Künstliche Intelligenz I

Wintersemester 2008/09

Inhaltsverzeichnis

Teil I.
Einführung

1. Was ist Künstliche Intelligenz

2. Chancen und Grenzen der Künstlichen Intelligenz

Teil II.
Suche

3. Grundlagen und Beispiele zu Suche

Zustand: Modell + *hidden context*

Zustandsüberführung, -raum, -graph

Informationsqualität: Vollständigkeit (des Zustandsgraphen), Verlässlichkeit (Rauschen)

Expansion eines Knotens

4. Uninformierte Suche

Impliziter Suchraum (z.B. Karte)

Uninformierte Suche

Zyklen → *closed list* mit besuchten Knoten

Tiefensuche

Parameter für Aufwandsschätzung:

- Kosten für Auswahl eines Knotens C_L
- Kosten für Backtracking C_B
- Kosten für Wahl des richtigen Knotens C_R
- Wahrscheinlichkeit richtig/falsch

$$\alpha(\underbrace{H}_{\text{Höhe}}) = p \cdot (C_L + \alpha(H-1)) + q \cdot (C_L + \underbrace{\max(H-1)}_{\substack{\text{falscher Teilbaum:} \\ \text{komplette Suche}}} + C_B + C_R + \alpha(H-1))$$

Tiefensuche in vollständigen Graphen:
worst case: $\sum_{i=1}^{N} i \in \mathcal{O}(N^2)$

$\rightsquigarrow$ Erwartungswert: wie viele Knoten müssen expandiert werden?
$\Rightarrow n \cdot H_n - n$
H_n: harm. Reihe $\in \mathcal{O}(n \cdot \ln n), \quad n = 2^H - 2$

Breitensuche

Sarantos Kapidakis: Average-Case Analysis of Graph-Searching Algorithms
`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.42.9246&rep=rep1&type=pdf`

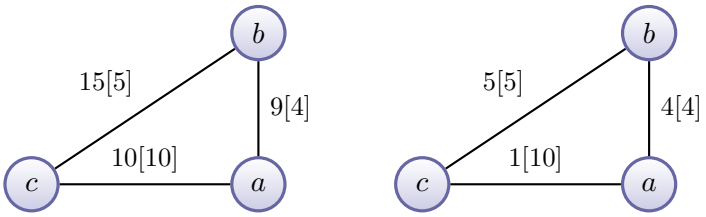
5. Heuristische Suche

Heuristik: $\hat{h} : \mathbb{R}^N \rightarrow \mathbb{R}$

A*

```
while (not len(open) == 0):
    min = open[0]
    for x in open:
      if ((g[x] + graph.heur[x]) < (g[min] + graph.heur[min])):
        min = x

    open.remove(min)
    if (min == target):
      return ways[target]

    closed.append(min)

    for x in graph.successors[min]:
      if ((not x in open) and (not x in closed)):
        open.append(x)
        ways[x] = ways[min][:]
        ways[x].append(x)
        g[x] = g[min] + graph.costs[(min, x)]

      elif (g[min] + graph.costs[(min, x)] < g[x]):
        g[x] = g[min] + graph.costs[(min, x)]
        ways[x] = ways[min][:]
        ways[x].append(x)

        if (x in closed):
          open.append(x)
          closed.remove(x)
```

Optimistische Heuristik: Restkostenschätzung $\leq$ tatsächliche Kosten
$\rightarrow$ Falls so eine schlechtere Lösung gefunden wird, ist die bessere Möglichkeit garantiert mit einer besseren Schätzung in der open-Liste!
(Beweis: 5.14)



Gesuchter Weg: $a \rightarrow c$, Kanten: *geschätzt[tatsächlich]*

Links: Nicht optimistisch: kürzester abc, gewählter ac
Rechts: Optimistisch: Es wird zwar zuerst ac ausgewählt, das aber noch korrigiert.

Zulässiger Suchgraph:

- Nachfolger k: $0 < k < \infty$
- Kosten C: $0 < \varepsilon < C$
- Optimistische Heuristik: $\hat{h}(n) \leq h(n)$

„Backtracking“ bei A*? Oft vermeidbar durch *monotone* Heuristik:
$\forall$ Kanten$(x, y) : \hat{h}(x) - \hat{h}(y) \leq c(x, y)$
$\rightarrow$ Restkostenschätzung darf nicht schneller fallen als die tatsächlichen Kosten

Bellmansches Optimalitätsprinzip: „Jeder Teilpfad der Lösung ist optimal“
$\hat{f}(x) = \min_{(x,y)\in G}(c(x, y) + \beta \cdot \hat{f}(y)$

6. Optimierte heuristische Suche und heuristische Suche nach dem Optimum

IDA*

BFS: Speicherbedarf wächst exponentiell mit Tiefe des Ziels!

IDA*: DFS mit heuristisch beschränkter Suchtiefe

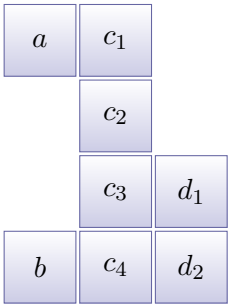
- Initialisieren: $f_{max} = \hat{f}(n_0) = \hat{h}(n_0)$ (optimistisch!)
- Solange keine Lösung gefunden wurde: $f_{max} = \min_{n_i} \hat{f}(n_i)$
  für in diesem Durchgang besuchte n_i

Clustern von Knoten

Probleme: Mehrdeutigkeit; Gebäude mit mehreren Eingängen = 1 Knoten

Kosten zwischen Clustern:

- Minimum-Abstand:
  $\overline{AC} = \overline{BC} = \overline{CD} = 1 \Rightarrow \overline{AD} = \overline{BD}$?
- Maximin-Abstand:
  max{min{Kosten für alle Eintrittspunkte}} (A*)
  Um $\overline{CD}$ herauszufinden, betrachtet man alle Möglichkeiten, wo man hergekommen sein könnte, bestimmt für jede den kürzesten Weg und nimmt den schlimmsten Fall an.
  ($\overline{CD} = \overline{c_1, c_2, c_3, d_1} = 3$)
- Average-Minimum-Abstand: Analog zu Maximin, es wird am Ende aber nicht das Maximum sondern der Durchschnitt gewählt.

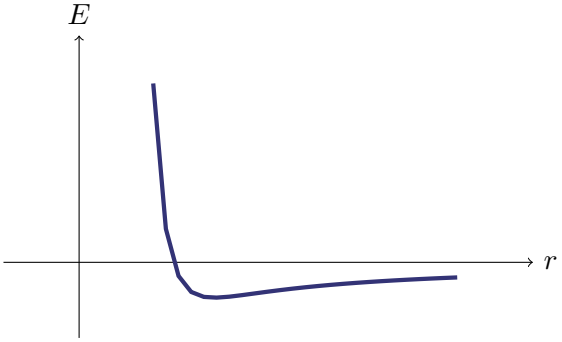


Teil III.
Suche mit unvollständiger Information

7. Adversiale Suche mit unvollständiger Information

Chasing and Evading

- Coulomb-Kräfte: $F = \frac{1}{4\pi\varepsilon_0}\frac{q_1 q_2}{r^2}$
- Lennard-Jones-Potential: $E = -\frac{A}{r^n} + \frac{B}{r^m}$



Minimax-Algorithmus

Voraussetzung: Abwechselnde Züge, Nullsummenspiel

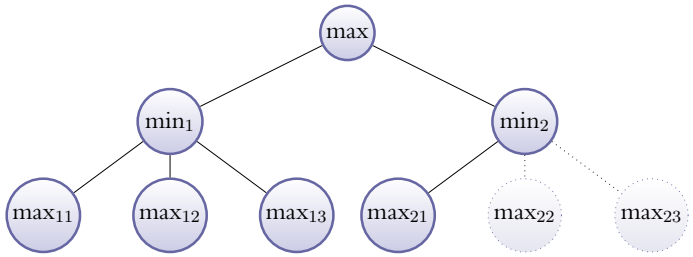
Die Maus versucht in ihrem Zug ihren Gewinn zu maximieren unter der Annahme dass die Katze das gleiche tut ($\rightsquigarrow$ minimal für Maus).

max-Entscheidung setzt bekannte min-Entscheidung voraus etc. $\Rightarrow$ Rekursion

Zufall? $\Rightarrow$ Einen Zufallsknoten für jede Möglichkeit.

Alpha-Beta-Suche

Unter Umständen kann man bestimmte Teilbäume ignorieren: Ist bei der max-Suche eine Ebene tiefer ein min-Kandidat gefunden worden, der kleiner gleich einem bereits feststehenden Minimum ist, braucht der restliche Teilbaum nicht untersucht werden, da das Maximum von einem anderen Teilbaum bestimmt werden wird.



$$\max_{21} < \min_1 \quad \Rightarrow \quad \min_2 < \min_1 \quad \Rightarrow \quad \max\{\min_1, \min_2\} \neq \min_2$$

8. Aktionsplanung bei unsicheren Sensorwerten

Probleme von A* bei Nichtdeterminismus:

- Zustand und Zielzustände müssen ohne Unsicherheit bekannt sein.
- Alle Aktionen müssen deterministisch sein.
- Der komplette Suchraum muss bekannt sein.
- Die Planung müsste ohne Messungen von Sensoren möglich sein.

$\Rightarrow$ Iteration Pfadsuche – Messungen,
Neuplanung bei Abweichungen

Mathematisches Modell eines Roboters

- Pose (x, y, Θ)
- Überführungsfunktion $\delta(x_1, u) = x_2$

Bewegungsspur: $(x_\tau, y_\tau, \Theta_\tau)_{0 \leq \tau \leq t}$
Steueranweisung: $u_t = (v_t, \omega_t)$

Aktionsplanung bei Unsicherheit

(Thrun, 14; Russell, Norvig)

(siehe Kapitel 13)

9. Lokale Optimierung: Positionsbestimmung mithilfe verrauschter Sensorwerte

Formel-Wiederholung

Totale Wahrscheinlichkeit: $p(x) = \int p(x|y) \cdot p(y)\ dy$
Bedingte Wahrscheinlichkeit: $p(x|y) = p(x, y)/p(y)$
Bayes-Formel: $p(x|y) = p(y|x) \cdot p(x)\ /\ p(y)$
1D-Gauss: $\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}(\frac{x-\mu}{\sigma})^2\right)$

Feste Landmarken Θ_i, Karte $\Theta = \{\Theta_1, \ldots, \Theta_N\}$

Annahmen:

- Eine Aktion u_t pro Zeitpunkt
- Eine Beobachtung z_t pro Zeitpunkt
- s_t abhängig nur von s_{t-1}, u_t (Bewegungsmodell)
- Jede Beobachtung liefert Informationen über genau eine Landmarke
- Beobachtung $z_t = f(\Theta_{n_t}, s_t)$ (Messmodell)

Positionsschätzung:

$$p(s_t, \Theta | z^t, u^t, n^t)$$
$$\hat{s_t} = \underset{s}{\operatorname{argmax}}\, p(s_t = s, \Theta | z^t, u^t, n^t)$$

Warum $p(\Theta)$?

Bayes-Filter

Vorhersage des nächsten Zustandes aus dem Bewegungsmodell:

$$\overline{\text{bel}}(x_t) = \int p(x_t|u_t, x_{t-1}) \cdot \text{bel}(x_{t-1})\ dx_{t-1}$$

Korrektur durch Einbeziehung der Messung:

$$\text{bel}(x_t) = \eta \cdot p(z_t|x_t) \cdot \overline{\text{bel}}(x_t)$$

η: Normalisierungsfaktor, damit $\int \text{bel}(x_t)dx_t = 1$ wird ($\rightarrow$ Wahrscheinlichkeit)

Alternativ:

$$p(s_t, \Theta|z^t, u^t, n^t) = \eta \cdot p(z_t|s_t, \Theta, n_t) \cdot \int p(s_t|s_{t-1}, u_t) \cdot p(s_{t-1}, \Theta|z^{t-1}, t^{t-1}, n^{t-1})\ ds_{t_1}$$

Beispiel: Thrun, S. 29

Markov-Annahme

Partikelfilter

Repräsentation der zu komplexen Verteilungen durch Partikel.
Update der s_{t-1}-Partikel: $p(s_t|s_{t-1}, u_t)$ ($\rightsquigarrow$ Bewegungsmodell)
Integration der Messung durch Resampling: Alle Partikel in einen Topf, mit Zurücklegen und $p(z_t|s_t)$ als Gewicht neu ziehen.

10. Lernen von Karten

Kalman-Filter

Durch das Integral ist der Bayes-Filter ist für praktische Zwecke kaum geeignet ⇒ Kalman!

Voraussetzungen:

- state transition probability / measurement probability müssen linear sein

$$\begin{array}{ll} p(s_t|u_t, s_{t-1}) & p(z_t|s_t) \\ s_t = a_t s_{t-1} + b_t u_t + \varepsilon_t & z_t = c_t s_t + \delta_t \end{array}$$

  ε_t: Rauschen $\in \mathcal{N}(0, \rho^2)$
  δ_t: Rauschen $\in \mathcal{N}(0, \nu^2)$
- Modellierung von bel(s_t) durch Normalverteilung $\mathcal{N}(\mu_t, \sigma_t^2)$

Algorithmus:

$$\tilde{\mu}_t = a_t \mu_{t-1} + b_t u_t$$
$$\tilde{\sigma_t}^2 = a_t^2 \sigma_{t-1}^2 + \rho_t^2$$

$$k_t = \frac{\tilde{\sigma_t}^2 c_t}{c_t^2 \tilde{\sigma_t}^2 + \nu_t^2}$$

$$\mu_t = (1 - k_t c_t)\tilde{\mu}_t + k_t z_t$$
$$\sigma_t^2 = (1 - k_t c_t)\tilde{\sigma_t}^2$$

Kalman-Gain k_t: „how much we will trust the observed landmarks“

Herleitung k_t: min $E[|s_t - \hat{s}_t|^2]$ (Schätzfehler) minimieren ($\frac{\partial \mathrm{tr}(\Sigma_t)}{\partial k_t} = 0$)

Fast-SLAM

- Schätze eine neue repräsentative Stichprobe für die Aufenthaltswahrscheinlichkeit (Partikelfilter)
- Aktualisiere die Kalman-Filter für die bekannten Landmarken
- Berechne die Gewichtung für jedes Element der Stichprobe
- Konstruiere (durch simuliertes Ziehen mit Zurücklegen) eine neue gleichverteilte Stichprobe (eine Hypothese darf aber von mehr als einem Element repräsentiert werden)

Teil IV.
Suche in abstrakten Räumen

11. Suche in abstrakten Räumen: Strukturierung durch Suchgraphen

Interessant: Vorbedingungen und Effekte von Aktionen
Kosten- und Nutzenfunktionen?

Erwartungen:

- Skalierbarkeit (neue Aktionen?)
- Diagnosefähigkeit (warum?)
- Erklärungsfähigkeit (was muss ich tun, damit ...?)

Frames

Stereotype Situation ⇒ Graph: *subeventof*, *capableof*, *do*

Scripts

Abstrakte Erzählkontexte, einige atomare Handlungen (z. B. *PTRANS*)

S asks for menu:

```
S MTRANS signal to W
W PTRANS W to table
S MTRANS 'need menu' to W
W PTRANS W to menu

W PTRANS W to table
W ATRANS menu to S
```

(NKRL)

Adäquatheitskriterien

1. Ontologische Adäquatheit: alle interessanten Aspekte darstellbar?
2. Erkenntnistheoretische Adäquatheit: lassen sich von Menschen benutzte Wissensstrukturen wiederfinden?
3. Heuristische Adäquatheit: sind Strukturen effizient bearbeitbar von einem Problemlöser
4. Beherrschbare Wissensaquisitionssemantik

Modellierung von Emotionen

Beispiel Robertino:
Fear, *Anger*, *Surprise*, *Anticipation* haben Auswirkungen auf *Precision*, *Power*, *Delay*, *Accuracy*

Endliche Automaten

Nachteile von endlichen Automaten für die NPC-Modellierung:

- Modellierung komplex
- Zur Laufzeit nicht veränderbar
- Verhalten nur von Eingabe und aktuellem Zustand abhängig
- $\Rightarrow$ evtl. vorhersagbar

12. Planung von Handlungen mit beschränktem Horizont

Goal-Oriented Behavior

Ziele	**Prioritäten**	**Handlungen**	**Effekte**	**Ressourcen**
eat	4	get-snack	eat-3/sleep+2	
sleep	3	sleep-on-sofa	eat+2/sleep-2	

Auswahl?

a) Höchste Priorität
b) Gewichtete Summe:

$$\text{aktion} = \underset{a}{\text{argmax}}\, \text{wert}(a) = \underset{a}{\text{argmax}} \sum_{x \in \text{Ziele}} \text{pri}(x) \cdot \text{effekt}(x)$$

c) Stress minimieren:

$$\text{aktion} = \underset{a}{\operatorname{argmin}}\, \text{stress}(a) = \underset{a}{\operatorname{argmin}} \left(\sum_{x \in \text{Ziele}} \text{pri}(x) - \text{effekt}(x, a) \right)^2$$

Problem: Sackgassen bei beschränkten Ressourcen
(Beispiel: Magier kann sich vollständig heilen oder den Oger töten)

Tiefensuche mit Planungshorizont? Nicht optimistisch / monoton...

Dominanz von Prioritäten: Eine Aktion dominiert eine andere, wenn sie in allen Zielen einen besseren Wert hat.

Maximale Elemente: Max. El. werden nicht dominiert (andere Elemente, die in einer Koordinate besser sind, müssen dafür in einer anderen schlechter sein)
$\Rightarrow$ Sortieren und überprüfen

Algorithmus:

- Nimm die k-te Dimension und sortiere die Punkte aufsteigend nach ihren Werten in dieser Dimension.
- Bilde zwei Teilmengen aus den Punkten, so dass die Punkte der einen Menge alle Punkte der anderen Menge in der k-ten Koordinate dominieren.
- Projiziere beide Teilmengen auf die Dimension $k-1$ und finde alle Maxima in $k-1$ Dimensionen.
- Eliminiere alle Punkte aus der Teilmenge, die in $k-1$ Koordinaten von den Punkten der anderen Menge dominiert werden.

13. Planung von Handlungen bei unvollständiger Information

(Thrun, Kapitel 14 und 15)

MDP und POMDP

x_t	Zustände
u_t	Aktionen
$p(x_{t+1}\|x_t, u_t)$	Effekte der Aktionen
$r(x_t, u_t)$	(Immediate-)Reward-Funktion (z.B. $+100$ falls u_t zum Ziel führt, -1 sonst)
z_t	Beobachtung (nach Aktion u_t) [POMDP]
$p(z_t\|x_i)$	Wahrscheinlichkeit im Zustand x_i z_t zu beobachten
π	Control Policy MDP: $u_t = \pi(x_t)$ POMDP: $u_t = \pi(z_{1:t-1}, u_{1:t-1})$
T	Planungshorizont
$\gamma \in [0; 1]$	Discount Factor (interessant v.a. für $T = \infty$)
V_T	Cumulative Payoff
R_T	Expected Cumulative Payoff:

$$R_T = \mathcal{E}\left[\sum_{\tau=1}^{T} \gamma^\tau r_{t+\tau}\right]$$

$$R_T(x_t) = \mathcal{E}\left[\sum_{\tau=1}^{T} \gamma^\tau r_{t+\tau} | x_t\right]$$

$$R_T^\pi(x_t) = \mathcal{E}\left[\sum_{\tau=1}^{T} \gamma^\tau r_{t+\tau} | (u_t = \pi(z_{1:t+\tau-1}, u_{1:t+\tau-1})\right]$$

MDP Markov Decision Process

Der Zustand x_t ist ohne Unsicherheit feststellbar, die Policy also nur abhängig von x_t: $\pi(x_t) = u_t$

POMDP Partially Observable Markov Decision Process

Der Zustand x_t ist nicht direkt beobachtbar, stattdessen arbeitet man mit dem *Belief* $b_t = (p(x_1), \ldots, p(x_n))$ und Beobachtungen z_t ($\rightsquigarrow p(z_t|x_i)$) $\Rightarrow$ $u_t = \pi(z_{1:t-1}, u_{1:t-1})$

Damit kann die Reward-Funktion nicht mehr wie bei MDP ausgewertet werden:
Ersatz: $r(b_t, u_t) = \mathcal{E}_{x_t}[r(x_t, u_t)]$

Für zwei Zustände: $r(b_t, u_t) = b_{t,p(x_1)} r(x_1, u_t) + b_{t,p(x_2)} r(x_2, u_t)$

Value Iteration

VI für MDPs

Rekursion über Planungshorizont T:

- $T = 1$:

$$V_1(x_t) = \gamma \cdot \max_{u_t} r(x_t, u_t)$$
$$\pi_1(x_t) = \operatorname*{argmax}_{u_t} r(x_t, u_t)$$

- $T > 1$:

$$V_T(x_t) = \gamma \cdot \max_{u_t} \left(r(x_t, u_t) + \int V_{T-1}(x_{t+1}) p(x_{t+1}|x_t, u_t) dx_{t+1} \right)$$
$$\pi_T(x_t) = \operatorname*{argmax}_{u_t} \left(r(x_t, u_t) + \int V_{T-1}(x_{t+1}) p(x_{t+1}|x_t, u_t) dx_{t+1} \right)$$

$\rightsquigarrow$ Dynamische Programmierung!
Für $T = \infty$: Bellman-Gleichgewicht!

Aus einem optimalen V_T erhält man unmittelbar (greedy argmax) eine optimale Policy!

VI für POMDPs

Statt mit $V_T(x_t)$ arbeiten wir hier mit $V_T(b_t)$ (stückweise linear).

Außerdem müssen Sensorwerte einbezogen werden: wird die Beobachtung z_i gemacht, hat das Auswirkungen auf b_t nach dem Sensormodell $p(z_t|x_i)$:

$$p'(x_i) = p(x_i|z_t) = \frac{p(z_t|x_i) \cdot p(x_i)}{p(z_t)}$$

(TODO)

Summary (Thrun):

- POMDPs are characterized by multiple types of uncertainty: Uncertainty in the control effects, uncertainty in perception, and uncertainty with regards to the environment dynamics. However, POMDPs assume that we are given a probabilistic model of action and perception.
- The value function in POMDPs is defined over the space of all beliefs robots might have about the state of the world. For worlds with N states, this belief is defined over

the $(N-1)$-dimensional belief simplex, characterized by the probability assigned to each of the N states.

- For finite horizons, the value function is piecewise linear in the belief space parameters. It is also continous and convex. Thus, it can be represented as a maximum of a collection of finitely many linear functions. Further, these linear constraints are easily calculated.
- The POMDP planning algorithm computes a sequence of value functions, for increasing planning horizons. Each such calculation is recursive: Given the optimal value function at horizon $T-1$, the algorithm proceeds to computing the optimal value function at horizon T.
- Each recursive iteration combines a number of elements. The action choice is implemented by maximizing over sets of linear constraints, where each action carries its own set.
  The anticipated measurement is incorporated by combining sets of linear constraints, one for each measurement.
  The prediction is then implemented by linearly manipulating the set of linear constraints.
  Payoff is generalized into the belief space by calculating its expectation, which once again is linear in the belief space parameters. The result is a value backup routine that manipulates linear constraints.
- We find that the basic update produces intractably many linear constrains. Specifically, in each individual backup the measurement step increases the number of constraints by a factor that is exponential in the number of possible measurements. Most of these constraints are usually passive, and omitting them does not change the value function at all.

14. Lernen von optimalen Strategien

Unüberwachtes Lernen: durch Ausprobieren und Bewerten durch die Reward-Funktion (← Domänen-Wissen!) wird die Bewertungsfunktion nach und nach gelernt.

(Beispiel: Katz-und-Maus-Spiel mit verschiedenen Reward-Funktionen)

Probleme: Zustandsraum, Anzahl der erlaubten Aktionen und die Zahl der Variablen im Lösungsraum sind häufig zu groß.

⇒ Punktuelle Aufgabe der partiellen Beobachtbarkeit: nach jeder Steueranweisung ist die Umgebung vollständig beobachtbar → MDP.

⇒ Approximative Repräsentation des Suchraums (z.B. repräsentative Stichprobe → Partikelfilter, Monte Carlo)

Monte-Carlo-Approximation

(Thrun, 16.4)

Bewertungsfunktion: $V_T : \chi \to \mathbb{R}$
We need a representation for V that can be updated using some particle set, but then provides a value for other particle sets, which the MC-POMDP algorithm never saw before. In other words, we need a learning algorithm. MC-POMDP use a *nearest neighbor* algorithm using locally weighted interpolation when interpolating between different beliefs.

$$V_T(\xi) = \frac{\sum_{1 \le k \le K} \frac{1}{d_k} V_T(\chi_k)}{\sum_{1 \le k \le K} \frac{1}{d_k}}$$

(χ_k ist der k-te Nachbar von ξ im Abstand d_k)

- Stichprobe χ der Länge M aus dem aktuellen *Belief* ziehen
- Aktualisierung der Bewertungsfunktion für einen Partikel:
  Für jede Aktion zufällig verschiedene Folgezustände und Messungen ziehen und V entsprechend aktualisieren.
  $\Rightarrow$ „oft“ wiederholen.
- Simulation der besten Aktion:
  Einen zulässigen Folgezustand und eine Messung dafür ziehen, mit dem Partikelfilter eine neue Annahme erzeugen.
  Bewertungsfunktion aktualisieren und „sehr oft“ wiederholen.
- Das Ganze noch einmal mit einer anderen Partikelmenge für die Startannahme versuchen.

Teil V.
Knowledge Representation and Reasoning

15. Wissensrepräsentation: Einführung

16. Wissensrepräsentation – Resolution

Pränexform: Quantoren außen

Skolemform: $\exists$ ersetzen durch Funktion: $\forall x \exists y : p(x, y) \Rightarrow \forall x : p(x, f(x))$

Eigenschaften der Skolemform:
$\exists \Phi : \Phi \models F \Rightarrow \exists \Phi : \Phi \models \text{Skolem}(F)$
$\neg \forall \Phi : \Phi \models F \Rightarrow \Phi \models \text{Skolem}(F)$
(erfüllbarkeitsäquivalent)

Satz von Skolem:
Formel in Pränexform erfüllbar $\Leftrightarrow$ erfüllbar in Skolemform

Herbrand-Universum zu Φ (Φ in Skolemform):

- alle Konstanten in Φ sind im HU (keine Konstanten $\rightarrow$ ein neues Symbol)
- f n-stelliges Funktionssymbol, $t_1, \ldots, t_n$ Terme $\Rightarrow f(t_1, \ldots, t_n)$ im HU

$\Rightarrow$ alle Terme, die in allquantifizierte Variablen eingesetzt werden können!

Herbrand-Expansion:
abzählbar unendliche Formelmenge statt Quantoren

Endlichkeits-/Kompaktheitssatz:
erfüllbar $\Leftrightarrow$ jede endliche Teilmenge erfüllbar

17. Repräsentation von Handlungen in Logik zweiter Stufe

Second Order: unentscheidbar!

Fluenten: Prädikate/Fakten, die sich von Situation zu Situation ändern können.

Effekt-Axiome:
do : Situationen $\rightarrow$ Situationen
$\text{fragile}(x, s) \rightarrow \text{broken}(x, \textbf{do}(\text{drop}(r, x), x))$

(notwendige) Vorbedingungen: $\textbf{poss}(\text{pickup}(r, x), s) \rightarrow \forall x \forall y \ldots$

Qualification Problem: Hinreichende Bedingungen? Man müsste *alle* Vorbedingungen auflisten können! (Nicht-Monotonie?)

Frame Problem: Die Frage, wie in einer Formalisierung einer Aktion ausgedrückt werden kann, welche Fakten unverändert bleiben (Frame-Axiome).
„an object's color remains unchanged after picking something up"
Probleme:

- So viele Axiome ($\mathcal{O}(F \cdot A)$, F Anzahl Fluenten, A Anzahl Aktionen) explizit angeben?
- Mit so vielen Axiomen effizient schlussfolgern?

18. Golog – Situationenkalkül in Logik erster Stufe

(Reiter)

(Beispiel: Educational Database, Reiter)

Regression

Die Regression $\mathcal{R}[W]$ einer Anfrage W ist wie folgt definiert:

1. wenn W atomar ist:
   - W ist situationsunabhängig: $\mathcal{R}[W] = W$
   - W ist ein Fluent der Form $F(t, S_0)$: $\mathcal{R}[W] = W$
   - W ist eine Vorbedingung der Form $\text{poss}(A(t), s)$.
     Mit der Vorbedingung $\text{poss}(A(x), \sigma)$:- $\Pi_A(x, \sigma)$ gilt:
     $\mathcal{R}[\text{poss}(A(t), s)] = \mathcal{R}[\Pi_A(t, s)]$
   - W ist ein Fluent der Form $F(t, do(a, s))$.
     Mit dem Nachfolgeaxiom $F(x, do(\alpha, \sigma))$:- $\Phi_F(x, \alpha, \sigma)$ gilt:
     $\mathcal{R}[F(t, do(a, s))] = \mathcal{R}[\Phi_F(t, a, s)]$

2. wenn W nicht atomar ist:
   - $\mathcal{R}[\neg W] = \neg\mathcal{R}[W]$
   - $\mathcal{R}[W_1 \wedge W_2] = \mathcal{R}[W_1] \wedge \mathcal{R}[W_2]$
   - $\mathcal{R}[\exists v : W] = \exists v : \mathcal{R}[W]$

Regressionstheorem

Ausgangspunkt ist die Menge $\mathcal{D}$ an Vorbedingungen und Nachfolgeaxiomen („theory of actions“) und eine Anfrage W. $\mathcal{D}_{S_0}$ ist die ursprüngliche Theorie, die folgende Voraussetzungen erfüllen muss:

- keine Formel quantifiziert über Situationen
- keine Formel enthält $\text{poss}(\alpha, s)$
- keine Formel enthält $\text{do}(\alpha, s)$
- Namen für Konstanten sind eindeutig

Dann gilt das Regressionstheorem:

$$\mathcal{D} \models W \;\Leftrightarrow \mathcal{D}_{S_0} \models \mathcal{R}[W]$$

Prolog

Lloyd-Topor (Reiter, 5.2)
Durch die Lloyd-Topor-Transformationen können Vorbedingungen und Nachfolgeaxiome immer in Klauselform (→ Lloyd-Topor-Normalform), also ein Prolog-Programm, überführt werden.

Satz von Clark (Reiter, 5.1, 5.2)
Unter Voraussetzung der Closed World Assumption stellt Prolog ein korrektes und vollständiges Beweisverfahren für Basic Action Theories zur Verfügung.

Komplexe Aktionen

(Reiter, Kapitel 6.1)

Primitive Aktionen $\text{Do}(a, s, s') := \text{poss}(a[s], s) \wedge s' = \text{do}(a[s], s)$

Überprüfen von Bedingungen $\text{Do}(\phi?, s, s') := \phi[s] \wedge s = s'$

Sequenzen von Aktionen $\text{Do}(\delta_1; \delta_2, s, s') := \exists\sigma : \text{Do}(\delta_1, s, \sigma) \wedge \text{Do}(\delta_2, \sigma, s')$

Nichtdeterministische Auswahl aus zwei Aktionen $\text{Do}(\delta_1|\delta_2, s, s') := \text{Do}(\delta_1, s, s') \vee \text{Do}(\delta_2, s, s')$

Nichtdeterministisches Belegen von Parametern einer Aktion

$$\mathrm{Do}((\pi x)\,\delta(x), s, s') := \exists \xi : \mathrm{Do}(\delta(\xi), s, s')$$

Nichtdeterministische Iteration

$$\begin{aligned}&\mathrm{Do}(\delta^*, s, s') :=\\ &\forall P : (\forall \sigma_1 : P(\sigma_1, \sigma_1) \wedge \forall \sigma_1, \sigma_2, \sigma_3 : \mathrm{Do}(\delta, \sigma_1, \sigma_2) \wedge P(\sigma_2, \sigma_3) \rightarrow P(\sigma_1, \sigma_3))\\ &\qquad \rightarrow P(s, s')\end{aligned}$$

Das ist nichts anderes als das Bilden der transitiven Hülle $[\mathrm{TC}_{\sigma_1,\sigma_2}\mathrm{Do}(\delta, \sigma_1, \sigma_2)](s, s')$.

if und while

Damit kann man `if` und `while` definieren:

$$\texttt{if } \phi \texttt{ then } \delta_1 \texttt{ else } \delta_2 := [\phi?; \delta_1] \,|\, [\neg\phi?; \delta_2]$$
$$\texttt{while } \phi \texttt{ do } \delta := [\phi?; \delta]^*; \neg\phi$$

Prozeduren

Programm:

$$\begin{aligned}&\Phi :=\\ &\quad \texttt{proc } P_1(\mathbf{v_1})\ \delta_1\ \texttt{endProc};\\ &\quad \ldots\\ &\quad \texttt{proc } P_n(\mathbf{v_n})\ \delta_n\ \texttt{endProc};\\ &\quad \delta_0\end{aligned}$$

Vorgehensweise:

- Auswerten der Parameter in der aktuellen Situation:

$$\mathrm{Do}(P(t_1, \ldots, t_n), s, s') := P(t_1[s], \ldots, t_n[s], s, s')$$

- Auswerten des Programms:

$$\begin{aligned}&\mathrm{Do}(\Phi, s, s') := \forall P_1, \ldots, P_n : (\\ &\qquad \bigwedge_{i=1}^{n} \forall \sigma_1, \sigma_2, \mathbf{v_i} : \mathrm{Do}(\delta_i, \sigma_1, \sigma_2) \rightarrow P_i(\mathbf{v_i}, \sigma_1, \sigma_2)\\ &\qquad \rightarrow \mathrm{Do}(\delta_0, s, s')\\ &)\end{aligned}$$

Golog by Example

Komplexe Aktionen in Golog

```
1  do(E1 : E2, S, S1) :- do(E1, S, S2), do(E2, S2, S1).
2  do(?(P), S, S) :- holds(P, S).
3  do(E1 # E2, S, S1) :- do(E1, S, S1) ; do(E2, S, S2).
4  do(star(E), S, S1) :- S1 = S ; do(E : star(E), S, S1).
5  do(pi(V, E), S, S1) :- sub(V, _, E, E1), do(E1, S, S1).
6  do(E, S, do(E, S)) :- primitive_action(E), poss(E, S).
7
8  do(if(P, E1, E2), S, S1) :- do((?(P) : E1) # (?(-P) : E2), S, S1).
9  do(while(P, E), S, S1) :- do(star(?(P) : E) : ?(-P), S, S1).
```

Variablen in Aktionen

(Reiter, 6.3.1)

`some(n, on(n))` $\rightsquigarrow$ `do(?(P), S, S) :- holds(P, S)` $\rightsquigarrow$ ...

```
1  holds(A, S) :-
2    restoreSitArg(A, S, F), F
3    ;
4    not restoreSitArt(A, S, F), isAtom(A), A.
5  isAtom(A) :- not (A = -W ; A = (W1 & W2) ; A = (W1 => W2) ; % ...
6
7  holds(P & Q, S) :- holds(P, S), holds(Q, S).
8  holds(P v Q, S) :- holds(P, S) ; holds(Q, S).
9  holds(P => Q, S) :- holds(-P v Q, S).
10 % ...
11 holds(all(V, P), S) :- holds(-some(V, -P), S).
12 holds(some(V, P), S) :- sub(V, _, P, P1), holds(P1, S).
```

`sub` zerlegt den Term in seine syntaktischen Bestandteile und ersetzt die Variable `V` durch ein neues Symbol ($\rightarrow$ _).

(Lloyd-Topor-Transformationen!)

The Elevator Controller

(Reiter, 6.3.4)

```
% Primitive control actions
primitive_action(turnoff(N)).  % Turn off call button N.
primitive_action(open).        % Open elevator door.
primitive_action(close).       % Close elevator door.
primitive_action(up(N)).       % Move elevator up to floor N.
primitive_action(down(N)).     % Move elevator down to floor N.

% Definitions of Complex Control Actions
proc(goFloor(N), ?(currentFloor(N)) # up(N) # down(N)).
proc(serve(N), goFloor(N) : turnoff(N) : open : close).
proc(serveAfloor, pi(n, ?(nextFloor(n)) : serve(n))).
proc(park, if(currentFloor(0), open, down(0) : open)).

% control is the main loop. So long as there is an active call
% button, it serves one floor. When all buttons are off, it
% parks the elevator.
proc(control, while(some(n, on(n)), serveAfloor) : park).

% Preconditions for Primitive Actions.
poss(up(N), S) :- currentFloor(M, S), M < N.
poss(down(N), S) :- currentFloor(M, S), M > N.
poss(open, S).
poss(close, S).
poss(turnoff(N), S) :- on(N, S).

% Successor State Axioms for Primitive Fluents.
currentFloor(M, do(A,S)) :- A = up(M) ; A = down(M) ;
              not A = up(N), not A = down(N), currentFloor(M,S).
on(M, do(A,S)) :- on(M,S), not A = turnoff(M).

% Initial Situation. Call buttons: 3 and 5. The elevator is at floor 4.
on(3, s0). on(5, s0). currentFloor(4, s0).

% nextFloor(N,S) is an abbreviation that determines which of the
% active call buttons should be served next. Here, we simply
% choose an arbitrary active call button.
nextFloor(N,S) :- on(N,S).

% Restore suppressed situation arguments.
restoreSitArg(on(N), S, on(N,S)).
restoreSitArg(nextFloor(N), S, nextFloor(N,S)).
restoreSitArg(currentFloor(M), S, currentFloor(M,S)).
```

(http://www.cs.toronto.edu/cogrobo/kia/simpleElevator)

19. POMDP im Situationenkalkül: Synthese zwischen stochastischer und symbolischer Modellierung

(Reiter, 12)

Unsicheres Wissen

Logische Unsicherheit (Disjunktion, Existenzquantifikation, Negation)
Ein Ereignis tritt sicher ein – aber welches?

Probabilistische Unsicherheit
Unsicherheit bei jedem Elementarereignis:

$$P(\text{goTo}(\text{start})) = 0.35$$
$$P(\text{getLost}(\text{end})) = 0.125$$

Zur Integration in den Situationenkalkül werden Elementarereignisse als deterministisch angenommen:
`choice(giveCoffee(p), a) := a = giveCoffeeT(p) ∨ a = giveCoffeeF(p)`

Stochastisches Golog

```
stDo(go(office(sue)) : giveMail(sue) : giveCoffee(sue) : nil, P, s0, S).
  P = 0.93
  S = do(giveCoffee(sue), do(giveMail(sue), do(endUpAt(office(sue)), s0)))
```

(PO)MDP im Situationenkalkül

Die initiale Wahrscheinlichkeitsverteilung wird durch Aufzählung aller Möglichkeiten mit entsprechenden Wahrscheinlichkeiten angegeben:

```
1 init(S) :- S = s01 ; S = s02 ; S = s03 ; s = s04.
2 loc(mailroom, s01). mailPresent(P, s01) :- P = pat ; P = sue.
3 coffeeRequested(_, s01).
4 initProb(s01, P) :- P is 0.4 * 0.9.
```

Reward-Funktion:

```
reward(puMail,S,R) :- R = 20.
reward(giveMail(P),S,R) :- P = ann, R = 80 ; P = alf, R = 100 ;
                           person(P), not P = ann, not P = alf, R = 50.
reward(giveCoffeeS(P),S,R) :- P = ann, R = 60 ; P = alf, R = 90 ;
                              person(P), not P = ann, not P = alf, R = 50.
reward(giveCoffeeF(P),S,R) :- P = ann, R = -40 ; P = alf, R = -60 ;
                              person(P), not P = ann, not P = alf, R = -30.
reward(endUpAt(L),S,R) :- R = 25.
reward(getLost(L),S,R) :- R = -100.
cost(puMail,S,C) :- C = 0.0 .
cost(giveMail(P),S,C) :- C = 0.0 .
cost(giveCoffeeS(P),S,C) :- C = 0.0 .
cost(giveCoffeeF(P),S,C) :- C = 0.0 .
cost(endUpAt(L),S,C) :- loc(L0,S), dist(L0,L,D), C is 0.25 * D.
cost(getLost(L),S,C) :- loc(L0,S), dist(L0,L,D), C is 0.50 * D.
```

Value Iteration

Bewertung einer Aktion a in einer Situation s:

$$\text{value}(\text{do}(a, s)) = \text{value}(s) + \text{reward}(a, s) - \text{cost}(a, s)$$

Erwartungswert einer Folge γ von Aktionen:

$$\text{evalue}(\gamma) = \sum_{s \in I} \text{initProb}(s) \cdot \left(\sum_{(p,\sigma) \in A} p \cdot \text{value}(\sigma) \right)$$

I: die Menge aller initialen Situationen
$A(s) = \{(p, \sigma) \mid \mathcal{D} \models \text{stDo}(\gamma : \text{nil}, p, s, \sigma)\}$:
die Menge aller Wahrscheinlichkeiten für die aus γ und s resultierenden Situationen

In Prolog kann diese Berechnung mit dem `findall`-Prädikat erfolgen.

MDP

Mit jeder Aktion und den ihr zugeordneten Elementarereignissen

$$\text{choice}(\alpha, c) \;:=\; (c = \nu_1) \vee \cdots \vee (c = \nu_k)$$

wird eine *sense*-Aktion assoziiert:

$$\text{choice}(\text{sense}(\alpha), c) \;:=\; (c = \text{observe}(\nu_1)) \vee \cdots \vee (c = \text{observe}(\nu_k))$$

POMDP

Unterschiede zu MDP:

- Unsicherheit in der initialen Situation
- Fehlerbehaftete Sensoren:

  $$\text{choice}(\text{sense}(\text{go}(l)), c) \;:=\; \exists \lambda : \text{location}(\lambda) \wedge (c = \text{observe}(\text{loc}(\lambda))$$

  Die Wahrscheinlichkeitsverteilung dazu wird nach dem Muster

  $$p = \text{prob}_0(\text{observe}(\text{loc}(\lambda)), \text{sense}(\text{go}(l)), s)$$

  (z.B. durch einfache Aufzählung) angegeben.
- Von den Sensoren (z. B. wegen Dunkelheit) nicht beobachtbare Ereignisse:

  $$\begin{aligned}&\text{choice}(\text{sense}(\text{giveCoffee}(\text{person})), c) \;:= \\ &\quad c = \text{observe}(\text{giveCoffeeSuccess}(\text{person})) \vee \\ &\quad c = \text{observe}(\text{giveCoffeeFailure}(\text{person})) \vee \\ &\quad c = \text{observe}(\text{giveCoffeeUnknown}(\text{person}))\end{aligned}$$

Bewertung

Vorteil: Die stochastische Modellierung eines POMDP wird mit einer expliziten Semantik der zulässigen Aktionen kombiniert.

Symbolische vs. stochastische Modellierung:

- Mit logischen Mitteln kann Semantik explizit formuliert werden.
  Aber: Aufwand, Expressivität und Komplexität setzen Grenzen.
- Die Optimierungsidee der stochastischen Modellierung garantiert, dass immer eine Lösung gefunden wird.
  Aber: Wie viel ist sie in der Praxis Wert?

Teil VI.
Regelbasierte Systeme

20. Regelbasierte Systeme: Grundlagen

Grundlagen

Beispiel: Vorhersage des nächsten Schlages
`if ((schlag(t-2) == X) and (schlag(t) == Y)) then schlag(t) = Z`

Bestandteile:

- Working Memory (Gedächtnis)
- Algorithmus, der überprüft, ob die Prämissen einer Regel erfüllt sind
- Algorithmus, der eine Regel auswählt, falls mehrere möglich sind
- Algorithmus, der aus der ausgewählten Regel Schlüsse zieht und den Speicher aktualisiert (z.B. je nach Korrektheit der Vorhersage Gewichte der Regeln aktualisieren)

Vorwärtsverkettung Transitive Verknüpfung von Regeln:
$A \to B, B \to C, A \Rightarrow C$

Vorwärtsverkettung: Die entspr. Folie (20.9) scheint damit nicht viel zu tun zu haben?

Konfliktresolution Auswahl einer mehrerer möglicher Regeln: die erste, eine zufällige, die mit einer besten Bewertung, ...

Metaregeln bewerten die Qualität des Regeleinsatzes und wählen im Konfliktfall eine Regel aus (z.B. „bevorzuge Regeln, die billige Verfahren vorschlagen“, „bevorzuge Regeln, die bisher oft erfolgreich waren“, ...).

Priorisierung von Regeln (Ordnen linear oder in Baumstruktur):

- Aktualität: möglichst viele neue Fakten
- geringstes Alter: möglichst wenig alte Fakten
- Vermeiden von wiederholten Regelanwendungen
- Vermeiden von Mehrfachinstantiierungen

Spezialisierung Eine Regel R_1 ist spezieller als eine Regel R_2 wenn

- R_1 mindestens so viele Prämissen hat wir R_2
- zu jeder Prämisse in R_2 gibt es mindestens eine Prämisse in R_1, die von dieser Prämisse subsumiert wird
- R_1 und R_2 sind nicht identisch

21. Regelbasiertes Programmieren: Details zum Rete-Algorithmus

Komplexität bei naiver Prämissenüberprüfung:

$$\mathcal{O}(\text{Regeln} \cdot \text{Fakten}^{\text{Vorbedingungen}})$$

Verbesserung: Rete-Algorithmus

A Rete-based expert system builds a network of nodes, where each node (except the root) corresponds to a pattern occurring in the left-hand-side (the condition part) of a rule. The path from the root node to a leaf node defines a complete rule left-hand-side. Each node has a memory of facts which satisfy that pattern.

As new facts are asserted or modified, they propagate along the network, causing nodes to be annotated when that fact matches that pattern. When a fact or combination of facts causes all of the patterns for a given rule to be satisfied, a leaf node is reached and the corresponding rule is triggered.

Aufbau

Alpha-Netzwerk Das A.-N. besteht aus SELECT-Knoten, die von „oben“ kommende Elemente aus dem Working Memory (Fakten) mit dem zum Knoten gehörenden Literal aus der Vorbedingung unifizieren oder auch ausfiltern, falls dies nicht gelingt.
Die gefundenen Substitutionen werden im Alpha Memory gespeichert und später durch JOIN-Knoten kombiniert (oder auch direkt an einen Produktionsknoten weitergegeben).

Beispiel:
Der SELECT-Knoten zu `hindernis(R, links)` würde zu den Fakten `hindernis(robertino, links)` und `hindernis(robertina, links)` zwei mögliche Substitutionen `[R/robertino]` und `[R/robertina]` finden.

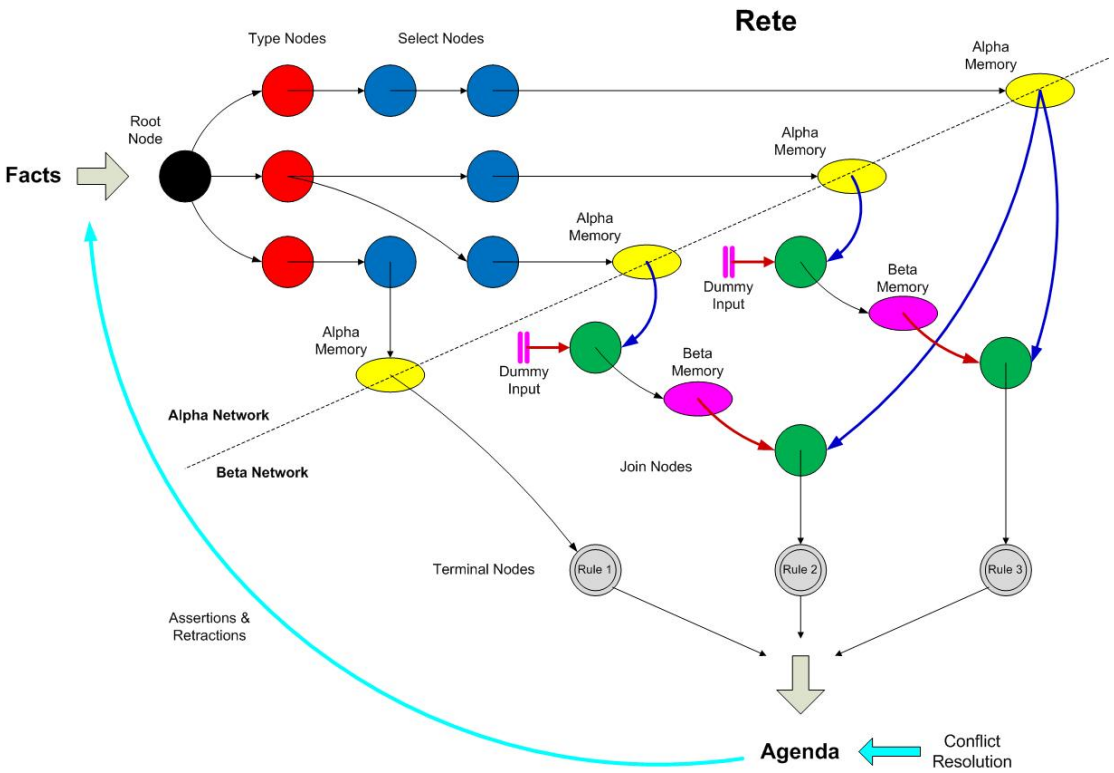
Beta-Netzwerk Im B.-N. werden die Daten aus dem Alpha-Memory zu *partial matches* für die Prämisse kombiniert und in Beta Memories abgelegt.

Wesentliches Element ist der JOIN-Knoten, der als Eingabe normalerweise einen Alpha-Speicher (einzelne Substitutionen) und einen Beta-Speicher (Liste von Substitutionen) hat und versucht, die Substitutionen zu einer neuen konsistenten Substitutionsliste zu kombinieren. Falls die Substitutionen tatsächlich verträglich sind, landet das Ergebnis in einem Beta Memory oder am Ende bei einem Produktionsknoten.

Beispiel:
$\{[\texttt{x/a}]\} + \{[\texttt{y/b}], [\texttt{z/c}]\} = \{[\texttt{x/a}], [\texttt{y/b}], [\texttt{z/c}]\}$
$\{[\texttt{x/a}]\} + \{[\texttt{x/b}]\}$: nicht verträglich

Produktionsknoten lösen Aktionen aus, durch die (ggf. nach Konfliktresolution, falls mehrere Produktionsknoten feuern!) Fakten verändert werden können.

Im einfachsten Fall berechnet man dann das gesamte Netz neu oder man merkt sich beispielsweise, welche Substitutionen von welchen Fakten abhängig waren, und aktualisiert nur den betroffenen Teil.



Bewertung

Vorteile:

- It reduces or eliminates certain types of redundancy through the use of node sharing.
- It stores partial matches when performing joins between different fact types. This, in turn, allows production systems to avoid complete re-evaluation of all facts each time changes are made to the production system's working memory. Instead, the production system needs only to evaluate the changes (deltas) to working memory.
- It allows for efficient removal of memory elements when facts are retracted from working memory.

Probleme:

- Reihenfolge der Einzelbedingungen bestimmt, welche Verschmelzungen möglich sind und wie kompakt der Graph ist.
- Änderung des Netzes durch zur Laufzeit erzeugte neue Regeln ist problematisch.
- Neue Regeln reagieren nicht auf den bisherigen Speicherinhalt.
- Ändern des Speicherinhalts erfordert Löschen und Wiedereinfügen eines Datenelementes $\Rightarrow$ teuer.
- Hoher Speicherverbrauch.

22. Lernen von Regeln

Möglichkeiten, Regeln zu lernen:

- Induktiv: Lernen aus einer Beispielmenge
- Deduktiv: Ableiten neuer Regeln aus bereits bekannten Regeln und Fakten

Separate and Conquer

Ziel: Regeln der Form $\alpha_1 \wedge \cdots \wedge \alpha_n \rightarrow \text{OK}$
$\Rightarrow$ Stück für Stück die *positiven* Beispiele abdecken

Vorgehen:

- Eine Regel $\alpha_i \rightarrow \text{OK}$ definieren, die (fast) nur positive Beispiele benutzt
- Solange die Regel auch (zu viele) negative Beispiele positiv bewerten würde, muss die Prämisse um weitere Bedingungen erweitert werden: $\alpha_i \wedge \alpha_j \rightarrow \text{OK}$ etc.

Auswahl einer Variablen:

$$r_\alpha = \frac{n_\alpha^+}{n_\alpha} = \frac{\#(\text{positive Beispiele, in denen } \alpha \text{ wahr ist})}{\#(\text{alle Beispiele, in denen } \alpha \text{ wahr ist})}$$

Die Grundmenge muss natürlich nach jeder Auswahl eines α entsprechend eingeschränkt werden.

$\rightsquigarrow$ GSCA (Generic Separate and Conquer Algorithm)

Induktive Logikprogrammierung

„Positive Beispiele + Negative Beispiele + Hintergrundwissen ⇒ Regeln“

Hintergrundwissen (Karten-Beispiel): `junction(x)`, `easy(x,y)`

GSCA für Prädikatenlogik funktioniert im Prinzip genau wie im aussagenlogischen Fall, nur mit dem zusätzlichen Freiheitsgrad der Variablenbindung.

Um Regeln für `easy(x,y)` zu lernen, kann man beispielsweise zuerst `easy(x,y) :- junction(x)` versuchen und das dann zu `easy(x,y) :- junction(x), junction(y)` erweitern, was nur noch positive Beispiele abdeckt.

Bewertung der Qualität einer Regelmenge

- Relative Häufigkeit: $\frac{n_c}{n}$
  n_c: Anzahl der korrekt benutzten Beispiele
  n: Anzahl aller von der Regelmenge benutzten Beispiele
- m-Schätzung
- Entropie

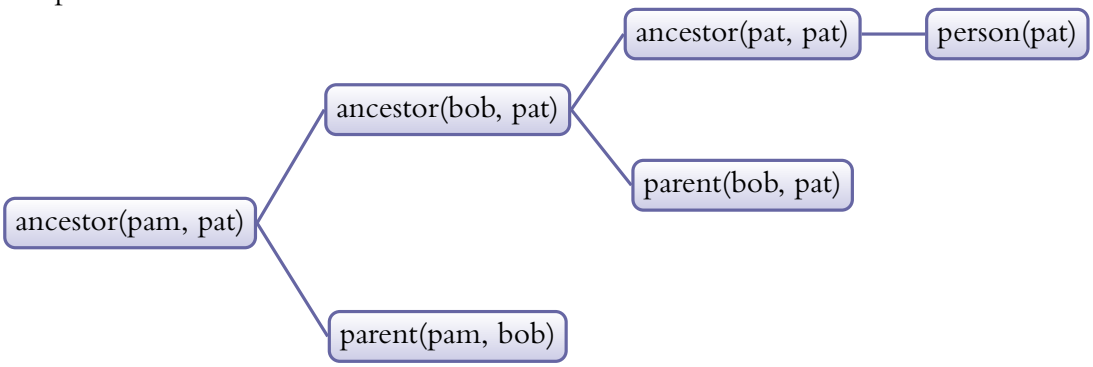
Explanation-Based Generalization

(`http://www.dvs.tu-darmstadt.de/teaching/dke/2008/vorlesung/ebl.pdf`)

Vorgehensweise:

1. Das Trainingsbeispiel wird mit Hilfe der vorhandenen Domänentheorie bewiesen („erklärt“) ⇒ Resolution.
2. Generalisierung: Der Beweis wird verallgemeinert, indem Konstanten durch Variablen ersetzt werden ⇒ Regression. (Mitchell, Kapitel 11.2.1.2)
3. Die Konjunktion der generalisierten Fakten, die im Beweis verwendet werden, bildet den neuen Regelrumpf (d.h. die Konjunktion aller Blätter im Beweisbaum).

Beispiel:



Aus diesem Beweis von `ancestor(pam, pat)` ergibt sich die neue Regel
`ancestor(X, Y) :- parent(X, Z), parent(Z, Y), person(Y).`

23. Lernen von Regeln: Algorithmen für First-Order-Regeln

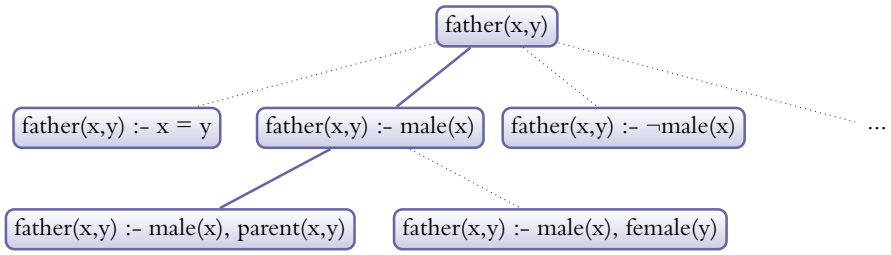
(Mitchell, Kapitel 10.5)

FOIL

Gleiches Prinzip wie bei GSCA.
FOIL lernt Mengen von Klauseln, die Horn-Klauseln ähneln. Unterschiede: keine Funktionen, negierte Literale im Regelrumpf.

Literale, die zu einer Regel hinzugefügt werden können:

- $P(v_1, \ldots, v_n)$, wenn alle v_i Variablen sind und mindestens ein v_i schon in einem anderen Literal der Regel benutzt wird
- Identitäten $v_i = v_j$, falls beide Variablen schon in der Regel vorkommen
- das negierte Literal zu einem so entstandenen Literal



Steuerung der Suche: Foil-Gain

Ziel: Entropiereduktion bei der Auswahl der Literale durch Maximierung des Foil-Gains:

$$\text{FoilGain} = t \cdot \left(\log_2 \frac{p_1}{p_1 + n_1} - \log_2 \frac{p_0}{p_0 + n_0} \right)$$

p_0, n_0: Anzahl pos./neg. Beispiele für die alte Regel
p_1, n_1: Anzahl pos./neg. Beispiele für die neue Regel
t: pos. Beispiele für die neue Regel? ($= p_1$?)

Lernen rekursiver Regeln

Wichtig: Es muss einen streng monoton fallenden Parameter geben!

Teil VII.
Fuzzy-Systeme

24. Fuzzy-Mengen und Fuzzy-Relationen

Arbeitsweise von Fuzzy-Systemen

- Fuzzyfizierung der konkreten Eingangsgrößen:
  200 km/h $\rightarrow$ 0.7 sehr schnell
- Inferenz und Komposition der Regeln
- Defuzzyfizierung der konkreten Ausgangsgrößen:
  0.7 sehr schnell $\rightarrow$ 200 km/h

Hintergrund

Eine Fuzzy-Menge A über X wird definiert durch eine Zugehörigkeitsfunktion

$$\mu_A : X \rightarrow [0; 1]$$

Schreibweise für endliche Fuzzy-Mengen:

$$A = 0.5/x_1 + 0.6/x_2 + 0.8/x_3$$

Relevante Operationen:
Konjunktion, Disjunktion, Negation

Man versucht, möglichst viele der Axiome einer Booleschen Algebra auf Fuzzy-Systeme zu übertragen. Beispielsweise mit dem Kontradiktionsgesetz ($A \cap A^c = \varnothing$) ist dies allerdings nicht möglich.

t-Norm und t-Co-Norm

t-Norm ($\top$, „Konjunktion“)

- neutrales Element 1
- monoton wie min: $a \leq b \wedge b \leq c \Rightarrow \min(a, c) \leq \min(b, d)$
- kommutativ wie min: $\min(x, y) = \min(y, x)$
- assoziativ wie min: $\min(x, \min(y, z)) = \min(\min(x, y), z)$

t-Co-Norm ($\bot$, „Disjunktion“)

- neutrales Element 0
- monoton wie max: $a \leq b \wedge b \leq c \Rightarrow \max(a, c) \leq \max(b, d)$
- kommutativ wie max: $\max(x, y) = \max(y, x)$
- assoziativ wie max: $\max(x, \max(y, z)) = \max(\max(x, y), z)$

Beispiele für t-(Co-)Normen:

- min und max
- Einstein-Produkt $\frac{xy}{1+(1-x)(1-y)}$ und Einstein-Summe $\frac{x+y}{1+xy}$
- beschränkte Differenz $\max(0, x + y - 1)$ und beschränkte Summe $\min(1, x + y)$

Ein Regler mit $\top = \min$ und $\bot = \max$ heißt *Mamdani*-Regler.

Fuzzy-Relationen

Eine Fuzzy-Relation R wird definiert durch eine Zugehörigkeitsfunktion μ_R:

$$\mu_R : A_1 \times \cdots \times A_N \to [0; 1]$$
$$R = \int_{A_1 \times \cdots \times A_N} \mu_R(a_1, \ldots, a_N)/(a_1, \ldots, a_N)$$

Fuzzy-Relationen sind nur spezielle Fuzzy-Mengen, also kann man sie entsprechend miteinander verknüpfen (t-(Co-)Norm):

$$\mu_{R_1 \cap R_2}(a_1, \dots\ a_N) = \min(\mu_{R_1}(a_1, \dots, a_N), \mu_{R_2}(a_1, \dots, a_N))$$

25. Fuzzy-Regeln und Fuzzy-Inferenz

Implikationen

```
if (temperatur == normal) then ventil = halb_offen
```

Generalisierter Modus Ponens

Aus $A(x)$, $A(x) \rightarrow B(x) \ \Rightarrow\ B(x)$ wird die Relation $A \circ R_{A \rightarrow B}$.

Die Semantik von

```
if A(x) then B(x) else C(x)
```

lässt sich in der Prädikatenlogik wie folgt ausdrücken:

$$(A(x) \rightarrow B(x)) \wedge (\neg A(x) \rightarrow C(x)) \quad \Leftrightarrow$$
$$(A(x) \wedge B(x)) \vee (\neg A(x) \wedge C(x))$$

Das kann man auch im Fall von Fuzzy-Relationen leicht ausrechnen.

$$R \ \subseteq\ (A \times B) \cup (A^c \times C)$$

Lässt man den *else*-Fall weg, erhält man:

$$\mu_{A \rightarrow B}(x, y) = \mu_{A \wedge B}(x, y) = \top(\mu_A(x), \mu_B(y))$$

Prinzipiell kann man bei der Definition eines Fuzzy-Systems zwischen der t-Norm für die Konjunktion ($\top_1$) und der für die Inferenz ($\top_2$) unterscheiden. Das ist aber nicht notwendig und wird beim Mamdani-Regler ($\top_1 = \top_2 = \min$, $\perp = \max$) auch nicht gemacht.

Fuzzy-System

(Borgelt, Kapitel 10.1)

Regelbasis Eine Fuzzy-Regelbasis $\mathcal{R}$ ist eine Menge von Fuzzy-Regeln R_i. Jede Regel R ist ein Tupel aus Fuzzy-Mengen:

$$R_i = (\mu_1, \ldots, \mu_N, \nu_1, \ldots, \nu_M)$$

μ_i: Fuzzy-Menge über dem Wertebereich der Eingabevariablen x_i
ν_i: Fuzzy-Menge über dem Wertebereich der Ausgabevariablen y_i

Fuzzy-System Ein Fuzzy-System ist eine Abbildung

$$F_{\mathcal{R}} : X_1 \times \cdots \times X_N \to Y_1 \times \cdots \times Y_M$$

zu einer gegebenen Regelbasis $\mathcal{R}$.

Defuzzyfizierung

(Borgelt, Kapitel 10.1)

$$\text{ventil} = 0.2/\text{etwas\_offen} + 0.8/\text{offen} + 0.4/\text{weit\_offen}$$
$\Rightarrow$ tatsächlicher Wert?

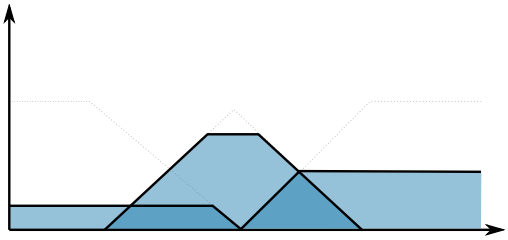
Erfüllungsgrad einer Regel R_i ist die konjunktive Verknüpfung aller Konjunktionsglieder aus dem Antezedens:

$$\tau_{R_i} = \top_1(\mu_1(x_1), \ldots, \mu_N(x_N))$$
$$\tau_{R_i} = \min(\mu_1(x_1), \ldots, \mu_N(x_N))$$

Vorgehensweise bei der Defuzzyfizierung einer Ausgabegröße:

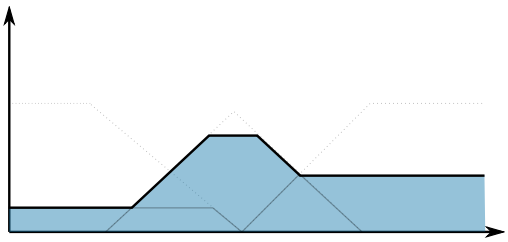
1. Ausgabe-Fuzzy-Menge der einzelnen Regeln R_i bestimmen.
   $\mu(y)$ ist die Zugehörigkeitsfunktion der Ausgabegröße.

   $$\nu_{R_i}(y) = \top_2(\tau_{R_i}, \mu(y))$$



2. Gesamtausgabe des Reglers durch disjunktive Verknüpfung der Ausgaben der einzelnen Regeln bestimmen:

$$\nu(y) = \perp(\nu_{R_1}, \ldots, \nu_{R_M})$$



3. Defuzzifizierung der entstandenen Fuzzy-Menge:

Schwerpunktmethode (Center of Gravity / Center of Area)

$$y = \frac{\int x \cdot \nu(x) dx}{\int \nu(x) dx}$$

(numerisch recht aufwändig)

Mean-of-Maxima-Methode

Jede Regel wird einmalig für Erfüllungsgrad 1 im Antezedens nach der Schwerpunktmethode defuzzyfiziert. Später wird das mit den tatsächlichen Erfüllungsgraden gewichtete Mittel über die berechneten Werte berechnet.

$$y_R = \text{schwerpunkt}(R, \tau_R = 1)$$
$$y = \sum_R \tau_R \cdot x_R$$

Die konkrete Form der Fuzzy-Mengen im Konsequens spielt also keine Rolle – man könnte stattdessen auch einfach die Werte angeben, über die das gewichtete Mittel gebildet werden soll. Dieses Prinzip wird bei *Sugeno*-Reglern verfolgt.

26. Fuzzy-Regel-Systeme

(siehe Kapitel 25)

Lernen von Fuzzy-Regeln

Bei Diskretisierung der Eingabe auf $m + 2$ Werte, q Eingabe-Fuzzy-Mengen und n Eingabegrößen gibt es $\binom{m}{q}^n \cdot 2^{(q+1)^n}$ mögliche Regelbasen...

$\Rightarrow$ Heuristische Einschränkungen ($\rightarrow$ Dreiecksfunktionen, ...)

Überwachtes Lernen

Gegeben: Beispiele (p, t) mit Eingabewerten $p \in \mathbb{R}^N$ und Zugehörigkeitsgraden $t \in [0; 1]^M$ (M verschiedene Klassen).

Lernen der Regeln:

- Initialisierung: Der Wertebereich der Ein- und Ausgabegrößen wird manuell in gleichmäßig verteile Fuzzy-Mengen partitioniert. Dabei werden sich überlappende Dreiecksfunktionen verwendet.
- Generate and Test: Generiere eine Menge von Regeln, bewerte jede davon, eliminiere schlechte Regeln.
  - Generieren von *if*-Teilen
  - Einordnen in die am besten passende Klasse
  - Bewertung und Auswahl $\rightarrow$ Regelbasis; entweder durch Vorgabe einer maximalen Anzahl von Regeln oder durch auswahl so vieler Regeln wie notwendig sind um alle Beispiele abzudecken.

27. Lernen von Fuzzy-Mengen aus Daten

Möglichkeiten zur Optimierung

Adaptive Regelgewichte auf Erfüllungsgrad oder Ausgabe:
Ein Mamdani-Regler mit gewichteten Regel ist äquivalent zu einem Mamdani-Regler ohne gewichtete Regeln aber mit geeignet modifizierter Zugehörigkeitsfunktion.

Training von Fuzzymengen anhand von Beispieldaten:

Problem bei der Suche nach einer Fehlerfunktion: max und min beim Mamdani-Regler sind nicht differenzierbar.

- Sum of squared errors

$$\text{SSE} = \frac{1}{2} \sum_{(p,t)} \sum_{j=0}^{m} (t_j - y_j)^2$$

- Fuzzy-Fehlerfunktion
  Linguistische Variable „Ergebnis korrekt“:

$$C(t, y) = \exp(-\frac{a \cdot (t - y)^2}{d_{\max}}) \in [0; 1]$$

  ($d_{\max}$ ist der maximale Fehler bei gegebenen Beispielen)
  Fehler:

$$\text{FE}(t, y) = \text{sign}(t - y) \cdot (1 - C(t, y)) \in [-1; 1]$$

  Gesamtfehler:

$$\text{SAFE} = \sum_{(p,t)} \text{FE}(t, y(p))$$

Strukturoptimierung und Pruning von Regelbasen:

Vier Kriterien werden herangezogen um Regeln zu eliminieren:

- *Korrelation*: Welche Eingabevariable korreliert am wenigsten mit der Ausgabe einer Regel? $\Rightarrow$ diese Variable ist überflüssig.
- *Klassifikationshäufigkeit*: Entferne die Regel, die am seltensten den maximalen Zugehörigkeitsgrad unter den Ausgaben aller Regeln annimmt. Regeln, die fast nie zum Zug kommen, könnten überflüssig sein.
  (Gefahr: Regeln für Ausnahmen!)
- *Redundanz*: Entferne die linguistische Variable, die am seltensten den minimalen von 0 verschiedenen Zugehörigkeitsgrad in einer aktiven Regel aufweist. Variablen, die die *if*-Teile von Regeln fast immer „übererfüllen“, beeinflussen die Ausgabe der Regelbasis nicht.
- *Unschärfe*: Entferne die Fuzzy-Menge mit der größten Trägermenge aus allen Regelbedingungen. Die entsprechende Zugehörigkeitsfunktion hat tendenziell hohe Varianz und führt daher zu starkem Rauschen in der Ausgabe.

Iteratives Vorgehen: Anwendung einer Regel, Neubewertung der Regelbasis, ...

A Formeln

Bayes-Formel: $p(x|y) = \frac{p(y|x) \cdot p(x)}{p(y)}$

Bewegungsmodell: $p(x_t|u_t, x_{t-1})$
Sensormodell: $p(z_t|x_t)$

Bayes-Filter:

$$\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1}) \cdot bel(x_{t-1}) dx_{t-1}$$
$$bel(x_t) = \eta \cdot p(z_t|x_t) \cdot \overline{bel}(x_t)$$

Kalman-Filter:

$$\begin{array}{ll} p(s_t|u_t, s_{t-1}) & p(z_t|s_t) \\ s_t = a_t s_{t-1} + b_t u_t + \varepsilon_t & z_t = c_t s_t + \delta_t \end{array}$$

$$\tilde{\mu}_t = a_t \mu_{t-1} + b_t u_t$$
$$\tilde{\sigma_t}^2 = a_t^2 \sigma_{t-1}^2 + \rho_t^2$$

$$k_t = \frac{\tilde{\sigma_t}^2 c_t}{c_t^2 \tilde{\sigma_t}^2 + \nu_t^2}$$

$$\mu_t = (1 - k_t c_t)\tilde{\mu}_t + k_t z_t$$
$$\sigma_t^2 = (1 - k_t c_t)\tilde{\sigma_t}^2$$

$(k_t : \min E[|s_t - \widehat{s_t}|^2])$

Value Iteration (MDP):

$$V_1(x_t) = \gamma \cdot \max_{u_t} r(x_t, u_t)$$
$$V_T(x_t) = \gamma \cdot (r(u_t, x_t) + \int V_{T-1}(x_{t+1}) \cdot p(x_{t+1}|x_t, u_t) dx_{t+1}$$

FOIL:

$$\text{FoilGain} = t \cdot \left(\log_2 \frac{p_1}{p_1 + n_1} - \log_2 \frac{p_0}{p_0 + n_0} \right)$$

Literatur

[Russell, Norvig] Stuart Russell, Peter Norvig: Künstliche Intelligenz

[Nilsson] Nils Nilsson: Artificial Intelligence — A New Synthesis

[Thrun] Sebastian Thrun, Wolfram Burgard, Dieter Fox: Probabilistic Robotics

[Reiter] Raymond Reiter: Knowledge in Action

[Mitchell] Tom Mitchell: Machine Learning

[Borgelt] Christian Borgelt et al.: Neuro-Fuzzy-Systeme