# Einführung in die
# Künstliche Intelligenz
## SS09 - Prof. Dr. J. Fürnkranz

TECHNISCHE UNIVERSITÄT DARMSTADT

**Beispiellösung für das 1. Übungsblatt (28.04.2009)**

### Aufgabe 1 Staubsauger-Agent

a) 1. Bei dem beschriebenen Agenten handelt es sich um einen simplen reflexbasierten Agenten, der anscheinend sinnvoll seine Aufgabe verrichtet. Im Gegensatz zu dem Agenten aus der Vorlesung, stößt es jedoch immer gegen eine Wand bevor es sich umdreht und in den nächsten Raum wandert. Auch mithilfe von Sensor 3 ist es nicht möglich, dies zu vermeiden. Die Schwierigkeit liegt darin, dass die Aktion *Left* bzw. *Right* bei diesem Agenten durch zwei sequentielle Aktionen abgebildet werden muss (*drehe(180), vorwärts*). Dies ist jedoch bei dem beschriebenen simplen reflexivbasierte Agent nicht möglich, da er nie weiß, welche Aktion er einen Schritt zuvor gemacht hat.

   2. `sensor[2]==wahr --> sauge`
      `sensor[1]==wahr --> drehe(90)`
      `--> vorwärts`

   3. Mit einem deterministischen simplen reflexiven Agenten ist es nicht möglich, alle Räume der 3x3 Welt abzufahren. Man betrachte z.B. den Fall, in dem die Ausgangsposition des Staubsauger-Agenten in einer Ecke liege. Jedoch ist es möglich alle Räume mindestens einmal zu erreichen, in dem man die Drehung zufällig macht.

b)
- **Simple reflex agent**: Der simple reflexive Agent ist nicht in der Lage die bekannten Informationen auszunutzen, da seine Informationen und darauf basierend seine Aktionen lediglich aus Wahrnehmungen hergeleitet werden können. Das heisst, die Informationsverarbeitung des beschriebenen Agenten ist beschränkt auf seine Sensoren und das vorhandene Wissen kann nicht geeignet auf die Sensoren abgebildet werden. Für beide Teilaufgaben wäre das Programm aus 1a) (2.) erweitert um die Abfrage (`sensor[3]==wahr -> abschalten`) eine sinnvolle Lösung.
- **Model-based reflex agent**: Der Model-based Agent besitzt eine interne Weltrepräsentation. Darin kann vorab die bekannte Welt abgebildet werden und explizit eine Aktionsfolge programmiert werden. Für (1) ist es also möglich ein Programm zu implementieren, dass die minimale Aktionsfolge (`drehe(90), vorwärts, sauge, drehe(180), vorwärts, abschalten`) zurückgibt. Für (2) wird es etwas komplizierter. Um für beliebige 2×2 Welten den schnellsten Staubsauger Agenten zu implementieren, müsste man für alle möglichen Startpositionen (4), Startausrichtungen (4), Verschmutzungsverteilungen ($2^4$) explizite Aktionsfolgen implementieren. Also für insgesamt $4*4*2^4 = 256$ verschiedene Weltkonfigurationen müsste eine explizite Lösung einprogrammiert werden (Durch Ausnutzung einiger Symmetrieeigenschaften könnte man wahrscheinlich die Anzahl etwas verringern).
- **Goal-based agent**: Im Gegensatz zum Model-based Agenten sieht das Konzept des Zielbasierten Agenten vor, dass es zukünftige Aktionen und deren Konsequenzen verarbeiten kann. Somit kann der zielbasierte Agent selber einen Plan erstellen, das ein bestimmtes Performanz-Maß maximiert/minimiert. Das hat hier den Vorteil, dass das Programm des zielbasierteren Agenten für diese Aufgabe sehr kurz sein kann. Das Programm kann im Prinzip identisch für (1) und (2) sein, das intern ein Suchalgorithmus einsetzt, um die beste Aktionsfolge zu bestimmen.

c) 1. *Hinweis: Die Aufgabe war leider etwas ungenau formuliert. Es sollte angenommen werden, dass die Räume ausreichend langsam schmutzig werden. Ein Beispiel: wenn der Agent $x$ Zeiteinheiten benötigt, um die Welt abzufahren und zu säubern, werden die Räume im Mittel erst nach $2 \cdot x$ Zeiteinheiten schmutzig.*

   Ein rationaler Agent würde versuchen die Welt abzufahren und sich dabei merken, wo er gegen eine Wand gefahren ist (dabei saugt er natürlich dreckige Räume). Zu einem gewissen Zeitpunkt hat er sich einen vollständigen Raumplan in seiner internen Weltrepräsentation hergeleitet, so dass er nie mehr gegen eine Wand fährt. Ab diesen Zeitpunkt würde er bis in alle Ewigkeit die Räume traversieren und säubern.

   Obwohl zunächst das Fahren gegen eine Wand stark bestraft wird, wird es bei dem beschriebenen Agenten nur einmal vorkommen. Auf lange Sicht gesehen relativiert sich diese Bestrafung. Da hier die Zeit nicht beschränkt wurde und die Pluspunkte pro sauberes Zimmer sich ewig aufsummieren werden, ist diese Herangehensweise im Allgemeinen *rationaler* als bewusst vorsichtig einen kleinen Teil der Welt zu säubern.

   Ein Beispiel dafür, dass es wichtig war eine genauere Angabe über die Verschmutzungsfrequenz zu machen, verdeutlicht folgendes Extrembeispiel: Wenn der Raum sofort schmutzig wird, (d.h. nach der Säuberung eines Raumes ist es in der nächsten Zeiteinheit wieder schmutzig) wäre es rational nichts zu tun, oder (äquivalent dazu) ständig den aktuellen Raum zu säubern. Aber unrational wäre es eine vörwärts Bewegung zu tätigen, da das Risiko besteht gegen eine Wand zu fahren.

2. Auch für diese Teilaufgabe ist es wichtig zu wissen, wie schnell die Räume schmutzig werden. Falls sie wieder *ausreichend langsam* schmutzig werden, ist die selbe Strategie wie aus (1) rational. Sei $n$ die Anzahl der Räume und wir gehen von dem worst-case aus, dass alle Räume zu Beginn schmutzig seien. Bis wir alle Räume gesäubert haben seien $x$ Zeiteinheiten verstrichen, in denen wir $x + x - 1 + x - 2 + \cdots + 0$ Minuspunkte aber auch zugleich $0+1+\cdots+x-1+x$ Pluspunkte für die Räume erhalten haben. Somit haben sich bis jetzt *nur* die negativen Aktionskosten von $-2 \cdot x$ aufsummiert (wir ignorieren hier die Wandkosten aus Einfachheitsgründen). Hier kann man sich nun wieder eine ausreichend lange *Verschmutzungsfrequenz* überlegen, so dass der Agent von einen positivem Erwartungswert ausgehen kann.

## Aufgabe 2 Modellierung

a) **Ketten und Zustände**

| | | |
|---|---|---|
| Kette: | $(f,l,s)$ | $f$ Form ($linie/kreis$),<br>$l$ Länge,<br>$s$ Status ($auf/zu$) |
| Zustand: | $\{(n,f,l,s),(n',f',l',s'),\dots\}$ | $n$ Anzahl vorhandener $(f,l,s)$-Ketten |

**Operatoren**

| | |
|---|---|
| AUFMACHEN$(f,l)$: | verwandelt eine Kette $(f,l,$zu$)$ in zwei Ketten $(linie,l-1,zu)$ und $(linie,1,auf)$ |
| VERBINDEN$(l,l',f)$: | verwandelt drei Ketten $(linie,l,zu)$, $(linie,l',zu)$ und $(linie,1,auf)$ in eine Kette $(f,l+l'+1,zu)$ |

Merke: $(f,l,s)$ mit $l=0$ ist eine leere Kette!

b)

| | |
|---|---|
| Ausgangszustand: | $\{(4,linie,3,zu)\}$ |
| Zielzustand: | $\{(1,kreis,12,zu)\}$ |

c)

**Lösungspfad**

$$
\begin{array}{ll}
\{(4, linie, 3, zu)\} & \\
\downarrow & \text{AUFMACHEN}(linie, 3) \\
\{(3, linie, 3, zu), (1, linie, 2, zu), (1, linie, 1, auf)\} & \\
\downarrow & \text{AUFMACHEN}(linie, 2) \\
\{(3, linie, 3, zu), (1, linie, 1, zu), (2, linie, 1, auf)\} & \\
\downarrow & \text{AUFMACHEN}(linie, 1) \\
\{(3, linie, 3, zu), (3, linie, 1, auf)\} & \\
\downarrow & \text{VERBINDEN}(3, 3, linie) \\
\{(1, linie, 7, zu), (1, linie, 3, zu), (2, linie, 1, auf)\} & \\
\downarrow & \text{VERBINDEN}(7, 3, linie) \\
\{(1, linie, 11, zu), (1, linie, 1, auf)\} & \\
\downarrow & \text{VERBINDEN}(11, 0, kreis) \\
\{(1, kreis, 12, zu)\} &
\end{array}
$$

## Aufgabe 3 Suchalgorithmen

Da der graphische Suchbaum zu groß ist, werden die einzelnen Iterationen der Suchalgorithmen in den folgenden Aufgaben schriftlich aufgezählt. Dabei wird angegeben welcher aktueller Knoten evaluiert wird und welche Knoten bei der Expansion in welcher Reihenfolge in die *fringe* eingereiht werden.

a) ohne *closed-list*

1. Breadth-First
   expandiere *wswwss_* : *wsw_ssw* , *wsww_ss* , *wswws_s*
   expandiere *wsw_ssw* : *_swwssw* , *w_wsssw* , *ws_wssw* , *wsws_sw* , *wswss_w* , *wswwss_*
   expandiere *wsww_ss*: *w_wwsss* , *ws_wwss* , *wsw_wss* , *wswws_s* , *wswwss_*
   expandiere *wswws_s*: *ws_wsws* , *wsw_sws* , *wsww_ss* , *wswwss_*
   expandiere *_swwssw*: *s_wwssw* , *ws_wssw* , *wsw_ssw*
   expandiere *w_wsssw*: *_wwsssw* , *ww_ssssw* , *wsw_ssw* , *wsws_sw*
   expandiere *ws_wssw*: *_swwssw* , *w_swssw* , *wsw_ssw* , *wssw_sw* , *wssws_w*
   expandiere *wsws_sw*: *w_wsssw* , *ws_swsw* , *wsw_ssw* , *wswss_w* , *wswsws_*
   expandiere *wswss_w*: *ws_ssww* , *wsw_ssw* , *wsws_sw* , *wswssw_*
   expandiere *wswwss_*: *wsw_ssw* , *wsww_ss* , *wswws_s*
   expandiere *w_wwsss*: *_wwwsss* , *ww_wsss* , *www_sss*
   alle auf Tiefe 2 noch expandieren und dann nochmal alle auf Tiefe 3 bis zum Zielknoten, dann kann man erst abbrechen.
2. Uniform-Cost
   expandiere *wswwss_*: *wsw_ssw* , *wsww_ss* , *wswws_s*
   und sortiere nach Kosten (in Expandierungsreihenfolge): *wsww_ss*, *wswws_s*, *wsw_ssw*
   expandiere *wsww_ss*: *w_wwsss*, *ws_wwss*, *wsw_wss*, *wswws_s*, *wswwss_*
   expandiere *wswws_s*: *ws_wsws*, *wsw_sws*, *wsww_ss*, *wswwss_*
   expandiere *wsw_ssw*: *_swwssw*, *w_wsssw*, *ws_wssw*, *wsws_sw*, *wswss_w*, *wswwss_*
   und sortiere nach Kosten:
   *ws_wwss*, *wsw_wss*, *wswws_s*, *wswwss_*, *wsw_sws*, *wsww_ss*, *wswwss_*, *w_wwsss*, *ws_wsws*, *w_wsssw*, *ws_wssw*, *wsws_sw*, *wswss_w*, *_swwssw*, *wswwss_*
   expandiere *ws_wwss*: *_swwwss*, *w_swwss*, *wsw_wss*, *wsww_ss*, *wsssww_s*
   expandiere *wsw_wss*: *_swwwss*, *w_wswss*, *ws_wwss*, *wsww_ss*, *wswsw_s*, *wswsws_*
   expandiere *wswws_s*: *ws_wsws*, *wsw_sws*, *wsww_ss*, *wswwss_*
   expandiere *wswwss_*: *wsw_ssw*, *wsww_ss*, *wswws_s*
   expandiere *wsw_sws*: *_swwsws*, *w_wsssws*, *ws_wsws*, *wswssw_*, *wswws_s*, *wsws_ws*
   expandiere *wsww_ss*: *w_wwsss*, *ws_wwss*, *wsw_wss*, *wswws_s*
   expandiere *wswwss_*: *wsw_ssw*, *wsww_ss*, *wswws_s*
   expandiere *w_wwsss*: *_wwwsss*, *ww_wsss*, *www_sss*
3. Depth-First
4. Iterative-Deepening
   $l = 0$: *wswwss_*
   $l = 1$: expandiere *wswwss_* auf Tiefe 0: *wsw_ssw*, *wsww_ss*, *wswws_s* (Tiefe 1)

$l = 2$: expandiere *wswwss_* auf Tiefe 0:
expandiere *wsw_ssw* auf Tiefe 1: *_swwssw*, *w_wsssw*, *ws_wssw*, *wsws_sw*, *wswss_w*, *wswwss_* (Tiefe 2)
expandiere *wsww_ss* auf Tiefe 1: *w_wwsss*, *ws_wwss*, *wsw_wss*, *wswws_s*, *wswwss_* (Tiefe 2)
expandiere *wswws_s* auf Tiefe 1: *ws_wsws*, *wsw_sws*, *wsww_ss*, *wswwss_* (Tiefe 2)
$l = 3$: expandiere *wswwss_* auf Tiefe 0:
expandiere *wsw_ssw* auf Tiefe 1:
expandiere *_swwssw* auf Tiefe 2: *s_wwssw*, *ws_wssw*, *wsw_ssw* (Tiefe 3)
expandiere *w_wsssw* auf Tiefe 2: *_wwsssw*, *ww_ssswv*, *wsw_ssw*, *wsws_sw* (Tiefe 3)
expandiere *ws_wssw* auf Tiefe 2: *_swwssw*, *w_swssw*, *wsw_ssw*, *wssw_sw*, *wssws_w* (Tiefe 3)
expandiere *wsws_sw* auf Tiefe 2: *w_wsssw*, *ws_swsw*, *wsw_ssw*, *wswss_w*, *wswsws_* (Tiefe 3)
expandiere *wswss_w* auf Tiefe 2: *ws_sssww*, *wsw_ssw*, *wsws_sw*, *wswssw_* (Tiefe 3)
expandiere *wswwss_* auf Tiefe 2: *wsw_ssw*, *wsww_ss*, *wswws_s* (Tiefe 3)
expandiere *wsww_ss* auf Tiefe 1:
expandiere *w_wwsss* auf Tiefe 2: *_wwwsss*, *ww_wsss*, *www_sss* (Tiefe 3)

b) mit *closed list*, alle blau markierten Knoten würden in der nächsten Iteratition nicht expandiert werden, da sie bereits in der *closed list* enthalten sind oder im Laufe der aktuellen Iteration eingefügt werden.

1. Breadth-First
   Tiefe 0:
   expandiere *wswwss_* : *wsw_ssw* , *wsww_ss* , *wswws_s*
   Tiefe 1:
   expandiere *wsw_ssw* : *_swwssw* , *w_wsssw* , *ws_wssw* , *wsws_sw* , *wswss_w* , *wswwss_*
   expandiere *wsww_ss*: *w_wwsss* , *ws_wwss* , *wsw_wss* , *wswws_s* , *wswwss_*
   expandiere *wswws_s*: *ws_wsws* , *wsw_sws* , *wsww_ss* , *wswwss_*
   Tiefe 2:
   expandiere *_swwssw*: *s_wwssw* , *ws_wssw* , *wsw_ssw*
   expandiere *w_wsssw*: *_wwsssw* , *ww_ssswv* , *wsw_ssw* , *wsws_sw*
   expandiere *ws_wssw*: *_swwssw* , *w_swssw* , *wsw_ssw* , *wssw_sw* , *wssws_w*
   expandiere *wsws_sw*: *w_wsssw* , *ws_swsw* , *wsw_ssw* , *wswss_w* , *wswsws_*
   expandiere *wswss_w*: *ws_sssww* , *wsw_ssw* , *wsws_sw* , *wswssw_*
   expandiere *w_wwsss*: *_wwwsss* , *ww_wsss* , *www_sss*
   alle auf Tiefe 2 noch expandieren und dann nochmal alle auf Tiefe 3 bis zum Zielknoten, dann kann man erst abbrechen.
2. Uniform-Cost
   Tiefe 0:
   expandiere *wswwss_*: *wsw_ssw* , *wsww_ss* , *wswws_s*
   und sortiere nach Kosten (in Expandierungsreihenfolge): *wsww_ss*, *wswws_s*, *wsw_ssw*
   Tiefe 1:
   expandiere *wsww_ss*: *w_wwsss*, *ws_wwss*, *wsw_wss*, *wswws_s*, *wswwss_*
   expandiere *wswws_s*: *ws_wsws*, *wsw_sws*, *wsww_ss*, *wswwss_*
   expandiere *wsw_ssw*: *_swwssw*, *w_wsssw*, *ws_wssw*, *wsws_sw*, *wswss_w*, *wswwss_*
   und sortiere nach Kosten:
   *ws_wwss*, *wsw_wss*, *wswws_s*, *wsw_sws*, *w_wwsss*, *ws_wsws*, *w_wsssw*, *ws_wssw*, *wsws_sw*, *wswss_w*, *_swwssw*, *wswwss_*
   Tiefe 2:
   expandiere *ws_wwss*: *_swwwss*, *w_swwss*, *wsw_wss*, *wsww_ss*, *wsssww_s*
   expandiere *wsw_wss*: *_swwwss*, *w_wswss*, *ws_wwss*, *wsww_ss*, *wswsw_s*, *wswsws_*
   expandiere *wswws_s*: *ws_wsws*, *wsw_sws*, *wsww_ss*, *wswwss_*
   expandiere *wsw_sws*: *_swwsws*, *w_wsssws*, *ws_wsws*, *wswssw_*, *wswws_s*, *wsws_ws*
   expandiere *w_wwsss*: *_wwwsss*, *ww_wsss*, *www_sss*
3. Depth-First
4. Iterative-Deepening
   zu beachten ist hier, dass man in jedem Limit wieder eine neue *closed list* verwenden muss
   $l = 0$: *wswwss_*
   $l = 1$: expandiere *wswwss_* auf Tiefe 0: *wsw_ssw*, *wsww_ss*, *wswws_s* (Tiefe 1)
   $l = 2$: expandiere *wswwss_* auf Tiefe 0:

expandiere *wsw_ssw* auf Tiefe 1: *_swwssw*, *w_wsssw*, *ws_wssw*, *wsws_sw*, *wswss_w*, *wswwss_* (Tiefe 2)
expandiere *wsww_ss* auf Tiefe 1: *w_wwsss*, *ws_wwss*, *wsw_wss*, *wswws_s*, *wswwss_* (Tiefe 2)
expandiere *wswws_s* auf Tiefe 1: *ws_wsws*, *wsw_sws*, *wsww_ss*, *wswwss_* (Tiefe 2)
$l = 3$: expandiere *wswwss_* auf Tiefe 0:
expandiere *wsw_ssw* auf Tiefe 1:
expandiere *_swwssw* auf Tiefe 2: *s_wwssw*, *ws_wssw*, *wsw_ssw* (Tiefe 3)
expandiere *w_wsssw* auf Tiefe 2: *_wwsssw*, *ww_sssw*, *wsw_ssw*, *wsws_sw* (Tiefe 3)
expandiere *ws_wssw* auf Tiefe 2: *_swwssw*, *w_swssw*, *wsw_ssw*, *wssw_sw*, *wssws_w* (Tiefe 3)
expandiere *wsws_sw* auf Tiefe 2: *w_wsssw*, *ws_swsw*, *wsw_ssw*, *wswss_w*, *wswsws_* (Tiefe 3)
expandiere *wswss_w* auf Tiefe 2: *ws_ssww*, *wsw_ssw*, *wsws_sw*, *wswssw_* (Tiefe 3)
expandiere *wsww_ss* auf Tiefe 1:
expandiere *w_wwsss* auf Tiefe 2: *_wwwsss*, *ww_wsss*, *www_sss* (Tiefe 3)