Multi Agenten Systeme

VU SS 00, TU Wien

Teil 1 (Kapitel 1–4) basiert auf

> **Multi-Agent Systems** (Gerhard Weiss), MIT Press, June 1999.

Es werden allgemeine Techniken und Methoden dargestellt (BDI-, Layered-, Logic based Architekturen, Decision Making, Kommunikation/Interaktion, Kontrakt Netze, Coalition Formation).

Teil 2 (Kapitel 5–9) basiert auf

> **Heterogenous Active Agents** (Subrahmanian, Bonatti, Dix, Eiter, Kraus, Özcan and Ross), MIT Press, May 2000.

Hier wird ein spezifischer Ansatz vorgestellt, der formale Methoden aus dem logischen Programmieren benutzt, aber nicht auf PROLOG aufsetzt (Code Call Mechanismus, Aktionen, Agenten Zyklus, Status Menge, Semantiken, Erweiterungen um Beliefs, Implementierbarkeit).

Übersicht

1. Einführung, Terminologie
2. 4 Grundlegende Architekturen
3. Distributed Decision Making
4. Contract Nets, Coalition Formation
5. *IMPACT* Architecture
6. Legacy Data and Code Calls
7. Actions and Agent Programs
8. Regular Agents
9. Meta Agent Programs

Kapitel 2. Die 4 grundlegenden Ansätze

Überblick

2.1 Reaktive Agenten

2.2 BDI Agenten

2.3 Schichtenarchitekturen

2.4 Logik-basierte Architekturen

2 Die 4 grundlegenden Ansätze

2.1 Reaktive Agenten

Intelligentes Verhalten ist *Interaktion des Agenten mit seiner Umgebung*. Es entsteht durch Aufsplittung in einfachere Interaktionen.

Subsumptions-Architekturen:

- Decision making wird realisiert durch **zielgerichtete Verhaltensweisen** (**Behaviours**): jedes Behaviour ist individuelle Aktion.
  nichtsymbolische Implementierung.
- Viele Behaviours können gleichzeitig angewandt werden. Wie wird selektiert?
  Impl. durch Subsumptions-Hierarchien, Schichten. Obere Schichten repräsentieren abstraktes Verhalten.

Formales Modell

- **see**: wie bisher, aber enge Verzahnung zwischen Wahrnehmung und Aktion: keine Transformation des Inputs .
- **action**: Menge von Behaviours und inhibition Relation.

$$Beh := \{\langle \mathbf{c}, \mathbf{a}\rangle : \mathbf{c} \in \mathbf{P}, \mathbf{a} \in \mathbf{A}\}.$$

$\langle \mathbf{c}, \mathbf{a}\rangle$ "feuert" falls $\mathbf{see}(\mathbf{s}) \in \mathbf{c}$ ($\mathbf{c}$ steht für "condition").

$$\prec \subseteq Ag_{rules} \times Ag_{rules}$$

heißt Inhibitions-Relation, $Ag_{rules} \subseteq Beh$. Wir verlangen von $\prec$ daß es eine totale Ordnung ist. $\mathbf{b_1} \prec \mathbf{b_2}$ bedeutet: b_1 inhibits b_2, **$\mathbf{b_1}$ hat Priorität**.

```
1.  function action(p : P) : A
2.  var fired : ℘(R)
3.  var selected : A
4.  begin
5.          fired := {(c, a) | (c, a) ∈ R and p ∈ c}
6.          for each (c, a) ∈ fired do
7.                  if ¬(∃(c', a') ∈ fired such that (c', a') ≺ (c, a)) then
8.                          return a
9.                  end-if
10.         end-for
11.         return null
12. end function action
```

Example 2.1 (Planetenerkundung)

Auf einem fernen Planeten sollen Gold gesucht und Proben zum Raumschiff gebracht werden. Man weiss nicht wo das Gold ist. Es gibt mehrere autonome Fahrzeuge die eingesetzt werden können. Wegen der Topographie des Planeten können die Fahrzeuge keine Informationen austauschen.

- Das Mutter-Raumschiff sendet Radiosignale aus: **Gradientenfeld**.
- Fahrzeuge können **indirekt** miteinander kommunizieren: sie setzen radioaktive Steine ab. Diese können auch aufgesammelt werden.

Low Level Behaviour: (1) **If** detect an obstacle **then** change direction.

2. Schicht: (2) **If** Proben an Bord **and** an Basis **then** lade aus.
(3) **If** Proben an Bord **and** nicht an Basis **then** wandere Gradient entlang.

3. Schicht: (4) **If** Probe gefunden **then** sammle sie auf.

4. Schicht: (5) **If** true **then** bewege dich zufällig.

Natürlich mit der Ordnung

$$(1) \prec (2) \prec (3) \prec (4) \prec (5).$$

Das klappt hervorragend, falls das Gold zufällig verteilt ist. Was wenn es in Clustern vorkommt?

Low Level Behaviour: (1) **If** detect an obstacle **then** change direction.

2. Schicht: (2) **If** Proben an Bord **and** an Basis **then** lade aus.
(3) **If** Proben an Bord **and** nicht an Basis **then** wirf 2 radiaktive Krümel ab und wandere Gradient entlang.

3. Schicht: (4) **If** Probe gefunden **then** sammle sie auf.
(5) **If** radioaktive Krümel gefunden **then** nimm eines auf und wandere Gradient entlang.

4. Schicht: (6) **If** true **then** bewege dich zufällig.

Natürlich mit der Ordnung

$$(1) \prec (2) \prec (3) \prec (4) \prec (5) \prec (6).$$

Pro: Einfach, ökonomisch, effizient, robust, elegant.

Contra:

- Ohne Vorstellung über Umgebung müssen Agenten Info über eigene lokale Umgebung haben.
- Entscheidungen nur aufgrund lokaler Info.
- Wie ist Lernen möglich?
- Zusammenhang zw. Agenten, Umgebung, Behaviours ist nicht klar.
- Agenten mit ≤ 10 Behaviours sind handhabbar. Aber je mehr Schichten desto komplizierter.

2.2 BDI-Architektur

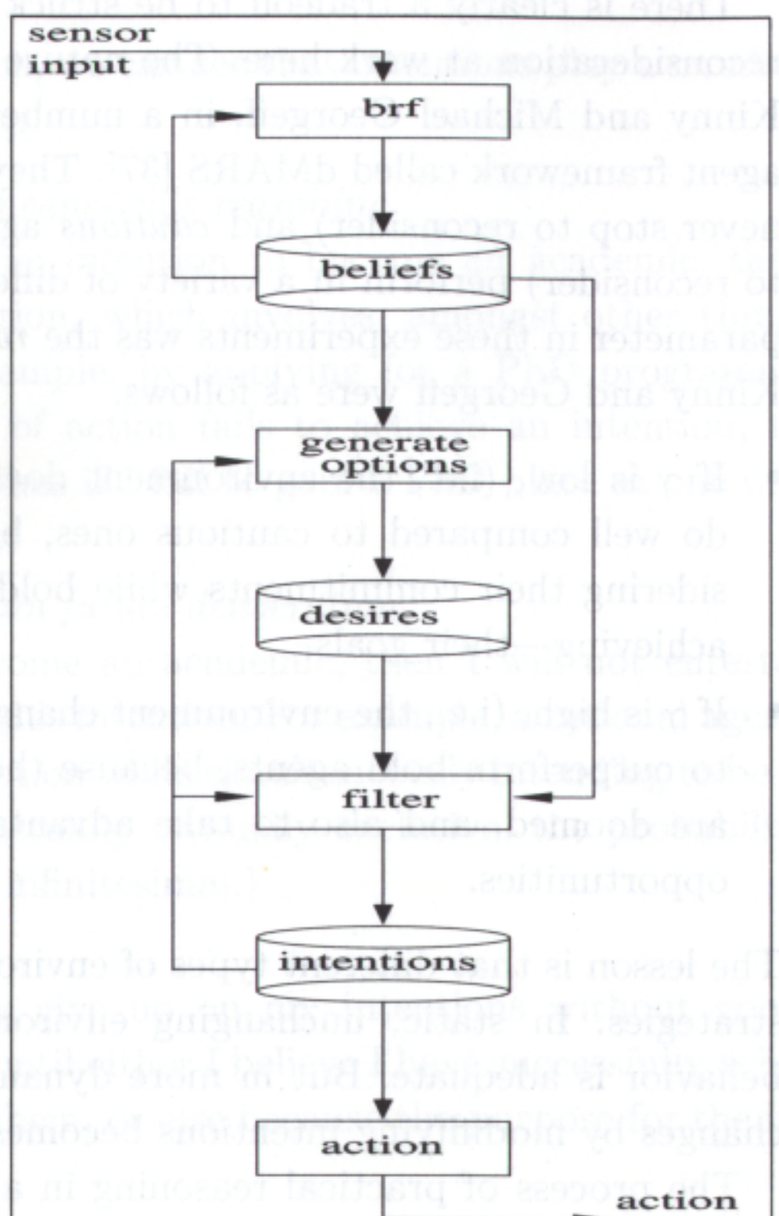
Belief, **D**esire, **I**ntention.

Von Zeit zu Zeit müssen Intentionen überprüft werden. Aber sie sollten auch verfolgt werden. (**Pro-aktiv vs. reaktiv**).
Extreme: *Sture Agenten*, *Unsichere Agenten*.

Was ist besser? Das hängt von der Umgebung ab!!
Sei γ die Rate, mit der sich die Welt ändert.

1. γ klein: Sturheit zählt sich aus.
2. γ groß: Unsicherheit ist besser.

Belief: Vorlesungen besuchen bringt was,
Desire: Besuche Dix-Vorlesung, lese Bücher,
Intention: Wissen über verteilte Systeme


sensor
input
brf
beliefs
generate
options
desires
filter
intentions
action
action
output

```
1.  function action(p : P) : A
2.  begin
3.      B := brf(B, p)
4.      D := options(D, I)
5.      I := filter(B, D, I)
6.      return execute(I)
7.  end function action
```

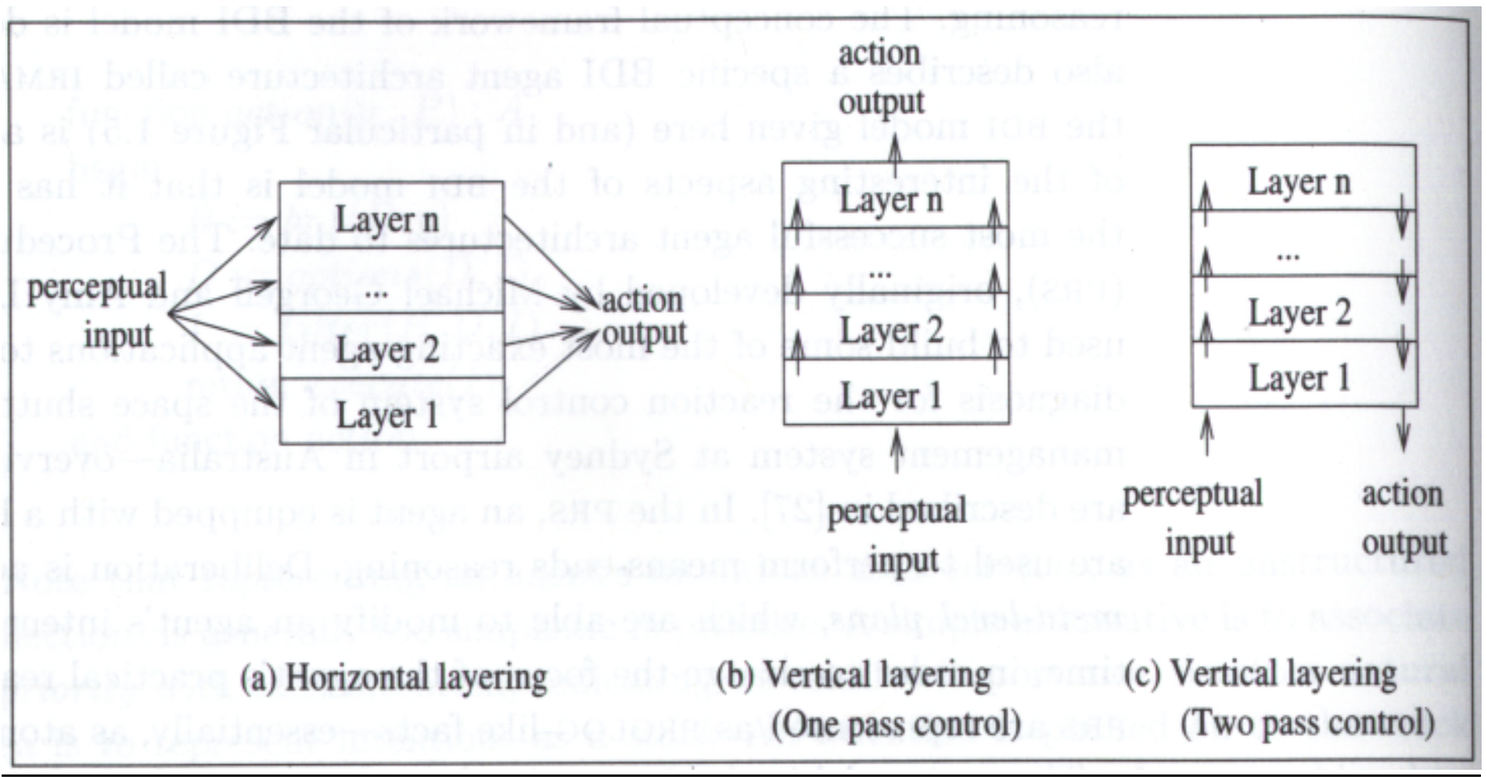
$$(B, D, I) \text{ wobei } B \subseteq Bel, D \subseteq Des, I \subseteq Int$$

I kann als stack repräsentiert werden (damit Prioritäten verfügbar).

- BDI geht auf Bratman/Israel/Pollack (88) zurück.
- PRS (*procedural reasoning system*, Georgeff/Lansky) benutzt BDI.
  Anwendungen: Space Shuttle (Diagnose), Sydney Airport (Luftverkehr).
- BDI-Logiken: Rao et al. 91-96

2.3 Schichtenbasierte Architekturen

Mindestens 2 Schichten: reaktiv, pro-aktiv.



Horizontal:

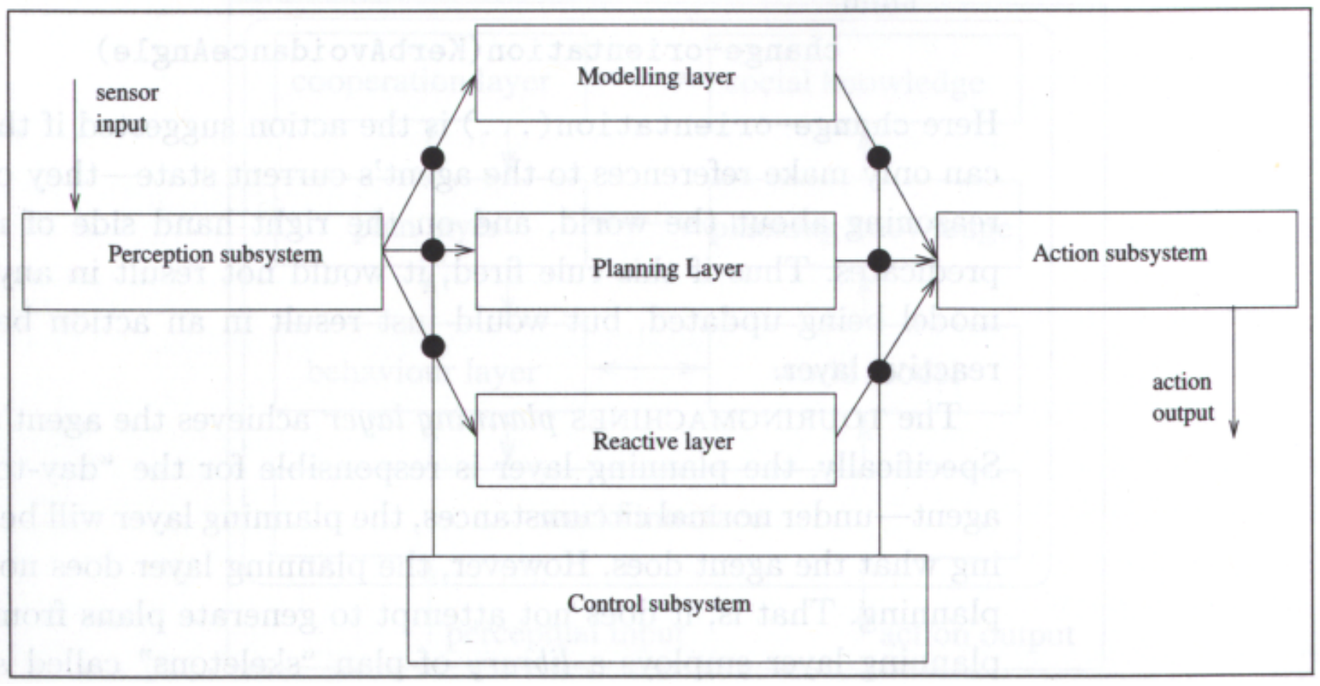
- einfach (n Verhaltensweisen, n Schichten),
- globales Verhalten evtl. nicht konsistent ,
- Interaktion zwischen Schichten: m^n (m = # Aktionen pro Schicht)
- Man braucht Kontrollsystem.

Vertikal:

- Nur $m^2(n-1)$ Interaktionen zwischen Schichten.
- Nicht so robust: fällt eine Schicht aus, bricht alles zusammen.

Touring Maschine

Autonomes Fahrzeug.



Regel 1: *Bordstein vermeiden*

falls $is\_in\_front(curb, observer)$ and
$speed(observer) > 0$ and
$seperation(curb, observer) < curb\_threshold$
dann $change\_orientation(curb\_avoidance\_angle)$

Planning-Schicht: Pro-aktives Verhalten

Modeling Schicht: Bild der Welt (updated), beliefs, ändert Planning-Goals

Control Subsystem: Entscheidet wer aktiv ist. Z.B. sorgt dafür daß gewisse Wahrnehmungen nie an bestimmte Schichten gelangen.

Schichtenbasierte Ansätze haben keine klare Semantik und die horizontale Interaktion ist schwierig.

2.4 Logik-basierte Architekturen

Symbolische KI: Symbolische Repräsentation, etwa in PL1. Ausnutzung von Deduktion. **Agent als Theorembeweiser**.

Traditionell: Theorie über Agenten. Implementierung als schrittweiser Prozess (Software Eng.) über viele Abstraktionsebenen.

Symb. KI: Betrachte Theorie selbst als **ausführbare Spezifikation**.

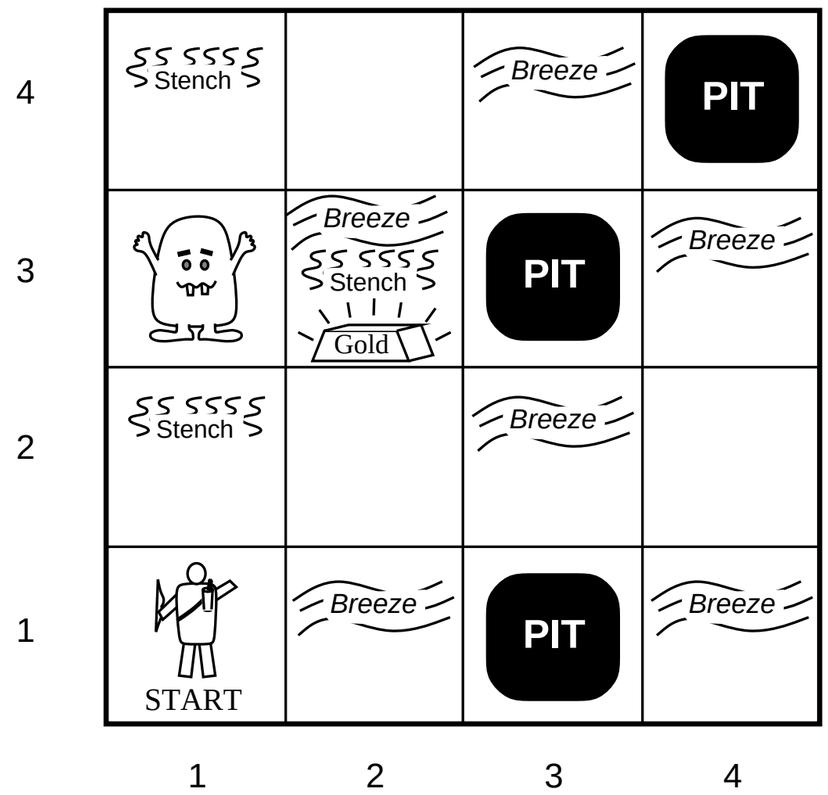
Interner Zustand: *Knowledge Base* (KB) D (Datenbank).

- $\mathbf{see : S \longrightarrow P}$,
- $\mathbf{next : D \times P \longrightarrow D}$,
- $\rightsquigarrow$ **nächste Folie action-Funktion.**

```
1.  function action(Δ : D) : A
2.  begin
3.      for each a ∈ A do
4.          if Δ ⊢_ρ Do(a) then
5.              return a
6.          end-if
7.      end-for
8.      for each a ∈ A do
9.          if Δ ⊬_ρ ¬Do(a) then
10.             return a
11.         end-if
12.     end-for
13.     return null
14. end function action
```

Aussagenlogik

Die Wumpus-Welt in AL



1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 **OK**	2,2	3,2	4,2
1,1 **A** **OK**	2,1 **OK**	3,1	4,1

(a)

A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2 **OK**	2,2 **P?**	3,2	4,2
1,1 **V** **OK**	2,1 **A** **B** **OK**	3,1 **P?**	4,1

(b)

1,4	2,4	3,4	4,4
1,3 **W!**	2,3	3,3	4,3
1,2 **A** **S** **OK**	2,2 **OK**	3,2	4,2
1,1 **V** **OK**	2,1 **B** **V** **OK**	3,1 **P!**	4,1

(a)

A = *Agent*
B = *Breeze*
G = *Glitter, Gold*
OK = *Safe square*
P = *Pit*
S = *Stench*
V = *Visited*
W = *Wumpus*

1,4	2,4 **P?**	3,4	4,4
1,3 **W!**	2,3 **A** **S G** **B**	3,3 **P?**	4,3
1,2 **S** **V** **OK**	2,2 **V** **OK**	3,2	4,2
1,1 **V** **OK**	2,1 **B** **V** **OK**	3,1 **P!**	4,1

(b)

Festlegung der Sprache:

$S_{i,j}$	stinkt
$B_{i,j}$	ist kalt
$Grube_{i,j}$	ist eine Grube
$Gl_{i,j}$	glitzert
$W_{i,j}$	enthält Wumpus

Allgemeines Wissen:

$$\neg S_{1,1} \longrightarrow (\neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,1})$$
$$\neg S_{2,1} \longrightarrow (\neg W_{1,1} \wedge \neg W_{2,1} \wedge \neg W_{2,2} \wedge \neg W_{3,1})$$
$$\neg S_{1,2} \longrightarrow (\neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,2} \wedge \neg W_{1,3})$$

$$S_{1,2} \longrightarrow (W_{1,3} \wedge W_{1,2} \wedge W_{2,2} \wedge W_{1,1})$$

Wissen nach dem 3. Zug:

$$\neg S_{1,1} \wedge \neg S_{2,1} \wedge S_{1,2} \wedge \neg B_{1,1} \wedge \neg B_{2,1} \wedge \neg B_{1,2}$$

Können wir herleiten, dass der Wumpus in (1,3) sitzt? Ja, etwa mit Resolution oder unserem Hilbert-Kalkül.

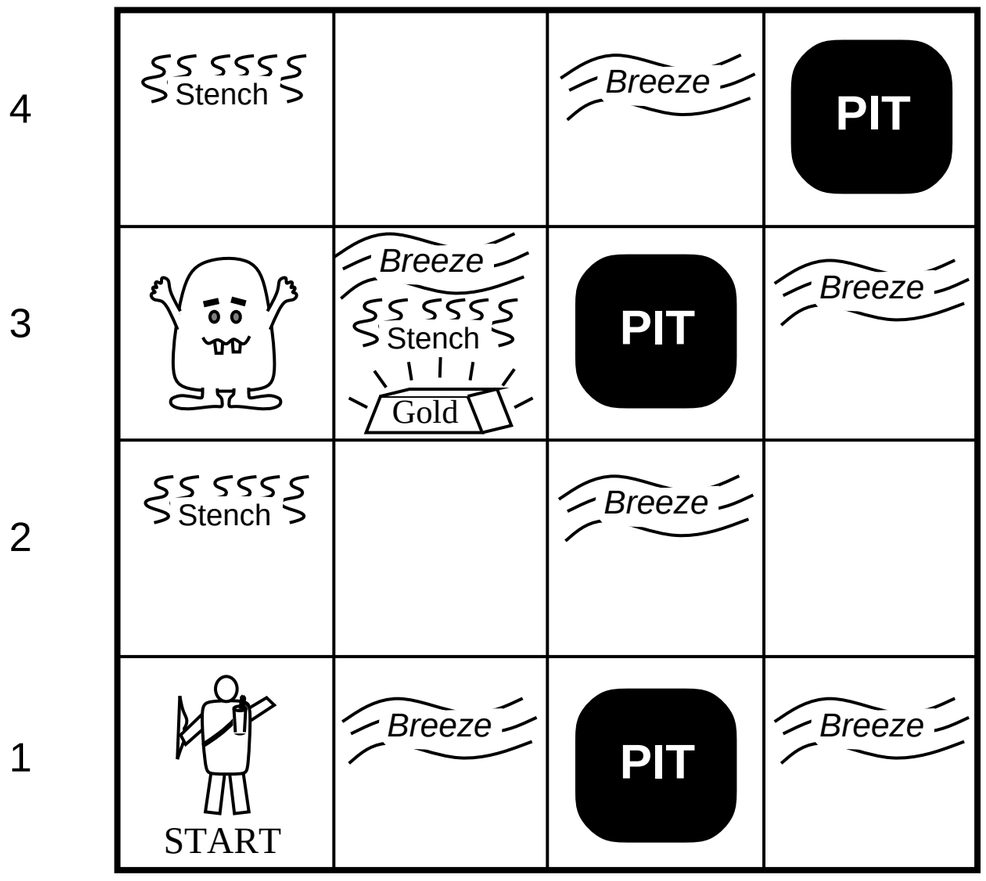
Eigentlich wollen wir mehr: zu gegebener Situation möchten wir gerne die **beste Aktion** auswählen. Das aber geht in AL nicht. Jedoch können wir für jede Aktion prüfen, ob sie ausgeführt werden sollte oder nicht. Dazu benötigen wir zusätzliche Axiome:

Zusätzliche Axiome:

$$A_{1,1} \wedge East \wedge W_{2,1} \longrightarrow \neg Forward$$

$$A_{1,1} \wedge East \wedge Grube_{2,1} \longrightarrow \neg Forward$$

$$A_{i,j} \wedge Gl_{i,j} \longrightarrow Take_{Gold}$$


4
3
2
1
Stench
Breeze
PIT
Breeze
Stench
Gold
PIT
Breeze
Stench
Breeze
START
Breeze
PIT
Breeze
1
2
3
4

Der Situationskalkül

Wie können wir eine dynamische, sich entwickelnde Welt repräsentieren?
Wie axiomatisieren wir die Wumpus-Welt in PL1?

function KB-AGENT(*percept*) **returns** an *action*
  static: *KB*, a knowledge base
    t, a counter, initially 0, indicating time

  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*

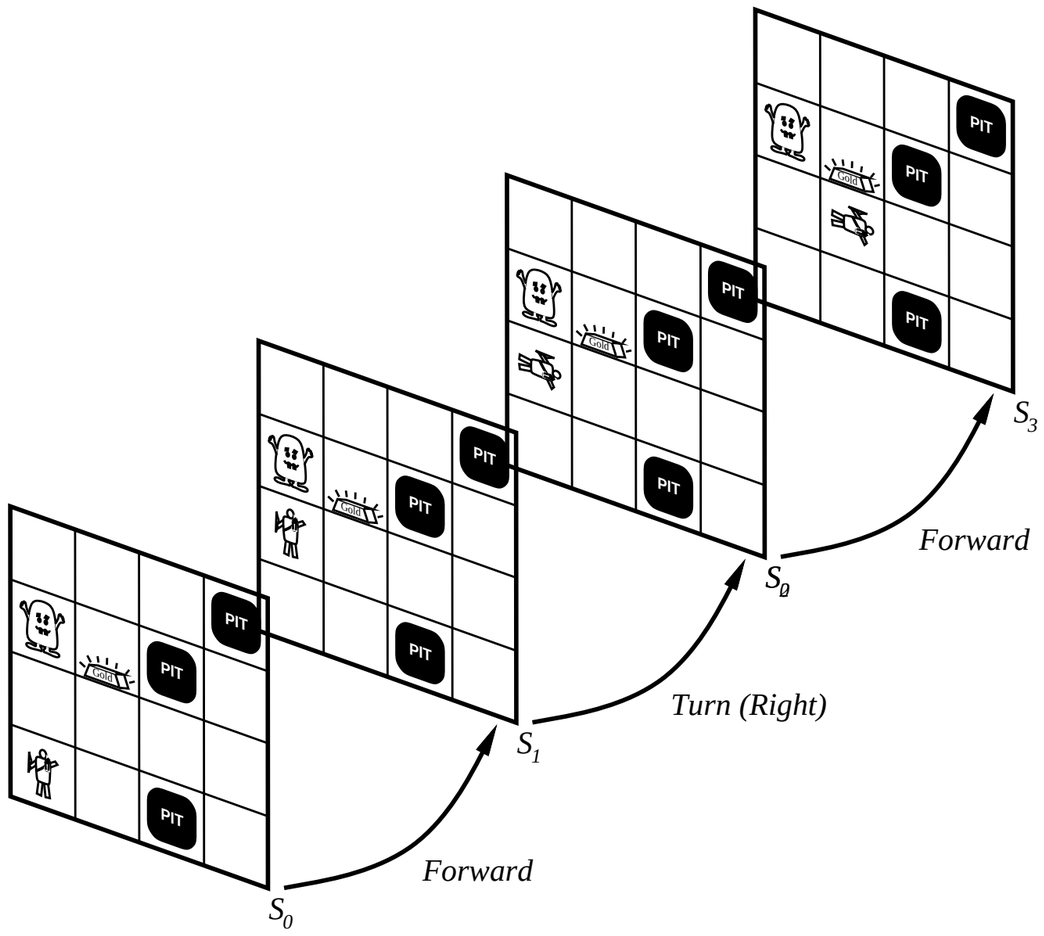
Idee: Um Aktionen, bzw. deren Effekte konsistent zu beschreiben, stellen wir die Welt als Folge von Situationen (Schnappschüsse der Welt) dar. Dazu müssen wir jedes Prädikat um ein zusätzliches Argument erweitern.

Wir benutzen ein Funktionssymbol

result(aktion,situation)

als Term für die Situation die ensteht, wenn in *situation* die Aktion *aktion* ausgeführt wird (**Historie**).

Aktionen: $Turn\_right$, $Turn\_left$, $Foreward$, $Shoot$, $Grab$, $Release$, $Climb$.


PIT
Gold
S_3
Forward
S_0
Turn (Right)
S_1
Forward
S_0

Wir brauchen auch ein Gedächtnis, in Form eines 3-stelligen Prädikates

$$\mathbf{At}(\mathbf{person}, \mathbf{location}, \mathbf{situation})$$

wobei *person* entweder *Wumpus* oder der *Agent* ist und *location* für den momentanen Ort, codiert als Paar [i,j] steht.

Wichtige Axiome sind die sogenannten

"Successor-state Axioms",

die beschreiben, welche Auswirkungen Aktionen auf Situationen haben. Die allgemeine Form dieser Axiome ist

true afterwards $\Longleftrightarrow$ an action made it true
or it is already true and
no action made it false

Axiome über $At(p,l,s)$:

$$At(p,l,result(a,s)) \quad \leftrightarrow \quad ((l = location\_ahead(p,s) \wedge \neg Wall(l) \wedge a = Forward) \vee (At(p,l,s) \wedge \neg a = Forward))$$

$$At(p,l,s) \quad \rightarrow \quad Location\_ahead(p,s) = Location\_toward(l, Orient.(p,s))$$

$$Wall([x,y]) \quad \leftrightarrow \quad (x = 0 \vee x = 5 \vee y = 0 \vee y = 5)$$

$$Location\_toward([x,y],0) = [x+1,y]$$
$$Location\_toward([x,y],90) = [x,y+1]$$
$$Location\_toward([x,y],180) = [x-1,y]$$
$$Location\_toward([x,y],270) = [x,y-1]$$

$$Orient.(Agent,s_0) = 90$$
$$Orient.(p,result(a,s)) = d \quad \leftrightarrow \quad \big((a = turn\_right \wedge d = mod(Orient.(p,s)-90,360))$$
$$\vee(a = turn\_left \wedge d = mod(Orient.(p,s)+90,360))$$
$$\vee(Orient.(p,s) = d \wedge \neg(a = Turn\_right \vee a = Turn\_left))$$

Dabei ist $mod(x,y)$ die eingebaute “modulo”-Funktion, die jedes x auf einen eindeutigen Wert zwischen 0 und y setzt.

Axiome über Wahrnehmungen, sinnvolle neue Begriffe durch Definitionen:

$$
\begin{aligned}
Percept([Stench, b, g, u, c], s) &\rightarrow Stench(s) \\
Percept([a, Breeze, g, u, c], s) &\rightarrow Breeze(s) \\
Percept([a, b, Glitter, u, c], s) &\rightarrow At\_Gold(s) \\
Percept([a, b, g, Bump, c], s) &\rightarrow At\_Wall(s) \\
Percept([a, b, g, u, Scream], s) &\rightarrow Wumpus\_dead(s)
\end{aligned}
$$

$$
\begin{aligned}
At(Agent, l, s) \wedge Breeze(s) &\rightarrow Breezy(l) \\
At(Agent, l, s) \wedge Stench(s) &\rightarrow Smelly(l)
\end{aligned}
$$

$$
\begin{array}{lcl}
Adjacent(l_1,l_2) & \leftrightarrow & \exists d\ l_1 = Location\_toward(l_2,d) \\
Smelly(l_1) & \rightarrow & \exists l_2\ At(Wumpus,l_2,s) \wedge (l_2 = l_1 \vee Adjacent(l_1,l_2))
\end{array}
$$

$$
\begin{array}{lcl}
Percept([none,none,g,u,c],s) \wedge At(Agent,x,s) \wedge Adjacent(x,y) & \rightarrow & OK(y) \\
(\neg At(Wumpus,x,t) \wedge \neg Pit(x)) & \rightarrow & OK(y)
\end{array}
$$

$$
\begin{array}{lcl}
At(Wumpus,l_1,s) \wedge Adjacent(l_1,l_2) & \rightarrow & Smelly(l_2) \\
At(Pit,l_1,s) \wedge Adjacent(l_1,l_2) & \rightarrow & Breezy(l_2)
\end{array}
$$

Axiome zur Beschreibung von Aktionen:

$$
\begin{aligned}
Holding(Gold, result(Grab, s)) &\leftrightarrow (At\_Gold(s) \vee Holding(Gold, s)) \\
Holding(Gold, result(Release, s)) &\leftrightarrow \Box \\
Holding(Gold, result(Turn\_right, s)) &\leftrightarrow Holding(Gold, s) \\
Holding(Gold, result(Turn\_left, s)) &\leftrightarrow Holding(Gold, s) \\
Holding(Gold, result(Forward, s)) &\leftrightarrow Holding(Gold, s) \\
Holding(Gold, result(Climb, s)) &\leftrightarrow Holding(Gold, s)
\end{aligned}
$$

Alle Effekte müssen sorgsam beschrieben werden.

Axiome die Präferenzen zwischen Aktionen beschreiben:

$$
\begin{array}{lcl}
Great(a,s) & \rightarrow & Action(a,s) \\
(Good(a,s) \wedge \neg\exists b\, Great(b,s)) & \rightarrow & Action(a,s) \\
(Medium(a,s) \wedge \neg\exists b\, (Great(b,s) \vee Good(b,s))) & \rightarrow & Action(a,s)
\end{array}
$$

$$
\begin{array}{lcl}
At(Agent,[1,1],s) \wedge Holding(Gold,s) & \rightarrow & Great(Climb,s) \\
At\_Gold(s) \wedge \neg Holding(Gold,s) & \rightarrow & Great(Grab,s) \\
At(Agent,l,s) \wedge \neg Visited(Location\_ahead(Agent,s)) \wedge & & \\
\wedge OK(Location\_ahead(Agent,s)) & \rightarrow & Good(Forward,s)
\end{array}
$$

$$
Visited(l) \quad \leftrightarrow \quad \exists s\, At(Agent,l,s)
$$

Eigentlich geht es nicht nur darum das Gold zu finden, sondern auch wieder lebend zurückzukehren. Man braucht also noch Axiome wie
$Holding(Gold,s) \rightarrow Go\_back(s)$.

Wichtige Probleme der Wissensrepräsentation

Es haben sich 3 sehr wichtige Repräsentationsprobleme bei der Axiomatisierung einer sich ändernden Welt herausgestellt:

Frame-Problem: **Aktionen ändern meist nur sehr wenig. Man braucht sehr viele Aktionen um invariante Eigenschaften zu beschreiben.**

Ideal wäre es, nur was sich **nicht ändert** zu axiomatisieren und dann eine Aussage der Form “Sonst ändert sich nichts” hinzuzunehmen.

Qualification-Problem: **Das Aufzählen aller Bedingungen, unter denen eine Aktion Erfolg hat, ist nötig.** Z.B.

$$\forall x \quad (\textbf{Vogel}(x) \quad \wedge\neg\textbf{Pinguin}(x) \wedge \neg\textbf{Tot}(x)\wedge$$
$$\wedge\neg\textbf{Strauss}(x) \wedge \neg\textbf{Flügel\_gebr.}(x)\wedge$$
$$\wedge\ldots$$
$$\longrightarrow \textbf{Fliegt}(x)$$

Ideal wäre es, nur “Normalerweise fliegen Vögel” aufzuschreiben.

Ramification-Problem: **Wie sollen implizite Konsequenzen von Aktionen gehandhabt werden?** Z.B $Grab(Gold)$ *Gold* kann radioaktiv verseucht sein. Die Aktion $Grab(Gold)$ ist dann jedenfalls nicht optimal.

Programmieren versus Knowledge Engineering.

Programmieren	Knowledge Engineering
Wähle Prog.-sprache	Wähle **Logik**
Schreibe Programm	Definiere **Knowledge Base**
Schreibe Compiler	Implementiere **Kalkül**
Arbeite Programm ab	Leite neue **Fakten** ab

References

Apt, K., H. Blair, and A. Walker (1988). Towards a Theory of Declarative Knowledge. In J. Minker (Ed.), *Foundations of Deductive Databases and Logic Programming*, pp. 89–148. Washington DC: Morgan Kaufmann.

Arens, Y., C. Y. Chee, C.-N. Hsu, and C. Knoblock (1993). Retrieving and Integrating Data From Multiple Information Sources. *International Journal of Intelligent Cooperative Information Systems 2*(2), 127–158.

Bayardo, R., et al. (1997). Infosleuth: Agent-based Semantic Integration of Information in Open and Dynamic Environments. In J. Peckham (Ed.), *Proceedings of ACM SIGMOD Conference on Management of Data*, Tucson, Arizona, pp. 195–206.

Brink, A., S. Marcus, and V. Subrahmanian (1995). Heterogeneous Multimedia Reasoning. *IEEE Computer 28*(9), 33–39.

Chawathe, S., et al. (1994, October). The TSIMMIS Project: Integration of Heterogeneous Information Sources. In *Proceedings of the 10th Meeting of the Information Processing Society of Japan*, Tokyo, Japan. Also available via

anonymous FTP from host db.stanford.edu, file /pub/chawathe/1994/tsimmis-overview.ps.

Genesereth, M. R. and S. P. Ketchpel (1994). Software Agents. *Communications of the ACM 37*(7), 49–53.

Rogers Jr., H. (1967). *Theory of Recursive Functions and Effective Computability*. New York: McGraw-Hill.

Wiederhold, G. (1993). Intelligent Integration of Information. In *Proceedings of ACM SIGMOD Conference on Management of Data*, Washington, DC, pp. 434–437.

Wilder, F. (1993). *A Guide to the TCP/IP Protocol Suite*. Artech House.