Lernen von $\mathcal{ALC}$-Konzepten in Beschreibungslogiken und Ontologien

Jens Lehmann

Institut für Künstliche Intelligenz
Fakultät Informatik
Technische Universität Dresden

21. Juni 2006

Gliederung:

- Einführung in Beschreibungslogiken, insbesondere $\mathcal{ALC}$
- Ontologiesprache OWL-DL
- Erklärung des Lernproblems für Konzepte in Beschreibungslogiken
- Anwendungsszenarien
- Ansätze zur Lösung des Lernproblems
- Ausblick

Beschreibungslogik

- *Beschreibungslogiken* (engl. *Description Logics*) ist der Name für eine Familie von Sprachen zur Wissensrepräsentation
- Fragmente Prädikatenlogik erster Ordnung
- weniger mächtig als Prädikatenlogik, aber dafür entscheidbare Inferenzprobleme
- intuitive variablenfreie Syntax

Wissensmodellierung

- Repräsentation von Wissen mit Rollen, Konzepten und Objekten/Individuen
- Objekt:
  - Konstante
  - Beispiele: Mary, John
- Konzept:
  - einstelliges Prädikat
  - Menge von Objekten
  - Beispiele: Vater, Student
- Rolle:
  - zweistelliges Prädikat
  - beschreibt Verbindung zwischen Objekten
  - Beispiele: ist-Schwester-von, ist-Kind-von

Wissensbasen

- Wissen über eine Domain wird in Wissensbasis mit folgender Struktur gespeichert:

Wissensbasis

TBox („terminology“), z.B.

$\texttt{Woman} \equiv \texttt{Human} \sqcap \texttt{Female}$

$\texttt{Mother} \equiv \texttt{Female} \sqcap \exists \texttt{hasChild}.\top$

$\texttt{HappyFather} \sqsubseteq \texttt{Father} \sqcap \forall \texttt{hasChild}.\texttt{Female}$

ABox („assertions“), z.B.

$\texttt{Mother}(\texttt{MONICA})$

$\texttt{hasChild}(\texttt{MONICA}, \texttt{JESSICA})$

Syntax von $\mathcal{ALC}$-Konzepten

- es sei N_R eine Menge von Rollennamen und N_C eine Menge von Konzeptnamen ($N_R \cap N_C = \emptyset$)
- Konzepte sind dann induktiv wie folgt definiert:
  1. Jeder Konzeptname ist ein Konzept (atomares Konzept).
  2. Falls C und D $\mathcal{ALC}$-Konzepte sind und $r \in N_R$ eine Rolle, dann sind folgende Ausdrücke auch $\mathcal{ALC}$-Konzepte:
     - $\top$ (Top), $\bot$ (Bottom)
     - $C \sqcup D$ (Disjunktion), $C \sqcap D$ (Konjunktion), $\neg C$ (Negation)
     - $\forall r.C$ (universelle Restriktion), $\exists r.C$ (existentielle Restriktion)

Syntax von Wissensbasen

- Axiome (TBox):
  - Äquivalenz (C, D Konzepte): $C \equiv D$
    Beispiel: $\texttt{Mother} \equiv \texttt{Female} \sqcap \exists \texttt{hasChild}.\top$
  - Ungleichheit (C, D Konzepte): $C \sqsubseteq D$
    Beispiel: $\texttt{HappyFather} \sqsubseteq \texttt{Father} \sqcap \forall \texttt{hasChild}.\texttt{Female}$
- Zusicherungen (ABox):
  - Konzeptzusicherung (C Konzept, a Objekt): $C(a)$
    Beispiel: $\texttt{Mother}(\texttt{MONICA})$
  - Rollenzusicherung (r Rolle, a, b Objekte): $r(a, b)$
    Beispiel: $\texttt{hasChild}(\texttt{MONICA}, \texttt{JESSICA})$
- neben Syntax muss natürlich auch Semantik definiert werden (wird hier weggelassen)

Reasoning

- TBox:
  - *Subsumption zwischen Konzepten*: $C \sqsubseteq_{\mathcal{T}} D$ bedeutet, dass D allgemeiner als C bzgl. $\mathcal{T}$ ist, z.B. $\texttt{Mother} \sqsubseteq_{\mathcal{T}} \texttt{Woman}$
  - *Erfüllbarkeit von Konzepten*: z.B. ist $\texttt{Male} \sqcap \texttt{Female}$ unerfüllbar, wenn die zugehörige TBox das Axiom $\texttt{Male} \equiv \neg\texttt{Female}$ enthält
- ABox (und TBox):
  - *Konsistenz*: ABox ist konsistent, falls sie widerspruchsfrei ist
  - *Instanzproblem*: es wird getestet, ob ein Objekt zu einem Konzept gehören muss
  - *Retrievalproblem*: zu einem Konzept werden alle Instanzen gefunden

OWL

- Ontologie = formales System zur Wissensrepräsentation mit Konzepten und Relationen
- OWL ist Akronym für Web Ontology Language
- 3 Varianten: OWL-Lite, OWL-DL, OWL-Full
- OWL-DL basiert auf Beschreibungssprache $\mathcal{SHOIN}(D)$ (mächtiger als $\mathcal{ALC}$)
- weit verbreiteter Standard im Semantic Web
- viele Ontologie-Editoren (z.B. Protégé) und Reasoner (z.B. KAON2, Pellet, Racer, FACT) existieren für OWL-DL

OWL

Darstellung von $\texttt{Mother} \equiv \texttt{Female} \sqcap \exists \texttt{hasChild}.\top$:

```
<owl:Class rdf:ID="mother">
 <owl:equivalentClass>
  <owl:Class>
   <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:ID="female"/>
    <owl:Restriction>
     <owl:onProperty>
      <owl:ObjectProperty rdf:ID="hasChild"/>
     </owl:onProperty>
     <owl:someValuesFrom rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
    </owl:Restriction>
   </owl:intersectionOf>
  </owl:Class>
 </owl:equivalentClass>
</owl:Class>
```

Lernproblem

- Zielkonzept (`Target`) wird aus gegebener Wissensbasis $\mathcal{K}$ als Hintergrundwissen, sowie positiven und negativen Beispielen gelernt
- Beispiele haben die Form `Target`(a) (a ist ein Objekt)
- es sei E^+ die Menge der positiven Beispiele, E^- die Menge der negativen Beispiele
- Ziel: finden einer Definition *Def* der Form `Target` $\equiv C$, so dass sich $\mathcal{K}' \models E^+$ und $\mathcal{K}' \not\models E^-$ für $\mathcal{K}' = \mathcal{K} \cup \{Def\}$ ergibt, d.h. die positiven Beispiele folgen aus $\mathcal{K}'$ und die negativen Beispiele folgen nicht aus $\mathcal{K}'$

Mögliche Anwendungen

- ähnliche Anwendungsgebiete wie in der Induktiven Logikprogrammierung (ILP), bei der Logikprogramme aus Beispielen und Hintergrundwissen gelernt werden
- ILP wird z.B. in Biologie, Medizin und Sprachverarbeitung angewandt
- inkrementelles Lernen von OWL-Ontologien im Semantic Web
- Vorteile gegenüber manuellem Definieren von Konzepten:
  - Definition ist konsistent mit dem bereits in der Wissensbasis gespeicherten Wissen
  - Bearbeiter der Wissensbasis kennt manchmal genaue Definition nicht oder ist sich unsicher, ob er alle relevanten Details erfasst hat

Beispiel 1 - Vater

$\texttt{Male} \equiv \neg\texttt{Female}$	$\texttt{Male}(\texttt{MARC})$
	$\texttt{Male}(\texttt{STEPHEN})$
$\texttt{hasChild}(\texttt{STEPHEN},\texttt{MARC})$	$\texttt{Male}(\texttt{JASON})$
$\texttt{hasChild}(\texttt{MARC},\texttt{ANNA})$	$\texttt{Male}(\texttt{JOHN})$
$\texttt{hasChild}(\texttt{JOHN},\texttt{MARIA})$	$\texttt{Female}(\texttt{ANNA})$
$\texttt{hasChild}(\texttt{ANNA},\texttt{JASON})$	$\texttt{Female}(\texttt{MARIA})$
	$\texttt{Female}(\texttt{MICHELLE})$

positiv:$\{\texttt{STEPHEN}, \texttt{MARC}, \texttt{JOHN}\}$

negativ:$\{\texttt{JASON}, \texttt{ANNA}, \texttt{MARIA}, \texttt{MICHELLE}\}$

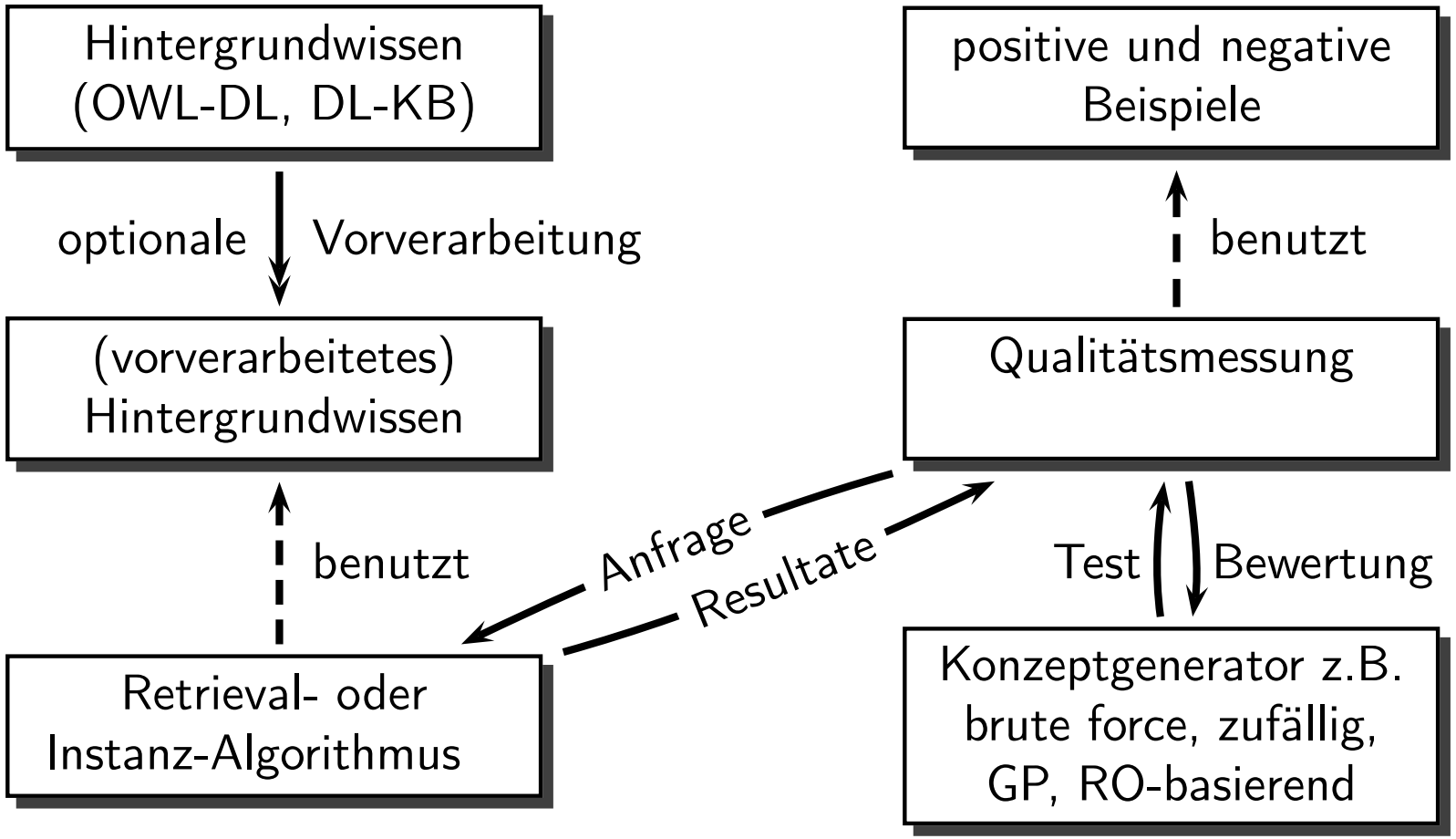
gelerntes Konzept: $\texttt{Male} \sqcap \exists\texttt{hasChild}.\top$

Beispiel 2 - Weinontologie

- es sei eine Ontologie über Weine gegeben
- Bearbeiter der Wissensbasis möchte neues Konzept für Weine mit Chardonnay-Geschmack hinzufügen
- dazu wählt er einige positive Beispiele aus: Mount Adam Chardonnay, Corton Montrachet White Burgundy, Chateau De Merseault, ...
- andere Weine wählt er als negative Beispiele aus: Longridge Merlot, Marietta Zinfandel, Chianti Classico, ...
- Lernsystem schlägt Definition vor, z.B.:

$$\exists \texttt{hasColor.White} \sqcap \exists \texttt{hasBody.}(\texttt{Full} \sqcup \texttt{Medium})$$
$$\sqcap \exists \texttt{hasFlavor.}(\texttt{Strong} \sqcup \texttt{Moderate})$$

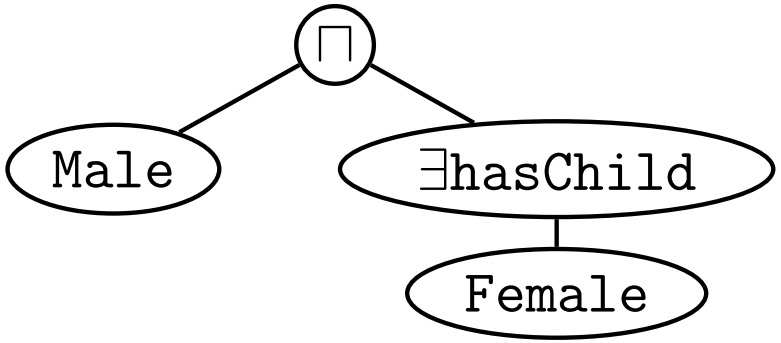
Framework zum Lernen



Genetische Programmierung

Funktionsweise:

- $\mathcal{ALC}$-Konzepte werden als Bäume betrachtet
- es wird zufällig eine Menge von Bäumen erzeugt (Population)
- die Population wird fortlaufend verändert, indem gute Bäume ausgewählt (entspricht natürlicher Auslese nach Darwin) und von genetischen Operatoren verändert werden
- Beispiel für Baumdarstellung von $\texttt{Male} \sqcap \exists \texttt{hasChild}.\texttt{Female}$:



Genetische Programmierung - Eigenschaften

- kann mit verrauschten Beispieldaten umgehen (d.h. Beispiele können fehlerhaft oder widersprüchlich sein)
- besonders geeignet, wenn Näherungslösungen akzeptabel sind
- gesamter Algorithmus ist stochastisch (verschiedene Durchläufe können verschiedene Resultate ergeben)
- benötigt schnelle Algorithmen zur Abschätzung/Berechnung der Qualität einer Lösung
- Erfolg hängt wesentlich von den eingesetzten genetischen Operatoren ab

Refinement-Operatoren

In einem geordneten Raum $(S, \preceq)$ ist ein downward (upward) Refinement-Operator ρ eine Abbildung von S auf 2^S, so dass für jedes $C \in S$ gilt: $C' \in \rho(C)$ impliziert $C' \preceq C$ ($C \preceq C'$). C' ist eine *Spezialisierung* (*Generalisierung*) von C.

- $\mathcal{ALC}$-Refinement-Operatoren kann man definieren, indem man $\mathcal{ALC}$-Konzepte geordnet nach Subsumption betrachtet
- Beispiele für Subsumption zwischen $\mathcal{ALC}$-Konzepten:
  - $\texttt{Male} \sqcap \forall \texttt{hasChild.Female} \sqsubseteq \texttt{Male}$
  - $\texttt{Male} \not\sqsubseteq_{\mathcal{T}} \texttt{Female}$
  - $\texttt{Female} \not\sqsubseteq_{\mathcal{T}} \texttt{Male}$

Refinement-Operatoren

- Warum ist Subsumption eine geeignete Ordnung?
  - Alle Beispiele, die aus einem Konzept folgen, folgen auch aus einem allgemeineren Konzept.
- Wie löst man das Lernproblem mit z.B. einem downward Refinement-Operator?
  - man beginnt die Suche beim allgemeinsten Konzept ($\top$), aus dem alle positiven und negativen Beispiele folgen
  - durch Anwendung des Operators gelangt man zu spezielleren Konzepten
  - dies wird solange durchgeführt bis man ein Konzept findet, aus dem alle positiven, aber keine negativen Beispiele folgen
  - einfaches Suchverfahren (z.B. Breitensuche, Tiefensuche) ist dabei nicht ausreichend; es muss eine geeignete Heuristik eingesetzt werden

Refinement-Operatoren - Eigenschaften

- Refinement-Operatoren und geeignete Heuristiken sind relativ komplex (deshalb wird hier kein konkreter Operator vorgestellt)
- es gibt eine Reihe von Eigenschaften von Refinement-Operatoren, die bestimmen wie gut sie zum Lernen geeignet sind
- in einem Teil der Diplomarbeit wurden bzw. werden diese Eigenschaften untersucht (Fragestellung: Welche Kombinationen von Eigenschaften sind möglich bzw. schließen sich aus?)

Refinement-Operatoren - Suchprozess

Warum ist die Suche schwierig?

- unendlicher Suchraum
- um „sinnvolle“ Operatoren zu definieren ist es günstiger, dass Konzepte auf eine unendliche Menge von anderen Konzepten abgebildet werden (= unendliche Breite des Suchbaums)
- $\mathcal{ALC}$ ist “relativ ausdrucksstark“, da aussagenlogisch abgeschlossen (über Konstruktoren $\neg$, $\sqcap$ und $\sqcup$)

Refinement-Operatoren - charakteristische Eigenschaften

- Refinement-Operatoren mit geeigneten Heuristiken ermöglichen intelligente und effiziente Suche
- deterministischer Algorithmus
- theoretische Garantien im Sinne von „Falls das Lernproblem eine Lösung hat, dann wird diese gefunden.“ sind möglich
- Verwendung von Refinement-Operatoren ist Methode der Wahl für das Lernen in Ontologien (da dort keine verrauschten Daten vorliegen und ein deterministischer Algorithmus benötigt wird)

Ausblick und Ziele

- Theorie zur Lösung des Lernproblems weiterentwickeln (wird während Diplomarbeit gemacht)
- vollständige Implementierung (momentan funktionsfähiger Prototyp für genetische Programmierung vorhanden)
- Einbettung in einen Ontologieeditor
- Erweiterung der erlaubten Konzeptkonstruktoren um weitere Features von OWL:
  - Zahlenrestriktionen ($\geq n\ r.C$, $\leq n\ r.C$, $= n\ r.C$)
  - abgeschlossene Mengen von Objekten in Konzeptbeschreibungen (z.B. $\{\texttt{ITALY}, \texttt{FRANCE}, \texttt{SPAIN}\}$)