, 2x left_leg, crown}**
- **2-stellige Relationen {brother, on_head}**
- **Unäre Relationen {person, crown, king}**
- **Unäre Funktion {left_leg}**



FOL — Beispiel: Interpretation

- **Intendierte Interpretation:**
  - R → Richard Löwenherz, J → König John, Crown → Krone
  - brother → Bruderrelation, usw.
- **Viele weitere Interpretationen möglich!**



FOL — Quantoren

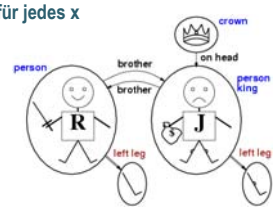
- **Eigenschaften von Objekten aufzählen ist sehr aufwändig:**
  z.B. „Ein König ist immer auch eine Person“
  - King(J) ⇒ Person(J).
  - King(Gustaf) ⇒ Person(Gustaf).
  - …
- **Mittels Quantoren lassen sich Eigenschaften von Mengen von Objekten eleganter ausdrücken, z.B.**
  - ∀x King(x) ⇒ Person(x). (Alle Könige sind Personen)
  - ∃x Crown(x) ∧ OnHead(x, J) (J trägt eine Krone)

FOL — Semantik von Quantoren

∀x P(x) ist wahr genau dann, wenn P(x) wahr für jedes Objekt x.

- 5 mögliche Instanzen für x: R, J, Crown, left_leg(R), left_leg(J).

- **∀x King(x) ⇒ Person(x) ist wahr gdw. King(x) ⇒ Person(x) ist wahr für jedes x**
  - Beweisführung über Wahrheitstabelle
- **∀x King(x) ∧ Person(x) ist falsch!**
  - z.B. für x = Crown unerfüllbar.



FOL — Semantik von Quantoren

∃x P(x) ist wahr gdw. P(x) wahr für mindestens ein Objekt x.

- **∃x King(x) ∧ Person(x) erfüllbar für x=J**

- **Die Verwendung der logischen Implikation ist üblicherweise nicht gewollt, da zu ausdrucksschwach.**
  - Beispiel:
    ∃x Crown(x) ⇒ OnHead(x, J) ist wahr, jedoch für viele Instanzen sinnlos, z.B. Crown(R) ⇒ OnHead(R, J).

FOL — Eigenschaften von Quantoren

- **∃x ∀y P(x, y) ist nicht dasselbe wie ∀y ∃x P(x, y)**
  - Z.B. Interpretation: Loves(x, y) ⟶ x liebt y
  - **∃x ∀y Loves(x, y)**
    „Es gibt eine Person, die alle Personen auf der Welt liebt“
- **Es gelten die deMorgan’schen Regeln:**
  - ∀x ¬P ⇔ ¬∃x P
  - ¬∀x P ⇔ ∃x ¬P
  - ∀x P ⇔ ¬∃x ¬P
  - ¬∀x ¬P ⇔ ∃x P

∀x Likes(x, IceCream)
⇔
¬∃x ¬Likes(x, IceCream)

FOL — Gleichheit

- **$t_1 = t_2$ ist wahr, gdw. t_1 und t_2 auf dasselbe Objekt referenzieren.**

- **z.B.: “R hat mindestens 2 Brüder”:**
  ∃x,y Brother(x, R) ∧ Brother(y, R) ∧ ¬(x = y)

- **Hingegen bedeutet ∃x,y Brother(x, R) ∧ Brother(y, R)**
  “R hat mindestens einen Bruder” (erfüllbar auch für x=y)

Gliederung

Logikbasierte (wissensbasierte) Agenten, revisited

function KB-Agent(percept) **returns** action

static KB, t // Knowledge Base, time counter

TELL(KB, Make_Percept_Sentence(percept, t))

action := **ASK**(KB, Make_Action_Query(t))

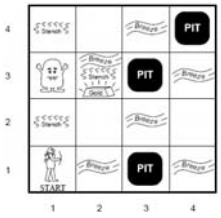
TELL(KB, Make_Action_Sentence(action, t))

t := t + 1

return action

FOL-Beispiele — Agenten in der Wumpus Welt

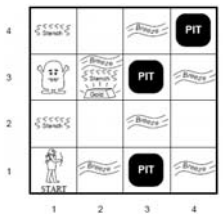
- **Sensoren für Stench, Breeze, Glitter**
- **Aktionen: Linksdrehung, Rechtsdrehung, Vorwärts laufen, Greifen**
- **Anfrage nach bester Aktion**



Charakterisierung der Wumpus-Umgebung:
¬beobachtbar ∧ deterministisch ∧ ¬episodisch ∧ statisch ∧ diskret ∧ single-agent

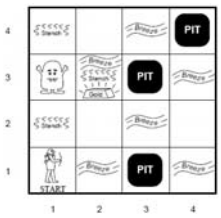
FOL-Beispiele — Repräsentation der Wumpus Welt

- **Sensorik: Percept(Stench, Breeze, None, 5)**
- **Aktionsterme: Turn(Right), Turn(Left), Forward, Grab.**
- **Anfrage (beste Aktion): ∃a BestAction(a, 5)**
  - ASK antwortet mit z.B. a = Forward
- **Wahrnehmung**
  - ∀b,g,t Percept(Smell, b, g, t) ⇒ Smell(t)
  - ∀s,g,t Percept(s, Breeze, g, t) ⇒ Breeze(t)
  - ∀s,b,t Percept(s, b, Glitter, t) ⇒ Glitter(t)



FOL-Beispiele — Repräsentation der Wumpus Welt

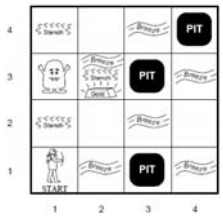
- **Reflex-Verhalten**
  - ∀t Glitter(t) ⇒ BestAction(Grab, t)
- **Reflex mit internem Zustand**
  - ∀t Glitter(t) ∧¬Holding(Gold, t) ⇒ BestAction(Grab, t)
- **Holding(Gold, t) kann nicht über Perceptions beobachtet werden, muss also im Zustand / in der Wissensbasis vermerkt werden**



FOL-Beispiele — Repräsentation der Wumpus Welt

„Nachbarschaftsbeziehung“:

∀x,y,a,b *Adjacent*([x,y], [a,b]) ⇔
[a,b] ∈ { [x+1,y], [x-1,y], [x,y+1], [x,y-1] }



(die Listennotation […] ist trivial auf FOL-Terme abbildbar)
(die Repräsentation von Mengen befindet sich im Anhang)
(Annahme: unzulässige Felder werden zurückgewiesen)

FOL-Beispiele — Versteckte Eigenschaften in der Wumpus-Welt

„Nachbarfelder von Grube sind windig“:

- **“Diagnostische Regeln”**
  - … schließen von beobachteten Effekten auf versteckte Ursachen
  - ∀s Breese(s) ⇒ ∃r Adjacent(r, s) ∧ Pit(r)
  - ∀s ¬Breese(s) ⇒ ¬∃r Adjacent(r, s) ∧ Pit(r)
- **“Kausale” Regeln**
  - … schließen von (versteckten) Eigenschaften auf Wahrnehmungen
  - ∀r Pit(r) ⇒ (∀s Adjacent(r, s) ⇒ Breeze(s))
  - ∀s (∀r Adjacent(r, s) ⇒ ¬Pit(r)) ⇒ ¬Breeze(s)

Gliederung

- Einleitung
- Prädikatenlogik
  - Formalismus
  - Beispiele
- Inferenzen in der Prädikatenlogik
  - Propositionale Inferenz
  - Unifikation
  - Resolution
- Zusammenfassung

FOL mit propositionaler Inferenz — Idee

1. Überführung einer FOL-Wissensbasis in eine Wissensbasis mit Aussagenlogik
2. Danach Anwendung eines aussagenlogischen Inferenzmechanismus
   - z.B. Model Checking oder Resolutionsmethode

FOL mit propositionaler Inferenz — Allquantoren auflösen

- Inferenzregel für Allquantoren: Ersetze jede allquantifizierte Variable mit allen Ground-Termen der Wissensbasis:
- $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$ wird ersetzt durch:

$King(J) \wedge Greedy(J) \Rightarrow Evil(J)$
$King(R) \wedge Greedy(R) \Rightarrow Evil(R)$
$King(Brother(R,J)) \wedge Greedy(Brother(R,J)) \Rightarrow Evil(Brother(R,J))$
$King(Brother(J,R)) \wedge Greedy(Brother(J,R)) \Rightarrow Evil(Brother(J,R))$
usw.

FOL mit propositionaler Inferenz — Existenzquantoren auflösen

- Inferenzregel für Existenzquantoren: Ersetze jede existenzquantifizierte Variable mit einem neuen Konstantensymbol:
- $\exists x\ Crown(x) \wedge OnHead(x, J)$ wird ersetzt durch:

$Crown(C1) \wedge OnHead(C1, J)$.

- Die resultierende Wissensbasis WB' ist nicht äquivalent zur ursprünglichen WB, aber erfüllbar gdw. WB erfüllbar.

FOL mit propositionaler Inferenz — Propositionalisierung

- Alle Atome der Wissensbasis werden als Propositionen betrachtet.

Die Wissensbasis

$King(J) \wedge Greedy(J) \Rightarrow Evil(J)$.

$King(Brother(R,J)) \wedge Greedy(Brother(R,J)) \Rightarrow Evil(Brother(R,J))$.

enthält die 6 Propositionen:

{ King(J), Greedy(J), Evil(J), King(Brother(R,J)),
Greedy(Brother(R,J)), Evil(Brother(R,J)) }

Gliederung

- Einleitung
- Prädikatenlogik
  - Formalismus
  - Beispiele
- Inferenzen in der Prädikatenlogik
  - Propositionale Inferenz
  - Unifikation
  - Resolution
- Zusammenfassung

Unifikation — Idee

⇨ **Aus der Wissensbasis**

King(x) ∧ Greedy(x) ⇒ Evil(x).
King(John).
Greedy(John).

kann Evil(John) direkt, ohne Generierung unnötiger Propositionen, abgeleitet werden, …

⇨ **… durch Substitution von Variablen.**

Unifikation — Substitution

⇨ **Eine Substitution θ ist eine endliche Menge der Form $\{v_1/t_1, \ldots, v_n/t_n\}$, wobei jedes v_i eine Variable ist und jedes t_i ein Term, der verschieden ist von v_i und die Variablen $v_1, \ldots, v_n$ disjunkt sind.**

→ Jedes Element v_i/t_i stellt eine Bindung von v_i dar.

⇨ **Sei $\theta = \{v_1/t_1, \ldots, v_n/t_n\}$ eine Substitution und E ein Ausdruck. Dann ist Eθ der Ausdruck, der gewonnen wird aus E, indem man simultan jedes Vorkommen der Variablen v_i in E durch den Term t_i ersetzt.**

Unifikation — Unifikator

⇨ **Sei S eine endliche Menge von einfachen Ausdrücken. Eine Substitution θ ist ein Unifikator für S, wenn Sθ eine einelementige Menge ist.**

Beispiel:

⇨ S = { a, P(b), P(Q(c)) } | S | = 3

⇨ θ = { a / P(Q(z)), b / Q(z), c / z }

⇨ Sθ = { P(Q(z)), P(Q(z)), P(Q(z)) }
= { P(Q(z)) } | Sθ | = 1

Unifikation — Beispiel

p	q	θ
Knows(John, x)	Knows(John, Jane)	{ x/Jane }
Knows(John, x)	Knows(y, OJ)	
Knows(John, x)	Knows(y, Mom(y))	
Knows(John, x)	Knows(x, OJ)	

⇨ **Unify(α ,β) = θ wenn αθ = βθ**

Unifikation — Beispiel

p	q	θ
Knows(John, x)	Knows(John, Jane)	{ x/Jane }
Knows(John, x)	Knows(y, OJ)	{ x/OJ, y/John }
Knows(John, x)	Knows(y, Mom(y))	
Knows(John, x)	Knows(x, OJ)	

Unifikation — Beispiel

p	q	θ
Knows(John, x)	Knows(John, Jane)	{ x/Jane }
Knows(John, x)	Knows(y, OJ)	{ x/OJ, y/John }
Knows(John, x)	Knows(y, Mom(y))	{ y/John, x/Mom(John) }
Knows(John, x)	Knows(x, OJ)	

Unifikation — Beispiel

p	q	θ
Knows(John, x)	Knows(John,Jane)	{ x/Jane }
Knows(John, x)	Knows(y, OJ)	{ x/OJ, y/John }
Knows(John, x)	Knows(y, Mom(y))	{ y/John, x/Mom(John) }
Knows(John, x)	Knows(x, OJ)	{ FAIL }

Unifikation — MGU

- 2 mögliche Substitutionen für unify(*Knows(John,x), Knows(y,z)*) :
  - (1) α = {y/John, x/z } und
  - (2) α = {y/John, x/John, z/John}.
- Unterschied: Unifier (1) ist genereller als (2).
- θ ist **Most General Unifier (MGU)**, wenn für jeden Unifikator ρ von S eine Substitution γ existiert sodass ρ = θγ.
  - → Für jede Unifikation existiert genau ein MGU

Unifikationsalgorithmus

```
Eingabe: eine Literalmenge L ≠ Ø
θ := { }; (leere Substitution)
while | Lθ | > 1 do
    Suche in Lθ die erste Position, an der sich zwei Literale
        L1 und L2 unterscheiden
    if keiner der beiden Terme ist eine Variable
        then stoppe mit „nicht unifizierbar“
        else Sei x die Variable in L1 und t der Term in L2
            if x kommt in t vor
                then stoppe mit „nicht unifizierbar“
                else θ := θ ∪ { x/t }
Ausgabe: θ ist MGU
```


Unifikationsalgorithmus — Terminierung und Korrektheit

- In jedem Schritt wird entweder durch Substitution die Menge der Variablen in Lθ kleiner, oder es wird ein Paar nicht unifizierbarer Literale gefunden und das Verfahren bricht ab.
- Die while-Schleife wird nur dann erfolgreich verlassen, wenn | Lθ | = 1, d.h. wenn ein Unifikator gefunden ist.
- Das heißt, der Algorithmus terminiert und ist korrekt.

Unifikationsalgorithmus — Beispiel

0. $L = \{Q(x, f (A)), Q(B, y)\}$ $\theta = \{ \}$ $|L\theta| = 2$
1. $L_1 = Q(x, f(A))$ $L_2 = Q(B, y)$
   $t_1 = x$ $t_2 = B$ (unterschiedliche Terme)
   $\theta = \{x/B\}$ (Substitution)
2. $L_1\theta = Q(B, f(A))$ $L_2\theta = Q(B, y)$ $|L\theta| = 2$
   $t_1 = f(A)$ $t_2 = y$
   $\theta = \{ x/B, y/f(A) \}$
3. $L_1\theta = Q(B, f(A))$ $L_2\theta = Q(B, f(A))$ $|L\theta| = 1$

Gliederung

Resolution, revisited

⇨ **Robinson'sche Resolutionsregel der Aussagenlogik:**

→ Seien $L_1, \ldots, L_n$ und $L^*_1, \ldots, L^*_m$ Literale mit $L = L_i = \neg L^*_j$ für ein i mit $1 \le i \le n$ und ein j mit $1 \le j \le m$. Dann gilt:

$$(L_1 \vee \ldots \vee L_i \vee \ldots \vee L_n) \wedge (L^*_1 \vee \ldots \vee \neg L_i \vee \ldots \vee L^*_n)$$
$$\models$$
$$L_1 \vee \ldots \vee L_{i-1} \vee L_{i+1} \vee \ldots \vee L_n \vee L^*_1 \vee \ldots \vee L^*_{i-1} \vee L^*_{i+1} \vee \ldots \vee L^*_n$$

⇨ **Eine Klauselmenge ist unerfüllbar genau dann, wenn die leere Klausel abgeleitet werden kann.**

FOL-Resolution — Idee

⇨ **Anwendung des Resolutionsverfahrens für die Prädikatenlogik**

→ Umgang mit Quantoren

→ Umgang mit Variablen

⇨ **Ansatz**

1. Bildung der Konjunktiven Normalform
2. Erweiterung der aussagenlogischen Resolution um die Behandlung von Variablen durch Unifikation

FOL-Resolution — Konjunktive Normalform

⇨ **KNF besteht aus einer Konjunktion von Disjunktionen von Literalen.**

→ Variablen in Literalen werden als allquantifiziert angenommen

⇨ **Alle Ausdrücke der Prädikatenlogik können in eine inferenziell äquivalente KNF transformiert werden**

→ Hauptproblem: Existenzquantoren eliminieren

→ Das Verfahren zur KNF-Transformation besteht aus 5 Schritten…

FOL-Resolution — Konvertierung in KNF

⇨ **"Jeder, der alle Tiere liebt, wird von irgendjemandem geliebt":**

→ $\forall x\, [\forall y\, Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\, Loves(y,x)]$

→ y bezeichnet links und rechts der Implikation unterschiedliche Variablen!

1. **Implikationen eliminieren**

$$\forall x\, [\forall y\, \neg Animal(y) \vee Loves(x,y)] \Rightarrow [\exists y\, Loves(y,x)]$$
$$\Leftrightarrow$$
$$\forall x\, [\neg \forall y\, \neg Animal(y) \vee Loves(x,y)] \vee [\exists y\, Loves(y,x)]$$

FOL-Resolution — Konvertierung in KNF

→ $\forall x\, [\neg \forall y\, \neg Animal(y) \vee Loves(x,y)] \vee [\exists y\, Loves(y,x)]$

2. **Negierte Quantoren auflösen:**

$$\forall x\, [\exists y\, \neg(\neg Animal(y) \vee Loves(x,y))] \vee [\exists y\, Loves(y,x)]$$
$$\Leftrightarrow \forall x\, [\exists y\, \neg\neg Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y\, Loves(y,x)]$$
$$\Leftrightarrow \forall x\, [\exists y\, Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y\, Loves(y,x)]$$

$\neg\forall x\, p \equiv \exists x\, \neg p$	$\neg\exists x\, p \equiv \forall x\, \neg p$

FOL-Resolution — Konvertierung in KNF

→ $\forall x\, [\exists y\, Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y\, Loves(y,x)]$

3. **Variablen standardisieren**

→ $\forall x\, [\exists y\, Animal(y) \wedge \neg Loves(x,y)] \vee [\exists \mathbf{z}\, Loves(\mathbf{z},x)]$

4. **Existenzquantifizierte Variablen skolemisieren durch Skolemfunktion über alle beeinflussenden allquantifizierten Variablen**

→ $\forall x\, [Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee Loves(G(x),x)$

FOL-Resolution — Konvertierung in KNF

→ $\forall x\,[Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee Loves(G(x),x)$

5. **Universalquantoren entfernen und KNF-Konnektoren bilden**

$[Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee Loves(G(x),x)$

$\Leftrightarrow$

$[Animal(F(x)) \vee Loves(G(x),x)]$
$\wedge\,[\neg Loves(x,F(x)) \vee Loves(G(x),x)]$

FOL-Resolution — Beispiel

⇨ **Es ist ein Verbrechen für einen Amerikaner, Waffen an feindliche Staaten zu verkaufen. Nono besitzt Raketen, die es sämtlich von Colonel West kaufte. Raketen sind Waffen. West ist Amerikaner. Nono ist Feind Amerikas.**

⇨ **Frage:**
(Wie) kann bewiesen werden, dass West ein Verbrecher ist?

FOL-Resolution — Beispiel: Formalisierung

⇨ **$\forall x,y,z$ American(x) $\wedge$ Weapon(y) $\wedge$ Sells(x, y, z) $\wedge$ Hostile(z) $\Rightarrow$ Criminal(x).**

⇨ **$\exists x$ Owns(Nono, x) $\wedge$ Missile(x)**

⇨ **Missile(x) $\wedge$ Owns(Nono, x) $\Rightarrow$ Sells(West, x, Nono)**

⇨ **Missile(x) $\Rightarrow$ Weapon(x)**

⇨ **American(West)**

⇨ **Enemy(Nono, America)**

⇨ **Enemy(x, America) $\Rightarrow$ Hostile(x)** (Hilfsimplikation)

FOL-Resolution — Beispiel in KNF

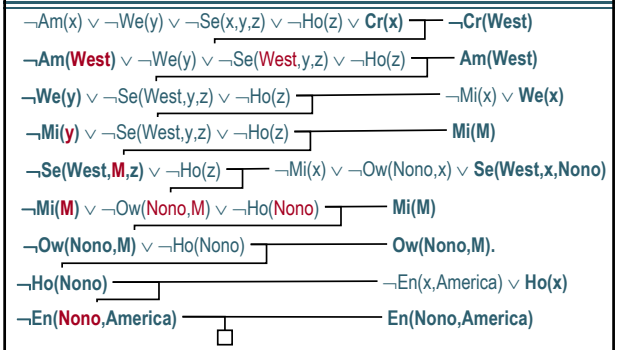
Wissensbasis in KNF:

⇨ **$\neg$American(x) $\vee$ $\neg$Weapon(y) $\vee$ $\neg$Sells(x, y, z) $\vee$ $\neg$Hostile(z) $\vee$ Criminal(x).**
$\neg$Missile(x) $\vee$ $\neg$Owns(Nono, x) $\vee$ Sells(West, x, Nono).
$\neg$Enemy(x, America) $\vee$ Hostile(x).
$\neg$Missile(x) $\vee$ Weapon(x).
Owns(Nono, M). **Missile(M).**
American(West). **Enemy(Nono, America).**

Anfrage negiert in WB aufnehmen:

⇨ **$\neg$Criminal(West).**

Resolutionsbeweis in FOL verwendet Unifikation



Gliederung

Zusammenfassung

- **FOL besitzt stärkere Ausdruckskraft als Aussagenlogik**
  - Objekte und Relationen als semantikbehaftete Primitive
  - Hinreichend mächtig, um die Wumpus Welt zu beschreiben
- **Unifikation macht logische Ausdrücke durch Substitution einander ähnlich**
  - alle FOL-Inferenzverfahren nutzen Unifikation
- **Konstruktion der KNF**
  - insbesondere Skolemisierung existenzquantifizierter Variablen
- **Resolution ist konstruktive Beweisführung**

Ausblick

- **Inferenzen mittels Forward Chaining und Backward Chaining, wie in der VL Planen besprochen, sind in FOL ebenso möglich.**
- **Komplexitätsproblem: Kombinatorische Explosion beim Durchsuchen des Resolutionsgraphen**
  - Zumeist sind nur wenige Resolventen zur Ableitung der leeren Klausel nötig
  - Heuristiken zur Reihenfolgebestimmung
  - Resolutionsrestriktionen (z.B. SLD-Resolution in PROLOG)
  - Weitere Heuristiken zur effizienteren Resolution in Russell/Norvig

DAI-Labor
TU Berlin

Grundlagen der Künstlichen Intelligenz

Logikbasierte Agenten 2: FOL

24.11.2005

nächster Termin: Nichtmonotones Schließen

Stefan Fricke
stefan.fricke@dai-labor.de

A|O|T
Agententechnologien in betrieblichen Anwendungen und der Telekommunikation

Anhänge

FOL-Beispiele — Verwandschaftsbeziehungen

- **Brüder sind Geschwister**
  $\forall x,y\ Brother(x, y) \Leftrightarrow Sibling(x, y)$
- **Geschwisterschaft ist symmetrisch**
  $\forall x,y\ Sibling(x, y) \Leftrightarrow Sibling(y, x)$
- **Eine Mutter ist ein weibliches Elternteil**
  $\forall m,c\ Mother(c) = m \Leftrightarrow (Female(m) \wedge Parent(m, c))$
- **Ein Onkel (o) ist der Bruder (b) eines Elternteils (p)**
  $\forall x,o\ Uncle(o, x) \Leftrightarrow \exists p\ Parent(p, x) \wedge Brother(o, p)$

FOL-Beispiele — Mengen

1. $\forall s\ Set(s) \Leftrightarrow (s = \{\}) \vee (\exists x,s_2\ Set(s_2) \wedge s = \{x|s_2\})$ (Konstruktion)
2. $\neg\exists x,s\ \{x|s\} = \{\}$ (Konsistenz)
3. $\forall x,s\ x \in s \Leftrightarrow s = \{x|s\}$ (Elemente)
4. $\forall x,s\ x \in s \Leftrightarrow \exists y,s_2\ (s = \{y|s_2\} \wedge (x = y \vee x \in s_2))$ (Elemente)
5. $\forall s_1,s_2\ s_1 \subseteq s_2 \Leftrightarrow (\forall x\ x \in s_1 \Rightarrow x \in s_2)$ (Teilmenge)
6. $\forall s_1,s_2\ (s_1 = s_2) \Leftrightarrow (s_1 \subseteq s_2 \wedge s_2 \subseteq s_1)$ (Gleichheit)
7. $\forall x,s_1,s_2\ x \in (s_1 \cap s_2) \Leftrightarrow (x \in s_1 \wedge x \in s_2)$ (Schnittmenge)
8. $\forall x,s_1,s_2\ x \in (s_1 \cup s_2) \Leftrightarrow (x \in s_1 \vee x \in s_2)$ (Vereinigung)

Unifikationsalgorithmus

```
function UNIFY(x, y, θ) returns a substitution to make x and y identical
  inputs: x, a variable, constant, list, or compound
          y, a variable, constant, list, or compound
          θ, the substitution built up so far
  if θ = failure then return failure
  else if x = y then return θ
  else if VARIABLE?(x) then return UNIFY-VAR(x, y, θ)
  else if VARIABLE?(y) then return UNIFY-VAR(y, x, θ)
  else if COMPOUND?(x) and COMPOUND?(y) then
      return UNIFY(ARGS[x], ARGS[y], UNIFY(OP[x], OP[y], θ))
  else if LIST?(x) and LIST?(y) then
      return UNIFY(REST[x], REST[y], UNIFY(FIRST[x], FIRST[y], θ))
  else return failure
```


Unifikationsalgorithmus

```
function UNIFY-VAR(var, x, θ) returns a substitution
  inputs: var, a variable
          x, any expression
          θ, the substitution built up so far
  if {var/val} ∈ θ then return UNIFY(val, x, θ)
  else if {x/val} ∈ θ then return UNIFY(var, val, θ)
  else if OCCUR-CHECK?(var, x) then return failure
  else return add {var/x} to θ
```


Forward chaining algorithm

```
function FOL-FC-ASK(KB, α) returns a substitution or false
  repeat until new is empty
      new ← { }
      for each sentence r in KB do
          (p1 ∧ ... ∧ pn ⇒ q) ← STANDARDIZE-APART(r)
          for each θ such that (p1 ∧ ... ∧ pn)θ = (p'1 ∧ ... ∧ p'n)θ
                       for some p'1, ..., p'n in KB
              q' ← SUBST(θ, q)
              if q' is not a renaming of a sentence already in KB or new then do
                  add q' to new
                  φ ← UNIFY(q', α)
                  if φ is not fail then return φ
      add new to KB
  return false
```


Backward chaining algorithm

```
function FOL-BC-ASK(KB, goals, θ) returns a set of substitutions
  inputs: KB, a knowledge base
          goals, a list of conjuncts forming a query
          θ, the current substitution, initially the empty substitution { }
  local variables: ans, a set of substitutions, initially empty
  if goals is empty then return {θ}
  q' ← SUBST(θ, FIRST(goals))
  for each r in KB where STANDARDIZE-APART(r) = (p1 ∧ ... ∧ pn ⇒ q)
          and θ' ← UNIFY(q, q') succeeds
    ans ← FOL-BC-ASK(KB, [p1, ..., pn|REST(goals)], COMPOSE(θ, θ')) ∪ ans
  return ans
```

SUBST(COMPOSE(α_1, α_2), p) = SUBST(α_2, SUBST(α_1, p))