Was bisher geschah

Wissensrepräsentation und -verarbeitung durch

- Künstliche Neuronale Netze (insbes. auch CNN)
- Zustandsübergangssysteme
- Klassische Logiken
- Logische Programme (Prolog)
  Beispiele zum Planen

Regeln

Regel: Implikation $r = (\varphi \to \psi)$, meist mit
$\varphi = (b_1 \wedge \ldots \wedge b_n \wedge \neg c_1 \wedge \ldots \wedge \neg c_m)$ und $\psi = h$
mit (aussagen- oder prädikatenlogischen) Atomen
$b_1, \ldots, b_n, c_1, \ldots, c_m, h$

Bestandteile der Regel r:

Kopf h (Folgerung)

Rumpf $b_1 \wedge \ldots \wedge b_n \wedge \neg c_1 \wedge \ldots \wedge \neg c_m$
(Voraussetzungen)
positive Voraussetzungen $b_1, \ldots, b_n$
negative Voraussetzungen $c_1, \ldots, c_m$

Logisches Programm (regelbasiertes System) besteht aus

- Wissensbasis (R, F) mit
  - Regelmenge R
  - Faktenmenge F
- Regelinterpreter, z.B. Prolog-Interpreter

Datalog

Datalog: Anfragesprache für relationale Datenbanken
(Tabellen repräsentieren Relationen, definieren Signatur)

FOL($\Sigma, \mathbb{X}$)-Fragment mit den folgenden Eigenschaften:

Syntax Σ_F enthält nur Konstantensymbole
(nullstellige Funktionssymbole),
keine Funktionssymbole höherer Stelligkeit

Semantik Interpretation über einer festen endlichen
Trägermenge,
meist Menge aller vorkommenden Konstantensymbole
Modelle: Mengen von Grundatomen (Konstanten)

Datalog: Syntax und Semantik

Datalog-Syntax:

Datalog-Term: Konstantensymbol oder Variable

Datalog-Atom: `p( t1, ... , tn )` mit n-stelligem Relationssymbol $p \in \Sigma_R$ und Termen `t1, ... , tn` (Variablen oder Konstanten)

Datalog-Klausel: Regel `h :- b1, ... , bn` mit Atomen `b1, ... , bn, h`
Datalog-Fakten sind Datalog-Klauseln mit $n = 0$.

Datalog-Wissensbasis: endliche Menge von Datalog-Klauseln

Datalog-Anfrage: Formel `?- b1, ... , bn` mit Atomen `b1, ... , bn`

übliche Semantik der klassischen Prädikatenlogik,
aber nur kleinstes Modell (bzgl. $\subseteq$) relevant

Semantik: Erweiterung der Faktenmenge

gegeben: Logisches Programm $P = (F, R)$ mit

- Faktenmenge $F \subseteq \text{Atom}(P)$ (interpretiert als Zustand)
  repräsentiert Menge aller Instanzen der Fakten,
  Menge von Grundatomen (Herbrand-Interpretation)
- Regelmenge R

Folge von Faktenmengen (Zuständen)

$$\begin{aligned} F_0 &= F \\ \forall i \in \mathbb{N} : F_{i+1} &= \{h \mid \exists (B \to h) \in R : F_i \models B\} \\ F^* &= \bigcup_{n \in \mathbb{N}} F_n \end{aligned}$$

datenorientierte Suche

Beispiel

aus der Wissensbasis

F Tom ist ein Baby.

F Tom ist männlich.

R1 Babies sind Kinder.

R2 Männliche Kinder sind Jungen.

R3 Weibliche Kinder sind Mädchen.

folgt (ohne gezielte Anfrage):

Tom ist ein Kind.

Tom ist ein Junge.

Regel feuert (ist anwendbar) in einer Faktenmenge gdw. Voraussetzung (Regelrumpf) erfüllt.

Verfahren: Schrittweise Erweiterung der Faktenmenge um gültige Fakten (Köpfe feuernder Regeln):

$$
\begin{aligned}
F_0 &= \{b(T), m(T)\} \\
F_1 &= \{b(T), m(T), k(T)\} \quad \text{wegen R1} \\
F_2 &= \{b(T), m(T), k(t), j(T)\} (= F_3) \quad \text{wegen R2}
\end{aligned}
$$

Konsequenzoperator für definite Programme

gegeben: definites Programm $P = (R, F)$ (Wissensbasis)
Faktenmenge $M \subseteq \text{Atom}(P)$
Jedes Programm P definiert seinen
Konsequenzoperator $T_P : 2^{\text{Atom}(P)} \to 2^{\text{Atom}(P)}$

$$\begin{aligned} T_P(M) &= \{h \mid b \to h \in P \text{ und } M \models b\} \\ &= \{h \mid (b_1 \wedge \cdots \wedge b_n) \to h \in P \text{ und } \{b_1, \ldots, b_n\} \subseteq M\} \end{aligned}$$

Diese definiert eine Folge von Faktenmengen F_i durch

$$\begin{aligned} F_P^0 &= \emptyset \\ F_P^{i+1} &= T_P(F_P^i) \\ &\vdots \\ F_P^* &= \bigcup_{i \in \mathbb{N}} F_P^i \end{aligned}$$

Fixpunkt-Semantik logischer Programme

Für definite Programme P:

- ist F_P^* der kleinste Fixpunkt des Operators T_P.
- gilt $F_P^* = \bigcap \mathrm{Mod}(P)$
- ist F_P^* das eindeutige kleinste Modell für P.
- Falls $F_P^{n+1} = F_P^n$ gilt, dann ist $F_P^* = F_P^n$.
- Für endliche (grundinstanziierte) Programme P wird $F_P^* = F_P^n$ nach endlich vielen Anwendungen von T_P erreicht.

Fixpunkt-Semantik des logischen Programmes P:

- Ein Atom a folgt genau dann aus P, wenn $a \in F_P^*$.
- Eine Formel φ folgt genau dann aus P, wenn φ in F_P^* gilt.

Schließen in klassischer Logik

Für eine Formelmenge $\Phi \subseteq \mathsf{FOL}(\Sigma, \mathbb{X})$ heißt die Formelmenge

$$C(\Phi) = \{\psi \in \mathsf{FOL}(\Sigma, \mathbb{X}) \mid \Phi \models \psi\}$$

Menge aller Konsequenzen aus Φ.

Formelmenge Φ mit $\Phi = C(\Phi)$ heißt deduktiv abgeschlossen.

In klassischer Logik gilt:
Aus $\Phi \subseteq \Psi$ folgt $C(\Phi) \subseteq C(\Psi)$.
Bei Erweiterung des Wissens bleiben alle Fakten, die vorher schon abgeleitet werden konnten, wahr.
(nur Erweiterung des Wissens, keine Revision)

Hülleneigenschaften

Ein Hüllenoperator ist ein Operator $f : 2^M \to 2^M$ mit den folgenden Eigenschaften (Hülleneigenschaften)

- Für alle Mengen $m, n \in 2^M$ folgt aus $m \subseteq n$, dass $f(m) \subseteq f(n)$ gilt.
  f ist monoton
- Für jede Menge $m \in 2^M$ gilt $m \subseteq f(m)$
  f ist extensiv
- Für jede Menge $m \in 2^M$ gilt $f(f(m)) = f(m)$
  f ist idempotent

In klassischer Logik ist C ein Hüllenoperator.

Unvollkommenes Wissen

einige mögliche Quellen der Unvollkommenheit:

- Aussagen mit unbekanntem Wahrheitswert
- Unvollständige Beschreibung der Situation
- Abstraktion von unwichtig erscheinenden Details
- Falsche Wahrnehmung
- Kein sicheres Wissen über zukünftige Aussagen
- natürlichsprachliche ungenaue Formulierungen

Schließen und Treffen von Entscheidungen oft trotzdem möglich.

Beispiel

Wissen Φ:

- Vögel können fliegen. $(\forall x(V(x) \rightarrow F(x)))$
- Tweety ist ein Vogel. $(V(t))$

Frage: Kann Tweety fliegen? $(F(t))$

zusätzliches Wissen Ψ: Es gibt Vögel, die nicht fliegen können, z.B.

- Pinguine sind Vögel $(\forall x(P(x) \rightarrow V(x)))$
- Pinguine können nicht fliegen $(\forall x(P(x) \rightarrow \neg F(x)))$

Problem: $\Phi \cup \Psi$ inkonsistent (enthält Widerspruch)

Lösungsansatz: „unnormale“ Vögel

- $\forall x(V(x) \wedge \neg U(x) \rightarrow F(x))$
- $\forall x(P(x) \rightarrow U(x))$

neue Information: Tweety ist ein Pinguin $(P(t))$

Negative Voraussetzungen

Problem: Wann gilt $\neg p$ in einer Faktenbasis F?
verschiedene Ansätze:

1. starke Negation: Faktenbasis enthält Literale
   $\neg p$ gilt genau dann, wenn $(\neg p) \in F$
   Vorteil: positive Antwort immer korrekt
   Probleme:
   - erfordert Verwaltung negativer Fakten in Faktenbasis
   - Was gilt, falls weder p noch $\neg p$ in F? (Unbestimmtheit)
   - Was gilt, falls sowohl p als auch $\neg p$ in F? (Inkonsistenz)
2. schwache Negation:
   Nicht aus der Wissensbasis ableitbare Aussagen werden als unwahr angenommen. (Freispruch aus Mangel an Beweisen)
   Vorteil: ergibt immer eine Antwort (zweiwertig)
   Problem: nach Erweiterung der Wissensbasis evtl. ungültig
3. Nutzer fragen
   Vorteil: Antwort führt zu Erweiterung des Wissens
   Nachteil: Was gilt, falls Nutzer keine Antwort gibt?

Closed World Assumption

CWA: Der Anwendungsbereich ist durch die Wissensbasis vollständig beschrieben.

Damit gilt insbesondere

- Jede im Anwendungsbereich gültige Aussage ist aus der Wissensbasis ableitbar.
- Jede nicht aus der Wissensbasis ableitbare Aussage gilt im Anwendungsbereich nicht.
  (also gilt ihre Negation)

entspricht der Idee der schwachen Negation

Regeln mit negativen Bedingungen

Regeln der Form

$$p_1 \wedge \cdots \wedge p_{n_i} \wedge \neg q_1 \wedge \cdots \wedge q_{m_i} \rightarrow h$$

mit

- positiven Bedingungen $p_1, \ldots, p_{n_i}$
- negativen Bedingungen $q_1, \ldots, q_{m_i}$

ist in der Faktenmenge F genau dann anwendbar, wenn

$$F \models (p_1 \wedge \cdots \wedge p_{n_i} \wedge \neg q_1 \wedge \cdots \wedge q_{m_i})$$

also

- $\{p_1, \ldots, p_{n_i}\} \subseteq F$ und
- $\{q_1, \ldots, q_{m_i}\} \cap F = \emptyset$

Vorwärtsverkettung auch möglich für Wissensbasen mit Regeln mit (schwacher) Negation

Nichtmonotones Schließen

Syntax: Wissensbasen mit negierten Atomen in Kopf und Rumpf

Problem beim Schließen mit Regeln mit negativen Bedingungen:

- Als falsch angenommene Voraussetzungen können sich später als wahr herausstellen.
- Voraussetzungen früher angewendeter Regeln gelten damit evtl. nicht mehr.

Ansätze zum Umgang mit unvollständigem Wissen

verschiedene Ansätze zur Definition einer intuitiven Semantik für Regelmengen mit negativen Voraussetzungen, z.B.:

- Stabile Modelle, Answer-Sets:
  Idee: Programm hat mehrere mögliche Modelle
  Aussage folgt aus Wissensbasis, wenn sie in einem / ausgewählten / allen Modellen wahr ist.
- Wohlfundierte Modelle
  Idee: Programm hat ein Modell mit drei Wahrheitswerten (wahr, falsch, unbekannt)
  Aussage folgt aus Wissensbasis, wenn sie in diesem Modell wahr ist.
- Truth-Maintenance-Systeme:
  Protokollierung aller zum Ableiten einer Formel verwendeten Voraussetzungen
  Bei späterer Feststellung der Unwahrheit einer Voraussetzung, alle daraus gezogenen Schlüsse löschen (rekursiv).

Normale logische Programme

(negative Voraussetzungen erlaubt)

(erweitertes) logisches Programm P (Wissensbasis) enthält:

- Menge R von Regeln der Form

$$p_1 \wedge \cdots \wedge p_{n_i} \wedge \neg q_1 \wedge \cdots \wedge q_{m_i} \rightarrow h$$

  mit Atomen p_i, q_i, h

spezielle Regeln:

- Regeln mit leerem Rumpf: h (Fakten)
- Regeln mit leerem Kopf: $p_1 \wedge \cdots \wedge p_{n_i} \wedge \neg q_1 \wedge \cdots \wedge q_{m_i}$ (Constraints)
  Abkürzung für $p_1 \wedge \cdots \wedge p_{n_i} \wedge \neg q_1 \wedge \cdots \wedge q_{m_i} \wedge \neg r \rightarrow r$ (unerfüllbar)

Beispiel: $\{p \rightarrow q, \neg q \rightarrow r, \neg r \rightarrow q, p\}$

Modelle normaler logischer Programme

Idee: P als Formelmenge

Herbrand-Interpretation eines normalen logischen Programmes P:
Menge I von Grundatomen (mit derselben Signatur wie P)
betrachtet als Aussagevariablen
Belegung der Aussagevariablen ist charakteristische Funktion von I

Beispiel: $P = \{p \rightarrow q, \neg q \rightarrow r, \neg r \rightarrow q, p\}$, $I = \{p, r\}$

Herbrand-Modell eines normalen logischen Programmes P:
Herbrand-Interpretation I mit $I \in \mathrm{Mod}(P)$
(Belegung = charakteristische Funktion)

Beispiel: $P = \{p \rightarrow q, \neg q \rightarrow r, \neg r \rightarrow q, p\}$

- $\{p, q, r\}, \{p, q\}$ sind Modelle für P
- $\{q, r\}, \emptyset$ sind keine Modelle für P

Auswahl intuitiver Modelle

Eigenschaften von Interpretationen I eines logischen Programmes P:

abgeschlossen unter P:
für jede Regelinstanz $B \rightarrow h$ aus P gilt:
falls $I \models B$, dann $h \in I$

begründet für jedes $p \in I$ existiert eine Ableitung (Begründung) für p in I

Eigenschaften von Modellen I eines logischen Programmes P:

- minimal
  (falls $J \subseteq I$ und $J \in \mathrm{Mod}(P)$, dann gilt $J = I$)

Intuitive Modelle: Modelle für P, die begründet und unter P abgeschlossen sind.

Gelfond-Lifschitz-Transformation

gegeben:

- normal logisches Programm P
- Modell I für P

Programmtransformation:

$$P^I = \left\{ p_1 \wedge \ldots \wedge p_m \rightarrow h \mid \begin{array}{l} p_1 \wedge \ldots \wedge p_m \wedge \neg q_1 \wedge \ldots \wedge \neg q_n \rightarrow h \\ \text{und} \{q_1, \ldots, q_n\} \cap I = \emptyset \end{array} \right\}$$

1. Alle Regeln mit negativen Bedingungen $\neg q_i$ mit $q_i \in I$ entfernen.
2. Alle negativen Bedingungen aus allen verbleibenden Regeln entfernen.

Für jedes normale logische Programm P und jede Interpretation I ist das I-Redukt P_I ein Programm ohne negative Bedingungen.

Der Konsequenzoperator T_{P_I} ist also monoton.

Stabile Modelle normaler logischer Programme

Idee: Auswahl einer Menge von intuitiven Modellen für normale logische Programme

Modell I für P heißt stabiles Modell, falls

$$I = T^*_{P^I}$$

Beispiel:

- $P = \{p \to q, \neg q \to r, \neg r \to q, p\}$
  - $\{p, q\}$ ist stabiles Modell für P, weil $P^{\{p,q\}} = \{p \to q, q, p\}$ und $T^*_{P^{\{p,q\}}} = \{p, q\}$
  - $\{p, q, r\}$ ist kein stabiles Modell für P, weil $P^{\{p,q,r\}} = \{p \to q, p\}$ und $T^*_{P^{\{p,q,r\}}} = \{p, q\}$
- $P' = \{\neg p \to q, \neg q \to p\}$
- $P'' = \{\neg p \to q, p \to q, \neg q \to p\}$
- $P''' = \{\neg p \to p\}$

Beispiel: gefärbte Graphen

Faktenbasis (Beschreibung des speziellen Problemes):

- Knotenmenge $V = \{v_1, \ldots, v_n\}$
  `ecke(v1), ..., ecke(vn)`
- Kantenmenge $E = \{(v_i, v_j), \ldots\}$
  `kante(vi, vj), . . .`
- Menge $C = \{r, g, b\}$ von Farben

Erzeugung der Kandidaten (jede Ecke genau eine Farbe):

```
farbe(X, r) :- ecke(X), not farbe(X, b), not farbe(X, g)
farbe(X, b) :- ecke(X), not farbe(X, r), not farbe(X, g)
farbe(X, g) :- ecke(X), not farbe(X, r), not farbe(X, b)
```

Bedingung für korrekte Färbung (Ausschlusskriterium):

```
:-  kante(X, Y ), farbe(X, Z), farbe(Y, Z)
```

Stabile Modelle repräsentieren Lösungen (korrekte Färbungen)

Erweiterte logische Programme

(sowohl starke $\overline{\ }$ als auch schwache Negation $\neg$ erlaubt)

Idee: p und $\overline{p}$ als unabhängige Atome betrachten
Konsistent durch Constraints garantieren (z.B. $p \wedge \overline{p} \rightarrow \mathbf{f}$

(erweitertes) logisches Programm P (Wissensbasis) enthält:

- Menge R von Regeln der Form

$$p_1 \wedge \cdots \wedge p_{n_i} \wedge \neg q_1 \wedge \cdots \wedge q_{m_i} \rightarrow h$$

  mit „Atomen“ p_i, q_i, h

spezielle Regeln:

- Regeln mit leerem Rumpf: h (Fakten)
- Regeln mit leerem Kopf: $p_1 \wedge \cdots \wedge p_{n_i} \wedge \neg q_1 \wedge \cdots \wedge q_{m_i}$ (Constraints)

Beispiel: $\{p \rightarrow \overline{q}, \neg\overline{q} \rightarrow r, \neg r \rightarrow \overline{q}, \overline{p}\}$

Answer Sets

Answer-Sets:
ausgewählte Modelle erweiterter logischer Programme

Eigenschaften:

- abgeschlossen unter P oder
  für ein Atom p gilt $\{p, \overline{p}\} \subseteq I$ (inkonsistent)
- begründet in P

Interpretation eines erweiterten logischen Programmes P:
Menge I von Grundliteralen (mit derselben Signatur wie P)

I ist Answer-Set für P gdw. Modell des I-Reduktes P^I ist (analog stabilen Modellen)

Beispiel: Terminplanung

Faktenbasis (Beschreibung des speziellen Problemes):

```
termin(m1), . . . , termin(mn)
zeit(t1), . . . , zeit(ts), raum(r1), . . . , raum(rm)
person(p1), . . . , person(pk)
mit(p1,m1), . . . , mit(p2,m3), . . .
```

Zuordnung von Zeiten und Räumen zu Terminen:

```
um(M, T)  :- termin(M), zeit(T), not um'(M, T)
um'(M, T) :-  termin(M), zeit(T), not um(M, T)
in(M,R)   :- termin(M), raum(R), not in'(M,R)
in'(M,R)  :- termin(M), raum(R), not in(M,R)
zeitvergeben(M) :-  um(M, T)
raumvergeben(M) :-  in(M,R)
```

Bedingungen:

```
:-  termin(M), not zeitvergeben(M)
:-  termin(M), not raumvergeben(M)
:-  termin(M), um(M, T), um(M, T'), T <> T'
:-  termin(M), in(M,R), in(M,R'), R <> R'
:-  in(M,X), in(M',X), um(M, T), um(M', T),M <> M'
:-  mit(P,M), mit(P,M'), M <> M', um(M, T), um(M', T)
```

Reading Group

- Esra Erdem, Volkan Patoglu, 2018:
  Applications of ASP in Robotics
- Tran Cao, Marcello Balduccini, 2018:
  Answer Set Planning in Single- and Multi-agent Environments
- Abels, D., Jordi, J., Ostrowski, M., Schaub, T., Toletti, A., Wanko, P. (2019):
  Train Scheduling with Hybrid ASP

  `https://www.cs.uni-potsdam.de/wv/publications/DBLP_conf/lpnmr/AbelsJOSTW19.pdf`