Reinforcement Learning unter Benutzung von Hintergrundwissen und Revisionstechniken mit Anwendung auf interaktive Objekterkennung

Diplomarbeit von: Thomas Leopold

Gutachter:

- Prof. Dr. Gabriele Kern-Isberner (Lehrstuhl 6)
- Dr. Gabriele Peters (Lehrstuhl 7)

Endversion vom: 2007-05-28

Hiermit erkläre ich, die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Literaturhilfsmittel benutzt zu haben. Diese Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Inhaltsverzeichnis

Inhaltsverzeichnis 4
Verzeichnis der Abkürzungen 7
1. Einleitung und Gliederung 8
1.1 Einleitung 8
1.2 Gliederung 9
2. Reinforcement Learning 11
2.1 Grundlagen 11
2.2 Die Komponenten im Detail 13
2.2.1 Die Umwelt 13
2.2.2 Der Agent 16
2.2.3 Das Belohnungssignal 16
2.3 Ansätze für Lernverfahren 21
2.3.1 Markov Decision Processes 21
2.3.2 Value Functions und Bellman Gleichungen 22
2.4 Lösungsverfahren der Dynamischen Programmierung 25
2.4.1 Allgemeines 25
2.4.2 Backward DP 26
2.4.3 Value Iteration 29
2.4.4 Policy Iteration 30
2.5 Monte Carlo Lernverfahren 32
2.5.1 Allgemeines 32
2.5.2 Policy Iteration bei bekannter Zustandsübergangsfunktion 32
2.5.3 Policy Iteration bei unbekannter Zustandsübergangsfunktion 34
2.6 Temporal-Difference Learning und *Q*-Learning 37
2.6.1 Allgemeines 37
2.6.2 *TD*(0) und *Q*-Learning 38
2.6.3 *TD*(λ) und *Q*(λ)-Learning 40
2.7 Vergleich der Verfahren 41
3. Wissen, Geglaubtes und Belief Revision 43
3.1 Grundlagen 43
3.2 Logiken und Inferenzsysteme 45
3.2.1 Klassische Logik 45

3.2.2 Reiter's Default-Logik und Antwortmengenprogrammierung 48
3.3 Belief Revision – Theorie 51
3.3.1 Knowledge Bases, Belief Sets und Belief States 51
3.3.2 AGM-Theorie 52
3.3.3 Revision von Formeln 54
3.3.4 Revision von Belief States 56
3.4 Belief Revision – Praxis: Revision von Belief Sets 58
3.4.1 Revision mittels Selektionsfunktionen 58
3.4.2 Revision mittels epistemischen Verwurzelungsrelationen oder -funktionen auf Formeln 59
3.4.3 Revision mittels minimalen Modellen 61
3.5 Belief Revision – Praxis: Revision von Belief States 62
3.5.1 Ordinale konditionale Rangfunktionen (OCFs) 62
3.5.2 Revision von OCFs mit propositionaler Information 65
3.5.3 Konditionale 67
3.5.4 Revision von OCFs mit konditionaler Information 70
3.5.5 OCFs für mehrwertige Variablen 73
4. Zwei-Level Lern-Ansätze 77
4.1 Vorüberlegungen basierend auf psychologischen Erkenntnissen 77
4.2 Lern-Modell CLARION 79
4.3 Sphinx-Learning 81
4.3.1 Grundlagen 81
4.3.2 Das Zustandssignal 84
4.3.3 Aufbau von κ und Wahl einer Aktion 88
4.3.4 Der Zustandswechsel 91
4.3.5 Aktualisierung der QTable und Revision von κ 91
4.3.6 Praxisnahe Varianten des Sphinx-Learnings 111
4.3.7 Verwendung von Hintergrundwissen 112
4.3.8 Implementierung 113
5. Analyse 114
5.1 Allgemeines 114
5.2 Gridworld mit Hindernissen und Farben 114
5.2.1 Vorstellung der Umwelt 114
5.2.2 Ergebnisse 116

5.2.3 Interpretation der Ergebnisse 129
5.3 Erkennung geometrischer Objekte 132
5.3.1 Vorstellung der Umwelt 132
5.3.2 Ergebnisse 135
5.3.3 Interpretation der Ergebnisse 141
5.4 Erkennung simulierter Objekte 142
5.4.1 Vorstellung der Umwelt 142
5.4.2 Ergebnisse 149
5.4.3 Interpretation der Ergebnisse 157
6. Ausblick und Resümee 159
6.1 Ausblick 159
6.2 Resümee 160
Anhang 162
A.1 Auflistung einiger der gefundenen Regeln 162
A.1.1 Gridworld mit Hindernissen und Farben 162
A.1.2 Erkennung geometrischer Objekte 168
A.1.3 Erkennung simulierter Objekte 170
Literaturverzeichnis 174

Verzeichnis der Abkürzungen

ASP:	Antwortmengenprogrammierung
bzgl.:	bezüglich
d. h.:	das heißt
DP:	Dynamische Programmierung
engl.:	englisch
excl.:	exklusive
i. A.:	im Allgemeinen
i. d. R.:	in der Regel
incl.:	inklusive
MC:	Monte Carlo
MDP:	Markov Decision Process
n. V.:	nach Voraussetzung
RL:	Reinforcement Learning
S.:	Seite
TD:	Temporal-Difference
u. a.:	unter anderem
usw.:	und so weiter
vs.:	versus
vgl.:	vergleiche
z. B.:	zum Beispiel

1. Einleitung und Gliederung

1.1 Einleitung

Einen Roboter in einer zum Teil bekannten, zum Teil unbekannten Umgebung sinnvoll handeln zu lassen, ist ein Schritt auf dem Weg, eine künstliche, intelligente Lebensform zu schaffen und so einen Menschheits(alb)traum zu erfüllen. Verschiedene Ideen, dieses Zwischenziel zu realisieren, werden schon seit Jahrzehnten entwickelt und sind hauptsächlich im großen Gebiet der 'Artificial Intelligence' zu finden.

Damit sich ein Roboter in seiner Umwelt ähnlich wie ein Mensch zurechtfinden kann, muss er seine Umwelt möglichst gut kennen und wissen, wie sein Handeln oder auch Nicht-Handeln diese beeinflusst. Zwei vielversprechende Ansätze, die zur Zeit meist nur getrennt voneinander existieren, sind das Reinforcement Learning einerseits und die dynamische Aktualisierung von Wissen durch Wahrnehmung und Schlussfolgerung andererseits.

Auf beiden Seiten gibt es Vorteile und Nachteile. So liefert das Reinforcement Learning oft sehr gute Ergebnisse in verschiedenen Typen von Umwelten, doch dauert das Training meistens sehr lange, denn es ist eine Trial-and-Error-Methodik, bei der der Roboter ohne Vorgabe von Verhaltensregeln von Grund auf lernen muss, wie er in welcher Situation agieren muss, damit er sein Ziel erreicht. Dem Roboter wird nur mitgeteilt, *was* sein Ziel ist, aber ihm werden keinerlei Regeln gegeben, die helfen könnten, *wie* das Ziel gefunden werden könnte. Ein häufig genanntes Argument für diesen Ansatz ist, dass er einsetzbar ist, wenn man wirklich keinerlei Ahnung hat, wie der Lösungsweg aussieht, oder dass man den Roboter nicht mit Regeln vorbelasten möchte, die einem als Mensch zwar intuitiv logisch erscheinen, aber die den Roboter zu einem Lösungsweg zwingen, der vielleicht nicht optimal ist. Ein weiterer Nachteil ist, dass für den Roboter jede Erfahrung eine Einzelerfahrung bleibt und es ohne weiteres nicht möglich ist, Erfahrungen zu verallgemeinern und so auch in unbekannten Situationen noch sinnvoll zu handeln. Was aber wäre,

- wenn es eine Methode gäbe, bei der man dem Roboter einige Regeln oder allgemeiner Wissen mitgeben könnte, das ihm, insbesondere am Anfang der Trainingsphase, wo er noch sehr unerfahren ist, hilft, sich für Aktionen zu entscheiden, die für ihn gut sind,
- wenn er dieses Wissen mit zunehmender Erfahrung entsprechend verfeinern und

ändern könnte, so dass Verhaltensregeln entstehen, die er später in unbekannten Situationen anwenden kann und die besser sind als die ursprünglichen, und

- wenn man ihm nachträglich Regeln vorschlagen könnte, die er dann selbstständig sofort zurückweist, weil er sie für falsch hält, aufnimmt, weil sie zu seiner Erfahrung passen oder ausprobiert, weil er noch nicht genug Information hat?

Es wäre eine Erweiterung des Reinforcement Learnings um eine Komponente für kognitive Fähigkeiten.

Wissensverarbeitung und die dazu gehörende Belief Revision haben den Vorteil, dass das Wissen/ das Geglaubte des Roboters relativ klar repräsentiert ist und sich gut zum Planen von Handlungsfolgen eignet, denn er kann darüber 'nachdenken' wie sich seine Umwelt ändert/ reagiert, wenn er sich für diese oder jene Aktion entscheidet. Wissen/ Geglaubtes kann durch neu hinzukommende Information erweitert werden, muss aber beim Erfahren von widersprüchlicher Information geändert werden bzw. die neue Information muss verworfen werden. Genau diese Entscheidung, welches Wissen der Roboter aufgeben soll bzw. wie er sein Wissen ändern soll, damit es nachher möglichst richtig wird, ist schwierig. Wie wäre es, wenn es möglich wäre,

- diese Entscheidung mit Hilfe von Erfahrungswerten zu treffen,
- das Geglaubte regelmäßig an Hand von Erfahrungswerten auf Richtigkeit zu prüfen und
- durch induktives Schlussfolgern aus Einzelerfahrungen allgemeinere Verhaltensregeln zu entdecken und somit neue Information zu generieren?

Ziel dieser Arbeit ist es, eine Verbindung zwischen beiden Gebieten herzustellen, die die jeweiligen Vorteile nutzt, die Schwächen reduziert und somit Synergie-Effekte erzielt. Konkret bedeutet das, dass man den Roboter sowohl mit einem Gedächtnis aus Einzelerfahrungen, als auch mit einer Komponente für kognitive Fähigkeiten ausstattet, die beide Hand-in-Hand arbeiten, nach dem Vorbild des Menschen.

1.2 Gliederung

In Kapitel 2 werden zunächst die Grundlagen und -ideen des Reinforcement Learnings erarbeitet und im Anschluss mehrere Lösungsverfahren vorgestellt und einander gegenübergestellt. Insbesondere wird dadurch das Q- bzw. $Q(\lambda)$-Learning hergeleitet, welches die positiven Eigenschaften der verschiedenen Lernverfahren

vereint.

Kapitel 3 beschäftigt sich mit der Darstellung, Verarbeitung und Revision von Wissen/ Geglaubtem. Es werden die damit verbundenen Probleme beleuchtet und gängige Lösungsansätze studiert.

Anschließend werden in Kapitel 4 beide Bereiche miteinander verzahnt und ein Konzept zur Implementation dieser Verbindung entwickelt und analysiert. Zuvor werden Vorüberlegungen basierend auf psychologischen Erkenntnissen getroffen und eine ähnliche in letzter Zeit entwickelte Methode, welche in diese Richtung zielt, präsentiert.

In Kapitel 5 wird das in Kapitel 4 Entwickelte an Hand von Beispielen/ Experimenten getestet, bewertet und mit der Leistung des reinen Reinforcement-Learnings verglichen.

Im letzten Kapitel (Kapitel 6) wird ein Ausblick für mögliche Weiterentwicklungen gegeben und resümiert.

2. Reinforcement Learning

2.1 Grundlagen

Reinforcement Learning (RL) (auch ‘optimierendes Lernen’ genannt) ist ein Lerntypus maschinellen Lernens, einem Teilbereich der Artificial Intelligence.

Ein RL-System besteht aus folgenden Komponenten:

- einer Umwelt (Environment)
- einem (oder mehreren) Agenten
- einem Belohnungssignal (Reward Signal)

Die Komponente, die in der Einleitung der Anschaulichkeit wegen noch als ‘Roboter’ bezeichnet wurde, wird verallgemeinert und von nun an als ‘Agent’ bezeichnet. Einen Agenten kann man sich als das Gehirn des Roboters vorstellen, das den Sensor-Input analysiert und anschließend den Körper anweist, Aktionen auszuführen.

Zitat 2.1.1:

"An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators." [RuN2003, S. 32]

Die Rolle der Umwelt ist es, dem Agenten eine Plattform bereitzustellen, in der er wahrnehmen und agieren kann. Sie ist das, was man sich intuitiv unter ihr vorstellt, vielleicht mit einer Ausnahme: Der Körper des Roboters gehört nicht zum Agenten, der ja nur das Gehirn ist, sondern zur Umwelt. Diese Betrachtung ist sinnvoll, da der Körper sich nicht immer so verhält, wie der Geist es sich vorstellt, und der Agent lernen muss, wie sich sein Körper verhält, genau wie er lernen muss, wie die restliche Umwelt reagiert.

Charakteristisch für ein RL-System ist das Belohnungssignal. Seine Aufgabe ist, dem Agenten mitzuteilen, ob seine Aktionen gut oder schlecht waren, damit er, wenn er das nächste Mal in die gleiche Situation gerät, weiß, ob er wie zuvor handeln soll oder sich für eine andere Aktion entschließen soll.

Der Ablauf des Lernens ist im Groben der folgende:

1. Die Umwelt befindet sich im Startzustand.
2. Der Agent betrachtet den aktuellen Zustand der Umwelt.
3. Er wählt unter Berücksichtigung seiner bisherigen Erfahrung, die aus vorherigen

Besuchen des aktuellen Zustands stammt, eine Aktion und führt sie aus.

4. Die Umwelt wechselt darauf hin in einen Folgezustand gemäß einer Zustandsübergangsfunktion.
5. Der Agent bekommt mittels des Belohnungssignals eine Belohnung (und verarbeitet diese).
6. Wenn der nun aktuelle Zustand kein Terminal-Zustand (End-Zustand) ist, weiter bei 2.

Definition 2.1.1:

Einen Durchlauf der Punkte 2 bis 5 nennt man ‘Schritt’ (‘Step’).

Definition 2.1.2:

Ein Durchlauf vom Startzustand bis zu einem Terminal-Zustand heißt ‘Episode’. Eine Epsiode besteht also i. d. R. aus mehreren Schritten.

Der Agent lernt/ trainiert, indem er viele Epsioden durchläuft und so das Verhalten seiner Umwelt immer besser kennen lernt.

Beispiel 2.1.1:

Ein klassisches Beispiel ist ein Dame spielender Agent [Sam1959]. Hier könnte man das RL-System wie folgt modellieren:

Der *Zustand* der Umwelt ist die aktuelle Konfiguration der Spielsteine.

An dem Spiel nehmen zwei Agenten teil, entweder ein computergesteuerter und ein menschlicher Agent oder zwei computergesteuerte Agenten, die gegeneinander spielen.

Ein Schritt aus Sicht des Agenten *A* beginnt, wenn er ziehen darf und endet, wenn sein Gegner *B* gezogen hat und er (*A*) seine Belohnung bekommen hat. Eine Episode entspricht einem ganzen Spiel. Das Lernen besteht darin, sehr viele Spiele zu spielen und so seine Umwelt – dazu gehört hier insbesondere die Spielweise des Gegners – kennen zu lernen.

Das Belohnungssignal liefert dem Agenten ein ‘gut’, wenn er gewinnt, und ein ‘schlecht’, wenn er verliert. Während des Spiels liefert es immer ein ‘neutral’ oder, wenn der Agent möglichst schnell gewinnen soll, ein ‘leicht negativ’. Es sei darauf hingewiesen, dass, wenn man den Agenten bestrafen würde, wenn er einen Stein

verliert, man ihn in seiner Handlungsfreiheit einschränken würde, denn vielleicht führt der Weg zum Gewinn darüber, viele Steine zu opfern. Hier wird das bereits in der Einleitung genannte Paradigma deutlich: Nur *was* das Ziel ist, wird dem Agenten mitgeteilt, nicht, *wie* er es erreichen soll.

2.2 Die Komponenten im Detail

2.2.1 Die Umwelt

Die Umwelt liefert dem Agenten ein Zustandssignal (seine Wahrnehmung) und reagiert auf seine Aktionen.

Sei S die Menge aller Zustände, die die Umwelt annehmen kann, (auch Zustandsraum genannt) und $A(s)$ die Menge aller Aktionen, die der Agent ausführen kann, wenn die Umwelt in Zustand s ist. Der Startzustand einer Episode wird mit s_0 bezeichnet. Verschiedene Episoden können verschiedene Startzustände haben.

Die Artificial Intelligence unterscheidet zwischen verschiedenen Typen von Umwelten ([RuN2003, S. 41f], [Rie2002, VE2, Folie 7]), welche sich aus Sicht des RL wie folgt charakterisieren lassen:

- deterministisch vs. indeterministisch: Sei die Umwelt zum Zeitpunkt t in Zustand $s_t = s \in S$. Wenn der Agent eine Aktion $a_t = a \in A(s)$ ausführt, wird die Umwelt in einen Zustand $s_{t+1} = s' \in S$ wechseln. Ist dieser Folgezustand durch das Tupel (s_t, a_t) eindeutig bestimmt, so ist die Umwelt deterministisch. Wenn s_{t+1} allerdings gemäß einer nicht trivialen Wahrscheinlichkeitsverteilung angenommen wird, ist die Umwelt indeterministisch. In diesem Fall kann der Agent nicht davon ausgehen, dass, wenn er im Zustand s jedesmal dieselbe Aktion a wählt, die Umwelt immer in denselben Folgezustand s' wechseln wird.
- diskret vs. kontinuierlich: Bei diskreten Umwelten ist S abzählbar. Insbesondere sind Umwelten mit endlichem Zustandsraum diskret.
- statisch vs. dynamisch: Statische Umwelten ändern ihren Zustand nur durch eine Aktion des Agenten. Dynamische Umwelten – diese können sich auch ohne eine Handlung des Agenten ändern – verhalten sich beim RL aus Sicht des Agenten wie indeterministische Umwelten, denn wenn der Agent längere Zeit nichts tut, die Umwelt sich aber trotzdem ändert, so hat der Agent den Eindruck, dass seine letzte Aktion zu unterschiedlichen Folgezuständen führen kann, da er als

Folgezustand den Zustand wertet, der unmittelbar vor seiner nächsten Aktion präsent ist.

- zugänglich vs. nicht zugänglich: Zugängliche Umwelten können vom Agenten vollständig wahrgenommen werden. Hier ist zwischen der Komponente 'Umwelt' des RL-Systems und der realen Umwelt zu unterscheiden. Die für den Agenten wahrnehmbare Umwelt ist die erstere, da er nur mit ihr in direktem Kontakt steht. Diese teilt sich ihm über das Zustandssignal mit, welches immer vollständig zugänglich ist. Basiert dieses Zustandssignal auf Werten aus der realen Umwelt, so können möglicherweise einige Werte nicht in Erfahrung gebracht werden. Dennoch wird ein konkretes und voll zugängliches Zustandssignal erstellt, welches der Agent liest. I. A. wird in diesem Fall die Umwelt des RL-Systems für den Agenten indeterministisch erscheinen.
- einzelner vs. mehrere Agenten: Es gibt Umwelten, in den genau ein Agent handelt, und welche, in denen mehrere Agenten handeln. Wenn es mehrere Agenten gibt, wird beim RL die Umwelt jedem Agenten i. A. indeterministisch erscheinen, da die anderen Agenten sie ebenfalls verändern und so aus Sicht eines Agenten bei gleichem Ausgangszustand und gleicher Aktion abhängig vom Verhalten der anderen Agenten unterschiedliche Folgezustände eintreten können.

Zudem gibt es beim RL noch einen weiteren Typ, nämlich Umwelten mit endlichem Horizont vs. Umwelten mit unendlichem Horizont. Erstere terminieren, wenn das Ziel erreicht ist, z. B. das Finden eines Ausgangs in einem Labyrinth. Das inverse Pendel-Problem ([RuN2003, S. 780f], [Rie2003, VE 11, Folie 3]) hingegen ist ein typisches Beispiel für eine Umwelt mit unendlichem Horizont, denn selbst wenn das inverse Pendel ausbalanciert ist, müssen immer noch kleine Korrekturen vorgenommen werden, denn ansonsten würde das Pendel doch wieder umfallen.

An dieser Stelle lässt sich ein RL-System wie folgt definieren:

Definition 2.2.1.1:

Ein 'RL-System' (genauer: das Verhalten eines RL-Systems) wird beschrieben durch eine Wahrscheinlichkeitsverteilung:

$$\Pr(s_0 = s, a_0 = a, r_1 = r, s_1 = s', a_1 = a', r_2 = r', \ldots).$$

Hierbei sind $s_0, s_1, \ldots$ die zu den Zeitpunkten $t = 0,1,\ldots$ angenommenen Zustände,

$s, s', \ldots$ Elemente aus S, $a_0, a_1, \ldots$ die zu den Zeitpunkten $t = 0,1,\ldots$ gewählten Aktionen, $a, a', \ldots$ Elemente aus der Menge der im jeweiligen Zustand möglichen Aktionen, $r_1, r_2, \ldots$ die nach dem Zustandswechel erhaltenen Belohnungen und $r, r', \ldots$ reelle Zahlen.

Definition 2.2.1.2:

Unter dem 'Modell des RL-Systems' versteht man die (deterministische oder indeterministische) Zustandsübergangsfunktion, durch die die Umwelt den Folgezustand bestimmt, und das (deterministische oder indeterministische) Belohnungssignal.

Für einige, aber nicht alle, RL-Lernverfahren ist es notwendig, dass der Agent das Modell der Umwelt kennt bzw. zumindest die Zustandsübergangsfunktion kennt.

Von großer Bedeutung ist beim RL eine mögliche Eigenschaft des RL-Systems – die so genannte Markov-Eigenschaft. Diese besagt, dass im deterministischen Fall der Folgezustand bzw. im indeterministischen Fall die Wahrscheinlichkeit für einen bestimmten Folgezustand nur vom aktuellen Zustand und der gewählten Aktion abhängig ist, nicht aber von der Historie des Systems.

Definition 2.2.1.3:

Unter der 'Markov-Eigenschaft' (in der einfachen Variante) versteht man:

$$\Pr\left(s_{t+1} = s' \middle| s_t, a_t\right) = \Pr\left(s_{t+1} = s' \middle| s_t, a_t, s_{t-1}, a_{t-1}, \ldots, \ldots, s_0, a_0\right).$$

Soll die Markov-Eigenschaft erfüllt werden, muss, falls die Historie des Systems von Bedeutung ist, diese mit in das Zustandssignal hineincodiert werden.

Beispiel 2.2.1.1:

Für einen Staub saugenden Roboter ist nicht nur seine aktuelle Position wichtig, sondern auch die Menge aller Positionen, an denen er bereits gesaugt hat. Damit die Markov-Eigenschaft erfüllt ist, könnte man bei n möglichen Positionen den Zustand so modellieren, dass er für jede Position angibt, ob sie bereits gesaugt wurde. Dies führt zu einem 2^n-großen Zustandsraum.

Je nach Autor ist die Markov-Eigenschaft noch ausführlicher definiert. An dieser Stelle reicht die eben gegebene Definition. Die ausführliche Variante wird in

Definition 2.2.3.3 gegeben.

2.2.2 Der Agent

Der Agent nimmt das von der Umwelt kommende Zustandssignal wahr und entscheidet sich anschließend zu einer Aktion. Diese Entscheidung entspricht einer Abbildung.

Definition 2.2.2.1:

Eine 'Policy' ist eine (deterministische oder indeterministische) Abbildung $\pi : S \to A$. Im Zusammenhang mit indeterministischen Policies, gibt die ebenfalls π genannte Funktion $\pi : S \times A \to [0;1]$ die Wahrscheinlichkeit an, dass die Policy π in s Aktion a wählt, d. h.:

$$\pi(s,a) := \Pr(\pi(s) = a).$$

Aufgabe des Agenten ist es, eine möglichst gute Policy zu finden, also zu lernen, wie er sich in der gegebenen Umwelt am besten verhält. Dies geschieht ausschließlich durch Interaktion mit der Umwelt. Hierbei kommt es bei vielen Lernverfahren zu einem Trade-off zwischen Exploration und Exploitation. Einerseits soll der Agent noch nicht erkundete Reaktionsweisen der Umwelt testen (Exploration), andererseits sollte er bereits erworbenes Wissen auch nutzen (Exploitation). In der Praxis wird dieses Problem gelöst, indem der Agent in jedem Schritt mit einer kleinen Wahrscheinlichkeit ε von seiner aktuellen Policy abweichen darf. In dem Fall, dass es sich um eine Umwelt mit endlichem Horizont handelt, kann man mit einem relativ großen ε starten und dieses dann mit wachsender Anzahl erlebter Episoden abkühlen lassen. Dies bewirkt, dass der Agent zu Beginn recht experimentierfreudig ist und mit zunehmendem Alter immer mehr nur das Erlernte anwendet.

2.2.3 Das Belohnungssignal

Das Belohnungssignal ist die Funktion, über die dem Agenten mitgeteilt wird, was gut und was schlecht ist. Sie kann ebenfalls deterministisch oder indeterministisch sein. Gewöhnlich wird sie als Bestandteil der Umwelt angesehen, obwohl man sie auch als eigenständige Komponente ansehen oder sogar dem Agenten zuordnen könnte.

Am Ende jedes Schritts bekommt der Agent eine Belohnung (einen Reward) r_{t+1} für die Wahl der Aktion a_t in Zustand s_t. Die korrekte Modellierung der Belohungsfunktion ist extrem wichtig (siehe Beispiel 2.1.1).

Zitat 2.2.3.1:

"The reward signal is your way of communicating to the agent *what* you want it to achieve, not *how* you want it achieved." [SuB1998, S. 56f]

An dieser Stelle lässt sich das Ziel des Agenten, das bisher nur mit Begriffen wie 'gut' und 'schlecht' umschrieben wurde, deutlich formulieren: Finde eine Policy, welche die erwartete Gesamtbelohnung, d. h. die Summe aller während einer Episode gesammelten Belohnungen, maximiert.

Wie folgendes Beispiel schnell verdeutlicht, reicht eine einfache Greedy-Methode nicht aus, um diese Aufgabe zu erfüllen:

Beispiel 2.2.3.1:

Der Agent *S* hat die Wahl, sich in die Sonne zu legen oder eine Vorlesung zu besuchen. Da er das Sonnenbaden als angenehm empfinden soll, wird er dafür großzügig belohnt. Andererseits soll er den Besuch der Vorlesung als weniger angenehm empfinden, weshalb die Belohnung für diese Aktion geringer ausfällt. Würde *S* rein greedy handeln, würde er sich jeden Tag für das Liegen in der Sonne entscheiden. Nach einiger Zeit bekäme er jedoch negative Belohnungen, da er sich weder Nahrung, Kleidung noch eine Wohnung leisten kann. Am Ende seines Lebens (der Episode) hätte er dann in der Summe sehr wenig Belohnungen angesammelt. Hätte *S* die Vorlesung öfter besucht, hätte er zwar kurzfristig wenig Belohnung erhalten, doch er hätte sich durch seine gewonnenen Fähigkeiten sein Leben finanzieren können und so später mehr Belohnung bekommen.

Dieses Beispiel wirft außerdem zwei allgemeine Probleme auf:

1. Dem Agenten muss ermöglicht werden, lokal im aktuellen Zustand zu erkennen, welche Aktion in Zukunft die größtmögliche Gesamtbelohnung bringen wird.
2. Wenn der Horizont unendlich ist, wird möglicherweise die Summe der Belohnungen ebenfalls unendlich und damit jede Policy aus Sicht des Agenten gleich gut.

Um Lösungsansätze für diese Probleme zu eröffnen, ist es sinnvoll, den Begriff des ‘Returns’ zu definieren.

Definition 2.2.3.1:

Der ‘Return des Zeitpunkts *t*’ ist die Summe der Belohnungen, die der Agent bis zum Ende der laufenden Episode noch bekommen wird:

$$R_t = r_{t+1} + r_{t+2} + \ldots + r_T .$$

Die gängigen Lernverfahren verwenden den Return (in irgendeiner Form), um lokal in Zustand *s* entscheiden zu können, welche Aktion *a* am günstigsten ist. Z. B. könnte man jedem Zustand den Durchschnitt aller von ihm aus erzielten Returns als Wert zuweisen und dann in Zustand s_t diejenige Aktion a_t wählen, die zu dem Zustand mit dem größten Wert, d. h. dem größten durchschnittlichen Return, führt, der von s_t erreichbar ist.

Beispiel 2.2.3.2:

Gegeben sei folgende Umwelt:

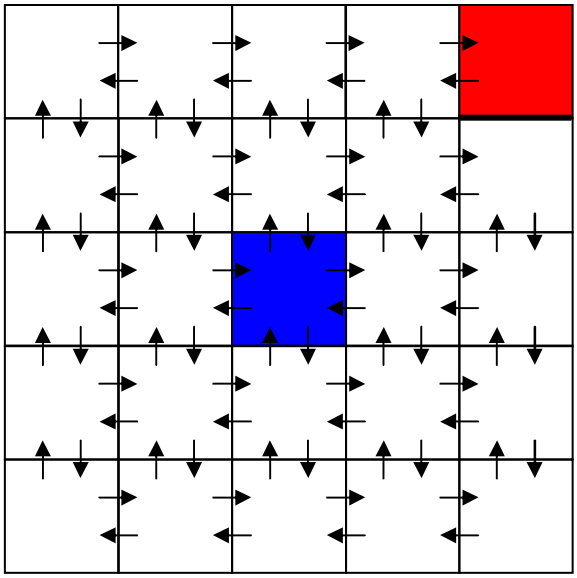


Abbildung 2.2.3.1

Der Zustand der Umwelt ist die Position des Agenten. Zu Beginn befindet sich der Agent auf dem blauen Feld. Seine Aufgabe ist es, in möglichst wenigen Schritten das rote Feld zu erreichen. Erreicht er das rote Feld, endet die Episode. Seine im jeweiligen Feld möglichen Aktionen sind durch ausgehenden Pfeile gekennzeichnet. Zwischen dem roten Feld und dem Feld darunter ist kein Übergang möglich, was in

der Realität einem Hindernis entsprechen würde. Für jeden Schritt bekommt der Agent eine Belohnung von -1 (was einer Strafe entspricht).

In der ersten Episode könnte der Agent den grün markierten Weg wählen:

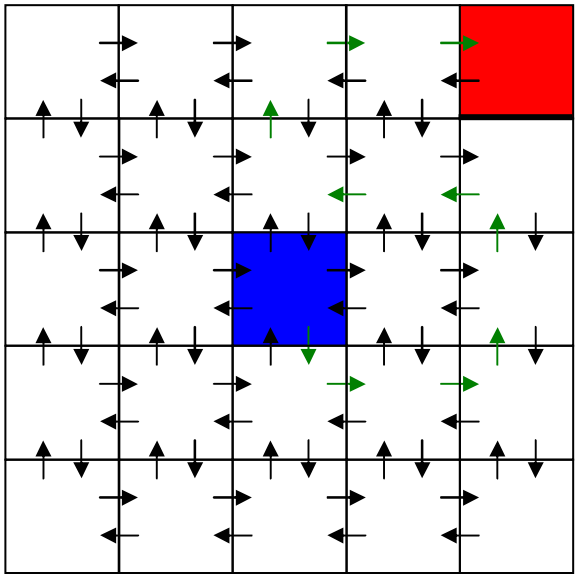


Abbildung 2.2.3.2

Der Return des Zeitpunkts *t* lässt sich dem jeweiligen Feld zuordnen, in dem er sich zu diesem Zeitpunkt befindet.

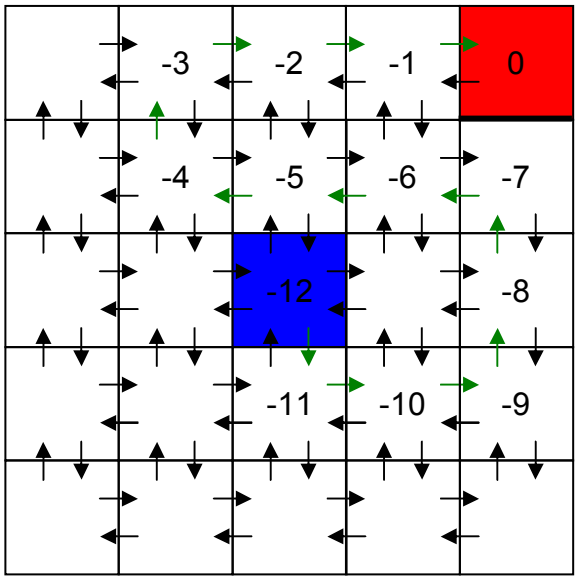


Abbildung 2.2.3.3

(Bemerkung: Auf Grund der Definition von R_t stehen die Return-Werte erst am Ende der Episode fest.)

Werden sehr viele Episoden durchlaufen und die Policy immer weiter verbessert, bis schließlich eine optimale Policy gefunden und oft probiert wurde, so ergibt sich

idealisierter Weise folgendes Bild, wenn im jeweiligen Feld nun nicht mehr der in der letzten Episode erzielte Return eingetragen wird, sondern der Durchschnitt der erzielten Returns über alle Episoden:

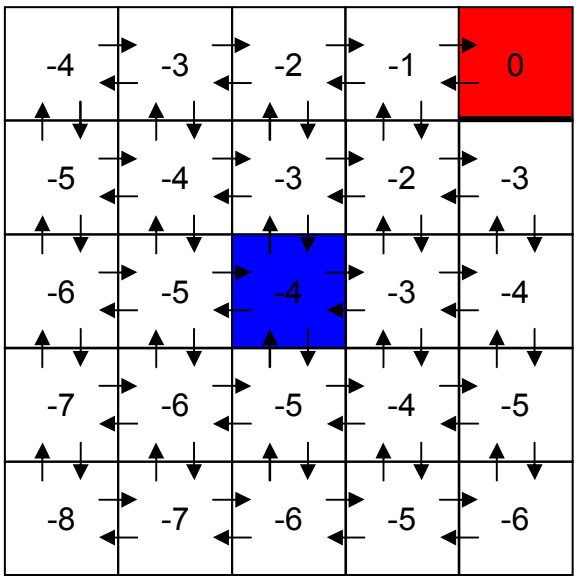


Abbildung 2.2.3.4

Diese Abbildung zeigt, dass sich nun lokal in jedem beliebigen Feld entscheiden lässt, welche Aktion gewählt werden soll – nämlich eine, die in ein Nachbar-Feld führt, dass unter allen Nachbar-Feldern den größten Wert hat, z. B. vom blauen Feld aus 'nach oben' oder 'nach rechts'. (Bemerkung: Da die Zahlen negativ sind, sind die Zahlen mit kleinem Betrag groß.)

Nun zum zweiten oben genannten Problem. Das Problem bei unendlichem Horizont ist, dass Belohnungen, die in ferner Zukunft erhalten werden, genauso wichtig sind, wie Belohnungen, die in naher Zukunft erhalten werden. Deswegen geht man über zum diskontierten Return, der den Einfluss von Belohnungen umso stärker dämpft, je weiter sie in der Zukunft liegen.

Definition 2.2.3.2:

Der '*diskontierte* Return des Zeitpunkts *t*' ist die durch einen Faktor $0 \leq \gamma \leq 1$ diskontierte Summe der Belohnungen, die der Agent zukünftig noch bekommen wird:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots.$$

Wählt man $\gamma = 1$, so erhält man den einfachen Return aus Definition 2.2.3.1. Wählt man $\gamma < 1$, so konvergiert γ^k für $k \to \infty$ gegen 0. Da die Belohnungen so gestaltet

werden, dass sie beschränkt sind, kann die diskontierte Summe nicht mehr unendlich werden, selbst wenn der Horizont unendlich ist. Bei $\gamma = 0$ würde nur die unmittelbar folgende Belohnung r_{t+1} zählen, was für einen greedy handelnden Agenten geeignet wäre.

Da zu Beginn dieses Unterkapitels der Begriff der Belohnung eingeführt worden ist, lässt sich nun die ausführliche Variante der bereits in Definition 2.2.1.3 genannten Markov-Eigenschaft formulieren. Der Unterschied zwischen beiden Varianten ist, dass in der ausführlichen Version zusätzlich gefordert wird, dass das Belohnungssignal nur vom aktuellen Zustand und der gewählten Aktion abhängig sein darf.

Definition 2.2.3.3:

Unter der 'Markov-Eigenschaft' (in der ausführlichen Variante) versteht man:

$$\Pr\left(s_{t+1} = s', r_{t+1} = r \middle| s_t, a_t\right) = \Pr\left(s_{t+1} = s', r_{t+1} = r \middle| s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, \ldots, r_1, s_0, a_0\right).$$

2.3 Ansätze für Lernverfahren

2.3.1 Markov Decision Processes

Definition 2.3.1.1:

Ein 'Markov Decision Process (MDP)' ist ein RL-System, welches die Markov-Eigenschaft erfüllt. Er heißt zudem 'endlich', wenn sowohl Zustands- als auch Aktionsraum endlich sind.

Für die vorgestellten Lernverfahren sollte man die Aufgabe als endlichen MDP formulieren (sofern möglich), damit die nachher genannten Konvergenzaussagen gelten und somit sichergestellt ist, dass das gewünschte Ergebnis auch gefunden wird.

Da die Markov-Eigenschaft Aussagen über die Wahrscheinlichkeitsverteilungen für Folgezustand und Belohnung trifft, erhalten folgende Größen eine abkürzende Schreibweise:

Definition 2.3.1.2:

Die *Wahrscheinlichkeitsverteilung* für den Folgezustand wird bezeichnet mit:

$$P_{ss'}^{a} := \Pr\left(s_{t+1} = s' \middle| s_t = s, a_t = a\right).$$

Definition 2.3.1.3:

Der *Erwartungswert* für die Belohnung wird bezeichnet mit:

$$r_{ss'}^{a} := \mathrm{E}\left(r_{t+1} = r \middle| s_t = s, a_t = a, s_{t+1} = s'\right).$$

Die *Wahrscheinlichkeitsverteilung* für die Belohnung $\Pr\left(r_{t+1} = r \middle| s_t = s, a_t = a, s_{t+1} = s'\right)$ erhält keine abkürzende Schreibweise.

2.3.2 Value Functions und Bellman Gleichungen

Ein Ansatz, der als Basis für viele Lösungsverfahren dient, sind die Value Functions. Diese ordnen einem Zustand s bzw. einem Zustand-Aktion-Paar (s,a) einen Wert zu, der im Wesentlichen eine Näherung des von ihm aus erzielbaren, maximalen erwarteten Returns ist.

Definition 2.3.2.1:

Der Wert eines Zustands in Abhängigkeit von der aktuellen Policy π ist definiert als:

$$V^{\pi}(s) := \sum_{a \in A(s)} \left(\pi(s,a) \sum_{s'} P_{ss'}^{a} \left(r_{ss'}^{a} + \gamma V^{\pi}(s') \right) \right).$$

Diese Gleichung heißt 'Bellmann Gleichung für V^{π}'.

Diese rekursive Gleichung ist sowohl für alle möglichen indeterministischen Fälle (mit deterministischen Fällen als Spezialfällen), als auch für Umwelten mit unendlichem Horizont (mit endlichem Horizont als Spezialfall) geeignet und lässt sich dennoch relativ leicht erklären: Der Teil $r_{ss'}^{a} + \gamma V^{\pi}(s')$ steht für den erwarteten diskontierten Return, der aus Zustand s erzielt wird, wenn Aktion a gewählt wird und s' der Folgezustand ist. Dieser setzt sich nach Definition 2.2.3.2 zusammen aus der erwarteten sofortigen Belohnung $r_{ss'}^{a}$ und der diskontierten Summe der noch folgenden Belohnungen, deren Erwartungswert nach rekursiver Definition der Bellman Gleichung für V^{π} in $V^{\pi}(s')$ steht. Da wegen des Indeterminismus verschiedene Folgezustände s' eintreten können, wird $r_{ss'}^{a} + \gamma V^{\pi}(s')$ mit $P_{ss'}^{a}$ gewichtet und über alle möglichen Folgezustände aufsummiert, so dass

$\sum_{s'} P^a_{ss'}\left(r^a_{ss'} + \gamma V^\pi(s')\right)$ dem erwarteten diskontierten Return entspricht, wenn in *s* die Aktion *a* gewählt wird. Die Policy des Agenten ist i. A. wiederum selber indeterminisch, so dass $\sum_{s'} P^a_{ss'}\left(r^a_{ss'} + \gamma V^\pi(s')\right)$ mit der Wahrscheinlichkeit, dass der Agent *a* in *s* wählt, ($\pi(s,a)$) gewichtet wird und entsprechend aufsummiert wird über alle in *s* möglichen Aktionen. Man erhält $\sum_{a \in A(s)} \left(\pi(s,a) \sum_{s'} P^a_{ss'}\left(r^a_{ss'} + \gamma V^\pi(s')\right)\right)$, den erwarteten Return für Zustand *s*.

Ziel des RL's ist es, eine optimale Policy zu finden. Um mathematisch auszudrücken, wann eine Policy besser als eine andere ist, wird definiert:

Definition 2.3.2.2:

$$\pi' \geq \pi :\Leftrightarrow V^{\pi'}(s) \geq V^\pi(s) \forall s \in S$$

Definition 2.3.2.3:

Eine optimale Policy wird mit 'π^*' bezeichnet und es muss gelten: $\pi^* \geq \pi$ für alle möglichen Policies π.

Analog zur Value Function V^π lässt sich auch eine Value Function Q^π für Zustand-Aktion-Paare (s,a) festlegen:

Definition 2.3.2.4:

Der Wert eines Zustand-Aktion-Paares (s,a), d. h. der Wert der Wahl von Aktion *a* in Zustand *s*, in Abhängigkeit von der aktuellen Policy π ist definiert als:

$$Q^\pi(s,a) := \sum_{s'} P^a_{ss'} \left(r^a_{ss'} + \gamma \sum_{a' \in A(s')} \pi(s',a') Q^\pi(s',a') \right).$$

Diese Gleichung heißt 'Bellmann Gleichung für Q^π'.

Die Motivierung dieser Gleichung erfolgt analog zur Herleitung von V^π.

Wegen der Definition $\pi' \geq \pi :\Leftrightarrow V^{\pi'}(s) \geq V^\pi(s) \forall s \in S$ führt der Weg zu einer optimalen Policy über die optimalen Value Functions V^* bzw Q^*, für die $V^*(s) \geq V^\pi(s) \forall \pi$ bzw. $Q^*(s,a) \geq Q^\pi(s,a) \forall \pi$ gelten soll.

Definition 2.3.2.5:

Die optimale Value Function 'V^*' ist definiert durch:

$$V^*(s) := \max_{\pi} V^{\pi}(s).$$

V^* ist wohldefiniert (d. h. das Maximum existiert) für endliche MDPs, denn wegen der Endlichkeit des Zustands- und des Aktionsraums gibt es auch nur endliche viele Policies.

Weil $V^*(s)$ immer den größtmöglichen Wert (über alle Policies) zugewiesen bekommt, erfüllt das so definierte V^* die gewünschte Eigenschaft $V^*(s) \geq V^{\pi}(s) \forall \pi$. Wieder ergibt sich Q^* auf entsprechende Weise:

Definition 2.3.2.6:

Die optimale Value Function 'Q^*' ist definiert durch:

$$Q^*(s,a) := \max_{\pi} Q^{\pi}(s,a).$$

Q^* ist wohldefiniert für endliche MDPs.

V^* und Q^* stehen in folgender Beziehung zueinander: $V^*(s) = \max_{a \in A(s)} Q^*(s,a)$. Diese Gleichung drückt aus, dass ein Zustand bei Verwendung der optimalen Policy genauso gut ist, wie das Maximum, welches man vom ihm aus durch eine Aktion erreichen kann.

Ziel ist es, eine Policy π^* zu finden, die V^* bzw. Q^* als Value Function hat. Eine solche Policy ist eine optimale Policy, d. h. diese lässt den Agenten in der gegebenen Umwelt optimal handeln. Benutzt ein Agent eine optimale Policy, ist er optimal an seine Umwelt angepasst.

Die Idee zur Konstruktion von π^* ist, π^* in *s* diejenige Aktion wählen zu lassen, die den maximalen erwarteten Return liefert, also

$$\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} P^a_{ss'} \left(r^a_{ss'} + \gamma V^*(s') \right)$$

bzw.

$$\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} P^a_{ss'} \left(r^a_{ss'} + \gamma \max_{a' \in A(s')} Q^*(s', a') \right).$$

Letzteres leitet sich aus Ersterem her wegen der Beziehung zwischen V^* und Q^*.

Diese Überlegungen führen zu den so genannten Bellman Optimalitäts-Gleichungen:

Definition 2.3.2.7:

Die 'Bellman Optimalitäts-Gleichung für V^*' lautet:

$$V^*(s) = \max_{a \in A(s)} \sum_{s'} P^a_{ss'} \left(r^a_{ss'} + \gamma V^*(s') \right).$$

Definition 2.3.2.8:

Die 'Bellman Optimalitäts-Gleichung für Q^*' lautet:

$$Q^*(s, a) = \sum_{s'} P^a_{ss'} \left(r^a_{ss'} + \gamma \max_{a' \in A(s')} Q^*(s', a') \right).$$

Dass die Bellman Optimalitäts-Gleichung für V^* bzw. Q^* konform zu Definition 2.3.2.5 bzw. Definition 2.3.2.6 ist, zeigt [SuB1998, S. 76].

Das Erkennungsmerkmal einer optimalen Policy π^* ist, dass gilt: $V^{\pi^*} = V^*$ bzw. $Q^{\pi^*} = Q^*$.

2.4 Lösungsverfahren der Dynamischen Programmierung

2.4.1 Allgemeines

Dynamische Programmierung (DP) ist als Lösungsverfahren für das RL eher von intellektuellem Reiz, da es in der Praxis schon bei kleinen RL-Aufgaben viel zu lange Rechenzeiten erfordern würde. Die Idee ist es, eine der Bellman Optimalitäts-Gleichungen zu lösen, d. h. Policies mit $V^{\pi^*} = V^*$ bzw. $Q^{\pi^*} = Q^*$ zu finden. Das Problem liegt allerdings darin, dass V^* bzw. Q^* unbekannt sind und erst errechnet werden müssen. Hat man V^* bzw. Q^* bestimmt, sind optimale Policies dann implizit gegeben.

Im Folgenden werden drei DP-Methoden vorgestellt, die diese Aufgabe unter der Voraussetzung, dass es sich um einen endlichen MDP handelt, lösen:

- Backward DP
- Value Iteration
- Policy Iteration

2.4.2 Backward DP

Backward DP ist anwendbar, wenn der Horizont endlich ist und das Modell des RL-Systems bekannt ist. Das Modell lässt sich im rein deterministischen Fall grafisch darstellen, beispielsweise:

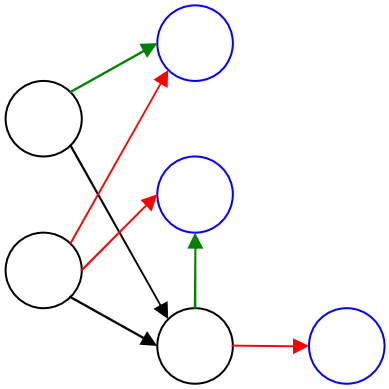


Abbildung 2.4.2.1

In diesem Beispiel repräsentieren Kreise Zustände und Pfeile Aktionen, die von einem Zustand zum Folgezustand führen. Die blauen Kreise stellen die Terminal-Zustände dar. Grüne Pfeile bedeuten eine Belohnung von 1, schwarze Pfeile eine Belohnung von 0 und rote eine von -1.

Da nach Voraussetzung der Horizont endlich ist, existieren Terminal-Zustände und von jedem Zustand aus ist ein Terminal-Zustand in höchstens *N* Schritten erreichbar. Backward DP ermittelt in *N* Iterationen die Funktionswerte $V^*(s)$ für alle Zustände. Zunächst werden alle Funktionswerte mit 0 initialisiert, d. h. $V_0^*(s) := 0 \forall s \in S$. Auf diese Weise haben schon alle Terminal-Zustände den richtigen Wert, denn von Terminal-Zuständen lassen sich keine weitere Belohnungen erwirtschaften, d. h. der Return und damit der Wert ist 0. Grafisch dargestellt sieht diese Initialisierung wie durch folgende Abbildung illustriert aus.

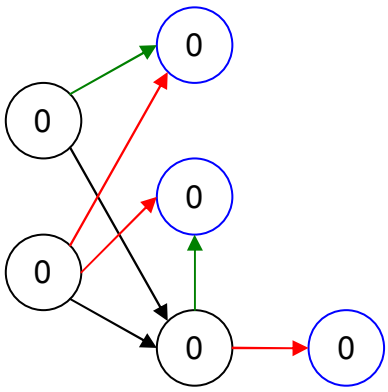


Abbildung 2.4.2.2

Da V^* gesucht ist, wird im Prinzip die Bellman Optimalitäts-Gleichung für V^* verwendet. Wegen des endlichen Horizonts wird eine Diskontierung nicht benötigt, also $\gamma = 1$ und somit gilt: $V^*(s) = \max_{a \in A(s)} \sum_{s'} P^a_{ss'} \left(r^a_{ss'} + V^*(s') \right)$. Backward DP berechnet V^* stufenweise von den Terminal-Zuständen ausgehend nach 'vorne'. Es benutzt die Werte der Berechnungsstufe $i-1$, bei der alle Zustände, deren maximale Entfernung zu den Terminal-Zuständen höchstens $i-1$ ist, schon ihren korrekten Wert $V^*(s)$ haben, um die Werte der Berechnungsstufe i zu erhalten, und orientiert sich dabei an der Bellman Optimalitäts-Gleichung für V^*. Die formale Update-Vorschrift ist also:

$$V^*_i(s) := \max_{a \in A(s)} \sum_{s'} P^a_{ss'} \left(r^a_{ss'} + V^*_{i-1}(s') \right).$$

Hier geht die Voraussetzung 'Modell bekannt' ein, denn sonst wäre $P^a_{ss'}$ oder $r^a_{ss'}$ unbekannt.

Für das Beispiel erhält man nach der ersten Iteration folgende Werte $V^*_1(s)$:

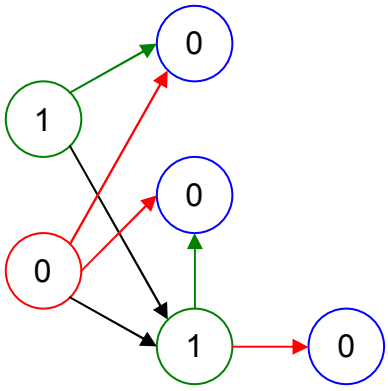


Abbildung 2.4.2.3

Nun ($i = 1$) haben bereits alle Zustände, die durch *höchstens* eine Aktion zu Terminal-Zuständen führen (hier grün markiert), den richtigen Wert. Der rot

gekennzeichnete Zustand, zu dem es einen Terminal-Zustand in Entfernung 2 gibt, hat den Wert 0 und somit noch nicht den richtigen Wert, weil bei seiner Berechnung noch der alte Wert V_0^* des unteren grünen Zustands verwendet wurde und dieser 0 war (siehe Abbildung 2.4.2.2).

Nach dem nächsten Iterationsschritt ergibt sich $V_2^*(s)$:

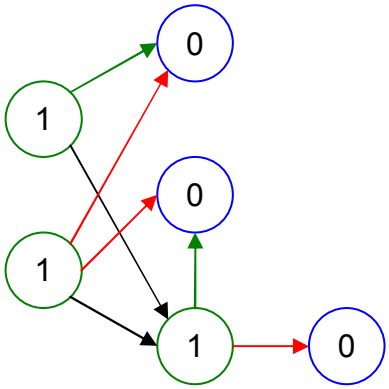


Abbildung 2.4.2.4

Auf diese Weise erhält man für alle Zustände nach N Iterationen ihren V^*-Wert (d. h. $V_N^* = V^*$) und eine optimale Policy ergibt sich implizit, indem sie in einem Zustand immer eine Aktion wählt, die zu einem Folgezustand führt, der den maximalen Wert aller Folgezustände hat (vgl. auch Abbildung 2.2.3.4). Im indeterministischen Fall sind hierbei noch die Übergangswahrscheinlichkeiten zu beachten, denn es kann besser sein, eine Aktion zu wählen, die sehr wahrscheinlich zu einem mittelmäßigen Folgezustand führt, als eine, durch die mit geringer Wahrscheinlichkeit ein sehr guter Folgezustand erreicht wird, aber mit hoher ein schlechter. Die Konstruktionsvorschrift für π^* lautet daher:

$$\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} P_{ss'}^a \left(r_{ss'}^a + V^*(s') \right).$$

Ein weiteres Problem im indeterministischen Fall liegt vor, wenn die Wahrscheinlichkeitsverteilungen $P_{ss'}^a$ oder $r_{ss'}^a$ unbekannt sind, das Modell also nicht vollständig bekannt ist. Dann müssen diese Werte zusätzlich noch stochastisch ermittelt werden, bevor Backward DP beginnen kann.

Algorithmus 2.4.2.1 (Backward DP):

1. $\forall s \in S : V_0^*(s) := 0$
2. For i = 1 To N:

2.1. $\forall s \in S : V_i^*(s) := \max_{a \in A(s)} \sum_{s'} P_{ss'}^a \left(r_{ss'}^a + V_{i-1}^*(s') \right)$

3. $\forall s \in S : \pi^*(s) := \operatorname{argmax}_{a \in A(s)} \sum_{s'} P_{ss'}^a \left(r_{ss'}^a + \underbrace{V_N^*}_{=V^*}(s') \right)$

Als einziges in dieser Arbeit vorgestellte Verfahren berechnet es V^* und damit π^* nach endlicher Laufzeit mit Sicherheit korrekt, setzt jedoch hohe Voraussetzungen, denn der Horizont muss endlich sein und das Modell des RL-System muss bekannt sein. Zusammenfassend lässt sich sagen, dass Backward DP auf Grund dieser Einschränkungen und der langen Rechenzeit in der Praxis uninteressant ist.

2.4.3 Value Iteration

Als nächstes DP-Verfahren sei die so genannte Value Iteration erwähnt. Sie ist im Wesentlichen eine Verallgemeinerung des Backward DP, bei dem eine Voraussetzung der endliche Horizont war. Dies setzt Value Iteration nicht voraus, benötigt aber, falls der Horizont unendlich ist, die Diskontierung. Die Update-Vorschrift lautet demnach hier:

$$V_i^*(s) := \max_{a \in A(s)} \sum_{s'} P_{ss'}^a \left(r_{ss'}^a + \gamma V_{i-1}^*(s') \right).$$

Eine weitere Folge des endlichen Horizonts war, dass es ein N gab, so dass von jedem Zustand nach spätestens N Schritten alle von ihm aus erreichbaren Terminal-Zustände erreicht wurden. Dadurch reichten N Iterationen, um die korrekten Werte für V^* zu ermitteln. Value Iteration betrachtet nun $N \to \infty$ und es lässt sich zeigen [DuW2001, S. 9], dass

$$\lim_{N \to \infty} V_N^*(s) = V^*(s) \forall s \in S.$$

In der Praxis wird allerdings nur solange iteriert, bis sich keiner der Funktionswerte mehr wesentlich ändert, d. h. bis alle Änderung unterhalb einer kleinen Schranke δ bleiben. Man hat dann eine gute Approximation für V^* und dadurch i. d. R. auch für π^*.

Algorithmus 2.4.3.1 (Value Iteration):

1. $i := 0$; $\forall s \in S : V_0^*(s) := 0$

2. Do

2.1. $i += 1$

2.2. $\forall s \in S : V_i^*(s) := \max_{a \in A(s)} \sum_{s'} P_{ss'}^a \left(r_{ss'}^a + \gamma V_{i-1}^*(s') \right)$

Loop Until $\forall s \in S : \left| V_i^*(s) - V_{i-1}^*(s) \right| < \delta$

3. $\forall s \in S : \pi^*(s) := \operatorname{argmax}_{a \in A(s)} \sum_{s'} P_{ss'}^a \left(r_{ss'}^a + \gamma \underbrace{V_i^*}_{\approx V^*}(s') \right)$

(Bemerkung zur Notation: $A \circ = B$, wobei A und B Objekte wie Zahlen oder Mengen sind und $\circ$ ein Operator wie +, - oder $\cup$ ist, bedeutet $A := A \circ B$. Im obigen Algorithmus bedeutet demnach $i += 1$, dass i um 1 inkrementiert wird.)

2.4.4 Policy Iteration

Das dritte und letzte DP-Verfahren ist die Policy Iteration und wendet eine andere Vorgehensweise an. Es wird nicht erst V^* berechnet und anschließend eine optimale Policy konstruiert, sondern es wird mit einer beliebigen *deterministischen* Policy π_0 begonnen und danach V^{π_0} näherungsweise berechnet (Policy Evaluation). Anschließend wird an Hand der Werte von V^{π_0} geprüft, an welchen Stellen man π_0 verbessern könnte (Policy Improvement). Es ergibt sich eine bessere, deterministische Policy π_1, welche wieder erst evaluiert und anschließend verbessert wird.

Während der Policy Evaluations-Phase für π_i lautet die Update-Vorschrift:

$$V_j^{\pi_i}(s) := \sum_{s'} P_{ss'}^{\pi_i(s)} \left(r_{ss'}^{\pi_i(s)} + \gamma V_{j-1}^{\pi_i}(s') \right),$$

wobei die Startwerte $V_0^{\pi_i}$ jeder Phase wieder beliebig initialisiert werden können, und es wird gelten:

$$\lim_{j \to \infty} V_j^{\pi_i}(s) = V^{\pi_i}(s) \forall s \in S.$$

Die aktuelle Policy Evaluations-Phase endet in der Praxis, wenn sich die Werte im Vergleich zur vorherigen Iteration nicht mehr bedeutend ändern und man daher i. d. R. eine gute Approximation für V^{π_i} ausgeben kann.

Algorithmus 2.4.4.1 (Policy Evaluation):

1. $j := 0$; $\forall s \in S : V_0^{\pi_i}(s) := 0$

2. Do

2.1. $j += 1$

2.2. $\forall s \in S : V_j^{\pi_i}(s) := \sum_{s'} P_{ss'}^{\pi_i(s)}\left(r_{ss'}^{\pi_i(s)} + \gamma V_{j-1}^{\pi_i}(s')\right)$

Loop Until $\forall s \in S : \left|V_j^{\pi_i}(s) - V_{j-1}^{\pi_i}(s)\right| < \delta$

3. $V^{\pi_i} := V_j^{\pi_i}$

In der darauf folgenden Policy Improvement-Phase wird die neue, ebenfalls deterministische Policy π_{i+1} konstruiert gemäß der Vorschrift:

$$\pi_{i+1}(s) := \arg\max_{a \in A(s)} \sum_{s'} P_{ss'}^{a}\left(r_{ss'}^{a} + \gamma V^{\pi_i}(s')\right).$$

Algorithmus 2.4.4.2 (Policy Improvement):

1. $\forall s \in S : \pi_{i+1}(s) := \arg\max_{a \in A(s)} \sum_{s'} P_{ss'}^{a}\left(r_{ss'}^{a} + \gamma V^{\pi_i}(s')\right)$

Diese Vorschrift verbessert die Policy in der Tat, denn:

$$\begin{aligned} V^{\pi_i}(s) &= \sum_{s'} P_{ss'}^{\pi_i(s)}\left(r_{ss'}^{\pi_i(s)} + \gamma V^{\pi_i}(s')\right) \\ &\leq \sum_{s'} P_{ss'}^{\pi_{i+1}(s)}\left(r_{ss'}^{\pi_{i+1}(s)} + \gamma V^{\pi_{i+1}}(s')\right) = V^{\pi_{i+1}}(s) \forall s \in S \\ &\underbrace{\Rightarrow}_{\text{Definition 2.3.2.2}} \pi_{i+1} \geq \pi_i \end{aligned}$$

Es gilt $\pi_{i+1} = \pi_i$ genau dann, wenn $\sum_{s'} P_{ss'}^{\pi_i(s)}\left(r_{ss'}^{\pi_i(s)} + \gamma V^{\pi_i}(s')\right) = \sum_{s'} P_{ss'}^{\pi_{i+1}(s)}\left(r_{ss'}^{\pi_{i+1}(s)} + \gamma V^{\pi_{i+1}}(s')\right) \forall s \in S$. Das wiederum bedeutet, dass bereits für π_i galt: $\pi_i(s) = \arg\max_{a \in A(s)} \sum_{s'} P_{ss'}^{a}\left(r_{ss'}^{a} + \gamma V^{\pi_i}(s')\right) \forall s \in S$.

Das Verfahren endet somit, wenn die neue Policy gleich der alten ist. Die Konvergenz der π_i gegen π^* ist gegeben wegen $\pi_{i+1} > \pi_i$ (außer im letzten Schritt) und der Tatsache, dass es nur endlich viele deterministische Policies gibt, weil es sich nach Voraussetzung um einen endlichen MDP handelt, d. h. Zustands- und Aktionsraum sind endlich.

Algorithmus 2.4.4.3 (Policy Iteration):

1. $i := 0$
2. Do
2.1. Policy Evaluation
2.2. Policy Improvement
2.3. $i += 1$
Loop Until $\pi_i = \pi_{i-1}$
3. $\pi^* := \pi_i$

2.5 Monte Carlo Lernverfahren

2.5.1 Allgemeines

Die großen Nachteile der DP-Verfahren sind, dass das Modell bekannt sein muss und die lange Rechenzeit. Die lange Rechenzeit wird u. a. dadurch verursacht, dass Näherungen für V^* bzw. V^{π_i} in jedem Iterationsschritt vollständig berechnet werden, d. h. für alle Zustände aus *S*. Monte Carlo (MC) Lernverfahren setzen statt auf lange Berechnung auf Lernen aus Beispiel-Episoden (engl.: sample episodes), welche sich gemäß der Wahrscheinlichkeitsverteilungen ergeben. Daher ist MC generell nur für RL-Aufgaben mit endlichem Horizont geeignet und benötigt in keinem Fall Kenntnis über die $r_{ss'}^a$-Werte, denn durch das Erleben von Beispiel-Episoden lernt der Agent das Belohnungssignal kennen.

2.5.2 Policy Iteration bei bekannter Zustandsübergangsfunktion

Sei zunächst die Zustandsübergangsfunktion der Umwelt gegeben. Wie bei der Policy Iteration der DP wird mit einer beliebigen Policy π_0 begonnen, diesmal jedoch eine Beispiel-Episode durchgespielt. Danach wird nur zu jedem in dieser speziellen Episode besuchten Zustand s_t der gewonnene Return R_t gemäß $R_t = r_{t+1} + r_{t+2} + \ldots + r_T$ berechnet, was nur wegen des endlichen Horizonts möglich ist, und $\tilde{V}^{\pi_0}(s_t) := R_t$ gesetzt, wobei $\tilde{V}^{\pi_0}$ die aktuelle Näherung für das zu bestimmende V^{π_0} ist. Anschließend werden weitere Episoden ausgeführt. Dabei werden andere Zustände erreicht und ihr $\tilde{V}^{\pi_0}$-Wert gesetzt. Wird ein Zustand erreicht, der bereits in vorherigen Episoden erreicht wurde, wird sein bisheriger $\tilde{V}^{\pi_0}$-Wert nicht einfach überschrieben, was einer Gedächtnisauslöschung gleich käme,

sondern durch ein gewichtetes Mittel zwischen altem Wert und dem aktuellen Return wie folgt ersetzt:

Sei s_t ein in der eben beendeten Episode besuchter Zustand, R_t der von s_t aus erzielte Return in dieser Episode und sei $v_{s_t}^{\pi_0}$ die Anzahl der Besuche des Zustands s_t *vor* dieser Episode unter Anwendung von π_0. Dann ist eine übliche und sinnvolle Update-Vorschrift:

$$\tilde{V}^{\pi_0}(s_t) := \frac{v_{s_t}^{\pi_0}}{v_{s_t}^{\pi_0}+1}\tilde{V}^{\pi_0}(s_t) + \frac{1}{v_{s_t}^{\pi_0}+1}R_t .$$

Der alte Wert geht also umso stärker in den neuen Wert ein, je öfter der Zustand zuvor besucht wurde, damit, falls das aktuelle R_t eine Ausnahme im Sinne von 'besonders gut' oder 'besonders schlecht' war, der bisherige Wert nicht zu sehr verfälscht wird. Ein so definiertes $\tilde{V}^{\pi_0}(s_t)$ entspricht genau dem Durchschnitt aller bisherigen Returns von s_t aus und konvergiert, wenn die Anzahl der Episoden gegen unendlich geht, nach dem 'Law of large numbers' [Fel1968] gegen $V^{\pi_0}(s_t)$.

Die Update-Vorschrift lässt sich durch einen zusätzlichen Parameter $0 < \alpha < 1$ verallgemeinern zu

$$\tilde{V}^{\pi_0}(s_t) := (1-\alpha)\tilde{V}^{\pi_0}(s_t) + \alpha R_t .$$

Ein derartiges α kann mit der Zeit abkühlen (z. B. $\alpha := \frac{1}{v_{s_t}^{\pi_0}+1}$) oder fix sein (z. B. $\alpha := 0.1$), wodurch allerdings die Konvergenzeigenschaft verloren gehen kann.

Diese Phase entspricht einer Policy Evaluation.

Algorithmus 2.5.2.1 (MC-Policy Evaluation mit *V*-Werten):

1. $\forall s \in S : \tilde{V}^{\pi_i}(s) := 0$, $v_s^{\pi_i} := 0$
2. Do

2.1. Durchlaufe eine Beispiel-Episode und berechne zu jedem besuchten Zustand $s_0,\ldots,s_T$ den erhaltenen Return R_t.

2.2. $\forall s_t \in \{s_0,\ldots,s_T\} : \tilde{V}^{\pi_i}(s_t) := \frac{v_{s_t}^{\pi_i}}{v_{s_t}^{\pi_i}+1}\tilde{V}^{\pi_i}(s_t) + \frac{1}{v_{s_t}^{\pi_i}+1}R_t$, $v_{s_t}^{\pi_i} += 1$

Loop Until ‘keine wesentlichen Veränderungen der $\tilde{V}^{\pi_i}$-Werte in den letzten Durchläufen’

3. $V^{\pi_i} := \tilde{V}^{\pi_i}$

An Hand der gewonnenen Werte wird nun eine Policy Improvement wie bei der Policy Iteration der DP durchgeführt, aus der π_1 hervorgeht. Diese Policy wird wiederum evaluiert und verbessert. Das Verfahren konvergiert aus denselben Gründen wie die Policy Iteration der DP gegen π^*, da der einzige Unterschied ist, dass die Approximation für V^{π_i} auf andere Art und Weise berechnet wird, was aber für die Konvergenzeigenschaft an sich irrelevant ist.

2.5.3 Policy Iteration bei unbekannter Zustandsübergangsfunktion

Sei nun die Zustandsübergangsfunktion der Umwelt unbekannt. In diesem Fall reichen die $\tilde{V}^{\pi_i}(s)$-Werte nicht, denn selbst wenn man das korrekte V^{π_i} kennen würde, so würde dieses Wissen ohne Zustandsübergangsfunktion nichts nutzen, d. h. man könnte die Policy nicht verbessern, da man zwar wüsste, in welchen Folgezustand man gerne wechseln möchte, aber nicht, durch welche Aktion man dorthin gelangt. Aus diesem Grund geht man in diesem Fall zu den $Q^{\pi_i}(s,a)$ -Werten über. Dann weiß man zwar immer noch nicht, in welchen Folgezustand man gelangt, doch man weiß, wie gut die Aktion *a* in *s* ist. Darin ist dann implizit gespeichert, ob erwartet wird, dass der Folgezustand gut oder schlecht ist. Die Update-Vorschrift während der Policy Evaluation-Phase lautet analog:

$$\tilde{Q}^{\pi_i}(s_t,a_t) := (1-\alpha)\tilde{Q}^{\pi_i}(s_t,a_t) + \alpha R_t \text{ mit } \alpha := \frac{1}{v^{\pi_i}_{(s_t,a_t)}+1},$$

wobei $v^{\pi_i}_{(s_t,a_t)}$ analog zu $v^{\pi_i}_{s_t}$ angibt, wie oft (s_t,a_t) während der Evaluationsphase von π_i zuvor angewendet wurde. Die neue Policy π_{i+1} ergibt sich in der Policy Improvement-Phase durch:

$$\pi_{i+1}(s) := \arg\max_{a\in A(s)} \tilde{Q}^{\pi_i}(s,a).$$

π_{i+1} wählt demnach in Zustand *s* diejenige Aktion *a*, die den höchsten erwarteten Return bringt.

Die Unzugänglichkeit der Zustandsübergangsfunktion und der daraus

resultierende Übergang zu Q^{π_i} werfen ein Problem auf. Auf Grund der jeweils deterministischen Policy approximiert das Verfahren nur die Q^{π_i}-Werte für Paare $(s,\pi_i(s))$, nicht aber für Paare (s,a) mit $a \neq \pi_i(s)$. Diese werden aber benötigt, um π_i zu verbessern. Eine bewährte Möglichkeit, dieses Problem zu lösen, ist, den Agenten in jedem Schritt der Episoden mit einer kleinen Wahrscheinlichkeit ε von seiner Policy abweichen zu lassen, d. h. eine *beliebige* Aktion $a \in A(s)$ zu wählen ($a = \pi_i(s)$ möglich). Dieses Verhalten lässt sich als eine indeterministische Policy $\tilde{\pi}_i$ ausdrücken:

Definition 2.5.3.1:

Eine Policy $\tilde{\pi}$ mit

$$\tilde{\pi}(s,a) := \begin{cases} 1-\varepsilon+\dfrac{\varepsilon}{|A(s)|}, \text{ falls } a = \pi(s) \\ \dfrac{\varepsilon}{|A(s)|}, \text{ falls } a \neq \pi(s) \end{cases}$$

heißt 'ε-greedy bzgl. π'.

Verwendet man in den Beispiel-Episoden ε-greedy Policies $\tilde{\pi}_i$, so wird dafür gesorgt, dass die Policy π_i genutzt wird (Exploitation), dennoch aber unbekannte Paare (s,a) erforscht werden (Exploration).

Auf diese Weise wird durch die Policy Iteration eine optimale *ε-greedy* Policy für das RL-Problem gefunden.

Algorithmus 2.5.3.1 (MC-Policy Evaluation mit *Q*-Werten):

1. $\forall\left(\underbrace{s}_{\in S},\ \underbrace{a}_{\in A(s)}\right): \tilde{Q}^{\tilde{\pi}_i}(s,a) := 0, v^{\tilde{\pi}_i}_{(s,a)} := 0$

2. Do

2.1. Durchlaufe eine Beispiel-Episode und berechne zu jedem besuchten Paar $(s_0,a_0),\ldots,(s_T,a_T)$ den erhaltenen Return R_t.

2.2. $\forall(s_t,a_t) \in \{(s_0,a_0),\ldots,(s_T,a_T)\}: \tilde{Q}^{\tilde{\pi}_i}(s_t,a_t) := \left(\dfrac{v^{\tilde{\pi}_i}_{(s_t,a_t)}}{v^{\tilde{\pi}_i}_{(s_t,a_t)}+1}\right)\tilde{Q}^{\tilde{\pi}_i}(s_t,a_t) + \dfrac{1}{v^{\tilde{\pi}_i}_{(s_t,a_t)}+1}R_t$,

$v^{\tilde{\pi}_i}_{(s_t,a_t)} += 1$

Loop Until ‘keine wesentlichen Veränderungen der $\tilde{Q}^{\tilde{\pi}_i}$-Werte in den letzten Durchläufen’

3. $Q^{\tilde{\pi}_i} := \tilde{Q}^{\tilde{\pi}_i}$

Algorithmus 2.5.3.2 (MC-Policy Improvement mit *Q*-Werten):

1. $\forall s \in S : \pi_{i+1}(s) := \arg\max_{a \in A(s)} Q^{\tilde{\pi}_i}(s,a)$

2. $\forall \left(\underbrace{s}_{\in S}, \underbrace{a}_{\in A(s)} \right) : \tilde{\pi}_{i+1}(s,a) := \begin{cases} 1 - \varepsilon + \dfrac{\varepsilon}{|A(s)|}, \text{falls } a = \pi_{i+1}(s) \\ \dfrac{\varepsilon}{|A(s)|}, \text{falls } a \neq \pi_{i+1}(s) \end{cases}$

Algorithmus 2.5.3.3 (MC-Policy Iteration mit *Q*-Werten):

1. $i := 0$
2. Do
2.1. Policy Evaluation
2.2. Policy Improvement
2.3. $i += 1$

Loop Until $\pi_i = \pi_{i-1}$

3. $\pi^* := \pi_i$

Sämtliche bisher vorgestellte Policy Evaluation-Algorithmen haben eine negative Gemeinsamkeit: Sie konvergieren erst im Unendlichen. Da eine Policy Evaluation nach jeder Verbesserung der Policy durchgeführt wird, kann die Rechenzeit bis zum Finden einer optimalen Policy sehr lang sein.

Es ist möglich, nach jeder Episode die Policy zu verbessern, ohne sie vorher ausgiebig zu evaluieren. Begonnen werden kann wie üblich mit einer beliebigen Policy π, genauer mit der dazu gehörenden, ε-greedy Policy $\tilde{\pi}$:

Algorithmus 2.5.3.4 (Monte Carlo):

1. $\forall \left(\underbrace{s}_{\in S}, \underbrace{a}_{\in A(s)} \right) : \tilde{Q}(s,a) := 0, v_{(s,a)} := 0$

2. Do

2.1. Durchlaufe eine Beispiel-Episode unter Verwendung von $\tilde{\pi}$ und berechne zu jedem besuchten Paar $(s_0, a_0), \ldots, (s_T, a_T)$ den erhaltenen Return R_t.

2.2. $\forall (s_t, a_t) \in \{(s_0, a_0), \ldots, (s_T, a_T)\} : \tilde{Q}(s_t, a_t) := \left(\frac{v_{(s_t, a_t)}}{v_{(s_t, a_t)} + 1} \right) \tilde{Q}(s_t, a_t) + \frac{1}{v_{(s_t, a_t)} + 1} R_t$,

$v_{(s_t, a_t)} += 1$

2.3. $\forall s_t \in \{s_0, \ldots, s_T\} : \pi(s_t) := \arg\max_{a \in A(s_t)} \tilde{Q}(s_t, a)$

2.4. $\forall \left(s_t, \underbrace{a}_{\in A(s_t)} \right) : \tilde{\pi}(s_t, a) := \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|A(s_t)|}, \text{falls } a = \pi(s_t) \\ \frac{\varepsilon}{|A(s_t)|}, \text{falls } a \neq \pi(s_t) \end{cases}$

Loop Until ‘π in den letzten Durchläufen unverändert’

3. $\pi^* := \pi$

Bei diesem Algorithmus werden erstmals eine globale Policy π und ein globales $\tilde{Q}$ verwaltet, dessen Werte nach einer Änderung der Policy erhalten bleiben.

Das Verfahren konvergiert nicht gegen eine sub-optimale Policy. Würde es gegen eine solche konvergieren, so würde im 3. Schritt der Schleife die Policy geändert werden und damit würden sich die $\tilde{Q}$-Werte der neuen Policy anpassen, was der Konvergenz widersprechen würde. Ob es allerdings konvergiert, wurde bisher nicht bewiesen.

2.6 Temporal-Difference Learning und *Q*-Learning

2.6.1 Allgemeines

Temporal-Difference (TD) Learning verbindet die Vorteile von DP und MC. MC hat die Nachteile, dass immer eine komplette Episode durchgespielt werden muss, bis ein Update oder eine Verbesserung durchgeführt werden kann, und damit auch, dass der Horizont endlich sein muss, aber die Vorteile, dass es aus Beispiel-Episoden lernen kann und das Modell des RL-Systems nicht benötigt. DP führt die Updates nach jedem Schritt mit Hilfe des Wertes des Folgezustands durch und setzt dank Diskontierungsmöglichkeit keinen endlichen Horizont voraus, kann aber nicht aus Beispielen lernen und braucht das Modell des RL-Systems.

TD braucht i. A. das Modell des RL-Systems nicht und lernt an Hand von Beispiel-Episoden. Die einfachste TD-Methode ($TD(0)$) benutzt wie DP den Wert des Folgezustands bzw. des Zustand-Aktion-Folgepaars. Dies wird verallgemeinert zu $TD(\lambda)$, welches noch Werte weiterer Folgezustände bzw. Zustand-Aktion-Folgepaare berücksichtigt, jedoch nicht so, dass erst das Ende der Episode abgewartet werden muss, wie bei MC. Daher muss auch der Horizont nicht mehr als endlich vorausgesetzt werden.

2.6.2 *TD*(0) und *Q*-Learning

Es gibt sowohl einen $TD(0)$-Algorithmus zur Berechnung von *V*-Werten, als auch einen, für *Q*-Werte. Hier wird nur der Algorithmus für *Q* betrachtet, weil *V*-Werte nur nutzbar sind, wenn die Zustandsübergangsfunktion der Umwelt bekannt ist. Die zentrale Update-Formel dieses Algorithmus leitet sich aus der Update-Formel

$$\tilde{Q}^{\tilde{\pi}_i}(s_t, a_t) := (1-\alpha)\tilde{Q}^{\tilde{\pi}_i}(s_t, a_t) + \alpha R_t$$

der MC-Policy Evaluation her. Die Idee ist, R_t durch den erwarteten diskontierten Return $r_{t+1} + \gamma \tilde{Q}^{\tilde{\pi}_i}(s_{t+1}, a_{t+1})$ zu ersetzen.

Algorithmus 2.6.2.1 (*TD*(0)-Policy Evaluation):

1. $\forall\left(\underbrace{s}_{\in S}, \underbrace{a}_{\in A(s)}\right): \tilde{Q}^{\tilde{\pi}_i}(s,a) := 0$
2. $t := 0$; Wähle einen Startzustand s_0 und eine Startaktion $a_0 := \tilde{\pi}_i(s_0)$.
3. Do While 's_t ist kein Terminal-Zustand':

3.1. Führe im aktuellen Zustand s_t die aktuelle Aktion a_t aus.

3.2. Erhalte Belohnung r_{t+1}, gelange in Folgezustand s_{t+1} und wähle eine Folgeaktion $a_{t+1} := \tilde{\pi}_i(s_{t+1})$.

3.3. $\tilde{Q}^{\tilde{\pi}_i}(s_t, a_t) := (1-\alpha)\tilde{Q}^{\tilde{\pi}_i}(s_t, a_t) + \alpha\left(\underbrace{r_{t+1} + \gamma \tilde{Q}^{\tilde{\pi}_i}(s_{t+1}, a_{t+1})}_{\text{diskontierter Näherungswert von } R_t}\right)$

3.4. $t += 1$

4. Falls 'wesentliche Veränderungen der $\tilde{Q}^{\tilde{\pi}_i}$-Werte in den letzten Durchläufen': zurück zu 2
5. $Q^{\tilde{\pi}_i} := \tilde{Q}^{\tilde{\pi}_i}$

Die $\tilde{Q}^{\tilde{\pi}_i}$-Werte konvergieren für $\alpha := \frac{1}{v_{(s_t,a_t)}}$, wobei $v_{(s_t,a_t)}$ diesmal für die *aktuelle* Anzahl bisheriger Besuche von (s_t, a_t) steht (also incl. des Besuchs in der aktuellen Episode), wegen $\sum_{v_{(s_t a_t)}=1}^{\infty} \frac{1}{v_{(s_t a_t)}} = \infty$ [Bro1999, Beispiel 6.5] und $\sum_{v_{(s_t a_t)}=1}^{\infty} \left(\frac{1}{v_{(s_t a_t)}} \right)^2 = \sum_{v_{(s_t a_t)}=1}^{\infty} \frac{1}{v_{(s_t a_t)}^2} < \infty$ [Bro1999, Beispiel 6.12(2)] nach [JJS1994] gegen die $Q^{\tilde{\pi}_i}$-Werte. (Bemerkung: Dieser Algorithmus ist ein *TD*(0)-Algorithmus und damit ein Spezialfall eines *TD*(λ)-Algorithmus.)

Wie bei Algorithmus 2.5.3.4 lassen sich auch hier Policy Evaluation und Policy Improvement integrieren. Da das Ende der Episode nicht abgewartet werden muss, ist dies sogar nach jedem Schritt jeder Episode möglich. Resultat ist das weit verbreitete *Q*-Learning [Wat1989], [WuD1992]:

Algorithmus 2.6.2.2 (*Q*-Learning):

1. $\forall \left(\underbrace{s}_{\in S}, \underbrace{a}_{\in A(s)} \right) : \tilde{Q}(s,a) := 0$

2. $t := 0$; Wähle einen Startzustand s_0.

3. Do While 's_t ist kein Terminal-Zustand':

3.1. Führe im aktuellen Zustand s_t die aktuelle Aktion $a_t := \tilde{\pi}(s_t)$ aus.

3.2. Erhalte Belohnung r_{t+1} und gelange in Folgezustand s_{t+1}.

3.3. $\tilde{Q}(s_t, a_t) := (1-\alpha)\tilde{Q}(s_t, a_t) + \alpha\left(r_{t+1} + \gamma \max_{a' \in A(s_{t+1})} \tilde{Q}(s_{t+1}, a') \right)$

3.4. $\pi(s_t) := \arg\max_{a \in A(s_t)} \tilde{Q}(s_t, a)$

3.5. $\forall a \in A(s_t) : \tilde{\pi}(s_t, a) := \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|A(s_t)|}, \text{falls } a = \pi(s_t) \\ \frac{\varepsilon}{|A(s_t)|}, \text{falls } a \neq \pi(s_t) \end{cases}$

3.6. $t += 1$

4. Falls 'π in den letzten Durchläufen verändert': zurück zu 2.

5. $\pi^* := \pi$

Für $\alpha := \frac{1}{v_{(s_t,a_t)}}$ konvergiert $\tilde{Q}$ gegen Q^* [JJS1994].

2.6.3 *TD*(λ) und *Q*(λ)-Learning

Der *Q*-Learning-Algorithmus (Algorithmus 2.6.2.2) lässt sich weiter verbessern. Wie an Hand der Update-Formel zu erkennen ist, wird nach jeder Aktion immer nur der *Q*-Wert des zuletzt besuchten Paares (s_t, a_t) erneuert, weshalb es mitunter lange dauert, bis sich die neue Erfahrung auf weit davor liegende Zustand-Aktion-Paare auswirkt.

$TD(\lambda)$-Methoden beheben diese Schwäche. Sie protokollieren auf gewisse Weise mit, welche Paare zuvor besucht worden sind, und updaten deren *Q*-Werte ebenfalls, wobei die Änderung umso schwächer ist, je weiter das Paar in der Vergangenheit liegt. Die Schwächung wird durch einen Parameter $0 \le \lambda \le 1$ geregelt. Die Konvergenz der $\tilde{V}$- bzw. $\tilde{Q}$-Werte für $0 \le \lambda < 1$ gegen die korrekten Werte ist bewiesen [JJS1994].

In dieser Arbeit wird nur der $Q(\lambda)$-Algorithmus von Watkins vorgestellt, der eine $TD(\lambda)$-Variante des *Q*-Learning-Algorithmus ist. Er ordnet jedem Zustand-Aktion-Paar eine Berechtigung (engl.: eligibility) $e(s,a)$ für das nächste Update zu. Wird ein Paar zum ersten Mal besucht, wird sein *e*-Wert auf 1 gesetzt. Nach jedem Schritt wird diese Berechtigung ähnlich wie beim Diskontieren reduziert. Wird das Paar im Laufe der Episode wieder besucht, wird der Wert wieder um 1 erhöht. Vor Beginn einer neuen Episode werden die Berechtigungen aller Paare auf 0 zurückgesetzt, weil zwei Episoden keinen kausalen Zusammenhang haben.

Es gibt ein erwähnenswertes Detail, das zu beachten ist. Da eine indeterministische, ε-greedy Policy $\tilde{\pi}$ anstelle der eigentlichen, deterministischen Policy π verwendet wird, würden $\tilde{Q}$-Werte von zuvor besuchten Zustand-Aktion-Paaren, die ja die Güte der Paare bzgl. π repräsentieren sollen, verfälscht, wenn $\tilde{\pi}$ sich auf Grund des Indeterminismus für eine andere Aktion als π entscheidet, um zu explorieren. Daher werden in einem solchen Fall alle *e*-Werte wieder auf 0 gesetzt, so als hätte eine neue Episode mit dem Zustand, in dem die zufällige Aktion gewählt wurde, als Startzustand begonnen.

Algorithmus 2.6.3.1 ($Q(\lambda)$-Learning):

1. $\forall\left(\underbrace{s}_{\in S},\ \underbrace{a}_{\in A(s)}\right): \tilde{Q}(s,a) := 0$

2. $\forall\left(\underbrace{s}_{\in S},\ \underbrace{a}_{\in A(s)}\right): e(s,a) := 0$; $t := 0$; Wähle einen Startzustand s_0.

3. Do While 's_t ist kein Terminal-Zustand':

3.1. Führe im aktuellen Zustand s_t die aktuelle Aktion $a_t := \tilde{\pi}(s_t)$ aus.

3.2. Erhalte Belohnung r_{t+1} und gelange in Folgezustand s_{t+1}.

3.3. $\delta := r_{t+1} + \gamma \max_{a' \in A(s_{t+1})} \tilde{Q}(s_{t+1}, a') - \tilde{Q}(s_t, a_t)$

3.4. $\forall(s,a)$ mit $e(s,a) > 0$: Falls in 3.1 $a_t \neq \pi(s_t)$ gewählt wurde: $e(s,a) := 0$; Else: $e(s,a) := \gamma\lambda e(s,a)$

3.5. $e(s_t, a_t) += 1$

3.6. $\forall(s,a)$ mit $e(s,a) > 0$: $\tilde{Q}(s,a) += \alpha e(s,a)\delta$

3.7. $\forall(s,a)$ mit $e(s,a) > 0$:

3.7.1. $\pi(s) := \arg\max_{b \in A(s)} \tilde{Q}(s,b)$

3.7.2. $\forall b \in A(s): \tilde{\pi}(s,b) := \begin{cases} 1 - \varepsilon + \dfrac{\varepsilon}{|A(s)|}, & \text{falls } b = \pi(s) \\ \dfrac{\varepsilon}{|A(s)|}, & \text{falls } b \neq \pi(s) \end{cases}$

3.8. $t += 1$

4. Falls 'π in den letzten Durchläufen verändert': zurück zu 2.

5. $\pi^* := \pi$

Es konnte bisher nicht nachgewiesen werden, ob dieser Algorithmus konvergiert. Ein Problem dabei stellen die Berechtigungen dar. In der Praxis liefert er jedoch schnell gute Ergebnisse.

2.7 Vergleich der Verfahren

Die vorgestellten Verfahren unterscheiden sich in ihren Voraussetzungen und der Art und Weise, wie die Werte aktualisiert werden. DP und *TD*(0) verwenden die unmittelbare Belohnung und den Wert des Folgezustands, um den wahren Return zu

schätzen. Diese Vorgehensweise wird als Bootstrapping bezeichnet. MC hingegen wartet das Ende der Episode ab, um den korrekten Return zu erfahren. *TD*(λ) füllt die Lücke zwischen diesen beiden Extremen. Im Fall $\lambda = 0$ entspricht es reinem Bootstrapping, im Fall $\lambda = 1$ entspricht es MC.

Es ergibt sich folgende Übersicht.

Tabelle 2.7.1 (Vergleich der Verfahren):

	unbekanntes Modell möglich	unendlicher Horizont möglich	Lernen aus Beispielen	Bootstrapping
DP	×	✓	×	✓
MC	✓	×	✓	×
TD(λ)	✓	✓	✓	✓ bis ×

3. Wissen, Geglaubtes und Belief Revision

3.1 Grundlagen

RL ist auf Grund der Art und Weise, wie der Agent Fähigkeiten erwirbt (nämlich Trial-and-Error), eine Low-Level-Lernmethodik und es findet sich in vielen Bereichen des Lebens. Man denke an ein neugeborenes Fohlen, das versucht aufzustehen, oder die Evolution an sich, die Vieles probiert und nur das überleben lässt, was sich als erfolgreich erweist.

Darüber hinaus haben einige Lebewesen zusätzlich ein höheres Level etwas zu lernen entwickelt. Dieses Level setzt ein gewisses Maß an Bewusstsein voraus, denn es ist nötig, dass sich das Lebewesen ein gewisses ‘Bild’ seiner Umwelt machen kann. Dieses ‘Bild’ (im Folgenden als ‘Modell’ bezeichnet) ist eine Hypothese des Lebewesens über den wahren aktuellen Zustand der Welt. Es wird gebildet z. B. durch Beobachtung und Nachdenken. Eine zentrale Rolle beim Nachdenken spielen die Logik und die Möglichkeit zur Formulierung von Begriffen.

‘Logik’ im Allgemeinen ist kaum zu definieren, da sie in den verschiedensten Disziplinen wie Mathematik, Informatik oder Philosophie verwendet wird und es dementsprechend viele Varianten und Spezialisierungen gibt. Eine Gemeinsamkeit ist jedoch, dass mit Hilfe von (Annahmen oder Fakten) und Regeln Schlussfolgerungen gezogen werden. Dieses Vorgehen nennt man ‘Inferenz’.

Beispiel 3.1.1:

Es ist bekannt, dass Wasser bei 100 °C beginnt zu kochen. Der Agent schaut auf ein Thermometer und liest ‘150 °’. Er schließt daraus, dass das Wasser kocht.

Diese rudimentäre Art des Schlussfolgerns ist jedoch für die reale Welt zu einfach, wie die folgende Erweiterung des Beispiels zeigt:

Beispiel 3.1.2:

Der Agent aus Beispiel 3.1.1 stellt durch einen Blick aufs Wasser fest, dass es *nicht* kocht. Jetzt ist er in der misslichen Lage, einen Widerspruch zu haben. (Das Wasser kocht und das Wasser kocht nicht.) Der klassischen Logik nach folgt aus dem Widerspruch alles. Der Agent ist jedoch schlauer und *revidiert* sein Wissen mit der neuen Information, statt diese einfach hinzuzunehmen. Er glaubt jetzt, dass das

Wasser nicht kocht und nicht mehr, dass es kocht. Doch das reicht noch nicht, denn es folgt aus der Temperatur von 150 °C immer noch, dass das Wasser kocht. Um zu verhindern, dass die Folgerung möglich ist, hat der Agent z. B. folgende Optionen:

- Wasser kocht nicht bei 150 °C.
- Die Flüssigkeit ist kein Wasser.
- Die Temperatur beträgt unter 100 °C.

Der Agent entscheidet sich, Letzteres zu glauben und nicht mehr, dass die Temperatur 150 °C beträgt, da er meint, es wäre am wahrscheinlichsten, dass die Grad-Anzeige am Thermometer in Fahrenheit angezeigt worden sein könnte. (150 °F entsprechen etwa 66 °C.) Auf diese Weise erhält er ein Modell der Welt, das in diesem Moment zu seinen Beobachtungen und Erfahrungen am besten passt.

Zu einem späteren Zeitpunkt könnte er feststellen, dass das Wasser plötzlich doch kocht und muss erneut sein Wissen revidieren, z. B. indem er die zu einfache Regel 'Wenn Wasser und heiß, dann kocht es.' verfeinert: 'Wenn Wasser und (heiß oder Luftdruck niedrig), dann kocht es.' (Wasser kocht im Vakuum schon weit unter 100 °C.)

An Hand dieser Beispiele wird auch klar, dass man zwischen 'Wissen' und 'Geglaubtem' unterscheiden sollte:

Definition 3.1.1:

'Wissen' sind sichere Tatsachen (oder zumindest Tatsachen, von denen man ausgeht, dass sie sicher sind), z. B. im mittelalterlichen Europa: 'Die Erde ist eine Scheibe.'

Definition 3.1.2:

'Geglaubtes' sind Vermutungen, die man im Moment für richtig hält, die man aber auch ohne sein Weltbild ins Wanken zu bringen aufgeben kann, z. B. 'Morgen wird es regnen.'.

Definition 3.1.3:

‘Belief Revision’ ist der Vorgang, Geglaubtes beim Hinzukommen neuer Information so anzupassen, dass hierdurch entstehende Widersprüche aufgelöst werden und das anschließend Geglaubte konsistent (d. h. widerspruchsfrei) und

wahrscheinlich richtig ist.

Der deutsche Begriff für ‘Belief Revision’ lautet untreffender Weise ‘*Wissens*revision’. Da aber nicht Wissen, sondern Geglaubtes revidiert wird, wird in dieser Arbeit der englische Ausdruck verwendet.

3.2 Logiken und Inferenzsysteme

3.2.1 Klassische Logik

Die klassische (Aussagen-)Logik ist die Grundlage aller Logiken. Ausdrücke in der klassischen Logik sind entweder ganz wahr oder ganz falsch. Gängige Symbole/ Operatoren und ihre Bedeutungen, um klassisch logische Ausdrücke zu bilden, sind der folgenden Tabelle zu entnehmen (vgl. [BuK2006, S. 42]).

Tabelle 3.2.1.1:

Syntax	in Worten	Semantik (... wahr genau dann, wenn:)
$\neg A$ (auch: $\overline{A}$)	not A	die Negation von A ist wahr (nicht: A ist nicht wahr)
$A \vee B$	A oder B	mindestens einer der beiden Ausdrücke A, B ist wahr
$A \wedge B$ (auch: AB)	A und B	beide Ausdrücke A, B sind wahr
$A \Rightarrow B$	aus A folgt B	wenn A wahr ist, ist auch B wahr
$A \Leftrightarrow B$	A und B sind äquivalent	A und B haben immer denselben Wahrheitswert

Definition 3.2.1.1:

Eine ‘atomare Formel’ ist eine Formel, die nur aus einer Bool’schen Variable besteht. Z. B. sind A, B und C atomare Formeln.

Eine nicht-negierte oder negierte atomare Formel wird als ‘Literal’ bezeichnet. Z. B. sind A, $\overline{B}$ und C Literale.

Eine ‘Formel’ ist ein aus Literalen und Operatoren (syntaktisch korrekt) zusammengesetzter Ausdruck, z. B. $\left(A \vee \overline{B}\right) \wedge C$.

(Vgl. BuK2006, S. 42.)

Definition 3.2.1.2:

Mit '*L*' ist die Menge aller logischen Formeln (über den im konkreten Fall gegebenen Variablen) gemeint.

Definition 3.2.1.3:

Eine 'Tautologie' ist eine Formel, die immer wahr ist, z. B. $A \vee \overline{A}$.

Definition 3.2.1.4:

Ein 'Widerspruch' ist eine Formel, die immer falsch ist, z. B. $A \wedge \overline{A}$.

Es lassen sich auch besondere Formeln, nämlich Regeln erstellen:

Definition 3.2.1.5:

Eine 'klassische Regel' hat die Form: $\textit{Prämisse} \Rightarrow \textit{Konsequenz}$, wobei *Prämisse* und *Konsequenz* Formeln sind.

Definition 3.2.1.6:

Eine Regel kann 'feuern', wenn ihre Prämisse erfüllt ist.

Definition 3.2.1.7:

Eine 'Knowledge Base' (KB) ist eine Menge von als wahr angenommenen Formeln, insbesondere auch Regeln.

Mit Hilfe einer KB lässt sich logisch schlussfolgern und so Information, welche nicht explizit, sondern nur implizit in der KB enthalten ist, erlangen.

Definition 3.2.1.8:

Lässt sich aus einer KB *KB* eine Formel *B* folgern, so schreibt man: '$KB \models B$'.

Die bekannteste Inferenz-Regel ist der Modus Ponens: $A \wedge (A \Rightarrow B) \models B$. Inferenz-Regeln werden üblicher Weise anders notiert, der Modus Ponens z. B. durch: $\frac{A, A \Rightarrow B}{B}$.

Die klassische Logik benutzt den Modus Ponens um von Bekanntem auf Neues zu schließen. Dabei gelten folgende Eigenschaft für die Inferenz:

- Inklusion: Aus der KB *KB* folgt alles, was explizit in ihr steht, d. h. $B \in KB$

impliziert $KB \models B$.

- Schnitteigenschaft: Wenn aus *KB* eine Formel *A* folgt und durch Hinzunahme von *A* zu *KB* anschließend *B* folgt, dann folgt *B* bereits aus *KB*, d. h. $KB \models A$ und $KB \cup \{A\} \models B$ impliziert $KB \models B$.
- Monotonie: Wenn *KB* ergänzt wird, so folgt immer noch alles, was zuvor schon folgte, d. h. $KB \models B$ impliziert $KB \cup \{A\} \models B$.

Die Monotonie-Eigenschaft macht, obwohl sie auf den ersten Blick sinnvoll erscheint, Belief Revision unmöglich, denn sie besagt, dass man einmal Geglaubtes in Zukunft immer glauben muss. Daher eignen sich grundsätzlich nur nicht-monotone Logiken für Belief Revision.

Definition 3.2.1.9:

Eine Menge *M* von Formeln heißt 'deduktiv abgeschlossen', wenn aus ihr nichts Neues gefolgert werden kann, also wenn $M \models B$ impliziert, dass $B \in M$. Man bezeichnet den 'deduktiven Abschluss' einer Menge von Formeln (z. B. einer KB) *KB* als $Cn(KB)$. *M* ist also deduktiv abgeschlossen, wenn $M = Cn(M)$.

Definition 3.2.1.10:

Ein 'Modell' ist eine Konjunktion von Literalen *aller* vereinbarten Variablen. Gibt es z. B. genau die Variablen *A*, *B* und *C*, so ist $A\overline{B}C$ ein mögliches Modell.

Definition 3.2.1.11:

Der 'Raum aller Modelle Ω' ist die 2^n-elementige Menge aller möglichen Modelle, wobei *n* die Anzahl der vereinbarten Variablen ist.

Definition 3.2.1.12:

Ein Modell ist ein 'Modell einer KB *KB*', wenn seine Variablenbelegungen mit den Folgerungen aus *KB* verträglich sind. Folgt aus *KB* beispielsweise *A* und $\overline{B}$, jedoch weder *C* noch $\overline{C}$, so sind genau die Modelle $A\overline{B}C$ und $A\overline{B}\overline{C}$ Modelle von *KB*.

Ein Modell ist ein 'Modell einer Formel *F*', wenn *F* für das Modell den Wahrheitswert *true* liefert. Ist z. B. $F = A\overline{B}$ und sind insgesamt die Variablen *A*, *B* und *C* vereinbart, so sind genau die Modelle $A\overline{B}C$ und $A\overline{B}\overline{C}$ Modelle von *F*.

In der Literatur wird eine Belegung aller vereinbarten Variablen mitunter nicht als ‘Modell’ bezeichnet, sondern als ‘Interpretation’. Der Begriff ‘Modell’ wird dann nur in Verbindung mit einem anderen Objekt *O* (z. B. einer KB oder einer Formel) in der Form ‘Eine Interpretation ist ein Modell eines Objektes *O*, wenn ...’ verwendet.

3.2.2 Reiter’s Default-Logik und Antwortmengenprogrammierung

Viele Nachteile der klassischen Logik werden durch Reiter’s Default-Logik [Rei1980] behoben. Insbesondere ist sie nicht-monoton.

Die Idee ist, Regeln ähnlich der klassischen Regeln zu benutzen, doch das Feuern so einzuschränken, dass es zwar standardmäßig geschieht, wenn die Prämisse erfüllt ist, man jedoch Ausnahmen angeben kann.

Definition 3.2.2.1:

Eine ‘Default-Regel’ hat die Form $\frac{Prämisse : Annahme_1, \ldots, Annahme_n}{Konsequenz}$ mit der Bedeutung: Wenn die Prämisse erfüllt ist und von keiner Annahme die Negation *bekannt* ist, dann kann die Regel feuern und es folgt die Konsequenz.

Beispiel 3.2.2.1:

Die Regel $\frac{GerätEingeschaltet : GerätHatStrom, GerätFunktionsfähig}{GerätLäuft}$ besagt, dass, wenn das Gerät eingeschaltet wurde, man prinzipiell davon ausgehen soll, dass es auch läuft, außer wenn man weiß oder zu einem späteren Zeitpunkt erfährt, dass das Gerät keinen Strom hat oder dass es beschädigt ist.

Außerdem wird Geglaubtes von den Regeln getrennt, d. h. bei der Inferenz wird eine Menge aus Geglaubtem *W* und eine Regel-Menge Δ verwaltet.

Definition 3.2.2.2:

Das Paar $T := (W, \Delta)$ heißt ‘Reiter’s Default-Theorie’.

Das Schlussfolgern in Reiter’s Default Logik geschieht mittels folgendem Algorithmus. Er verwendet eine anfangs leere Menge *Out*, in der sich der Algorithmus merkt, was sich auf Grund bereits gemachter Annahmen nicht bewahrheiten darf.

Algorithmus 3.2.2.1 (Default-Logik-Inferenz)

1. $In := Cn(W)$, $Out := \{\ \}$, $\Delta_{tmp} := \Delta$

2. Do While 'Es gibt eine Regel in Δ_{tmp}, die feuern kann.':

2.1. Wähle eine Regel $\frac{Prämisse : Annahme_1,\ldots,Annahme_n}{Konsequenz}$ aus Δ_{tmp}, welche unter dem momentan Geglaubten *In* feuern kann.

2.2. Füge die Konsequenz der Regel der Menge *In* hinzu ($In := Cn(In \cup Konsequenz)$), füge die Negationen der Annahmen der Regel der Menge *Out* hinzu.

2.3. If $In \cap Out \neq \{\ \}$: Fehlschlag und Abbruch, denn es hat sich die Negation einer Annahme, die zuvor gemacht wurde, als wahr herausgestellt.

2.4. Else: Entferne die gewählte Regel aus Δ_{tmp}, denn eine nochmalige Anwendung der Regel ist überflüssig, da die Konsequenz bereits glaubt wird.

Der Algorithmus endet regulär (d. h. nicht durch einen in 2.3 verursachten Abbruch), wenn keine Regel mehr feuern kann. In diesem Fall wurde, beginnend mit dem vorgegebenem Wissen *W*, alles gefolgert, was gefolgert werden konnte, und ist in *In* gespeichert. Doch dieses Ergebnis ist abhängig von der Wahl der nächsten Regel in Schritt 2.1, wie das folgende Beispiel zeigt:

Beispiel 3.2.2.2:

Es ist bekannt, dass Präsident Nixon ein Republikaner war. Ebenso war er Quaker. Also: $W := \{Republikaner, Quaker\}$. Außerdem sind republikanische Präsidenten i. d. R. keine Pazifisten. Daher die Regel: $\frac{Republikaner : \neg Pazifist}{\neg Pazifist}$. Schließlich sind Quaker i. d. R. Pazifisten: $\frac{Quaker : Pazifist}{Pazifist}$. Also: $\Delta := \left\{ \frac{Republikaner : \neg Pazifist}{\neg Pazifist}, \frac{Quaker : Pazifist}{Pazifist} \right\}$.

Zu Beginn können beide Regeln feuern. Wählt man die erste Regel zuerst, erhält man $In = Cn\{Republikaner, Quaker, \neg Pazifist\}$, $Out = \{Pazifist\}$ und $\Delta_{tmp} := \left\{ \frac{Quaker : Pazifist}{Pazifist} \right\}$. Weil $\neg Pazifist$ Element von *In* ist, kann die übrig

gebliebene Regel nun nicht mehr feuern. Der Algorithmus endet regulär.

Wählt man die zweite Regel zuerst, erhält man $In = Cn\{Republikaner, Quaker, Pazifist\}$, $Out = \{\neg Pazifist\}$ und $\Delta_{tmp} := \left\{\frac{Republikaner : \neg Pazifist}{\neg Pazifist}\right\}$. Wieder kann die übrig gebliebene Regel nicht mehr feuern. Der Algorithmus endet regulär.

Je nach gewählter Reihenfolge erhält man als Geglaubtes (im ersten Fall) $In = Cn\{Republikaner, Quaker, \neg Pazifist\}$ oder (im zweiten Fall) $In = Cn\{Republikaner, Quaker, Pazifist\}$.

Definition 3.2.2.3:

Das vom Default-Logik-Inferenz-Algorithmus beim regulären Beenden berechnete Ergebnis *In* nennt man eine 'Extension der Default-Theorie $T = (W, \Delta)$'.

Der Algorithmus berechnet i. d. R. unterschiedliche Extensionen. Weil es keinen Grund gibt, eine Extension einer anderen vorzuziehen, glaubt man letztendlich nur die Schnittmenge aller möglichen Extensionen.

Die Antwortmengenprogrammierung (answer set programming, ASP) ist Reiter's Default-Logik sehr ähnlich. Hier wird ein neuer Operator eingeführt.

Definition 3.2.2.4:

Mit 'not *A*' wird ausgedrückt, dass man die *Annahme* $\overline{A}$ macht. ($\overline{A}$ hingegen bedeutet, dass man $\overline{A}$ *weiß*.)

Definition 3.2.2.5:

Eine 'ASP-Regel' hat die Form $C \leftarrow A_1,\ldots,A_m,\text{not}B_1,\ldots,\text{not}B_n$ mit der Bedeutung: Wenn alle A_i wahr sind und man keines der B_i weiß, man also ihre Negationen annehmen kann, dann kann die Regel feuern und es folgt *C*.

Die ASP-Regel $C \leftarrow A_1,\ldots,A_m,\text{not}B_1,\ldots,\text{not}B_n$ lässt sich demnach in die Default-Regel $\frac{A_1 \wedge \ldots \wedge A_m : \overline{B}_1,\ldots,\overline{B}_n}{C}$ übersetzen.

3.3 Belief Revision – Theorie

3.3.1 Knowledge Bases, Belief Sets und Belief States

Es gibt verschiedene Arten, Geglaubtes zu repräsentieren. In Kapitel 3.2 wurden bereits KBs vorgestellt.

Definition 3.3.1.1:

Ein ‘Belief Set’ ist eine deduktiv abgeschlossene Menge von Formeln, die geglaubt werden. Ist z. B. KB eine KB, so ist $Cn(KB)$ ein Belief Set.

Für die Belief Revision spielt die so genannte epistemische Verwurzelung eine große Rolle. Darunter ist zu verstehen, wie stark man etwas glaubt bzw. wie stark man etwas bezweifelt.

Beispiel 3.3.1.1:

Im Beispiel 3.1.2 gab der Agent den Glauben daran auf, dass die Temperatur 150 °C betrug, denn dieses war bei ihm am wenigsten stark verwurzelt. Der Glaube daran, dass Wasser bei 100 °C kocht war epistemisch tiefer verwurzelt und wurde beibehalten.

Wird das Geglaubte ausschließlich durch eine KB oder ein Belief Set repräsentiert, so fehlt die epistemische Komponente. Es lässt sich zwar theoretisch auch so Revision betreiben, aber in diesem Fall wäre für den Agenten alles gleich tief verwurzelt. Er könnte also nicht das Löschen einer Information dem Löschen einer anderen bevorzugen. Daher wird die epistemische Information entweder extern bereitgestellt (z. B. in Form einer Relation, die regelt, welche von zwei Formeln glaubhafter ist als die andere) oder wird direkt in das Belief Set integriert.

Definition 3.3.1.2:

Ein 'Belief State' (auch 'Epistemic State' genannt) ist eine (Darstellung einer) Menge von Formeln mit der zusätzlichen Information, wie tief jede Formel epistemisch verwurzelt ist.

Wird epistemische Information verwendet, muss man zusätzlich definieren, wie tief eine Information verwurzelt sein muss, damit der Agent sie glauben darf bzw. muss.

Definition 3.3.1.3:

Ist E ein Belief State, so bezeichnet '$Bel(E)$' das Belief Set, welches der Agent glaubt.

3.3.2 AGM-Theorie

Die AGM-Theorie [AGM1985], benannt nach ihren Erfindern Alchourrón, Gärdenfors und Makinson, formuliert verschiedene Postulate für den Umgang mit neuer Information bzw. Geglaubtem. Es wird unterschieden zwischen der Expansion, Revision und Kontraktion.

Die Expansion bezeichnet die einfache Hinzunahme neuer Information. Sie überprüft nicht, ob die neue Information mit dem bereits Geglaubten vereinbar ist.

Mit Revision ist die Hinzunahme neuer Information mit entsprechender Anpassung des Geglaubten gemeint, so dass nach der Revision das Geglaubte konsistent ist.

Unter Kontraktion versteht man das Löschen einer Information aus dem Geglaubten. Insbesondere kommt nicht neue Information hinzu.

Sei im Folgenden K ein Belief Set und seien A und B einzelne Informationen.

Die AGM-Postulate (AGM +) für die Expansion lauten:

1. $K + A$ ist ein Belief Set. (Nach der Expansion eines Belief Set soll wieder ein Belief Set entstehen, damit die deduktive Abgeschlossenheit weiterhin gesichert ist.)
2. $A \in K + A$ (Die neue Information soll aufgenommen werden.)
3. $K \subseteq K + A$ (Es soll nichts gelöscht werden.)
4. $A \in K$ impliziert $K + A = K$. (Wenn A bereits in K liegt, ist nichts zu tun.)
5. $K \subseteq H$ impliziert $K + A \subseteq H + A$.
6. $K + A$ ist das kleinste Belief Set, so dass AGM +1 bis AGM +5 erfüllt sind. (Es soll nach der Expansion nicht mehr geglaubt werden als unbedingt nötig.)

Durch die AGM-Postulate ist die Expansion wie von [GuR1995] gezeigt eindeutig definiert durch:

$$K + A := Cn(K \cup \{A\})$$

Für die Revision sollen folgende Postulate (AGM *) gelten:

1. $K * A$ ist ein Belief Set. (Nach der Revision eines Belief Set soll wieder ein Belief

Set entstehen, damit die deduktive Abgeschlossenheit weiterhin gesichert ist.)

2. $A \in K * A$ (Die neue Information soll aufgenommen werden.)
3. $K * A \subseteq K + A$ (Nach einer Revision soll nicht mehr geglaubt, als nach einer einfachen Expansion geglaubt würde.)
4. $\overline{A} \notin K$ impliziert $K + A \subseteq K * A$. (Dieses Postulat ist im Zusammenhang mit AGM *3 zu sehen: Wenn nichts gegen A spricht, d. h. wenn $\overline{A} \notin K$, dann soll eine Expansion durchgeführt werden, denn zusammen mit AGM *3 gilt dann: $K * A = K + A$.)
5. $K * A$ ist konsistent genau dann, wenn A konsistent ist. (Widerspruchsfreie Information darf nicht zu einem inkonsistenen Belief Set führen.)
6. $A \equiv B$ impliziert $K * A = K * B$. (Dieselbe Information könnte unterschiedlich dargestellt sein, wie z. B. $P \Rightarrow F \equiv \overline{P} \vee F$. Dies darf nicht zu unterschiedlichen Belief Sets führen.)
7. $K * (A \wedge B) \subseteq (K * A) + B$ (vgl. AGM *3)
8. $\overline{B} \notin K * A$ impliziert $(K * A) + B \subseteq K * (A \wedge B)$ (vgl. AGM *4).

Die Postulate AGM *1 bis AGM *6 nennt man die Basispostulate der Revision.

Die Kontraktion wird geregelt durch diese Postulate:

1. $K - A$ ist ein Belief Set. (Nach der Kontraktion eines Belief Set soll wieder ein Belief Set entstehen, damit die deduktive Abgeschlossenheit weiterhin gesichert ist.)
2. $K - A \subseteq K$ (Es soll nur Information vergessen werden, aber keine neue hinzukommen.)
3. $A \notin K$ impliziert $K - A = K$. (Wenn A gar nicht in K liegt, ist nichts zu tun.)
4. Wenn A keine Tautologie ist, dann gilt: $A \notin K - A$. (Nach der Kontraktion soll A nicht mehr geglaubt werden.)
5. $K \subseteq (K - A) + A$ (Wird eine Information gelöscht und später wieder hinzugefügt, so soll das ursprünglich Geglaubte ausnahmslos wieder geglaubt werden.)
6. $A \equiv B$ impliziert $K - A = K - B$. (Dieselbe Information könnte unterschiedlich dargestellt sein, wie z. B. $P \Rightarrow F \equiv \overline{P} \vee F$. Dies darf nicht zu unterschiedlichen Belief Sets führen.)
7. $(K - A) \cap (K - B) \subseteq K - (A \wedge B)$
8. $A \notin K - (A \wedge B)$ impliziert $K - (A \wedge B) \subseteq K - A$.

Die Postulate AGM -1 bis AGM -6 nennt man die Basispostulate der Kontraktion.

AGM-Expansion, -Revision und -Kontraktion verfolgen das so genannte Minimal-Change-Paradigma, welches besagt, dass durch diese Operationen nur so wenig an *K* geändert werden soll, wie absolut notwendig.

Nach der Levi-Identität ([Lev1977.1], [Lev1977.2]) und der Harper-Identität [Har1977] sind Revision und Kontraktion zueinander dual. Das bedeutet, dass, wenn ein AGM-Kontraktionsoperator gegeben ist, so kann daraus ein AGM-Revisionsoperator konstruiert werden und anders herum [GuR1995]:

- Levi-Identität: $K * A = \left(K - \overline{A}\right) + A$
- Harper-Identität: $K - A = \left(K * \overline{A}\right) \cap K$

Die AGM-Postulate stellen allerdings nur eine grobe Vorgabe für Belief Revision dar, denn sie regeln erstens nur Operationen auf Belief Sets und lassen zweitens, wie das folgende Beispiel zeigt, noch zu viele Freiheitsgrade zu:

Beispiel 3.3.2.1:

Man betrachte den Revisionsoperator $K * A := \begin{cases} K + A, \text{falls } K \cup \{A\} \text{ konsistent ist} \\ Cn(\{A\}), \text{sonst} \end{cases}$.

Dieser erfüllt die AGM-Postulate, ist jedoch ein ungeeigneter Operator, denn sobald eine Information *A* aufgenommen werden soll, die zum bisher Geglaubtem im Widerspruch steht, wird alles restlos vergessen und nur noch *A* (und alles, was aus *A* folgt) geglaubt.

3.3.3 Revision von Formeln

Ein Belief Set lässt sich beschreiben durch eine Menge von Formeln. Oft ist ein Belief Set gerade durch eine Menge von Formeln (z. B. eine KB) definiert (siehe Definition 3.3.1.1). Daher ist es sinnvoll sich zu überlegen, wie Revision direkt auf Formel-Ebene betrieben werden kann.

Hier gibt es zum Einen die Basisrevision, d. h. Revision von KBs, welche durch die Postulate von Hansson ([Han1991], [Han1996], [K-I2005, Kapitel 1.6]) ein theoretisches Fundament erhält. Auf diese Art der Revision wird in dieser Arbeit jedoch nicht näher eingegangen. Zum Anderen gibt es die Formelrevision.

Bei der Formelrevision ist das Geglaubte durch eine einzige Formel repräsentiert,

z. B. durch $A \wedge B \wedge (\overline{B} \vee C)$, welche dann mit neuer Information revidiert wird, wodurch eine neue Formel entsteht, welche das *dann* Geglaubte repräsentiert.

Ist ein Belief Set *K* durch eine KB *KB* gegeben, welche aus den Formeln $A_1,\ldots,A_n$ besteht, dann ist $K = Cn(KB) = Cn(\{A_1,\ldots,A_n\}) = Cn(A_1 \wedge \ldots \wedge A_n)$. *K* ist also auch durch die Formel $A_1 \wedge \ldots \wedge A_n$ bestimmt.

Definition 3.3.3.1:

Ist $K = Cn(A_1 \wedge \ldots \wedge A_n)$, so nennt man die Formel $A_1 \wedge \ldots \wedge A_n$ den 'Core von *K*' ($Core(K)$).

Demnach ist es immer möglich, eine KB in eine Formel zu konvertieren und anders herum. $Core(K)$ ist bis auf Äquivalenz eindeutig bestimmt.

Ist ein Belief Set *K* durch die Menge der geglaubten Modelle gegeben, so gilt $Core(K) = \bigvee_{\omega \in Mod(K)} \omega$.

Sei $F = Core(K)$. Dann beschreiben folgende Postulate (RF) von Katsuno und Mendelzon ([KuM1991], [DuP1996, S. 3], [K-I 2005, Kapitel 1.5]) eine Grundlage für die Formelrevision:

1. $F * A \models A$ (vgl. AGM *2)
2. Wenn $F \wedge A$ erfüllbar ist, dann gilt: $F * A \equiv F \wedge A$ (vgl. AGM *3 und AGM *4).
3. Wenn *A* erfüllbar ist, dann ist auch $F * A$ erfüllbar (vgl. AGM *5).
4. Wenn $F_1 \equiv F_2$ und $A_1 \equiv A_2$, dann gilt: $F_1 * A_1 \equiv F_2 * A_2$ (vgl. AGM *6).
5. $(F * A) \wedge B \models F * (A \wedge B)$ (vgl. AGM *7)
6. Wenn $(F * A) \wedge B$ erfüllbar ist, dann gilt: $F * (A \wedge B) \models (F * A) \wedge B$ (vgl. AGM *8)

Die RF-Postulate sind äquivalent zu den AGM-Postulaten der Revision ([KuM1991], [DuP1996, S. 3], [K-I 2005, Kapitel 1.5]). Daher lässt sich ein RF-Revisionsoperator für Formeln in einen AGM-Revisionsoperator für Belief Sets umwandeln und anders herum:

- $K * A := Cn(Core(K) * A)$
- $F * A := Core(Cn(F) * A)$

3.3.4 Revision von Belief States

Die Revision von Belief States S orientiert sich weitestgehend an der Revision von Formeln und damit an der AGM-Revision. Die Postulate für die Epistemic State Revision (ESR) lauten ([DuP1996], [K-I2005, Kapitel 2.1]) (vgl. RF-Postulate):

1. $Bel(S * A) \models A$
2. Wenn $Bel(S) \wedge A$ erfüllbar ist, dann gilt: $Bel(S * A) \equiv Bel(S) \wedge A$.
3. Wenn A erfüllbar ist, dann ist auch $Bel(S * A)$ erfüllbar.
4. Wenn $S_1 = S_2$ und $A_1 \equiv A_2$, dann gilt: $Bel(S_1 * A_1) \equiv Bel(S_2 * A_2)$.
5. $Bel(S * A) \wedge B \models Bel(S * (A \wedge B))$
6. Wenn $Bel(S * A) \wedge B$ erfüllbar ist, dann gilt: $Bel(S * (A \wedge B)) \models Bel(S * A) \wedge B$.

Die ESR-Postulate sind nahezu identisch zu den RF-Postulaten. Es gibt nur einen Unterschied im 4. Postulat, bei dem $S_1 = S_2$ gefordert wird und nicht, wie es bei einer 1:1-Übersetzung von RF 4 lauten würde, $Bel(S_1) \equiv Bel(S_2)$. Diese Forderung wäre, wie aus dem nächsten Beispiel ersichtlich wird, zu schwach und würde zu unintuitiven Ergebnissen führen.

Beispiel 3.3.4.1:

Zwei New York Touristen möchten das Empire State Building besuchen. Tourist 1 befindet sich an der blau markierten Stelle, Tourist 2 an der rot markierten:

Abbildung 3.3.4.1 (Quelle: http://maps.google.com)

Tourist 1 ist im Belief State S_1 und glaubt, dass sein Ziel im Nordosten liegt. Er hält es für möglich, dass es im Südosten liegt, glaubt es aber nicht. Dass es im Nordwesten liegt, schließt er aus.

Tourist 2 ist im Belief State S_2 und glaubt ebenfalls, dass sein Ziel im Nordosten liegt. Er hält es für möglich, dass es im Nordwesten liegt, glaubt es aber nicht. Dass es im Südosten liegt, schließt er aus.

Es gilt demnach $Bel(S_1) \equiv Bel(S_2)$, aber nicht $S_1 = S_2$. Nach einiger Zeit des Laufens in nordöstliche Richtung revidieren beide Touristen mit der Information 'Das Ziel liegt nicht im Nordosten', da sie beide meinen, sie sind bereits weit genug nach Nordosten gelaufen und das Gebäude nun nicht mehr im Nordosten liegen kann. Würde die Bedingung des 4. ESR-Postulats $Bel(S_1) \equiv Bel(S_2)$ und nicht $S_1 = S_2$ sein, müssten auch nach der Revision beide dasselbe glauben, obwohl laut Beschreibung von S_1 bzw. S_2 intuitiv klar ist, dass Tourist 1 nun glauben wird, dass sein Ziel im Südosten liegt, bzw. dass Tourist 2 nun glauben wird, dass sein Ziel im Nordwesten liegt.

Ursache dafür ist, dass Belief States auch Information für nicht Geglaubtes enthalten. Wird durch Revision erzwungen, dass etwas nicht Geglaubtes geglaubt werden soll, so muss bei zwei unterschiedlichen Belief States im Anschluss an die Revision nicht notwendiger Weise dasselbe geglaubt werden, auch wenn vor der Revision das Geglaubte bei beiden identisch war. ESR 4 muss also $S_1 = S_2$ fordern und nicht nur $Bel(S_1) \equiv Bel(S_2)$.

3.4 Belief Revision – Praxis: Revision von Belief Sets

3.4.1 Revision mittels Selektionsfunktionen

Mit Selektionsfunktionen lässt sich eine Kontraktion definieren, aus welcher sich dann durch Verwendung der Levi-Identität eine Revision ergibt. Grundlegend ist hier der Begriff der 'maximalen Kontraktionsmenge'.

Definition 3.4.1.1:

Sei K ein Belief Set und A eine Information, welche nach der Kontraktion nicht (mehr) gefolgert werden können soll. M heißt 'maximale Kontraktionsmenge von K bzgl. A', wenn gilt:

- $M \subseteq K$ (Es soll nur Information vergessen werden, aber keine neue hinzukommen.)
- $A \notin Cn(M)$ (A kann nicht (mehr) aus M gefolgert werden.)
- $M \subset M' \subseteq K$ impliziert $A \in Cn(M')$. (Auf Grund des Minimal-Change-Paradigmas soll nur so wenig an K geändert werden, wie absolut notwendig. Es soll also nicht mehr gelöscht werden, als unbedingt nötig.)

Definition 3.4.1.2:

'$K \perp A$' bezeichnet die Menge aller maximalen Kontraktionsmengen von K bzgl. A.

Maximale Kontraktionsmengen sind prinzipiell geeignet, das Ergebnis einer Kontraktion $K - A$ zu sein. Eine Selektionsfunktion S wählt nun aus $K \perp A$ einige beste Kontraktionsmengen aus und man definiert:

Definition 3.4.1.3:

Die durch $K - A := \bigcap S(K \perp A)$ definierte Kontraktion nennt man 'Partial Meet-

Kontraktion'.

Der so definierte Kontraktionsoperator erfüllt die AGM-Basispostulate der Kontraktion [AGM1985]. Man kann eine Partial Meet-Kontraktion, die alle AGM-Postulate der Kontraktion erfüllt, gewinnen, indem man eine transitive Relation auf der Menge aller maximalen Kontraktionsmengen definiert.

Definition 3.4.1.4:

Die Menge aller maximalen Kontraktionsmengen von K (nun nicht mehr bzgl. einer Information A) ist '$M(K) := \bigcup_{A \in K, A \text{ keine Tautologie}} K \perp A$'.

Definition 3.4.1.5:

Seien $\leq$ eine *transitive* Relation über $M(K)$ und S definiert durch $S(K \perp A) := \{M \in K \perp A \mid \nexists M' \in K \perp A : M' > M\}$, dann heißt S 'relationale Selektionsfunktion' und die induzierte Partial Meet-Kontraktion 'relationale Partial Meet-Kontraktion'.

Relationale Partial Meet-Kontraktionen erfüllen alle AGM-Postulate der Kontraktion [AGM1985]. Mittels der Levi-Identität ergibt sich ein AGM-Revisions-Operator.

3.4.2 Revision mittels epistemischen Verwurzelungsrelationen oder -funktionen auf Formeln

Eine weitere Möglichkeit, Belief States K zu revidieren, bieten epistemische Verwurzelungsrelationen auf Formeln. Ist eine solche definiert worden, ist es möglich, zwei Formeln miteinander zu vergleichen und zu entscheiden, welche von beiden glaubhafter ist.

Definition 3.4.2.1:

Bei einer 'epistemischen Verwurzelungsrelation $\leq_E$ auf Formeln' bedeutet '$A \leq_E B$', dass B mindestens so sicher wie A angenommen wird.

Erfüllt $\leq_E$ die folgenden Epistemic Entrenchment-Postulate (EE), lassen sich daraus ein AGM-Revisions- und ein AGM-Kontraktionsoperator konstruieren:

1. $A \leq_E B, B \leq_E C$ impliziert $A \leq_E C$. ($\leq_E$ ist transitiv.)

2. $A \models B$ impliziert $A \leq_E B$. (Konsequenzen müssen mindestens so stark geglaubt werden wie ihre Prämissen.)
3. $A \leq_E A \wedge B$ oder $B \leq_E A \wedge B$ (Ziel ist die Eigenschaft $A_1 \wedge \ldots \wedge A_n =_E \min_i A_i$, welche dem Sprichwort 'Eine Kette ist genauso stark wie ihr schwächstes Glied.' entspricht. Wegen $A_1 \wedge \ldots \wedge A_n \models A_i$ für alle *i* gilt nach (EE 2) bereits $A_1 \wedge \ldots \wedge A_n \leq_E A_i$ für alle *i*. Dieses Postulat sorgt dafür, dass auch für mindestens ein A_i gilt: $A_1 \wedge \ldots \wedge A_n \geq_E A_i$.)
4. $A \leq_E B \forall B \in L$ genau dann, wenn $A \notin K$. (Alles, was nicht geglaubt werden soll, ist minimal verwurzelt.)
5. Wenn $A \geq_E B \forall B \in L$, dann ist *A* eine Tautologie. (Tautologien sind epistemisch am stärksten verwurzelt unter allen möglichen Formeln.)

Definition 3.4.2.2:

Durch eine epistemische Verwurzelungsrelation, welche die EE-Postulate erfüllt, lässt sich ein passendes Belief Set konstruieren: '$Bel(\leq_E) := \{A \in L : A >_E false\}$'. Es wird also alles geglaubt, was echt tiefer verwurzelt ist als der Widerspruch.

AGM-Operatoren ergeben sich wie folgt [GuM1988]:

- $B \in K - A$ genau dann, wenn $B \in K$ und ($A <_E A \vee B$ oder *A* ist Tautologie).
- $B \in K * A$ genau dann, wenn $\overline{A} <_E \overline{A} \vee B$ und $\overline{A}$ keine Tautologie ist.

Statt einer *Relation* kann auch eine epistemische Verwurzelungs*funktion* verwendet werden. Diese bildet Formeln auf das Intervall $[0;1]$ ab. Je größer die Zahl ist, desto stärker wird die Formel geglaubt. Formeln werden nicht direkt miteinander in Beziehung gesetzt, d. h. verglichen, sondern über die ihnen zugeordneten Werte.

Definition 3.4.2.3:

Eine 'epistemische Verwurzelungsfunktion $E : L \to [0;1]$' hat folgende Eigenschaften (vgl. Definition 3.4.2.1):

- $A \models B$ impliziert $E(A) \leq E(B)$.
- $E(A) \leq E(A \wedge B)$ oder $E(B) \leq E(A \wedge B)$
- $E(false) = 0$
- $E(A) = 1$ genau dann, wenn $A = true$

Der Zielraum muss nicht notwendigerweise das Intervall $[0;1]$ sein. Man kann auch $\mathrm{N} \cup \infty$ wählen.

Definition 3.4.2.4:

Eine 'epistemische, ordinale Verwurzelungsfunktion $E : L \to \mathrm{N} \cup \infty$' hat folgende Eigenschaften (vgl. Definition 3.4.2.3):

- $A \models B$ impliziert $E(A) \leq E(B)$.
- $E(A) \leq E(A \wedge B)$ oder $E(B) \leq E(A \wedge B)$
- $E(false) = 0$
- $E(A) = \infty$ genau dann, wenn $A = true$

Definition 3.4.2.5:

Durch eine epistemische Verwurzelungsfunktion oder eine epistemische, ordinale Verwurzelungsfunktion lässt sich ein passendes Belief Set konstruieren (vgl. Definition 3.4.2.2): '$Bel(E) := \{A \in L : E(A) > 0\}$'.

3.4.3 Revision mittels minimalen Modellen

Die letzte in dieser Arbeit vorgestellte Revisionsmöglichkeit von Belief Sets ist die Revision mittels minimalen Modellen.

Im Raum aller Modelle sind auch die Modelle des Belief Sets K enthalten. Außerdem gibt es die Modelle der neuen Information A. Da $A \in K * A$ sein soll, sind die Modelle von $K * A$ unter den Modellen von A zu suchen. Die Idee ist, $K * A$ durch diejenigen Modelle von A zu definieren, die möglichst 'wenig' von den Modellen von K entfernt sind, damit die durch die Revision verursachte Änderung gemäß dem Minimal-Change-Paradigma so klein wie möglich ist. Folgende Abbildung veranschaulicht das eben Beschriebene.

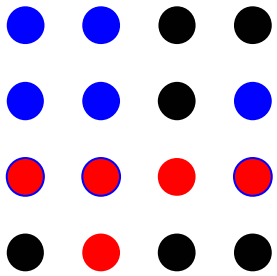


Abbildung 3.4.3.1

Die blauen Kreise repräsentieren die Modelle von K, die roten die von A und die

schwarzen die übrigen Modelle. Hier wären nur die blau umrandeten roten Modelle für die Definition von $K * A$ interessant, da sie am nächsten an K liegen.

Es muss zunächst ein Entfernungsbegriff festgelegt werden. Um die Entfernung von zwei Modellen zueinander bzw. von einem Modell zu einer Menge von Modellen zu definieren, eignet sich das Distanzmaß von Dalal [Dal1988]:

Definition 3.4.3.1:

Seien ω_1 und ω_2 zwei Modelle, dann ist ihre 'Distanz $dist(\omega_1, \omega_2)$ nach Dalal':

$dist(\omega_1, \omega_2) :=$ Anzahl unterschiedlicher Variablenbelegungen bei ω_1 und ω_2

Sei M eine Menge von Modellen und ω_2 ein Modell, dann ist ihre 'Distanz $dist(M, \omega_2)$ nach Dalal':

$$dist(M, \omega_2) := \min_{\omega_1 \in M} dist(\omega_1, \omega_2)$$

Sei nun *minDist* die kleinste vorkommende Entfernung eines Modells von A zu der Modellmenge von K, also $minDist := \min_{\omega \in Mod(A)} dist(Mod(K), \omega)$. Die Revision $K * A$ wird nun definiert durch:

$$Mod(K * A) := \{\omega \in Mod(A) | dist(Mod(K), \omega) = minDist\}$$

Die so definierte Revision ist eine AGM-Revision [Dal1988].

3.5 Belief Revision – Praxis: Revision von Belief States

3.5.1 Ordinale konditionale Rangfunktionen (OCFs)

Die Hauptaufgabe von OCFs [Spo1988] ist es, Belief States zu repräsentieren. Sie enthalten also epistemische Information. Dies geschieht, indem jedem Modell aus Ω eine natürliche Zahl zugeordnet wird, die angibt, wie stark das Modell geglaubt wird. Je kleiner diese Zahl ist, desto tiefer ist das Modell epistemisch verwurzelt. Das Geglaubte wird bestimmt durch die Menge aller Modelle, denen der Wert 0 zugeordnet ist.

Definition 3.5.1.1:

Eine 'ordinale konditionale Rangfunktion' κ ist eine Abbildung $\kappa : \Omega \to \mathbb{N} \cup \infty$,

wobei mindestens ein Modell auf 0 abgebildet wird, damit wenigstens ein Modell geglaubt wird.

Definition 3.5.1.2:

Die Modelle des Geglaubten einer OCF κ sind:

$$Mod(Bel(\kappa)) := \kappa^{-1}(0).$$

Beispiel 3.5.1.1:

In diesem Beispiel soll eine sinnvolle, zu Beispiel 3.1.1 passende OCF κ angegeben werden und diese anschließend revidiert werden. Die Variablen seien festgelegt durch:

- *W*: Die Flüssigkeit ist Wasser.
- *h*: Es ist heiß, d. h. mindestens 100 °C.
- *k*: Die Flüssigkeit kocht.

κ könnte so aussehen:

- $\kappa^{-1}(0) = \{Whk\}$
- $\kappa^{-1}(1) = \{W\overline{h}\,\overline{k}\}$
- $\kappa^{-1}(2) = \{\overline{W}hk, \overline{W}h\overline{k}\}$
- $\kappa^{-1}(3) = \{\overline{W}\,\overline{h}k, \overline{W}\,\overline{h}\,\overline{k}\}$
- $\kappa^{-1}(4) = \{Wh\overline{k}, W\overline{h}k\}$

Erklärung: Der Agent soll die Flüssigkeit als Wasser identifiziert haben, soll auf dem Thermometer die hohe Temperatur gesehen haben und soll annehmen, dass Wasser kocht, wenn es heiß ist, daher $\kappa^{-1}(0) = \{Whk\}$.

Da er dem Thermometer am wenigsten trauen soll, wird $\kappa^{-1}(1) = \{W\overline{h}\,\overline{k}\}$ gesetzt. So ist es immer noch nahe liegend, dass es sich um Wasser handelt und das Wasser kochen würde, wenn es heiß wäre.

Dass es sich bei der Flüssigkeit nicht um Wasser handelt, soll eher unwahrscheinlich sein. Ob diese Flüssigkeit dann kocht oder nicht, ist ungewiss, daher $\kappa^{-1}(2) = \{\overline{W}hk, \overline{W}h\overline{k}\}$.

Damit ein Modell aus $\kappa^{-1}(3) = \{\overline{W}\,\overline{h}k, \overline{W}\,\overline{h}\,\overline{k}\}$ wahr sein kann, hätten gleich zwei

Fehler passieren müssen, nämlich erstmal in der Temperaturmessung und außerdem in der Identifikation der Flüssigkeit. Das ist sehr unwahrscheinlich.

Schließlich ist es so gut wie unmöglich (zumindest soll der Agent dies annehmen), dass Wasser nicht kocht, wenn es heiß ist, oder dass es kocht, wenn es nicht heiß ist. Deswegen liegen diese beiden Modelle in $\kappa^{-1}(4) = \{Wh\bar{k}, W\bar{h}k\}$.

In Beispiel 3.1.2 stellt der Agent fest, dass die Flüssigkeit *nicht* kocht. Er revidiert seinen Belief State mit der Information $\bar{k}$. Das Ergebnis dieser Revision könnte sein:

- $(\kappa * \bar{k})^{-1}(0) = \{W\bar{h}\bar{k}\}$
- $(\kappa * \bar{k})^{-1}(1) = \{Whk, \bar{W}h\bar{k}\}$
- $(\kappa * \bar{k})^{-1}(2) = \{\bar{W}\bar{h}\bar{k}\}$
- $(\kappa * \bar{k})^{-1}(3) = \{\bar{W}hk, Wh\bar{k}\}$
- $(\kappa * \bar{k})^{-1}(4) = \{\bar{W}\bar{h}k\}$
- $(\kappa * \bar{k})^{-1}(5) = \{W\bar{h}k\}$

Der Agent glaubt nun das plausibelste Modell von eben, bei welchem $\bar{k}$ gilt. Genau genommen sind alle Modelle von $\bar{k}$ gleichmäßig so weit nach 'oben' gerutscht, dass ihre relativen Entfernungen zueinander gleich geblieben sind, aber das 'oberste' von ihnen jetzt in $(\kappa * \bar{k})^{-1}(0)$ und damit in $Bel(\kappa * \bar{k})$ liegt. Modelle von *k* sind jeweils um 1 nach 'unten' verschoben worden, so dass keines mehr in $(\kappa * \bar{k})^{-1}(0)$ liegt, aber dennoch ihre relativen Entfernungen erhalten geblieben sind.

Der Glaubensgrad bei OCFs ist zunächst nur für Modelle definiert. Über folgende Formel lassen sich auch Glaubensgrade von beliebigen Formeln bestimmen:

$$\kappa(A \vee B) = \min\{\kappa(A), \kappa(B)\}$$

Beispiel 3.5.1.2:

Angenommen, es sind genau die Variablen *A* und *B* vereinbart. Es gibt also die vier Modelle $\bar{A}\bar{B}$, $\bar{A}B$, $A\bar{B}$ und AB. Außerdem gilt $A \equiv A\bar{B} \vee AB$. $\kappa(A)$ lässt sich berechnen durch $\kappa(A) = \kappa(A\bar{B} \vee AB) = \min\{\kappa(A\bar{B}), \kappa(AB)\}$. Weil $A\bar{B}$ und AB Modelle sind, sind ihre κ-Werte direkt gegeben und $\kappa(A)$ ergibt sich als das Minimum der

beiden Werte.

Mit Hilfe dieser Formel lässt sich auch definieren, wann eine beliebige Formel geglaubt wird.

Definition 3.5.1.3:

'Aus einer OCF κ lässt sich eine Formel A ableiten ($\kappa \models A$)' genau dann, wenn $\kappa(A) = 0$ und $\kappa(\overline{A}) > 0$.

Die Bedingung $\kappa(\overline{A}) > 0$ in dieser Definition ist wichtig, denn wäre auch $\kappa(\overline{A}) = 0$, so gäbe es ein Modell von $\overline{A}$ in $\kappa^{-1}(0)$ und somit wäre $\overline{A}$ nicht ausgeschlossen als möglicher Teilzustand der Welt. Im Fall $\kappa(A) = 0$ und $\kappa(\overline{A}) = 0$ würde also weder A noch $\overline{A}$ geglaubt.

Zwischen OCFs und epistemischen, ordinalen Verwurzelungsfunktion (siehe Definitionen 3.4.2.4 und 3.4.2.5) bestehen Parallelen. Bei diesem Vergleich ist zu beachten, dass $E(A) < E(B)$ bedeutet, dass B plausibler ist als A. Bei OCFs bedeutet $\kappa(A) < \kappa(B)$ jedoch, dass A plausibler ist als B, weil bei epistemischen, ordinalen Verwurzelungsfunktion Aussagen mit großem Wert plausibler sind, während bei OCFs Aussagen mit kleinem Wert plausibler sind. Daher kehrt sich die Bedeutung von '$\leq$' um. Dies spiegelt sich auch in den Definitionen von $Bel(E)$ bzw. $Bel(\kappa)$ wider. Bei $Bel(E)$ wird geglaubt, was einen Wert echt größer als 0 hat, bei $Bel(\kappa)$, was genau den Wert 0 hat.

3.5.2 Revision von OCFs mit propositionaler Information

Soll eine OCF κ mit neuer, propositionaler Information (d. h. Information in Form einer Formel) A revidiert werden, so soll A im Anschluss an die Revision gemäß ESR 1 geglaubt werden. Eine Bedingung dafür, dass A von der neuen OCF $\kappa * A$ geglaubt wird, ist nach Definition 3.5.1.3, dass $(\kappa * A)(A) = 0$ gilt. Dies ist auf Grund der Tatsache, dass $A \equiv \bigvee_{\substack{\omega \in \Omega, \\ \omega \models A}} \omega$, und der Formel $\kappa(A \vee B) = \min\{\kappa(A), \kappa(B)\}$, deren Anwendung hier $(\kappa * A)(A) = \min_{\substack{\omega \in \Omega, \\ \omega \models A}} \{\kappa(\omega)\}$ lautet, genau dann der Fall, wenn es in $(\kappa * A)^{-1}(0)$ ein Modell gibt, das ein Modell von A ist.

Wegen des Minimal-Change-Paradigma sollte nur so wenig wie nötig geändert werden. Daher sollten sich u. a. die relativen Positionen der Modelle von A zueinander nicht ändern. Ein Teil der Revisionsoperation besteht darin, das glaubhafteste Modell von A zu finden, d. h. $\kappa(A)$ zu bestimmen. Dieses Modell soll nach der Revision geglaubt werden, muss also bei $\kappa * A$ den Wert 0 haben. Verschiebt man alle Modelle von A jeweils um $\kappa(A)$ nach 'oben', so gilt wie gewünscht $(\kappa * A)(A) = 0$ und die relativen Positionen sind erhalten geblieben.

Die zweite Bedingung dafür, dass A von der neuen OCF $\kappa * A$ geglaubt wird, ist $(\kappa * A)(\overline{A}) > 0$. Ist $\kappa(\overline{A}) = 0$, so müssen die entsprechenden Modelle von $\overline{A}$ aus der Menge der Modelle, die auf 0 abgebildet werden, entfernt werden. Dies geschieht, indem Modelle von $\overline{A}$ jeweils um mindestens 1 nach 'unten' verschoben werden. Auf diese Weise bleiben auch ihre relativen Positionen zueinander erhalten.

War bereits $\kappa(\overline{A}) > 0$, so ist überhaupt nichts zu tun, denn wegen $\kappa(true) = 0$ (*true* ist per Definition immer wahr und wird daher immer geglaubt) und $\kappa(true) = \kappa(A \vee \overline{A}) = \min\left\{\kappa(A), \underbrace{\kappa(\overline{A})}_{>0}\right\}$ muss $\kappa(A) = 0$ sein. Somit gilt bereits $\kappa \models A$ und wegen ESR 2 oder des Minimal-Change-Paradigmas muss $\kappa * A = \kappa$ gewählt werden.

Definition 3.5.2.1:

'$\kappa * A$' ist gegeben durch $\kappa * A := \kappa$, falls bereits $\kappa \models A$. Ansonsten ergeben sich die neuen Werte der Modelle ω durch:

$$(\kappa * A)(\omega) := \begin{cases} \kappa(\omega) - \kappa(A), \text{ falls } \omega \text{ ein Modell von } A \text{ ist} \\ \kappa(\omega) + 1, \text{ falls } \omega \text{ ein Modell von } \overline{A} \text{ ist} \end{cases}$$

Ein Beispiel für einen so definierten Revisionsoperator ist Beispiel 3.5.1.1.

Die eben definierte Revision erfüllt die ESR-Postulate [DuP1996, Theorem 5].

Die Revision lässt sich gemäß folgender Definition noch verallgemeinern:

Definition 3.5.2.2:

'$\kappa * (A, \alpha)$' bedeutet, dass A mit Sicherheit $\alpha \geq 1$ geglaubt werden soll. Dies

wiederum bedeutet, dass nach der Revision das glaubhafteste Modell von $\overline{A}$ mindestens den Abstand α zum glaubhaftesten Modell von A besitzen soll. Da der Wert von A nach der Revision gleich 0 sein wird, muss demnach $(\kappa^*(A,\alpha))(\overline{A}) \geq \alpha$ sein.

$\kappa^*(A,\alpha)$ ist gegeben durch $\kappa^*(A,\alpha) := \kappa$, falls bereits $\kappa(\overline{A}) \geq \alpha$ (dann muss gelten: $\kappa(A) = 0$ und somit $\kappa \models A$). Ansonsten ergeben sich die neuen Werte der Modelle ω durch:

$$(\kappa^* A)(\omega) := \begin{cases} \kappa(\omega) - \kappa(A), \text{ falls } \omega \text{ ein Modell von } A \text{ ist} \\ \kappa(\omega) + \alpha - \kappa(\overline{A}), \text{ falls } \omega \text{ ein Modell von } \overline{A} \text{ ist} \end{cases}$$

Modelle von $\overline{A}$ müssen jeweils um $\alpha - \kappa(\overline{A})$ verschoben werden, denn $(\kappa^*(A,\alpha))(\overline{A})$ soll nach der Revision gleich α sein. Hat das glaubhafteste Modell von $\overline{A}$ bei κ bereits den Wert $\kappa(\overline{A})$, so muss es (und alle anderen) nur noch um $\alpha - \kappa(\overline{A})$ unglaubhafter werden, damit es insgesamt den Glaubensgrad α haben wird ($\kappa(\overline{A}) + (\alpha - \kappa(\overline{A})) = \alpha$).

Der Fall $(\kappa^*(A,\alpha))(\overline{A}) > \alpha$ kommt nur dann vor, wenn schon $\kappa(\overline{A}) > \alpha$ war.

3.5.3 Konditionale

Bisher wurde vorrangig propositionale Information (z. B. 'Die Flüssigkeit ist Wasser.') sowohl bei der Festlegung des Geglaubten, d. h. der Definition von $Bel(\ldots)$, und der Revision behandelt. Es gibt allerdings auch konditionale Information (z. B. 'Wenn es Wasser ist und es heiß ist, dann kocht es.'), welche nun näher beleuchtet werden soll.

Definition 3.5.3.1

Ein 'Konditional $(B|A)$' ist ähnlich der klassischen Regel $A \Rightarrow B$. Wenn $(B|A)$ und A geglaubt werden, dann ist es auch plausibel, B zu glauben. Im Gegensatz zur klassischen Implikation, die entweder falsch oder richtig ist, sind Konditionale dreiwertig. Sie können entweder falsch oder richtig oder unentscheidbar sein. Außerdem ist es auf Grund der Nicht-Monotonie möglich, dass es bei A in Verbindung mit einem anderen Fakt nicht mehr plausibel ist, B zu glauben, oder

sogar $\overline{B}$ plausibel ist.

OCFs sind auch für die Darstellung und das Verarbeiten von konditionaler Information geeignet.

Definition 3.5.3.2:

Eine 'OCF κ akzeptiert das Konditional $(B|A)$ ($\kappa \models (B|A)$)' genau dann, wenn $B \in Bel(\kappa * A)$. Wie gewünscht wird *B* geglaubt, wenn *A* erfahren wird.

Folgendes Beispiel demonstriert den Unterschied zwischen der klassisch-logischen Regel $A \Rightarrow B$ und dem Konditional $(B|A)$.

Beispiel 3.5.3.1:

Sei κ gegeben durch:

- $\kappa^{-1}(0) = \{\overline{A}\,\overline{B}\,\overline{C}, \overline{A}\,\overline{B}C, \overline{A}BC\}$
- $\kappa^{-1}(1) = \{\overline{A}B\overline{C}, ABC\}$
- $\kappa^{-1}(2) = \{A\overline{B}C, A\overline{B}\,\overline{C}\}$
- $\kappa^{-1}(3) = \{AB\overline{C}\}$

κ akzeptiert das Konditional $(B|A)$, denn $B \in Bel(\kappa * A)$:

- $(\kappa * A)^{-1}(0) = \{ABC\}$
- $(\kappa * A)^{-1}(1) = \{\overline{A}\,\overline{B}\,\overline{C}, \overline{A}\,\overline{B}C, \overline{A}BC, A\overline{B}C, A\overline{B}\,\overline{C}\}$
- $(\kappa * A)^{-1}(2) = \{\overline{A}B\overline{C}, AB\overline{C}\}$

B wird daher plausibel, wenn *A* bekannt wird. Wird allerdings *A* in Verbindung mit $\overline{C}$ erfahren, so ist *B* nicht plausibel:

- $(\kappa * A\overline{C})^{-1}(0) = \{A\overline{B}\,\overline{C}\}$
- $(\kappa * A\overline{C})^{-1}(1) = \{\overline{A}\,\overline{B}\,\overline{C}, \overline{A}\,\overline{B}C, \overline{A}BC, AB\overline{C}\}$
- $(\kappa * A\overline{C})^{-1}(2) = \{\overline{A}B\overline{C}, ABC\}$
- $(\kappa * A\overline{C})^{-1}(3) = \{A\overline{B}C\}$

Tatsächlich ist dann sogar die Negation von *B*, also $\overline{B}$, plausibel.

Ebenso wie für Formeln lässt sich auch für Konditionale ein κ-Wert definieren. Ein gutes Maß dafür, wie sicher man bzgl. *B* gegeben *A* ist und damit wie sicher *B* aus *A* folgt, ist die Differenz $\kappa(AB) - \kappa(A)$. Je kleiner die Differenz ist, desto sicherer ist man bzgl. der Folgerung. Es ist daher sinnvoll zu definieren:

Definition 3.5.3.3:

Der 'Glaubensgrad $\kappa(B|A)$ eines Konditionals' ist

$$\kappa(B|A) := \kappa(AB) - \kappa(A) = (\kappa * A)(AB),$$

also der Glaubensgrad dafür, dass *B zusätzlich* zu *A* gilt, wenn *A* erfahren wird bzw. würde.

Manchmal ist es sinnvoll, eine Einschränkung von κ auf eine Teilmenge aller Modelle (z. B. die Menge aller Modelle, bei denen eine bestimmte Variable nicht negiert ist) zu betrachten.

Definition 3.5.3.4:

Die 'Konditionierung von κ auf eine Formel *F*' ist für Modelle ω von *F* definiert durch:

$$\kappa_F(\omega) := \kappa(\omega) - \kappa(F)$$

Modelle, die nicht Modelle von *F* sind, werden nicht betrachtet und erhalten daher keinen κ_F-Wert.

Beispiel 3.5.3.2:

Sei κ definiert durch:

- $\kappa^{-1}(0) = \{\overline{A}\,\overline{B}\}$
- $\kappa^{-1}(1) = \{\overline{A}B, A\overline{B}\}$
- $\kappa^{-1}(2) = \{AB\}$

Dann ist die Konditionierung von κ auf die Formel *A*:

- $\kappa_A^{-1}(0) = \{A\overline{B}\}$
- $\kappa_A^{-1}(1) = \{AB\}$

3.5.4 Revision von OCFs mit konditionaler Information

Zusätzlich zur Revision mit propositionaler Information lassen OCFs auch eine Revision mit Konditionalen zu. Nach der Revision mit $(B|A)$ soll gelten:

$$(\kappa * (B|A))(A\overline{B}) > (\kappa * (B|A))(AB).$$

(D. h. das Auftreten von *A* zusammen mit $\overline{B}$ ist weniger plausibel als das Auftreten von *A* zusammen mit *B*.)

Der Zusammenhang zwischen $(B|A)$ und $A \Rightarrow B$ wird hierbei nochmals deutlich. Aus der eben genannten Eigenschaft folgt wegen $(\kappa * (B|A))(AB) \geq 0$, dass $(\kappa * (B|A))(A\overline{B}) > 0$. Wegen

$$\begin{aligned} 0 &= (\kappa * (B|A))(true) = (\kappa * (B|A))\left(A\overline{B} \vee \underbrace{\overline{A\overline{B}}}_{\equiv \overline{A} \vee B \equiv A \Rightarrow B} \right) \\ &= \min\left\{ \underbrace{(\kappa * (B|A))(A\overline{B})}_{>0}, (\kappa * (B|A))(A \Rightarrow B) \right\} \end{aligned}$$

muss $(\kappa * (B|A))(A \Rightarrow B) = 0$ sein.

Insgesamt wird demnach $A \Rightarrow B$ im Anschluss an die Revision geglaubt, denn $(\kappa * (B|A))(A \Rightarrow B) = 0$ und $(\kappa * (B|A))(\overline{A \Rightarrow B}) = (\kappa * (B|A))(A\overline{B}) > 0$.

Die gewünschte Eigenschaft $(\kappa * (B|A))(A\overline{B}) > (\kappa * (B|A))(AB)$ sichert ebenfalls, dass, wenn anschließend noch mit *A* revidiert wird, auch *B* geglaubt wird. Dies ist der Fall, weil alle Modelle von *A* entweder Modelle von AB oder von $A\overline{B}$ sind und diese bei der Revision mit *A* gleichmäßig nach ‘oben’ verschoben werden und somit der Wert von $A\overline{B}$ auch nachher noch größer ist als der von AB. Formell ausgedrückt gilt:

$$\begin{aligned} &(\kappa * (B|A))(A\overline{B}) > (\kappa * (B|A))(AB) \\ &\Rightarrow ((\kappa * (B|A)) * A)(A\overline{B}) > ((\kappa * (B|A)) * A)(AB) = 0 \end{aligned}$$

Außerdem gilt:

$$\left(\left(\kappa * (B|A)\right) * A\right)\left(\overline{AB}\right) = \left(\left(\kappa * (B|A)\right) * A\right)\left(\overline{A}\,\overline{B} \vee \overline{A}B \vee A\overline{B}\right)$$

$$= \min\left\{ \underbrace{\left(\left(\kappa * (B|A)\right) * A\right)\left(\overline{A}\,\overline{B}\right)}_{\substack{>0,\text{ weil Modelle von } \overline{A} \text{ nach der} \\ \text{Revision mit } A \text{ nicht in Ebene 0} \\ \text{liegen können}}}, \underbrace{\left(\left(\kappa * (B|A)\right) * A\right)\left(\overline{A}B\right)}_{\substack{>0,\text{ weil Modelle von } \overline{A} \text{ nach der} \\ \text{Revision mit } A \text{ nicht in Ebene 0} \\ \text{liegen können}}}, \underbrace{\left(\left(\kappa * (B|A)\right) * A\right)\left(A\overline{B}\right)}_{>0,\text{ wie eben gezeigt}} \right\} > 0$$

AB und damit insbesondere B werden demnach insgesamt geglaubt. Diese Überlegungen führen zu folgender Definition, welche die gewünschte Eigenschaft realisiert.

Definition 3.5.4.1:

'$\kappa * (B|A)$' ist gegeben durch $\kappa * (B|A) := \kappa$, falls bereits $\kappa\left(A\overline{B}\right) > \kappa(AB)$ (dann muss gelten: $(\kappa * A)(B) = 0$ und $(\kappa * A)\left(\overline{B}\right) > 0$, somit $B \in Bel(\kappa * A)$ und $\kappa \models (B|A)$). Ansonsten ergeben sich die neuen Werte der Modelle ω durch:

$$\left(\kappa * (B|A)\right)(\omega) := \begin{cases} \kappa(\omega) - \kappa\left(\overline{A} \vee B\right), \text{ falls } \omega \text{ ein Modell von } \overline{A} \vee B \text{ ist} \\ \kappa(AB) + 1 - \kappa\left(\overline{A} \vee B\right) + \kappa(\omega) - \kappa\left(A\overline{B}\right), \text{ falls } \omega \text{ ein Modell von } A\overline{B} \text{ ist} \end{cases}$$

Erklärung: Im nicht trivialen Fall $\kappa\left(A\overline{B}\right) \leq \kappa(AB)$ soll nach der Revision gelten $\left(\kappa * (B|A)\right)\left(A\overline{B}\right) = \left(\kappa * (B|A)\right)(AB) + 1$, wobei sich die Addition der 1 statt einer größeren Zahl durch das Minimal Change-Paradigma ergibt. Das bedeutet, dass die Modelle von $A\overline{B}$ 'unter' das glaubhafteste Modell von AB verschoben werden, also ist sicherlich $\left(\kappa * (B|A)\right)\left(A\overline{B}\right) > 0$. Damit aber $\left(\kappa * (B|A)\right)(true) = 0$ ist, muss wie bereits eben gezeigt $\left(\kappa * (B|A)\right)\left(\underbrace{\overline{A} \vee B}_{\equiv A \Rightarrow B}\right) = 0$ sein. Dies wird erreicht, indem von κ ausgehend alle Modelle von $\overline{A} \vee B$ um $\kappa\left(\overline{A} \vee B\right)$ nach 'oben' rutschen. Daher: $\left(\kappa * (B|A)\right)(\omega) := \kappa(\omega) - \kappa\left(\overline{A} \vee B\right)$, falls ω ein Modell von $\overline{A} \vee B$ ist.

$\left(\kappa * (B|A)\right)(\omega) := \kappa(AB) + 1 - \kappa\left(\overline{A} \vee B\right) + \kappa(\omega) - \kappa\left(A\overline{B}\right)$, falls ω ein Modell von $A\overline{B}$ ist

ergibt sich durch folgende Überlegung: Modelle von $A\overline{B}$ sollen bei $\left(\kappa * (B|A)\right)$ eine Ebene 'unter' dem glaubhaftesten Modell von AB beginnen, daher der Teil '$\kappa(AB) + 1$'. Da das glaubhafteste Modell von AB auch ein Modell von $\overline{A} \vee B$ ist, ist

es im Vergleich zu κ durch die Revision um $\kappa(\overline{A} \vee B)$ Ebenen nach 'oben' verschoben worden. Deswegen liegt es bei $(\kappa * (B|A))$ auf Ebene $\kappa(AB) - \kappa(\overline{A} \vee B)$ und es ergibt sich für den $(\kappa * (B|A))$-Wert von Modellen von $A\overline{B}$ das Zwischenergebnis '$\kappa(AB) + 1 - \kappa(\overline{A} \vee B)$'. Da die Modelle von $A\overline{B}$ ihre relativen Abstände zueinander beibehalten sollen, muss noch ergänzt werden: '$+ \kappa(\omega) - \kappa(A\overline{B})$'.

Auch diese Revision lässt sich gemäß folgender Definition verallgemeinern.

Definition 3.5.4.2:

'$\kappa * ((B|A), \alpha)$' ist gegeben durch $\kappa * ((B|A), \alpha) := \kappa$, falls bereits $\kappa(A\overline{B}) \geq \kappa(AB) + \alpha$. Ansonsten ergeben sich die neuen Werte der Modelle ω durch:

$$(\kappa * (B|A))(\omega) := \begin{cases} \kappa(\omega) - \kappa(\overline{A} \vee B), \text{ falls } \omega \text{ ein Modell von } \overline{A} \vee B \text{ ist} \\ \kappa(AB) + \alpha - \kappa(\overline{A} \vee B) + \kappa(\omega) - \kappa(A\overline{B}), \text{ falls } \omega \text{ ein Modell von } A\overline{B} \text{ ist} \end{cases}$$

Beispiel 3.5.4.1:

Sei κ gegeben durch:

- $\kappa^{-1}(0) = \{A\overline{B}\}$
- $\kappa^{-1}(1) = \{\overline{A}\,\overline{B}, \overline{A}B\}$
- $\kappa^{-1}(2) = \{AB\}$

Dann ist $\kappa * ((B|A), 1)$:

- $(\kappa * ((B|A), 1))^{-1}(0) = \{\overline{A}\,\overline{B}, \overline{A}B\}$
- $(\kappa * ((B|A), 1))^{-1}(1) = \{AB\}$
- $(\kappa * ((B|A), 1))^{-1}(2) = \{A\overline{B}\}$

Wird nun noch eine Revision mit $(A, 1)$ durchgeführt, erhält man:

- $((\kappa * ((B|A), 1)) * (A, 1))^{-1}(0) = \{AB\}$
- $((\kappa * ((B|A), 1)) * (A, 1))^{-1}(1) = \{\overline{A}\,\overline{B}, \overline{A}B, A\overline{B}\}$

Es wird wie gewünscht B geglaubt. Die in diesem Beispiel durchgeführte Revision

ist ein Pendant zum Modus Ponens.

3.5.5 OCFs für mehrwertige Variablen

Definition 3.5.5.1:

Eine ‘mehrwertige Variable’ ist eine Variable, die Werte aus einer als ‘Domain’ bezeichneten Menge von möglichen Ausprägungen annehmen kann (allerdings nicht mehrere gleichzeitig). Eine Domain kann auch aus mehr als zwei Elementen bestehen.

OCFs sind zunächst nur für Bool’sche Variablen definiert, lassen sich aber mit einigen zusätzlichen Überlegungen auch bei mehrwertigen Variablen verwenden.

Der Schlüssel dazu liegt darin, die Negation über mehrwertigen Variablen zu definieren, denn bei OCF-Operationen ist die Negation von zentraler Bedeutung (z. B. bei der Bestimmung von $Bel(\kappa)$ oder bei der Revision).

Definition 3.5.5.2:

Sei eine mehrwertige Variable d durch die gleichnamige Domain $d := \{v_1, \ldots, v_n\}$ gegeben und sei $n \geq 2$. Dann sei ‘$\overline{d = v_k}$’ definiert durch:

$$\overline{d = v_k} :\equiv d = v_1 \vee \ldots \vee d = v_{k-1} \vee d = v_{k+1} \vee \ldots \vee d = v_n \equiv \bigvee_{\substack{1 \leq i \leq n, \\ i \neq k}} d = v_i$$

Allgemeiner lässt sich definieren für eine Indexmenge $I \underbrace{\subsetneq}_{\text{echte Teilmenge}} \{1, \ldots, n\}$:

$$\overline{\bigvee_{k \in I} d = v_k} :\equiv \bigvee_{\substack{i \in \{1, \ldots n\}, \\ i \notin I}} d = v_i$$

(Bemerkung: Im Folgenden wird der Name einer Domain auch als Name für die mit dieser Domain assoziierten, mehrwertigen Variable verwendet.)

Beispiel 3.5.5.1:

Sei $d := \{false, true\}$. Dann ist $\overline{d = true} \equiv d = false$

Sei nun $d := \{low, middle, high, veryHigh\}$. Dann ist $\overline{d = low \vee d = middle} \equiv d = high \vee d = veryHigh$.

Die Begriffe ‘Literal’ und ‘Formel’ lassen sich wie folgt definieren (vgl. Definition 3.2.1.1).

Definition 3.5.5.3:

Eine mehrwertige Variable, der genau einer ihrer möglichen Werte zugewiesen wurde, nennt man ein ‘Literal einer mehrwertigen Variable’. Sei z. B. $d_1 := \{low, middle, high, veryHigh\}$ die Domain der ebenfalls d_1 genannten Variable, dann ist $d_1 = middle$ ein Literal.

Eine ‘Formel bei mehrwertigen Variablen’ ist ein aus Literalen mehrwertiger Variablen und Operatoren (syntaktisch korrekt) zusammengesetzter Ausdruck. Sei d_1 wie eben definiert und d_2 durch $d_2 := \{false, true\}$. Dann ist z. B. $\left(d_1 = low \vee \overline{d_1 = high}\right) \wedge d_2 = true$ eine Formel.

Über die Regeln von DeMorgan können auch komplexe Negationen gebildet werden, wie das folgende Beispiel zeigt:

Beispiel 3.5.5.2:

Seien gegeben: $d_1 := \{low, middle, high, veryHigh\}$ und $d_2 := \{false, true\}$. Dann ist:

$$
\begin{aligned}
&\overline{\left(d_1 = low \vee \overline{d_1 = high}\right) \wedge d_2 = true} \\
&\equiv \overline{\left(d_1 = low \vee \overline{d_1 = high}\right)} \vee \overline{d_2 = true} \\
&\equiv \left(\overline{d_1 = low} \wedge d_1 = high\right) \vee d_2 = false \\
&\equiv \left(\left(d_1 = middle \vee d_1 = high \vee d_1 = veryHigh\right) \wedge d_1 = high\right) \vee d_2 = false \\
&\equiv d_1 = high \vee d_2 = false
\end{aligned}
$$

Auch der Begriff ‘Modell’ muss nicht wesentlich geändert werden (vgl. Definition 3.2.1.10):

Definition 3.5.5.4:

Ein ‘Modell bei mehrwertigen Variablen’ ist eine Konjunktion von Literalen *aller* vereinbarten Variablen. Gibt es z. B. genau die Domains $d_1 := \{low, middle, high, veryHigh\}$, $d_2 := \{false, true\}$ und $d_3 := \{false, true, unknown\}$ und die gleichnamigen, mehrwertigen Variablen, so ist $d_1 = middle \wedge d_2 = true \wedge d_3 = true$ ein Modell,

$(d_1 = low \vee d_1 = middle) \wedge d_2 = true \wedge d_3 = true$ oder $\overline{d_1 = middle} \wedge d_2 = true \wedge d_3 = true$ jedoch nicht.

Definition 3.5.5.5:

Der 'Raum aller Modelle Ω bei mehrwertigen Variablen' ist die $\prod_{i=1}^{n} |d_i|$-elementige Menge aller möglichen Modelle, wobei *n* die Anzahl der vereinbarten Domains ist.

Nach diesen Überlegungen und Definitionen können $Bel(\kappa)$ und Revisionen in der üblichen, in Kapitel 3.5 vorgestellten Weise berechnet werden. So sind z. B. die Modelle des Geglaubten von κ auch bei mehrwertigen Variablen gegeben durch $Mod(Bel(\kappa)) := \kappa^{-1}(0)$ (vgl. Definition 3.5.1.2).

Beispiel 3.5.5.3:

Seien gegeben: $d_1 := \{low, middle, high, veryHigh\}$ und $d_2 := \{false, true\}$. Sei eine OCF κ für mehrwertige Variablen gegeben durch:

- $$\begin{aligned}\kappa^{-1}(0) := \{&d_1 = low \wedge d_2 = false, d_1 = low \wedge d_2 = true,\\ &d_1 = middle \wedge d_2 = false, d_1 = middle \wedge d_2 = true,\\ &d_1 = high \wedge d_2 = false, d_1 = high \wedge d_2 = true,\\ &d_1 = veryHigh \wedge d_2 = false, d_1 = veryHigh \wedge d_2 = true\}\end{aligned}$$

Nach einer Revision mit dem Konditional $c := (d_2 = false | d_1 = high \vee d_1 = veryHigh)$ und einer gewünschten Verwurzelungstiefe von $\alpha = 2$ ergibt sich:

- $$\begin{aligned}(\kappa * (c,2))^{-1}(0) := \{&d_1 = low \wedge d_2 = false, d_1 = low \wedge d_2 = true,\\ &d_1 = middle \wedge d_2 = false, d_1 = middle \wedge d_2 = true,\\ &d_1 = high \wedge d_2 = false, d_1 = veryHigh \wedge d_2 = false\}\end{aligned}$$
- $(\kappa * (c,2))^{-1}(1) := \{\ \}$
- $(\kappa * (c,2))^{-1}(2) := \{d_1 = high \wedge d_2 = true, d_1 = veryHigh \wedge d_2 = true\}$

Es gilt wie gewünscht (siehe Definition 3.5.4.2):

$$\begin{aligned}&(\kappa * (c,2))((d_1 = high \vee d_1 = veryHigh) \wedge \overline{d_2 = false})\\ &\geq (\kappa * (c,2))((d_1 = high \vee d_1 = veryHigh) \wedge d_2 = false) + \underbrace{2}_{=\alpha}\end{aligned}$$

Nach einer weiteren Revision mit der propositionalen Information $p := (d_1 = high)$ und einer gewünschten Verwurzelungstiefe von $\alpha = 1$ ergibt sich:

- $(\kappa * (c,2) * (p,1))^{-1}(0) := \{d_1 = high \wedge d_2 = false\}$
- $(\kappa * (c,2) * (p,1))^{-1}(1) := \{d_1 = low \wedge d_2 = false, d_1 = low \wedge d_2 = true,$
  $d_1 = middle \wedge d_2 = false, d_1 = middle \wedge d_2 = true,$
  $d_1 = veryHigh \wedge d_2 = false\}$
- $(\kappa * (c,2) * (p,1))^{-1}(2) := \{d_1 = high \wedge d_2 = true\}$
- $(\kappa * (c,2) * (p,1))^{-1}(3) := \{d_1 = veryHigh \wedge d_2 = true\}$

Es gilt wie gewünscht $(d_1 = high) \in Bel(\kappa * (c,2) * (p,1))$. Außerdem wird die Konsequenz von *c* nun geglaubt, da die Prämisse durch *p* erfüllt ist. Formal gilt: $(d_2 = false) \in Bel(\kappa * (c,2) * (p,1))$.

4. Zwei-Level Lern-Ansätze

4.1 Vorüberlegungen basierend auf psychologischen Erkenntnissen

Analysiert man, wie Menschen lernen und sich Fähigkeiten aneignen, stellt man fest, dass es zwei Ebenen zu geben scheint [And1983], [Reb1989], [Reb1993]. Auf der einen Ebene, welche als Bottom-Level bezeichnet wird, lernt man *implizit* und erwirbt *prozedurales* Wissen. Man ist sich *nicht bewusst* darüber, was genau man gelernt hat und kann es kaum oder gar nicht in Worte fassen. Auf der anderen Ebene, welche als Top-Level bezeichnet wird, lernt man *explizit* und erwirbt *deklaratives* Wissen. Man ist sich *bewusst* darüber, welche Zusammenhänge man erkannt hat, und kann das Gelernte z. B. in Form von Regeln formulieren.

Eine Sonderform des deklarativen Wissens ist das *episodische* Wissen. Dieses Wissen ist nicht allgemeiner Natur, sondern auf spezielle Ereignisse, Situationen oder Objekte bezogen. Aktuelle Forschungen zeigen, dass auch einige Tiere zu dieser Form des deklarativen Wissens fähig sind. Ein Beispiel: Eine bestimmte Vogelsorte (Aphelocoma insularis, engl.: western scrub-jay), welche Nahrung versteckt, holt ein bestimmtes Objekt abhängig davon wieder hervor, wie lange es schon gelagert wurde und wie verderblich es ist [CuD1998]. Episodisches Wissen ermöglicht es u. a., sich spezielle Situationen zu merken, in denen sonst gültige Regeln nicht gelten.

Experimente zeigen (z. B. das Erlernen einer künstlichen Grammatik [Reb1989]), dass bei einfach zu erkennenden Zusammenhängen eher explizit gelernt wird, während bei komplexen Beziehungen eher implizit gelernt wird. Es ist allerdings keineswegs so, dass die beiden Lernmethoden vollkommen isoliert voneinander arbeiten. Jeder wird in seinem Leben erfahren haben, dass sie sich gegenseitig beeinflussen. Dies kann in beide Richtungen, also top-down oder bottom-up, geschehen, wie folgende Beispiele zeigen:

Beispiel 4.1.1:

Ein Schüler lernt im Mathematik-Unterricht eine neue Regel kennen, indem der Lehrer sie an die Tafel schreibt und erklärt. Das Lernen geschieht also im Top-Level. Der Schüler kann die neue Regel prinzipiell anwenden, indem er sich bei einer gegebenen Aufgabe bewusst Gedanken macht, wie die Regel anzuwenden ist.

Nachdem die Regel an Hand von vielen Aufgaben eingeübt worden ist, kann er weitere Aufgaben fast schon intuitiv lösen.

Hier handelt es sich um den top-down Fall, da die Fähigkeit, das Problem zu lösen, zuerst nur im Top-Level möglich ist und erst anschließend im Bottom-Level.

Beispiel 4.1.2:

Nach dem Erledigen seiner Hausaufgaben kauft sich der Schüler aus Beispiel 4.1.1 ein neues Computerspiel, bei dem er Pilot eines Raumschiffs mit Tarnvorrichtung ist. Im getarnten Zustand kann das Schiff allerdings nicht feuern. Zu Anfang verliert er immer recht schnell, wird jedoch mit der Zeit immer besser. Nach noch mehr Übung wird ihm folgende Regel bewusst: “Wenn ich drei oder mehr Gegner habe, sollte ich die Tarnung meines Raumschiffs aktivieren, mich anschließend direkt hinter einem der Gegner platzieren, die Tarnung aufheben und feuern.”

In diesem Fall fließt Information in bottom-up Richtung. Zuerst verbessern sich die Fähigkeiten des Spielers, ohne dass ihm bewusst ist, wodurch. Er lernt implizit, wie er sich verhalten muss, um zu gewinnen. Erst später werden ihm die Zusammenhänge deutlich, so dass er eine Taktik formulieren kann.

Zudem wurde festgestellt [SMB1989], dass sich beim impliziten Lernen, welches gerade in völlig unbekannten Situationen stattfindet, erst das prozedurale Wissen verbessert und das deklarative erst später. Daher spielt die bottom-up Richtung eine wichtige Rolle. Insbesondere bei menschlichem Lernen ist es von Vorteil, wenn der Lernende während des Lernens in ausgewogenem Maß versucht, das bereits Erlernte zu verbalisieren, weil so schneller deklaratives Wissen erlangt wird [SMB1989].

Deklaratives Wissen bietet noch einen Vorteil: Es wurde herausgefunden, dass der Erwerb deklarativen Wissens nicht nur das Lernen beschleunigt, sondern auch zu Fähigkeiten zur Problemlösung führt, welche denen, die durch rein implizites Lernen entwickelt werden, überlegen sind [WNB1989].

Ein leistungsfähiges, künstlich geschaffenes System zum Lernen und Erwerben von Fähigkeiten zum Lösen einer Aufgabe sollte nach diesen Überlegungen nicht auf nur implizites oder nur explizites Lernen beschränkt sein, sondern ebenfalls beide

Lerntypen verbinden. Eine Komponente sollte eine subsymbolische Darstellung von Erlerntem verwenden, welche das Bottom-Level repräsentiert. Eine weitere sollte das Erlernte mit Symbolen, welchen eine semantische Bedeutung zugeordnet werden kann, darstellen und so das Top-Level simulieren. Außerdem sollten episodische Erfahrungen berücksichtigt werden und ein ausgeglichener Informationsfluss zwischen diesen Komponenten gestaltet werden.

4.2 Lern-Modell CLARION

Sun, Merrill und Peterson entwickelten das Modell CLARION (Connectionist Learning with Adaptive Rule Induction ON-line) [SMP2001], [SPM1996], [SuP1997], [SuP1998]. CLARION ist ein bottom-up Modell, welches für das untere Level *Q*-Learning und für das obere Level eine Regelmenge verwendet. Die Regeln haben die Form 'Prämisse $\rightarrow$ Aktion', wobei die Prämisse durch das aktuelle Zustandssignal der Umwelt erfüllt wird oder nicht, und werden mit Hilfe der *Q*-Werte des unteren Level erstellt, verallgemeinert oder ggfs. wieder spezialisiert oder gelöscht. Im Vergleich mit menschlichem Verhalten in einer geeigneten Testumgebung zeigt CLARION Parallelen.

Die Autoren geben in [SMP2001] folgenden Haupt-Algorithmus an, bei dem die Aktionen des Agenten, wenn sie aus dem unteren Level betrachtet werden, mit a_i bezeichnet werden, und wenn sie aus dem oberen Level betrachtet werden, mit b_j bezeichnet werden.

Algorithmus 4.2.1 (CLARION):

1. Nehme den aktuellen Zustand s_t wahr.
2. Ermittle im unteren Level den Wert für jede in s_t mögliche Aktion a_i, also $Q(s_t, a_i)$.
3. Bestimme mittels der Regeln im oberen Level für alle möglichen Aktionen b_j, in wie weit sie bei der Wahrnehmung s_t folgen.
4. Wähle stochastisch eine passende Aktion *a*, basierend auf der Kombination der Werte der a_i (aus dem unteren Level) und der b_j (aus dem oberen Level).
5. Führe *a* aus, nimm den Folgezustand s_{t+1} und die Belohnung *r* wahr.
6. Aktualisiere die *Q*-Werte im unteren Level.

7. Aktualisiere die Regelmenge im oberen Level.

8. Zurück zu 1.

Zum Berechnen von $Q(s_t, a_i)$ und anschließend einer Wahrscheinlichkeitskeitsverteilung über alle Aktionen, welche später zur Wahl der endgültigen Aktion *a* benutzt wird, wird im unteren Level bei CLARION ein konnektionistisches Netzwerk verwendet. An die erste Schicht wird s_t angelegt, die vorletzte Schicht repräsentiert die *Q*-Werte und die letzte die Wahrscheinlichkeitsverteilung, welche auf der Boltzmann Verteilung $p(a|s_t) = \frac{e^{\frac{Q(s_t,a)}{\alpha}}}{\sum_i e^{\frac{Q(s_t,a_i)}{\alpha}}}$ basiert.

Im oberen Level wird das aktuelle Zustandssignal als Vektor der Länge *n* angesehen. Es gibt Domains $d_1,\ldots,d_n$ und jede Domain ist gegeben durch die Menge möglicher Werte, die die gleichnamige, zugeordnete, mehrwertige Variable annehmen kann: $d_i := \{v_{i.1},\ldots,v_{i.m_i}\}$. Jede Komponente des Zustandssignal-Vektors entspricht genau einer dieser Variablen. Der aktuelle Wert der *i*-ten Komponente des Vektors ist ein Element aus $\{v_{i.1},\ldots,v_{i.m_i}\}$ und stellt die aktuelle Ausprägung der entsprechenden Variablen dar. Die Prämisse einer Regel lässt sich schreiben als Konjunktion von Disjunktionen, die als Teilmengen $d_1' \subseteq d_1,\ldots,d_n' \subseteq d_n$ der Domains dargstellt sind, mit der Bedeutung: $\bigwedge_{i=1}^{n}\left(\bigvee_{v\in d_i'} v\right)$. (Bemerkung: Diese Darstellung ist einer Klauselform [BuK2006, S. 61] ähnlich.) Die Prämisse einer Regel ist somit erfüllt für einen Zustandssignal-Vektor $s_t = (v_1,\ldots,v_n)$, wenn gilt: $v_i \in d_i' \forall i \in \{1,\ldots,n\}$. Die Konsequenz einer Regel ist eine Aktion.

Bei CLARION werden Regeln verallgemeinert, indem genau eine Teilmenge d_i' um genau ein Element erweitert wird, so dass die Prämisse durch mehr Zustandssignal-Vektoren erfüllt wird. Umgekehrt werden Regeln spezialisiert, indem genau eine Teilmenge d_i' um genau ein Element verkleinert wird, so dass die Prämisse durch weniger Zustandssignal-Vektoren erfüllt wird. Die Menge aller

Regeln wird ebenfalls als konnektionistisches Netzwerk dargestellt, so dass auch hier die auszuführende Aktion stochastisch bestimmt ist.

In Schritt 7 des CLARION-Algorithmus (Algorithmus 4.2.1) wird Folgendes geleistet: Erstellt werden nur Regeln, bei denen die Prämisse genau einem Zustandssignal-Vektor entspricht, d. h. die d_i' sind ein-elementig. Eine entsprechende Regel '$s_t \rightarrow a$' wird erstellt, wenn im Zustand s_t das Ausführen der gewählten Aktion a 'gut' war, wobei 'gut' durch ein bestimmtes Kriterium definiert wird.

Zudem wird eine Statistik darüber geführt, wie oft jede Regel zu einem 'guten' Ergebnis geführt hat und wie oft nicht. Eine erfolgreiche Regel, d. h. eine Regel, die oft zu einem 'guten' Ergebnis und selten zu einem 'schlechten' Ergebnis führt, wird durch eine Verallgemeinerung ersetzt. Eine 'gute' Verallgemeinerung wird beibehalten und ggfs. weiter verallgemeinert, ansonsten wird sie wieder durch spezialisierte Regeln ersetzt.

CLARION setzt die in Kapitel 4.1 genannten, wünschenswerten Eigenschaften eines Zwei-Level Lernmodells weitestgehend um. Es wurde eine ausführliche Studie durchgeführt, in der die Leistung von CLARION mit der Leistung menschlicher Testpersonen verglichen wurde. Testumgebung war hierbei eine Navigationsaufgabe, bei der die Testperson bzw. CLARION mit Hilfe von einigen Anzeigen bzw. Werten durch ein Minenfeld hindurch einen Zielpunkt ansteuern sollte, wobei die Zeit für das Erreichen des Ziels beschränkt war. Es wurden verschiedene Konfigurationen getestet, bei denen z. B. das Training des einen Levels gegenüber dem des anderen intensiviert wurde. Beim Modell CLARION wurde dies durch das Einstellen gewisser Parameter erreicht, bei menschlichen Testpersonen durch psychologische Effekte. Die beiden sich jeweils ergebenden Lernkurven weisen bemerkenswerte Ähnlichkeiten auf. Außerdem leidet die Performance gleicher Maßen bei CLARION und Menschen darunter, wenn das Gleichgewicht beider Level zu sehr in die eine oder andere Richtung verzerrt wird.

4.3 Sphinx-Learning

4.3.1 Grundlagen

In dieser Arbeit stelle ich das von mir entwickelte Sphinx-Learning vor – ein Hybrid

aus $Q(\lambda)$-Learning und einer Komponente für das Abfragen und Revidieren von Geglaubtem, welches durch eine OCF κ repräsentiert wird. κ dient dem Agenten dazu, Hintergrundwissen und selbst erlernte Verhaltensregeln in Form von konditionaler Information zu speichern und assistiert ihm dadurch bei der Wahl seiner Aktionen. Dieses Lernverfahren wurde nach der bekannten Großen Sphinx von Gizeh benannt, welche ebenfalls ein Hybrid ist.

Im Folgenden wird das Vorgehen zum besseren Verständnis i. d. R. zunächst grob geschildert und anschließend ins Detail gegangen. Insbesondere wird in den erläuternden Beispielen meist nur erwähnt, *was* geschieht bzw. gemacht wird, ohne dass ausführlich darauf eingegangen wird, *wodurch* etwas geschieht bzw. *wie* etwas gemacht wird. Dies wird später an geeigneterer Stelle geschildert, wenn die grundlegenden Ideen präsentiert worden sind.

Sphinx-Learning findet, wie das $Q(\lambda)$-Learning, im Rahmen eines RL-Systems (siehe Kapitel 2.1 und 2.2) statt. Ziel des Agenten ist daher, für jeden Zustand $s \in S$ der Umwelt eine für ihn beste Aktion zu finden. Dies wird erreicht, indem der Agent durch Interaktion mit der Umwelt lernt. Wie CLARION setzt Sphinx-Learning auf eine Zwei-Level-Strategie. Der Ablauf eines Schrittes (im Sinne von Definition 2.1.1) ist im Wesentlichen der folgende:

1. Der Sphinx-Agent nimmt das von der Umwelt bereitgestellte, vollständig zugängliche Zustandssignal *s* wahr.
2. Der Agent fragt seine OCF κ ab, welche Aktionen, bezeichnet mit $A_\kappa(s) = \{a_1, \ldots, a_k\}$, bei einer solchen Wahrnehmung am plausibelsten sind.
3. Der Agent fragt aus seiner Tabelle, in der die *Q*-Werte gespeichert sind, (im Folgenden 'QTable' genannt) die Werte der Aktionen aus $A_\kappa(s)$ für Zustand *s* ab und bestimmt die Menge der besten Aktionen $A_{best}(s) \subseteq A_\kappa(s)$. (Ein normaler $Q(\lambda)$-Agent bestimmt die Menge der besten Aktionen aus der Menge *aller* möglichen Aktionen $A(s)$.)
4. Er wählt aus $A_{best}(s)$ zufällig eine Aktion *a* aus und führt sie aus.
5. Die Umwelt wechselt in den Folgezustand.
6. Der Agent erhält die Belohnung *r*.

7. Er aktualisiert die QTable.

8. Die neuen *Q*-Werte für Aktionen in *s* werden gelesen.

9. Anschließend revidiert der Sphinx-Agent κ. Diese Revision bewirkt i. d. R., dass beim nächsten Besuch des Zustands *s* die nun laut QTable besten Aktionen am plausibelsten sind, wobei vorherige Erfahrungen dafür sorgen können, dass einige beste Aktionen plausibler sind als andere. Außerdem wird versucht, Muster in den Zustandssignalen zu erkennen, bei deren Wahrnehmung bestimmte Aktionen generell besser sind als andere. Wird ein solcher Zusammenhang zwischen Muster und Aktionen erkannt, findet eine Revision mit einer generalisierten Regel statt. Wenn der nun aktuelle Zustand kein Terminal-Zustand ist, weiter bei 1.

Grafisch lässt sich dies wie in Abbildung 4.3.1.1 dargestellt veranschaulichen.

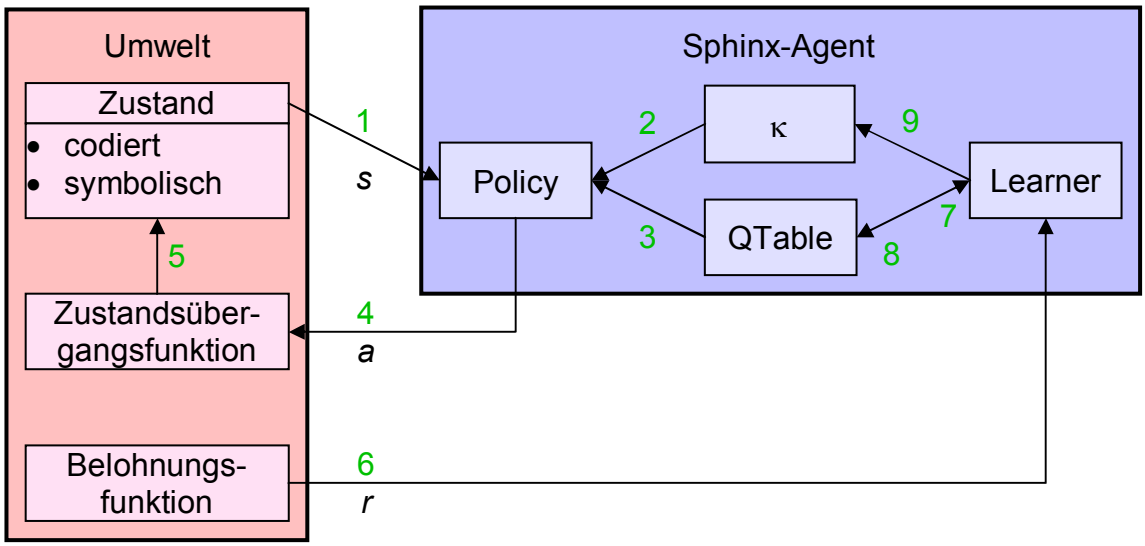


Abbildung 4.3.1.1

Eine grafische Darstellung eines Schrittes für ein normales RL-System mit einem $Q(\lambda)$-Agenten zeigt Abbildung 4.3.1.2.

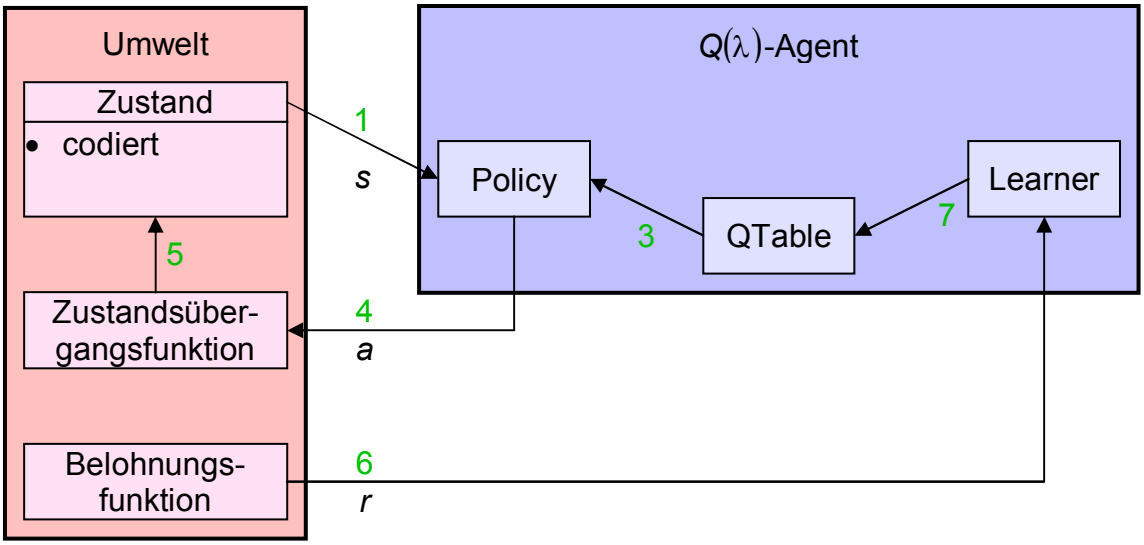


Abbildung 4.3.1.2

Abgesehen davon, dass ein $Q(\lambda)$-Agent keine OCF besitzt und dementsprechend die Punkte 2, 8 und 9 entfallen, ist das Zustandssignal im klassischen, endlichen und somit diskreten RL-System z. B. ein numerischer Wertevektor oder eine ID-Nummer. Beim Sphinx-Learning hingegen besteht es zudem aus einer symbolischen Beschreibung, der man als Anwender die semantische Bedeutung des Zustands direkt entnehmen kann. Zwischen der codierten und der symbolischen Beschreibung existiert eine bijektive Beziehung. Zu Beginn kann ein Sphinx-Agent den symbolischen Beschreibungen von Zuständen keine Bedeutungen zuordnen. Diese werden nach und nach durch die Revisionen in Punkt 9 gewonnen in dem Sinne, dass er lernt, den symbolischen Beschreibungen 'gute' Aktionen zuzuordnen.

Sphinx-Learning lernt demnach sowohl implizit als auch explizit und speichert sowohl prozedurales als auch episodisches und deklaratives Wissen. Somit wird das in Kapitel 4.1 beschriebene Lernkonzept umgesetzt.

4.3.2 Das Zustandssignal

Sei zunächst der Zustandsraum S endlich und somit diskret. Dann lässt sich jedem Zustand eine ein-eindeutige ID aus $\{0,\ldots,|S|-1\}$ zuordnen. Diese ID ist eine codierte Darstellung, welche für das übliche RL ausreicht. In der Praxis hat jedoch jeder Zustand eine semantische Bedeutung, die für das RL irrelevant ist und nicht beachtet oder ausgenutzt wird.

Beispiel 4.3.2.1 (Gridworld mit Hindernissen):

Gegeben sei folgende Umwelt:

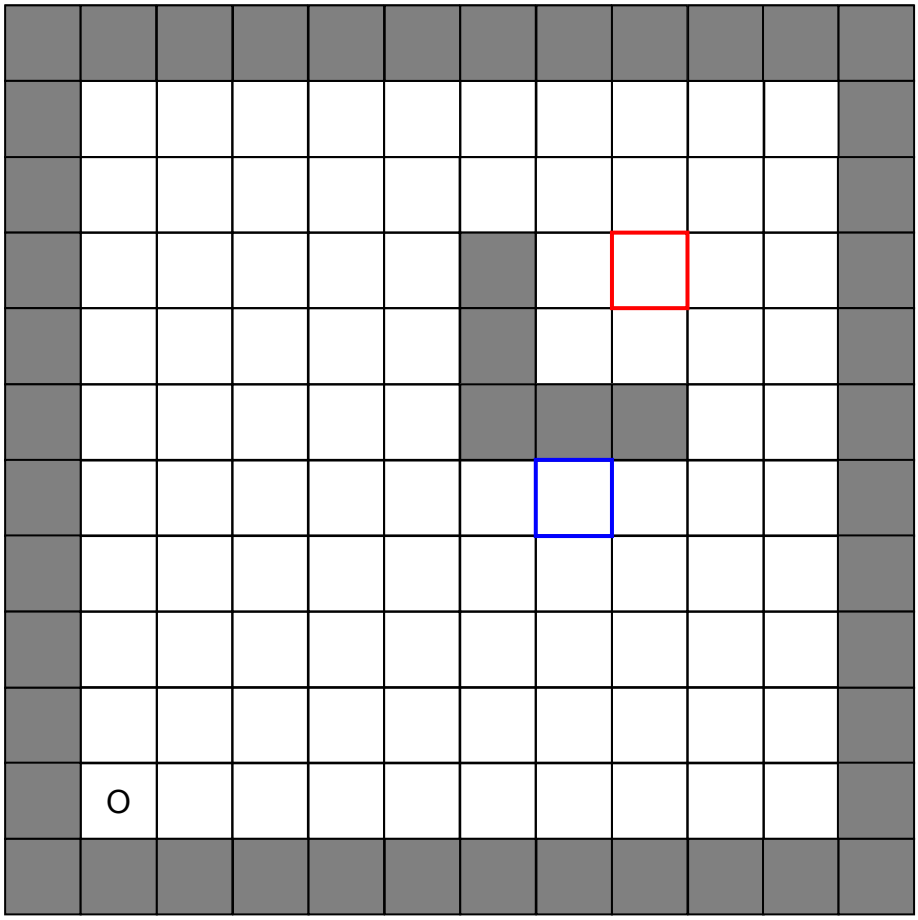


Abbildung 4.3.2.1

Aufgabe des Agenten ist, von einer zufälligen Startposition aus, z. B. dem blau umrandeten Feld, das rot umrandete Zielfeld zu erreichen. Graue Felder repräsentieren Hindernisse, z. B. Gegenstände oder Mauern. Eine erste Idee, das Zustandssignal der Umwelt zu definieren, wäre, einen Koordinatenursprung festzulegen (z. B. das *weiße*, mit 'O' gekennzeichnete Feld unten links) und jedes Feld durch ein Tupel (x, y) zu beschreiben. Das mit 'O' gekennzeichnete Feld hätte dann die Koordinaten $(0,0)$ und das Zielfeld die Koordinaten $(7,7)$. Die für jedes Feld eindeutige ID könnte man berechnen durch $ID := x \cdot 10 + y$, wobei die '10' die Anzahl der Zeilen, also möglicher *y*-Werte, ist.

Die zweite Idee ergibt sich durch das Betrachten der menschlichen Herangehensweise an dieses Problem. Menschen würden den Zustand nicht in Koordinaten ausdrücken, sondern, z. B. für das blau umrandete Feld mit den Koordinaten $(6,4)$, mit Begriffen wie:

- "Das Ziel liegt im Nordosten, aber mehr im Norden als im Osten."
- "Die Entfernung zum Ziel ist mittel."

- “Im Norden ist ein Hindernis. Alle anderen Richtungen sind frei.”

Analysiert man diese verbalen Beschreibungen, findet man, dass eine Zustandsbeschreibung eine Konjunktion von Werten aus Domains ist. Für diese Gridworld lassen sich folgende Domains zur Beschreibung des Zustands festliegen:

- *TargetNorthSouth* = {*Neither-Nor*, *NorthAhead*, *NorthAside*, *SouthAhead*, *SouthAside*}
- *TargetEastWest* = {*Neither-Nor*, *EastAhead*, *EastAside*, *WestAhead*, *WestAside*}
- *TargetDistance* = {*Far*, *Middle*, *Close*}
- *ObstacleNorth* = {*False*, *True*}
- *ObstacleEast* = {*False*, *True*}
- *ObstacleSouth* = {*False*, *True*}
- *ObstacleWest* = {*False*, *True*}

Die oben genannte, verbale Beschreibung von Feld $(6,4)$ lässt sich nun ausdrücken wie folgt:

- “Das Ziel liegt im Nordosten, aber mehr im Norden als im Osten.”: *TargetNorthSouth* = *NorthAhead* $\wedge$ *TargetEastWest* = *EastAside*
- “Die Entfernung zum Ziel ist mittel.”: *TargetDistance* = *Middle*
- “Im Norden ist ein Hindernis. Alle anderen Richtungen sind frei.”: *ObstacleNorth* = *True* $\wedge$ *ObstacleEast* = *False* $\wedge$ *ObstacleSouth* = *False* $\wedge$ *ObstacleWest* = *False*

Insgesamt hätte das Feld $(6,4)$ dann die symbolische Beschreibung:

$$
\begin{array}{llll}
 & \mathit{TargetNorthSouth} & = \mathit{NorthAhead} & \wedge\ \mathit{TargetEastWest} \quad = \mathit{EastAside} \\
\wedge & \mathit{TargetDistance} & = \mathit{Middle} & \wedge\ \mathit{ObstacleNorth} \quad = \mathit{True} \\
\wedge & \mathit{ObstacleEast} & = \mathit{False} & \wedge\ \mathit{ObstacleSouth} \quad = \mathit{False} \\
\wedge & \mathit{ObstacleWest} & = \mathit{False} &
\end{array}
$$

Aus dieser symbolischen Beschreibung lässt sich ein-eindeutig eine ID für das RL berechnen:

$$
\begin{aligned}
ID := & \underbrace{1}_{\hat{=}NorthAhead} \cdot |TargetEastWest| \cdot |TargetDistance| \cdot \ldots \cdot |ObstacleWest| \\
& + \underbrace{2}_{\hat{=}EastAside} \cdot |TargetDistance| \cdot |ObstacleNorth| \cdot \ldots \cdot |ObstacleWest| \\
& + \underbrace{1}_{\hat{=}Middle} \cdot |ObstacleNorth| \cdot |ObstacleEast| \cdot |ObstacleSouth| \cdot |ObstacleWest| \\
& + \underbrace{1}_{\hat{=}True} \cdot |ObstacleEast| \cdot |ObstacleSouth| \cdot |ObstacleWest| \\
& + \underbrace{0}_{\hat{=}False} \cdot |ObstacleSouth| \cdot |ObstacleWest| \\
& + \underbrace{0}_{\hat{=}False} \cdot |ObstacleWest| \\
& + \underbrace{0}_{\hat{=}False} \\
& = 1 \cdot (5 \cdot 3 \cdot 2 \cdot 2 \cdot 2 \cdot 2) \\
& + 2 \cdot (3 \cdot 2 \cdot 2 \cdot 2 \cdot 2) \\
& + 1 \cdot (2 \cdot 2 \cdot 2 \cdot 2) \\
& + 1 \cdot (2 \cdot 2 \cdot 2) \\
& + 0 \cdot (2 \cdot 2) \\
& + 0 \cdot (2) \\
& + 0 \\
& = 360
\end{aligned}
$$

Seien nun i. A. n Domains $d_0, \ldots, d_{n-1}$ zur Zustandsbeschreibung gegeben. Sei die i-te Domain gegeben durch m_i mögliche Werte, d. h. $d_i := \{v_{i.0}, \ldots, v_{i.m_i-1}\}$. Eine allgemeine Formel zur Berechnung der ID zur symbolischen Zustandsbeschreibung $d_0 = v_{0.k_0} \wedge \ldots \wedge d_{n-1} = v_{n-1.k_{n-1}}$ lautet dann:

$$
ID := \sum_{i=0}^{n-1} k_i \cdot \prod_{j=i+1}^{n-1} |d_j|
$$

Der folgende Algorithmus kann verwendet werden, um aus einer ID die symbolische Zustandsbeschreibung zurückzugewinnen.

Algorithmus 4.3.2.1:

1. For i = 0 to n-1:

1.1. $k_i := \left\lfloor \dfrac{ID}{\prod_{j=i+1}^{n-1} |d_j|} \right\rfloor$

1.2. $ID := ID \bmod \prod_{j=i+1}^{n-1} |d_j|$

Auf diese Weise werden alle k_i rekonstruiert, durch welche sich dann die symbolische Darstellung $d_0 = v_{0.k_0} \wedge \ldots \wedge d_{n-1} = v_{n-1.k_{n-1}}$ ergibt.

Als geeignetes Zustandssignal von der Umwelt an den Agenten ergibt sich für das Sphinx-Learning das Tupel aus codierter Darstellung (die ID) und symbolischer Darstellung (der konjunktive Term $d_0 = v_{0.k_0} \wedge \ldots \wedge d_{n-1} = v_{n-1.k_{n-1}}$).

Definition 4.3.2.1:

's_{ID}' bezeichnet die codierte und 's_{sym}' bezeichnet die symbolische Darstellung des Zustands *s*. Man kann *s* nun schreiben als $s = (s_{ID}, s_{sym})$.

Sei nun der Zustandsraum *S* unendlich. In einem solchen Fall bedient man sich beim klassischen RL der Funktionsapproximation, d. h. man schätzt $Q(s,a)$ mit Hilfe bekannter Werte von zu *s* ähnlichen Situation. Sphinx-Learning bietet eine ähnliche Möglichkeit: Die kontinuierlichen realen Werte werden durch Begriffe wie 'sehr ...', 'etwas ...', 'nicht ...' usw. diskretisiert und anschließend zu einem konjunktiven Term verküpft. Da die Domains nun endlich sind, lässt sich wie eben eine entsprechende ID berechnen.

4.3.3 Aufbau von κ und Wahl einer Aktion

Zusätzlich zu den Domains bzw. Variablen des Zustandssignals, die von der Umwelt, in welcher der Agent handeln soll, abhängen, gibt es beim Sphinx-Learning immer die ausgezeichnete Domain bzw. Variable *Action*. Die Domain *Action* repräsentiert die Menge aller dem Agenten zur Verfügung stehenden Aktionen, die allerdings wieder von der speziellen Umwelt abhängen.

Ein Modell von κ ist gemäß Definition 3.5.5.4 eine Konjunktion von Literalen *aller* Variablen, also der Variablen der Umwelt *und* der Variablen *Action*. Für die Gridworld mit Hindernissen (Beispiel 4.3.2.1) wäre ein Modell von κ:

	TargetNorthSouth	= *NorthAhead*	∧ *TargetEastWest*	= *EastAside*
∧	*TargetDistance*	= *Middle*	∧ *ObstacleNorth*	= *True*
∧	*ObstacleEast*	= *False*	∧ *ObstacleSouth*	= *False*
∧	*ObstacleWest*	= *False*	∧ *Action*	= *GoNorth*,

wobei die Domain *Action* definiert ist durch *Action* := {*GoNorth*, *GoEast*, *GoSouth*, *GoWest*}.

Der Raum aller Modelle Ω für κ ist demnach die Menge aller möglichen Zustandssignale (genauer: ihrer symbolischen Darstellungen), von denen jedes mit jeder möglichen Aktion konjunktiv kombiniert worden ist:

$$\Omega := \left\{ \omega = s_{sym} \wedge Action = a \middle| s = \left(s_{ID}, s_{sym}\right) \in S, a \in Action \right\}$$

Initialisiert wird κ als uniforme OCF, d. h. $\kappa(\omega) = 0 \forall \omega \in \Omega$.

Um eine Aktion zu wählen (vgl. Punkte 1 bis 4 in Abbildung 4.3.1.1), benutzt der Sphinx-Agent eine ε-greedy Policy, d. h. mit einer kleinen Wahrscheinlichkeit wird zur Exploration der Umwelt die nächste Aktion zufällig gewählt. Andernfalls wird eine beste Aktion gewählt, indem κ nach den plausibelsten Modellen von s_{sym} des aktuellen Zustandssignals durchsucht wird. Aus diesen Modellen werden die Aktionen extrahiert und dadurch die Menge $A_\kappa(s)$ gebildet.

Definition 4.3.3.1:

'$A_\kappa(s)$' ist die Menge der Aktionen, die das Top-Level in *s* für am plausibelsten/ besten hält. Formal sind die Elemente von $A_\kappa(s)$ diejenigen Aktionen, welche in den Modellen von $Bel(\kappa_{s_{sym}})$ als Belegungen der Variable *Action* vorkommen, wobei $\kappa_{s_{sym}}$ die Konditionierung von κ auf s_{sym} (siehe Definition 3.5.3.4) ist. Sie auch folgendes Beispiel.

Beispiel 4.3.3.1:

Sei die betrachtete Umwelt die in Beispiel 4.3.2.1 beschriebene Gridworld mit Hindernissen. Sei die aktuelle Position des Agenten $(6,4)$ und s_{sym} die entsprechende symbolische Darstellung dieses Zustands. Die Konditionierung von κ auf s_{sym} wäre zu Beginn des Lernens und ohne Hintergrundwissen:

- $$\left(\kappa_{|s_{sym}}\right)^{-1}(0) = \{ s_{sym} \wedge Action = GoNorth, s_{sym} \wedge Action = GoEast, \\ s_{sym} \wedge Action = GoSouth, s_{sym} \wedge Action = GoWest \}$$

Hier sind alle vier möglichen Aktionen gleichermaßen plausibel und es gilt: $A_\kappa(s) = \{GoNorth, GoEast, GoSouth, GoWest\}$.

Nach einiger Zeit des Lernens könnte die Konditionierung von κ auf s_{sym} diese Form annehmen:

- $\left(\kappa_{|s_{sym}}\right)^{-1}(0) = \{s_{sym} \wedge Action = GoEast\}$
- $\left(\kappa_{|s_{sym}}\right)^{-1}(1) = \{s_{sym} \wedge Action = GoSouth, s_{sym} \wedge Action = GoWest\}$
- $\left(\kappa_{|s_{sym}}\right)^{-1}(2) = \{\ \}$
- $\left(\kappa_{|s_{sym}}\right)^{-1}(3) = \{\ \}$
- $\left(\kappa_{|s_{sym}}\right)^{-1}(4) = \{s_{sym} \wedge Action = GoNorth\}$

Hier wäre: $A_\kappa(s) = \{GoEast\}$.

Es ist anzumerken, dass κ für die Bestimmung von $A_\kappa(s)$ nicht verändert wird. Insbesondere wird κ nicht mit der propositionalen Information s_{sym} revidiert.

Anschließend werden mittels der codierten Darstellung s_{ID} die *Q*-Werte aller Aktionen aus $A_\kappa(s)$ aus der QTable ermittelt und die wie folgt definierte Menge $A_{best}(s)$ bestimmt.

Definition 4.3.3.2:

'$A_{best}(s)$' ist die Menge derjenigen Aktionen aus $A_\kappa(s)$, welche die höchsten *Q*-Werte haben, und ist gegeben durch:

$$A_{best}(s) := \left\{ a_{best} \in A_\kappa(s) \middle| Q(s_{ID}, a_{best}) = \max_{a \in A_\kappa(s)} Q(s_{ID}, a) \right\}.$$

Es gilt: $A_{best}(s) \subseteq A_\kappa(s)$.

In den meisten Fällen besteht $A_{best}(s)$ nur aus einer einzigen Aktion, welche dann von der Policy gewählt wird. Ist $|A_{best}(s)| > 1$, wird zufällig eine Aktion aus $A_{best}(s)$ gewählt.

Formal ist die Policy π des Agenten gegeben durch:

$$\pi(s,a) := \begin{cases} \dfrac{1}{|A_{best}(s)|}, \text{falls } a \in A_{best}(s) \\ 0, \text{falls } a \notin A_{best}(s) \end{cases}.$$

Die zugehörige, ε-greedy Policy $\tilde{\pi}$ ist gegeben durch:

$$\tilde{\pi}(s,a) := \begin{cases} \frac{1-\varepsilon}{|A_{best}(s)|} + \frac{\varepsilon}{|A(s)|}, \text{falls } a \in A_{best}(s) \\ \frac{\varepsilon}{|A(s)|}, \text{falls } a \notin A_{best}(s) \end{cases}$$

An dieser Stelle erkennt man erstmals die Zusammenarbeit von beiden Komponenten, denn beide Komponenten filtern nacheinander die Menge aller Aktionen, um eine beste Aktion zu bestimmen.

4.3.4 Der Zustandswechsel

Nach der Wahl und Ausführung der Aktion findet in Punkt 5 (siehe Abbildung 4.3.1.1) der Zustandswechsel statt. Der Folgezustand wird, wie beim RL üblich, allein durch die Zustandsübergangsfunktion der Umwelt an Hand des noch aktuellen Zustands und der vom Agenten gewählen Aktion bestimmt. Der Agent hat an dieser Stelle keinen Einfluss.

Beim Sphinx-Learning muss nun eine neue symbolische Zustandsdarstellung und eine neue codierte Zustandsdarstellung berechnet werden. Bei dieser Berechnung darf die Umwelt wie beim klassischen RL Information verwenden, die dem Agenten nicht zur Verfügung steht. Die Gridworld mit Hindernissen aus Beispiel 4.3.2.1 würde beispielsweise intern die genaue Position des Agenten in Koordinatenform verwalten und daraus ein passendes Zustandssignal $s = (s_{ID}, s_{sym})$ generieren, welches dem Agenten zur Verfügung gestellt wird.

4.3.5 Aktualisierung der QTable und Revision von κ

Im Folgenden werden die Punkte 6 bis 9 aus Abbildung 4.3.1.1 behandelt. Nach dem Erhalt des Belohnungssignals r_{t+1} für die zuvor in s_t ausgeführte Aktion a_t wird die QTable aktualisiert gemäß Algorithmus 2.6.3.1. Die dort verwendeten Parameter γ und λ sind während des ganzen Lernens konstant. α hingegen ist vom jeweils vorliegenden Zustand-Aktion-Paar abhängig und ergibt sich für (s_t, a_t) durch:

$$\alpha := \max\left\{\frac{1}{v_{(s_t,a_t)}}, \alpha_{min}\right\},$$

wobei $v_{(s_t,a_t)}$ für die aktuelle Anzahl bisheriger Besuche von (s_t, a_t) steht und α_{min}

eine untere Schranke ist, die α nicht unterschreiten soll, z. B. $\alpha_{min} := 0.25$. Ein so definiertes α bewirkt, dass die *Q*-Werte anfangs den genauen Durchschnitt der wenigen, bisher gemachten Erfahrungen widerspiegeln, sich später jedoch schnell auf Änderungen einstellen können. Eine solche Änderung tritt z. B. in einem System auf, in dem mehr als ein Agent handelt. Ändert einer der Agenten sein Verhalten, so können sich auch die Werte der Aktionen in einigen Situationen für andere Agenten ändern. Auch in einem System mit nur einem Agenten können sich die Werte signifikant ändern, wenn der Agent beispielsweise eine 'Abkürzung' zum Ziel entdeckt. Die vorangegangen Aktionen erhalten dadurch einen deutlichen höheren Wert als vor der Entdeckung. Damit die neuen Werte schnell gelernt werden, sollte α deshalb nicht unter eine bestimmte Grenze fallen.

Nach der Aktualisierung der QTable wird κ revidiert (siehe Abbildung 4.3.1.1). Dabei gibt es folgende, sich gegenseitig nicht ausschließende, Möglichkeiten:

- Revision mit Information über eine schlechte Aktion in einem bestimmten Zustand
- Revision mit Information über eine schlechte Aktion in mehreren, ähnlichen Zuständen
- Revision mit Information über beste Aktionen in einem bestimmten Zustand
- Revision mit Information über beste Aktionen in mehreren, ähnlichen Zuständen

Definition 4.3.5.1:

'Eine Aktion *a* ist im Zustand *s* schlecht' aus Sicht des Sphinx-Agenten, wenn die Belohnung für *a* in *s* echt kleiner als -1 ausgefallen ist. Das bedeutet, dass die Belohnungsfunktion so zu gestalten ist, dass schlechtere Belohnungen als -1 nur dann vergeben werden, wenn *a* eine in *s* definitiv unerwünschte Aktion ist, beispielsweise das Anstoßen an ein Hindernis.

(Bemerkung: Statt -1 hätte auch eine andere negative Zahl als Grenze zwischen schlechten und nicht schlechten Aktionen festgelegt werden können, denn die Belohungsfunktion kann immer entsprechend skaliert werden. Der Lernvorgang wird hierdurch nicht beeinflusst.)

Die Konditionale, mit denen κ zum Speichern einer schlechten Aktion in einem bestimmten Zustand *s* revidiert wird, haben die Form $\left(\overline{Action = a} \middle| s_{sym}\right)$, wobei s_{sym} die symbolische Darstellung des Zustands *s* ist, in dem die Aktion *a* schlecht ist. Die

Interpretation dieses Konditionals ist, dass *a* in *s* nicht plausibel sein soll.

Die Konditionale, mit denen κ zum Speichern bester Aktionen in einem bestimmten Zustand *s* revidiert wird, haben die Form $\left(\bigvee_i Action = a_i \middle| s_{sym}\right)$, wobei jedes a_i eine laut den neuen *Q*-Werten beste Aktion in *s* ist. Die Disjunktion ergibt sich aus der Überlegung, dass *jede* Aktion in der Konsequenz des Konditionals momentan als eine beste in *s* gilt und sie deshalb alle in *s* plausibel sind. Die übrigen dem Agenten bekannten Aktionen sollen hingegen nicht plausibel sein, da sie als nicht optimal in *s* gelten.

Diese beiden Formen von Konditionalen realisieren den Erwerb von episodischem Wissen, denn die Regeln beziehen sich auf spezielle Situationen. Das allgemeinere deklarative Wissen wird durch die im Folgenden beschriebene Generalisierung erworben.

Definition 4.3.5.2:

Ein 'Muster' ist eine Konjunktion von Literalen *einiger* der *für das Zustandssignal* vereinbarten Variablen (also excl. der Variablen *Action*). Der Unterschied zur symbolischen Darstellung eines Zustands ist, dass bei einem Muster *nicht alle* Variablen vorhanden sein müssen.

Definition 4.3.5.3:

Seien *p* ein Muster und $s = (s_{ID}, s_{sym})$ ein Zustand. '*s* enthält *p*' genau dann, wenn gilt: $s_{sym} \models p$. Die Länge des Musters, d. h. die Anzahl der mit einem Wert belegten Variablen, wird mit '$|p|$' bezeichnet.

Stellt der Agent (wie später erläutert) fest, dass bei einem bestimmten Muster *p* in den symbolischen Darstellungen verschiedener Zustände eine bestimmte Aktion *a* immer schlecht ist, so revidiert er mit dem Konditional ($\overline{Action = a} \middle| p$). Die Interpretation dieses Konditionals ist, dass *a* bei der Wahrnehmung von *p* nicht plausibel sein soll. Gibt es mehrere schlechte Aktionen, wird mit mehreren, entsprechenden Konditionalen dieser Form revidiert.

Beschränken sich die besten Aktionen für Zustände, die das Muster *p* enthalten,

auf eine Menge $A_p := \{a_{p.1},\ldots,a_{p.k}\} \subset A$, wobei A die Menge aller Aktionen, die der Agent kennt, ist, so wird eine Revision mit $\left(\bigvee_{i=1}^{k} Action = a_{p.i} \middle| p\right)$ durchgeführt. (Bemerkung: Es gilt i. d. R. nicht, dass *jede* Aktion aus A_p in jedem Zustand *s*, der *p* enthält, eine beste ist. Die Revision ist so zu verstehen, dass bei der Wahrnehmung von *p* nur die Aktionen aus $A \setminus A_p$ ausgeschlossen werden sollen und die besten Aktionen im speziellen Zustand *s* nur noch in der Menge A_p gesucht werden sollen.)

Beispiel 4.3.5.1:

Angenommen, der Sphinx-Agent in der Gridworld mit Hindernissen aus Beispiel 4.3.2.1 stellt fest, dass in folgenden Situationen die Aktion *GoNorth* schlecht ist:

- *TargetNorthSouth* = *NorthAhead* ∧ *TargetEastWest* = *EastAside*
  ∧ *TargetDistance* = *Middle* ∧ *ObstacleNorth* = *True*
  ∧ *ObstacleEast* = *False* ∧ *ObstacleSouth* = *False*
  ∧ *ObstacleWest* = *False*
- *TargetNorthSouth* = *NorthAside* ∧ *TargetEastWest* = *EastAside*
  ∧ *TargetDistance* = *Middle* ∧ *ObstacleNorth* = *True*
  ∧ *ObstacleEast* = *False* ∧ *ObstacleSouth* = *False*
  ∧ *ObstacleWest* = *False*
- *TargetNorthSouth* = *SouthAside* ∧ *TargetEastWest* = *EastAhead*
  ∧ *TargetDistance* = *Middle* ∧ *ObstacleNorth* = *True*
  ∧ *ObstacleEast* = *False* ∧ *ObstacleSouth* = *False*
  ∧ *ObstacleWest* = *True*

So erkennt er u. a. das Muster $p := (ObstacleNorth = True)$ und prüft, ob momentan wenigstens eine Situation bekannt ist, in der *p* auch aufgetreten ist, aber in der *GoNorth* als nicht schlecht gewertet wurde. Ist dies nicht der Fall, ist dies ein Hinweis darauf, dass *GoNorth* allgemein nicht ausgeführt werden sollte, wenn *p* beobachtet wird, und es wird mit dem Konditional $\left(\overline{Action = GoNorth} \middle| OstacleNorth = True\right)$ revidiert.

Beispiel 4.3.5.2:

Gegeben sei wieder die Gridworld mit Hindernissen. Sei s_{sym} gegeben durch

$$
\begin{array}{llll}
 & TargetNorthSouth & = NorthAhead & \wedge\ TargetEastWest & = EastAside \\
\wedge & TargetDistance & = Middle & \wedge\ ObstacleNorth & = True \\
\wedge & ObstacleEast & = False & \wedge\ ObstacleSouth & = False \\
\wedge & ObstacleWest & = False & &
\end{array}
$$

und wähle der Agent (z. B. zur Exploration) die Aktion *GoNorth*, was zu einer Belohnung echt kleiner als -1 führt (siehe Definition 4.3.5.1).

Der Sphinx-Agent stellt auf Grund vorheriger Erfahrungen den Zusammenhang 'Wenn *ObstacleNorth* = *True*, dann $\overline{Action = GoNorth}$.' fest. Er revidiert zuerst mit der allgemeinen Regel $\left(\overline{Action = GoNorth} \middle| ObstacleNorth = True\right)$. Für Zustände, deren symbolische Darstellungen das Muster *ObstacleNorth* = *True* enthalten, sind nun i. d. R. nur noch die Aktionen *GoEast*, *GoSouth* und *GoWest* plausibel. Der Agent wird dann nicht noch einmal versuchen (außer z. B. in Fällen, bei denen sich die Policy für eine explorative Aktion entscheidet), nach Norden zu gehen, wenn dort ein Hindernis ist, auch wenn es sich um einen noch unbekannten Zustand handelt.

Anschließend wird mit der speziellen Regel $\left(\overline{Action = GoNorth} \middle| s_{sym}\right)$ revidiert. Dies ist ggfs. notwendig, denn nach der vorangegangen Revision mit $\left(\overline{Action = GoNorth} \middle| ObstacleNorth = True\right)$ gilt mit Sicherheit nur

$$
\begin{aligned}
&\kappa\left(ObstacleNorth = True \wedge Action = GoNorth\right) \\
&> \kappa\left(ObstacleNorth = True \wedge \overline{Action = GoNorth}\right)
\end{aligned}
$$

aber nicht notwendiger Weise auch

$$\kappa\left(s_{sym} \wedge Action = GoNorth\right) > \kappa\left(s_{sym} \wedge \overline{Action = GoNorth}\right).$$

Nach der Revision mit $\left(\overline{Action = GoNorth} \middle| ObstacleNorth = True\right)$ ist *GoNorth* in Verbindung mit s_{sym} zwar (zumindest) weniger plausibel als vor der Revision, dennoch können vorhergehende Revisionen, die s_{sym} betrafen und die *GoNorth* in s_{sym} plausibel erscheinen ließen, dafür sorgen können, dass *GoNorth* auch weiterhin noch in s_{sym} plausibel ist. Dies ist unerwünscht, wie die Belohnung, die kleiner als -1 war, belegt, und daher wird mit $\left(\overline{Action = GoNorth} \middle| s_{sym}\right)$ revidiert und so auch

$$\kappa\left(s_{sym} \wedge Action = GoNorth\right) > \kappa\left(s_{sym} \wedge \overline{Action = GoNorth}\right)$$

erreicht.

Danach findet der Agent zusätzlich den Zusammenhang

'Wenn *TargetNorthSouth* = *NorthAhead* $\wedge$ *TargetEastWest* = *EastAside*,
dann *Action* = *GoNorth* $\vee$ *Action* = *GoEast*.'

und revidiert entsprechend. Für u. a. s_{sym} leitet sich aus κ nun i. d. R. die Aktion *GoEast* ab, da es sowohl das Muster *ObstacleNorth* = *True* also auch das Muster *TargetNorthSouth* = *NorthAhead* $\wedge$ *TargetEastWest* = *EastAside* enthält. Für (insbesondere unbekannte) Zustände, deren symbolische Darstellung nur das Muster *TargetNorthSouth* = *NorthAhead* $\wedge$ *TargetEastWest* = *EastAside* enthalten, leitet sich i. A. ab, dass eine beste Aktion vermutlich in der Menge $\{GoNorth, GoEast\}$ liegt.

Falls zuvor schon andere Revisionen durchgeführt wurden, können für einige Zustände auch andere (z. B. speziellere) Aktionsmengen folgen. Was im Einzelfall plausibel ist, hängt davon ab, welche Regeln bereits hinzugefügt wurden und wie stark sie gemäß Definition 3.5.4.2 epistemisch verwurzelt wurden. Letzteres wird dadurch bestimmt, für wie sicher der Agent die Regel hält, was im Wesentlichen davon abhängt, wie oft die Regel laut seiner bisherigen Erfahrung zutraf und wie viele Gegenbeispiele gefunden wurden.

Zuletzt bestimmt der Sphinx-Agent noch die Menge der für speziell *s* laut aktuellen *Q*-Werten besten Aktionen. Sei diese Menge $\{GoEast\}$ und sei *GoEast* schon bei den letzten Besuchen von *s* eine beste Aktion gewesen, so dass davon auszugehen ist, dass *GoEast* tatsächlich eine beste Aktion in *s* ist. Dann wird revidiert mit dem Konditional $(Action = GoEast | s_{sym})$.

Um eine zuverlässige generalisierte Regel über eine schlechte Aktion *a* mit dem Muster *p* als Prämisse zu erstellen, ist es notwendig zu wissen, wie ‘oft’ *a* laut dem aktuellen Erfahrungsstand eine schlechte Aktion war, wenn *p* im Zustandssignal enthalten war, denn je ‘öfter’ dies der Fall war und je ‘seltener’ *a* bei der Wahrnehmung von *p* eine nicht schlechte Aktion war (d. h. je weniger Gegenbeispiele es gibt), desto eher kann man von der Richtigkeit des Zusammenhangs von *p* und $\overline{a}$ ausgehen.

Um den Begriff ‘oft’ zu formalisieren, gibt es zwei Ansätze. Der erste Ansatz ist, einen Zähler zu benutzen, der mitzählt, wie viele Male *p* im Zustandssignal

wahrgenommen wurde und anschließend die Aktion *a* gewählt und als schlecht im Sinne von Definition 4.3.5.1 bewertet wurde. Sei dieser Zähler $freq(\bar{a},p)$. Dieser Ansatz hat aber – wie sich in Experimenten gezeigt hat – in einigen Umwelten den Nachteil, dass, wenn ein Zustand, der *p* enthält, sehr häufig vorkommt, der Verdacht erhärtet wird, dass ein Zusammenhang zwischen *p* und $\bar{a}$ besteht, obwohl dieser in den seltener vorkommenden Zuständen, die auch *p* enthalten, nicht besteht. Daraus ergibt sich der zweite Ansatz, nämlich das Verfahren so zu modifizieren, dass für jeden Zustand, der *p* enthält, der Zähler $freq(\bar{a},p)$ nur einmal erhöht werden darf, so dass häufig vorkommende und selten vorkommende Zustände dasselbe Gewicht haben. Damit der Anwender wählen kann, welchen Ansatz er verwenden möchte, wird ein Bool'scher Parameter *useEveryVisitForGeneralization* verwendet. Wird er auf *True* gesetzt, wird Ansatz 1 verwendet, ansonsten Ansatz 2.

Da sich die Umwelt im Verlauf der Zeit ändern kann und so Aktionen, die in der Vergangenheit in einem Zustand *s*, der *p* enthält, schlecht waren, dies in der Gegenwart nicht mehr sein müssen, muss die Möglichkeit bestehen, $freq(\bar{a},p)$ zu korrigieren, wenn erfahren wird, dass *a* in *s* nun nicht mehr schlecht ist. Wird Ansatz 2 verwendet, muss der Zähler $freq(\bar{a},p)$ um 1 dekrementiert werden, weil er durch die vergangenen Erfahrungen in *s* nur einmal erhöht wurde. Bei Ansatz 1 wurde er jedes Mal, wenn Zustand *s* besucht und *a* ausgeführt wurde, um 1 erhöht. Da nun die Erfahrung gemacht wurde, dass *a* in *s* nicht mehr schlecht ist, muss der Zähler $freq(\bar{a},p)$ um *n* dekrementiert werden, wenn er zuvor *n*-mal durch das Ausführen von *a* in *s* inkrementiert wurde. Es wird also zusätzlich zu $freq(\bar{a},p)$ noch ein weiterer Zähler $bad(a,s)$ benötigt, der angibt, ob (bei Ansatz 2) bzw. wie oft (bei Ansatz 1) *s* dazu beigetragen hat, dass $freq(\bar{a},p)$ inkrementiert wurde. Für Ansatz 2 nimmt $bad(a,s)$ nur die Werte 0 (falls *s* noch nicht zur Erhöhung von $freq(\bar{a},p)$ beigetragen hat) oder 1 (falls *s* bereits zur Erhöhung von $freq(\bar{a},p)$ beigetragen hat) an. Für Ansatz 1 nimmt $bad(a,s)$ den Wert 0 an, falls *s* noch nicht zur Erhöhung von $freq(\bar{a},p)$ beigetragen hat. Ansonsten gibt $bad(a,s)$ an, wie oft *s* bereits zur Erhöhung von $freq(\bar{a},p)$ beigetragen hat. Gilt *a* in *s* nicht mehr als schlechte Aktion, wird $freq(\bar{a},p)$ um $bad(a,s)$ dekrementiert und $bad(a,s)$ wieder auf 0 gesetzt, weil *s* nun nicht mehr zur Erhöhung von $freq(\bar{a},p)$ beiträgt.

Zudem muss, wie bereits erwähnt, dem Agenten die Information zur Verfügung gestellt werden, wie viele Gegenbeispiele es gibt, d. h. wie 'oft' *a* laut dem aktuellen Erfahrungsstand eine nicht schlechte Aktion war, wenn *p* im Zustandssignal enthalten war. Hierzu werden zwei Zähler $freq(\text{not}\,\overline{a},p)$ bzw. $notBad(a,s)$ verwendet, welche analog zu $freq(\overline{a},p)$ bzw. $bad(a,s)$ definiert sind, d. h. $freq(\text{not}\,\overline{a},p)$ gibt an, wie 'oft' *a* eine nicht schlechte Aktion war, wenn *p* im Zustandssignal enthalten war, und $notBad(a,s)$ gibt an, ob (bei Ansatz 2) bzw. wie oft (bei Ansatz 1) *s* dazu beigetragen hat, dass $freq(\text{not}\,\overline{a},p)$ inkrementiert wurde.

Auch für eine zuverlässige generalisierte Regel über beste Aktionen A_p mit dem Muster *p* als Prämisse muss für jede der dem Agenten bekannten Aktionen die Information zugänglich sein, wie 'oft' sie laut dem aktuellen Erfahrungsstand eine beste Aktion war, wenn *p* im Zustandssignal enthalten war, bzw. wie 'oft' sie es nicht war. Hierzu werden Zähler $freq(a,p)$ und $best(a,s)$ bzw. $freq(\text{not}\,a,p)$ und $notBest(a,s)$ verwendet.

Um eine zuverlässige Regel über eine schlechte Aktion in einem bestimmten Zustand zu erstellen, wird ein Zähler $age(\overline{a},s)$ verwendet, der angibt, wie oft in Folge *a* als schlechte Aktion in *s* gewertet wurde. Je größer sein Wert ist, desto sicherer kann davon ausgegangen werden, dass *a* tatsächlich eine schlechte Aktion in *s* ist.

Um eine zuverlässige Regel über beste Aktionen in einem bestimmten Zustand zu erstellen, wird ein Zähler $age(a,s)$ verwendet, der angibt, wie oft in Folge *a* als beste Aktion in *s* gewertet wurde. Je größer sein Wert ist, desto sicherer kann davon ausgegangen werden, dass *a* tatsächlich eine beste Aktion in *s* ist.

(Bemerkung: Falls für die Generalisierung Ansatz 1 verwendet wird, gilt $age(\overline{a},s) = bad(a,s)$ und $age(a,s) = best(a,s)$.)

Folgende Definition gibt eine kompakte Übersicht über alle Zähler.

Definition 4.3.5.4:

'$bad(a,s)$' gibt an, ob bzw. wie oft in Folge *a* als schlechte Aktion in *s* gewertet wurde.

'$notBad(a,s)$' gibt an, ob bzw. wie oft in Folge *a* in *s* als nicht schlechte Aktion in *s* gewertet wurde.

‘$age(\overline{a},s)$’ gibt an, wie oft in Folge *a* als schlechte Aktion in *s* gewertet wurde.

‘$best(a,s)$’ gibt an, ob bzw. wie oft in Folge *a* als eine beste Aktion in *s* gewertet wurde.

‘$notBest(a,s)$’ gibt an, ob bzw. wie oft in Folge *a* als eine nicht beste Aktion in *s* gewertet wurde.

‘$age(a,s)$’ gibt an, wie oft in Folge *a* als eine beste Aktion in *s* gewertet wurde.

‘$freq(\overline{a},p)$’ gibt an, wie oft die Aktion *a* als eine schlechte Aktion bei Wahrnehmung des Musters *p* gewertet wurde. Es gilt: $freq(\overline{a},p) = \sum_{s \in S | s_{sym} \models p} bad(a,s)$.

‘$freq(\text{not}\,\overline{a},p)$’ gibt an, wie oft die Aktion *a* als eine nicht schlechte Aktion bei Wahrnehmung des Musters *p* gewertet wurde. Es gilt: $freq(\text{not}\,\overline{a},p) = \sum_{s \in S | s_{sym} \models p} notBad(a,s)$.

‘$freq(a,p)$’ gibt an, wie oft die Aktion *a* als eine beste Aktion bei Wahrnehmung des Musters *p* gewertet wurde. Es gilt: $freq(a,p) = \sum_{s \in S | s_{sym} \models p} best(a,s)$.

‘$freq(\text{not}\,a,p)$’ gibt an, wie oft die Aktion *a* als eine nicht beste Aktion bei Wahrnehmung des Musters *p* gewertet wurde. Es gilt: $freq(\text{not}\,a,p) = \sum_{s \in S | s_{sym} \models p} notBest(a,s)$.

(Bemerkung: Der Bruch $\frac{freq(a,p)}{freq(a,p) + freq(\text{not}\,a,p)}$ lässt sich im Wesentlichen als die relative Häufigkeit ansehen, dass *a* bei der Wahrnehmung von *p* eine beste Aktion ist. Entsprechendes gilt für $\frac{freq(\overline{a},p)}{freq(\overline{a},p) + freq(\text{not}\,\overline{a},p)}$.)

Zustandssignale enthalten sehr viele Muster, insbesondere dann, wenn sie aus vielen Variablen bestehen, was zu einem hohen Speicherbedarf und Rechenaufwand führt. Daher sollte für den Anwender die Möglichkeit bestehen, für die Revision mit Information über eine schlechte Aktion und die Revision mit Information über beste Aktionen jeweils eine maximal zu betrachtende Musterlänge festlegen zu können. Dies wird über die Parameter *badMaxPatternLength* und *bestMaxPatternLength*

realisiert. So werden nur Muster p mit $|p| \leq badMaxPatternLength$ bei der Revision mit Information über eine schlechte Aktion bzw. $|p| \leq bestMaxPatternLength$ bei der Revision mit Information über beste Aktionen berücksichtigt.

Zudem wird eine geeignete Datenstruktur benötigt, welche dem Agenten abhängig von Aktionen a und Mustern p mitteilt, welche Werte $freq(\overline{a}, p)$, $freq(\text{not}\,\overline{a}, p)$, $freq(a, p)$ und $freq(\text{not}\,a, p)$ haben.

Definition 4.3.5.5:

Seien die Domains $d_1 := \{v_{1.1}, \ldots, v_{1.m_1}\}$, ..., $d_n := \{v_{n.1}, \ldots, v_{n.m_n}\}$ gegeben. Dann ist ein zugehöriger 'Pattern-Frequency-Tree' ein Baum mit folgendem Aufbau:

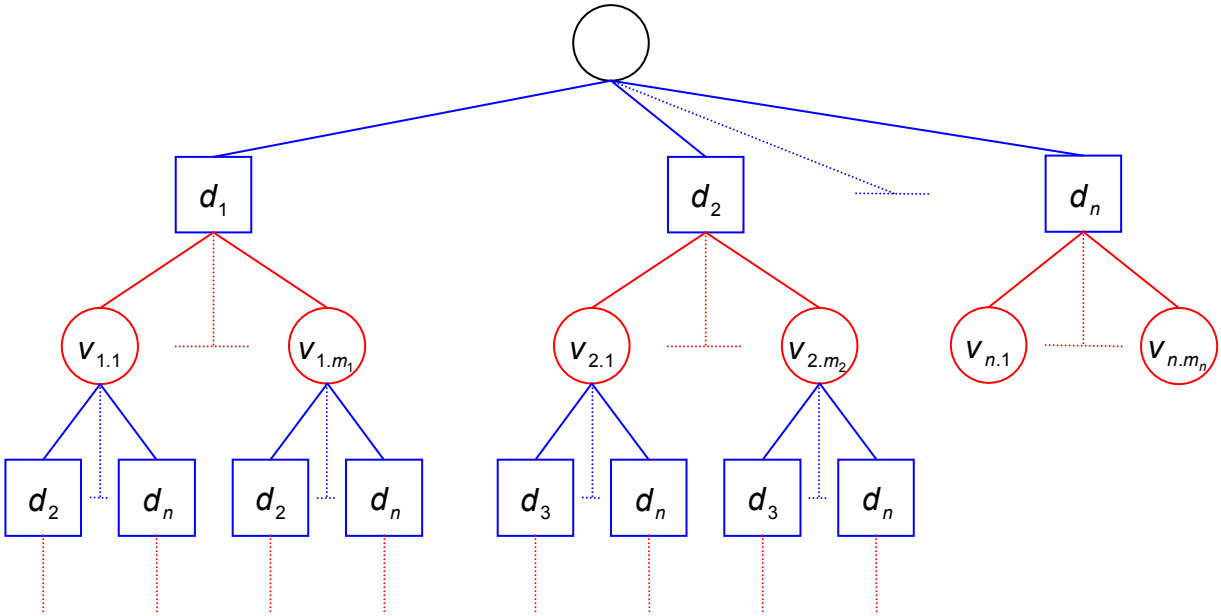


Abbildung 4.3.5.1

Im obersten Level sind alle Domains vertreten und werden aufgesplittet in ihre jeweiligen möglichen Werte. Alle Werte-Knote repräsentieren ein eindeutiges Muster p, welches sich durch den Pfad von der Wurzel bis zu dem Knoten ergibt. (Der oberste linke Werte-Knoten z. B. steht für das Muster $d_1 = v_{1.1}$). Unter den Werte-Knoten befindet sich wieder ein Level mit Domain-Knoten, wobei nur noch Domains mit einem höheren Index als dem des Parent-Domain-Knotens aufgeführt werden. (Daher endet auf der rechten Seite der Ast unter d_n schon im ersten Level.) Unter jedem Domain-Knoten befinden sich wieder alle Werte-Knoten der jeweiligen Domain.

Die Werte-Knoten des obersten Levels stellen alle Muster der Länge 1 dar, die

des zweiten Levels die Muster der Länge 2 usw.. (Der linkeste Werte-Knoten des 2. Levels (nicht in Abbildung 4.3.5.1 zu sehen) z. B. repräsentiert das Muster $d_1 = v_{1.1} \wedge d_2 = v_{2.1}$.)

Auf Grund dieses Aufbaus wird im Pattern-Frequency-Tree jedes Muster durch einen Werte-Knoten dargestellt. In den Werte-Knoten werden dann jeweils zwei Zahlen gespeichert, nämlich entweder $freq(\bar{a}, p)$ und $freq(\text{not}\,\bar{a}, p)$ oder $freq(a, p)$ und $freq(\text{not}\,a, p)$.

Pro bekannter Aktion werden zwei Pattern-Frequency-Trees erstellt – einer für die Werte $freq(\bar{a}, p)$ und $freq(\text{not}\,\bar{a}, p)$ und einer für die Werte $freq(a, p)$ und $freq(\text{not}\,a, p)$. Die maximal notwendige Tiefe liegt bei *badMaxPatternLength* bzw. *bestMaxPatternLength*, denn längere Muster sollen nicht betrachtet werden.

Im Folgenden werden die beiden zentralen Algorithmen des Sphinx-Learnings vorgestellt, welche zusammen die Punkte 8 und 9 von Abbildung 4.3.1.1 realisieren, nämlich die Revision von κ mit Hilfe der neuen *Q*-Werte. Der erste Algorithmus revidiert mit Information über eine schlechte Aktion, der zweite mit Information über beste Aktionen. Der Aufbau der Algorithmen ist ähnlich. Sei *s* der Zustand, in dem soeben die Aktion *a* ausgeführt wurde und *r* die erhaltene Belohnung. Zunächst wird versucht, generalisierte Regeln zu erstellen. Dazu werden die oben genannten Zähler aktualisiert und für jedes in *s* enthaltene Muster *p*, dessen Länge die maximal zu betrachtende Länge nicht überschreitet, geprüft, ob die *freq*-Zähler gewisse, von *p* abhängige Schwellwerte $t_{\ldots}(p)$ überschreiten bzw. nicht überschreiten und ggfs. eine entsprechende Regel in Form eines Konditionals gebildet und κ revidiert. Anschließend wird geprüft, ob sich eine Regel für den speziellen Zustand *s* erstellen lässt. Auch dies wird an Hand von Schwellwerten $t_{\ldots}$ entschieden. (Bemerkung: Wie genau die Schwellwerte definiert werden können, wird jeweils im Anschluss an den Algorithmus beschrieben. Ihre jeweilige Bedeutung wird durch die Kommentare im Algorithmus verdeutlicht.)

Algorithmus 4.3.5.1 (Revision mit Information über eine schlechte Aktion):

1. If $r < -1$: ‘Falls die Belohnung schlecht war

1.1. If $bad(a,s) = 0$ Or *useEveryVisitForGeneralization*: ‘Falls *a* in *s* noch nicht als schlechte Aktion gilt oder Zustände mehrfach Einfluss haben sollen

1.1.1. $\forall p | s_{sym} \models p$ und $|p| \leq badMaxPatternLength$: ‘Für alle zu betrachtenden Untermuster *p* von *s*

1.1.1.1. $freq(\text{not}\,\overline{a}, p) -= notBad(a,s)$ ‘Falls *a* in *s* bisher $notBad(a,s)$-mal als nicht schlechte Aktion gewertet wurde, wurde $freq(\text{not}\,\overline{a}, p)$ entsprechend oft um 1 inkrementiert. Dies muss rückgängig gemacht werden, da *a* wegen $r < -1$ nun als schlechte Aktion in *s* gelten soll.

1.1.1.2. $freq(\overline{a}, p) += 1$ ‘*p* wird einmal mehr mit $\overline{a}$ in Verbindung gebracht, da $s_{sym} \models p$ und $r < -1$.

1.1.1.3. If $freq(\overline{a}, p) \geq t_{-}(p)$ And $freq(\text{not}\,\overline{a}, p) \leq t_{\text{not}-}(p)$: Revidiere mit $(\overline{Action = a} | p)$. ‘Falls *p* oft genug mit $\overline{a}$ assoziiert wird und es nicht zu viele Gegenbeispiele gibt, revidiere mit $(\overline{Action = a} | p)$.

1.1.2. $notBad(a,s) := 0$ ‘*a* gilt nicht mehr als nicht schlechte Aktion in *s*.

1.1.3. $bad(a,s) += 1$ ‘*a* gilt ein weiteres Mal als schlechte Aktion in *s*.

1.2. $age(\overline{a}, s) += 1$ ‘*a* gilt einmal länger als schlechte Aktion in *s*.

1.3. If $age(\overline{a}, s) \geq t_{age-}$: Revidiere mit $(\overline{Action = a} | s_{sym})$. ‘Falls *a* in *s* lang genug als schlechte Aktion gilt, soll speziell in *s* die Aktion *a* nicht plausibel sein.

2. Else: ‘Falls die Belohnung nicht schlecht war

2.1. If $notBad(a,s) = 0$ Or *useEveryVisitForGeneralization*: ‘Falls *a* in *s* noch nicht als nicht schlechte Aktion gilt oder Zustände mehrfach Einfluss haben sollen

2.1.1. $\forall p | s_{sym} \models p$ und $|p| \leq badMaxPatternLength$: ‘Für alle zu betrachtenden Untermuster *p* von *s*

2.1.1.1. $freq(\overline{a}, p) -= bad(a,s)$ ‘Falls *a* in *s* bisher $bad(a,s)$-mal als schlechte Aktion gewertet wurde, wurde $freq(\overline{a}, p)$ entsprechend oft um 1 inkrementiert. Dies muss rückgängig gemacht werden, da *a* wegen $r \geq -1$ nun als nicht schlechte Aktion in *s* gelten soll.

2.1.1.2. $freq(\text{not}\,\overline{a}, p) += 1$ ‘*p* wird einmal mehr nicht mit $\overline{a}$ in Verbindung gebracht, da $s_{sym} \models p$ und $r \geq -1$.

2.1.2. $bad(a,s) := 0$ ‘*a* gilt nicht mehr als schlechte Aktion in *s*.

2.1.3. $notBad(a,s) += 1$ ‘*a* gilt ein weiteres Mal als nicht schlechte Aktion in *s*.

2.2. $age(\bar{a},s) := 0$ ‘*a* gilt nicht mehr als schlechte Aktion in *s*.

Der Schwellwert $t_(p)$ gibt – abhängig vom Muster *p* – an, welchen Wert $freq(\bar{a},p)$ mindestens haben muss, damit der Agent mit $\left(\overline{Action = a} \middle| p\right)$ revidieren kann. Eine sinnvolle Weise $t_(p)$ zu definieren ist, kurzen Mustern einen hohen Schwellwert zuzuweisen, da sie in vielen Zuständen auftreten und eine Revision mit einem kurzem Muster als Prämisse somit große Auswirkungen hat. Eine Formel, die diese Überlegung berücksichtigt, ist:

$$t_(p) := 2^{badMaxPatternLength - |p| + badOffset},$$

wobei *badOffset* den Schwellwert unabhängig von der Länge des Musters zusätzlich erhöht.

Eine weitere Heuristik zur Definition von $t_(p)$ ist, selten vorkommenden Mustern (unabhängig von ihrer Länge) einen kleineren Schwellwert zuzuweisen als häufigen Mustern, weil es möglicherweise nicht genügend Zustände, die *p* enthalten, gibt, so dass $freq(\bar{a},p)$ den Wert $t_(p) := 2^{badMaxPatternLength - |p| + badOffset}$ überschreiten kann, selbst wenn der Zusammenhang zwischen *p* und $\bar{a}$ besteht.

Sei $\Pr(p)$ die Wahrscheinlichkeit, mit der *p* bisher aufgetreten ist. Formal:

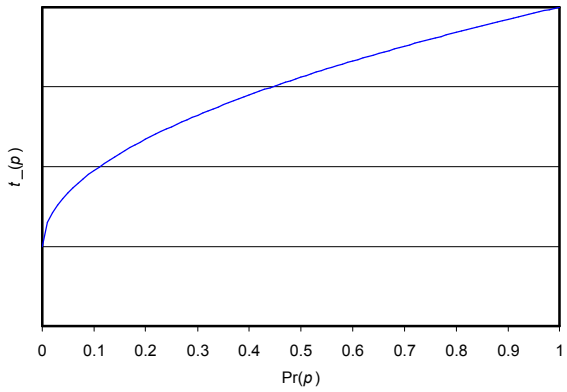
$$\Pr(p) := \begin{cases} \dfrac{\text{Anzahl erlebter, } \textit{verschiedener} \text{ Zustände, die } p \text{ enthielten}}{\text{Anzahl erlebter, } \textit{verschiedener} \text{ Zustände}}, \\ \quad \text{falls } useEveryVisitForGeneralization = False \text{ (Ansatz 2)} \\ \\ \dfrac{\text{Anzahl erlebter Zustände, die } p \text{ enthielten}}{\text{Anzahl erlebter Zustände}}, \\ \quad \text{falls } useEveryVisitForGeneralization = True \text{ (Ansatz 1)} \end{cases}$$

Definiert man

$$t_(p) := \left(\omega_ + (1 - \omega_)\sqrt{\Pr(p)}\right) 2^{badMaxPatternLength - |p| + badOffset},$$

wobei $\omega_{_} \in [0;1]$ eine Gewichtung darstellt, so werden beide Heuristiken kombiniert. Für einen festen Wert von $|p|$ (z. B. $|p| = badMaxPatternLength$) ergibt sich folgende, typische Schwellwertkurve.

Diagramm 4.3.5.1:



Der Parameter $\omega_{_}$ regelt u. a. den minimalen Schwellwert (bei $\Pr(p) = 0$). In Diagramm 4.3.5.1 wurde $\omega_{_} := \frac{1}{4}$ gewählt. Für $\omega_{_} := 1$ erhält man die ursprüngliche Heuristik $t_{_}(p) = 2^{badMaxPatternLength - |p| + badOffset}$. (Bemerkung: Statt der Wurzelfunktion ($\sqrt{\Pr(p)}$) können auch andere, monoton steigende Funktionen f mit den Eigenschaften $f(0) = 0$ und $f(1) = 1$ geeignet sein.)

Die Bedingung $freq(\text{not } \overline{a}, p) \leq t_{\text{not}_}(p)$ lässt eine Revision mit $(\overline{Action = a} | p)$ nur zu, wenn die Anzahl der Gegenbeispiele zu $(\overline{Action = a} | p)$ den Schwellwert $t_{\text{not}_}(p)$ nicht überschreitet. Oft reicht es, $t_{\text{not}_}(p) := 0$ zu setzen. Um die Allgemeinheit zu wahren, wurde in Algorithmus 4.3.5.1 statt des festen Wertes 0 die flexiblere Variante $t_{\text{not}_}(p)$ eingesetzt.

Damit eine Revision mit der Regel $(\overline{Action = a} | s_{sym})$ geschehen kann, sollte der Agent sich sicher sein, dass *a* tatsächlich eine in *s* schlechte Aktion ist. Ein Indiz dafür ist, dass *a* schon mehrmals in Folge als schlecht in *s* gewertet wurde, wenn also $age(\overline{a}, s)$ groß ist. Der Schwellwert $t_{age_}$ regelt hierbei, welchen Wert $age(\overline{a}, s)$ mindestens besitzen muss, damit mit $(\overline{Action = a} | s_{sym})$ revidiert werden kann. I. d. R. kann $t_{age_} := 1$ gesetzt werden.

Außer Information über eine schlechte Aktion soll der Sphinx-Agent auch Information über beste Aktionen verarbeiten. Sei für den folgenden Algorithmus für die Revision mit Information über beste Aktionen *s* der Zustand, in dem soeben eine Aktion ausgeführt wurde. Sei *bestActions* die Menge der laut den soeben (siehe Punkt 7 von Abbildung 4.3.1.1) aktualisierten *Q*-Werten besten Aktionen in *s*, d. h.:

$$bestActions := \left\{ a \in A \middle| Q(s,a) = \max_{\tilde{a} \in A} Q(s,\tilde{a}) \right\}.$$

Algorithmus 4.3.5.2 (Revision mit Information über beste Aktionen):

1. *patternFrequenciesHaveChanged* := *False* ‘Ein Flag, das später anzeigt, ob die *freq*-Werte geändert wurden.

2. $\forall a \in A$: ‘Für alle Aktionen *a*

2.1. If $a \in bestActions$: ‘Falls *a* eine beste Aktion in *s* ist

2.1.1. If $best(a,s) = 0$ Or *useEveryVisitForGeneralization*: ‘Falls *a* in *s* noch nicht als beste Aktion gilt oder Zustände mehrfach Einfluss haben sollen

2.1.1.1. $\forall p | s_{sym} \models p$ und $|p| \leq bestMaxPatternLength$: ‘Für alle zu betrachtenden Untermuster *p* von *s*

2.1.1.1.1. $freq(\text{not } a,p) -= notBest(a,s)$ ‘Falls *a* in *s* bisher $notBest(a,s)$-mal als nicht beste Aktion gewertet wurde, wurde $freq(\text{not } a,p)$ entsprechend oft um 1 inkrementiert. Dies muss rückgängig gemacht werden, da *a* wegen $a \in bestActions$ nun als beste Aktion in *s* gelten soll.

2.1.1.1.2. $freq(a,p) += 1$ ‘*p* wird einmal mehr mit *a* in Verbindung gebracht, da $s_{sym} \models p$ und $a \in bestActions$.

2.1.1.2. $notBest(a,s) := 0$ ‘*a* gilt nicht mehr als nicht beste Aktion in *s*.

2.1.1.3. $best(a,s) += 1$ ‘*a* gilt ein weiteres Mal als beste Aktion in *s*.

2.1.1.4. *patternFrequenciesHaveChanged* := *True* ‘Die *freq*-Werte wurden geändert.

2.2. Else: ‘Falls *a* eine nicht beste Aktion in *s* ist

2.2.1. If $notBest(a,s) = 0$ Or *useEveryVisitForGeneralization*: ‘Falls *a* in *s* noch nicht

als nicht beste Aktion gilt oder Zustände mehrfach Einfluss haben sollen

2.2.1.1. $\forall p | s_{sym} \models p$ und $|p| \leq bestMaxPatternLength$: 'Für alle zu betrachtenden Untermuster *p* von *s*

2.2.1.1.1. $freq(a,p) -= best(a,s)$ 'Falls *a* in *s* bisher $best(a,s)$-mal als beste Aktion gewertet wurde, wurde $freq(a,p)$ entsprechend oft um 1 inkrementiert. Dies muss rückgängig gemacht werden, da *a* wegen $a \notin bestActions$ nun als nicht beste Aktion in *s* gelten soll.

2.2.1.1.2. $freq(\text{not } a,p) += 1$ '*p* wird einmal mehr nicht mit *a* in Verbindung gebracht, da $s_{sym} \models p$ und $a \notin bestActions$.

2.2.1.2. $best(a,s) := 0$ '*a* gilt nicht mehr als beste Aktion in *s*.

2.2.1.3. $notBest(a,s) += 1$ '*a* gilt ein weiteres Mal als nicht beste Aktion in *s*.

2.2.1.4. *patternFrequenciesHaveChanged* := *True* 'Die *freq*-Werte wurden geändert.

3. If *patternFrequenciesHaveChanged*: 'Falls die *freq*-Werte geändert wurden

3.1. $\forall p | s_{sym} \models p$ und $|p| \leq bestMaxPatternLength$: 'Für alle zu betrachtenden Untermuster *p* von *s*

3.1.1. $mostSuccessfulActions := \{\ \}$ 'In die Menge *mostSuccessfulActions* werden die für *p* erfolgreichsten Aktionen aufgenommen.

3.1.2. $\forall a \in A$: 'Für alle Aktionen *a*

3.1.2.1. If $freq(a,p) \geq t_{+}(p)$ And $freq(\text{not } a,p) \leq t_{\text{not}+}(p)$: $mostSuccessfulActions := mostSuccessfulActions \cup \{a\}$ 'Falls *p* oft genug mit *a* assoziiert wird und es nicht zu viele Gegenbeispiele gibt, nimm *a* in *mostSuccessfulActions* auf.

3.1.3. If $mostSuccessfulActions \neq \{\ \}$ And $mostSuccessfulActions \neq A$ And $\max_{a \in mostSuccessfulActions} freq(a,p) \geq t_{\max+}(p)$: Revidiere mit $\left(\bigvee_{a \in mostSuccessfulActions} Action = a | p \right)$. 'Falls *mostSuccessfulActions* nicht leer ist und nicht alle Aktionen enthält und falls *p* oft genug mit mindestens einer

Aktion in *mostSuccessfulActions* assoziiert wird, revidiere mit $\left(\bigvee_{a \in mostSuccessfulActions} Action = a \middle| p \right)$.

4. $\forall a \in A$: 'Für alle Aktionen *a*

4.1. If $a \in bestActions$: $age(a,s) += 1$ 'Falls *a* eine beste Aktion in *s* ist, inkrementiere den *age*-Wert um 1,

4.2. Else: $age(a,s) := 0$ 'Falls *a* eine nicht beste Aktion in *s* ist, setze den *age*-Wert auf 0.

5. If $\max_{a \in bestActions} age(a,s) \geq t_{age+}$ And $bestActions \neq A$: Revidiere mit $\left(\bigvee_{a \in bestActions} Action = a \middle| s_{sym} \right)$. 'Falls mindestens ein *a* aus *bestActions* lang genug als eine in *s* beste Aktion gilt, sollen speziell in *s* die Aktionen aus *bestActions* plausibel sein.

Die Schwellwerte haben folgende Bedeutungen:

- $t_+(p)$: Damit eine Aktion in *mostSuccessfulAction* aufgenommen werden kann, muss ihr $freq(a,p)$-Wert den Schwellwert $t_+(p)$ erreichen. Wählt man $t_+(p) := 1$, wird jede Aktion *a*, die in mindestens einem Zustand *s* mit $s_{sym} \models p$ als eine beste Aktion gilt, in *mostSuccessfulAction* aufgenommen, sofern die zweite Aufnahmebedingung auch erfüllt ist. Ein höherer Schwellwert filtert Aktionen, die bei *p* nur selten beste Aktionen sind, heraus.
- $t_{\text{not}+}(p)$: Die Anzahl der Gegenbeispiele, d. h. der Wert von $freq(\text{not}\, a, p)$, darf den Schwellwert $t_{\text{not}+}(p)$ nicht überschreiten, damit *a* der Menge *mostSuccessfulAction* hinzugefügt werden kann. Bei $t_{\text{not}+}(p) := 0$ muss *a* immer eine beste Aktion sein, wenn *p* wahrgenommen wird. Bei $t_{\text{not}+}(p) := \infty$ entscheidet allein $t_+(p)$ über die Aufnahme von *a* in *mostSuccessfulAction*.
- $t_{\max+}(p)$: Dieser Parameter hat eine ähnliche Funktion wie $t_-(p)$ bei Algorithmus 4.3.5.1. Er gibt an, wie groß $freq(a,p)$ für mindestens eine Aktion *a* aus *mostSuccessfulActions* mindestens sein muss, damit die Revision mit $\left(\bigvee_{a \in mostSuccessfulActions} Action = a \middle| p \right)$ durchgeführt werden soll. Eine sinnvolle Formel für

$t_{\max+}(p)$ ist:

$$t_{\max+}(p) := \left(\omega_{\max+} + (1-\omega_{\max+})\sqrt{\Pr(p)}\right) 2^{bestMaxPatternLength-|p|+bestOffset},$$

wobei *bestOffset* bzw. $\omega_{\max+}$ analog zu *badOffset* bzw. ω_{-} sind.

- t_{age+}: Um eine Revision mit $\left(\bigvee_{a \in bestActions} Action = a \middle| s_{sym}\right)$ zuzulassen, sollte der Sphinx-Agent sich möglichst sicher sein, dass mindestens eine beste Aktion in *bestActions* liegt. Eine beste Aktion wird nach einiger Zeit des Lernens stabil nach jedem Besuch von *s* in *bestActions* liegen, während nicht beste Aktionen möglicherweise auch mal in *bestActions* liegen werden, sich aber nicht lange darin halten können. Ein hoher $age(a,s)$-Wert ist ein Indiz dafür, dass *a* in *s* tatsächlich eine beste Aktion ist. t_{age+} definiert nun – analog zu t_{age-} in Algorithmus 4.3.5.1 –, ab wann der Agent annehmen darf, dass in *bestActions* wenigstens eine tatsächlich beste Aktion liegt. Oft reicht schon ein Wert von 2 bis 4.

Die Bedingungen $mostSuccessfulActions \neq \{\ \}$, $mostSuccessfulActions \neq A$ bzw. $bestActions \neq A$ sind insofern wichtig, als Revisionen bei $mostSuccessfulActions = \{\ \}$, $mostSuccessfulActions = A$ bzw. $bestActions = A$ nicht zu einem Informationsgewinn führen. Z. B. würde $bestActions = A$ bedeuten, dass jede Aktion eine beste ist, was erstens unwahrscheinlich ist und zweitens zu Beginn und ohne Hintergrundwissen ohnehin angenommen wird.

Damit die vorgestellten Algorithmen gute Ergebnisse liefern, ist es notwendig, die Frage, wie tief die aufgenommenen Regeln bei den Revisionen epistemisch verwurzelt werden sollen, näher zu beleuchten. Die epistemische Verwurzelung wird realisiert gemäß Definiton 3.5.4.2. Da sowohl die RL- als auch die BR-Komponente des Sphinx-Learnings einen Parameter α verwenden (siehe Algorithmus 2.6.3.1 und Definition 3.5.4.2), wird im Folgenden zur besseren Unterscheidung der Parameter α des RL mit α_{RL} und der der BR mit α_{BR} bezeichnet.

Eine generalisierte Regel ist umso glaubwürdiger, je mehr Situationen es gibt, in denen sie sich bewahrheitet hat, d. h. je höher der Wert $freq(\overline{a},p)$ (für Revision mit Information über eine schlechte Aktion) bzw. $\max_{a \in mostSuccessfulActions} freq(a,p)$ (für Revision mit Information über beste Aktionen) ist. Bei neu gefundenen Regeln wird der

entsprechende *freq*-Wert i. A. den jeweiligen Schwellwert nur knapp überschreiten, während bewährte Regeln im Vergleich zum Schwellwert einen hohen *freq*-Wert besitzen. Durch dieses Verhältnis wird ein α_{BR} für die Revision mit der neuen Regel bestimmt. Damit die Werte für α_{BR} nicht zu schnell wachsen, wenn der *freq*-Wert im Vergleich zum Schwellwert mit der Zeit größer wird, und in Anlehnung an die von Goldszmidt und Pearl vorgeschlagene, logarithmische Staffelung der OCF-Ebenen [GuP1996], empfiehlt sich folgende Berechnungsvorschrift für α_{BR} bei der Revision mit der Regel $\left(\overline{Action = a} \middle| p\right)$:

$$\alpha_{BR} := \max\left\{1, \left\lfloor \log_{\max\{2, t_{-}(p)\}} freq(\overline{a}, p) \right\rfloor\right\}$$

(Bemerkung: Für $t_{-}(p) \geq 2$ vereinfacht sich diese Formel zu $\alpha_{BR} := \left\lfloor \log_{t_{-}(p)} freq(\overline{a}, p) \right\rfloor$ und die logarithmische Verbindung zwischen dem *freq*-Wert und dem Schwellwert wird deutlich. Da eine Revision mit $\left(\overline{Action = a} \middle| p\right)$ und somit die Berechnung vom zugehörigen α_{BR} nur ausgeführt wird, wenn $freq(\overline{a}, p) \geq \underbrace{t_{-}(p)}_{\geq 2,\, \text{n. V.}}$, gilt $\left\lfloor \log_{t_{-}(p)} freq(\overline{a}, p) \right\rfloor \geq 1$, so dass insgesamt beide $\max\{...\}$-Ausdrücke entfallen können.)

Analog wird α_{BR} bei der Revision mit der Regel $\left(\bigvee_{a \in mostSuccessfulActions} Action = a \middle| p \right)$ bestimmt:

$$\alpha_{BR} := \max\left\{1, \left\lfloor \log_{\max\{2, t_{\max+}(p)\}} \max_{a \in mostSuccessfulActions} freq(a, p) \right\rfloor\right\}$$

Spezielle Regeln für bestimmte Zustände werden glaubwürdiger, je öfter sie (zumindest in letzter Zeit) zutrafen. Diese Information stellt bei der Revision mit Information über beste Aktionen in einem betimmten Zustand der Wert $age(a, s)$ bereit. Je größer dieser relativ zum Parameter t_{age+} ist, desto eher kann bzw. soll davon ausgegangen werden, dass die Regel korrekt ist. Eine sinnvolle Berechnungsvorschrift für α_{BR} bei der Revision mit der Regel $\left(\bigvee_{a \in bestActions} Action = a \middle| s_{sym} \right)$ ist:

$$\alpha_{BR} := \max\left\{1, \left\lfloor \log_{\max\{2, t_{age+}\}} \max_{a \in bestActions} age(a,s) \right\rfloor\right\}$$

Analog kann α_{BR} für die Revision mit der Regel $\left(\overline{Action = a} \middle| s_{sym}\right)$ definiert werden durch:

$$\alpha_{BR} := \max\left\{1, \left\lfloor \log_{\max\{2, t_{age-}\}} age(\overline{a}, s) \right\rfloor\right\}$$

Der vollständige Lern-Algorithmus des Sphinx-Learnings ist im Folgenden aufgeführt (vgl. 2.6.3.1 ($Q(\lambda)$-Learning)). Dabei ist *BackgroundKnowledge* eine Menge von Regeln in Form von Konditionalen, welche der Anwender dem Agenten als Hintergrundwissen zur Verfügung stellt (siehe auch Kapitel 4.3.7).

Algorithmus 4.3.5.3 (Sphinx-Learning):

1. Erzeuge eine uniforme OCF κ, d. h. $\kappa(\omega) = 0 \forall \omega \in \Omega$ (wobei Modelle, wie in Kapitel 4.3.3 erwähnt, die Form $\omega = s_{sym} \wedge Action = a$ haben).
2. $\forall c \in BackgroundKnowledge : \kappa := \kappa * c$ 'Revidiere κ nacheinander mit allen Konditionalen des Hintergrundwissens.
3. $\forall \left(\underbrace{s}_{\substack{=(s_{ID}, s_{sym}) \\ \in S}}, \underbrace{a}_{\in A(s)} \right) : Q(s_{ID}, a) := 0$
4. $\forall (s,a) : e(s_{ID}, a) := 0$; $t := 0$; Wähle einen Startzustand s_0.
5. Do While 's_t ist kein Terminal-Zustand':

5.1. Führe im aktuellen Zustand s_t die aktuelle Aktion $a_t := \tilde{\pi}(s_t)$ (wie in Kapitel 4.3.3 beschrieben) aus.

5.2. Erhalte Belohnung r_{t+1} und gelange in Folgezustand s_{t+1}.

5.3. $\delta := r_{t+1} + \gamma \max_{a' \in A(s_{t+1})} Q((s_{t+1})_{ID}, a') - Q((s_t)_{ID}, a_t)$

5.4. $\forall (s,a)$ mit $e(s_{ID}, a) > 0$: Falls in 5.1 eine explorative Aktion gewählt wurde: $e(s_{ID}, a) := 0$; Else: $e(s_{ID}, a) := \gamma\lambda e(s_{ID}, a)$

5.5. $e((s_t)_{ID}, a_t) += 1$

5.6. $\forall (s,a)$ mit $e(s_{ID}, a) > 0$: $Q(s_{ID}, a) += \alpha_{RL} e(s_{ID}, a)\delta$

5.7. Führe Algorithmen 4.3.5.1 und 4.3.5.2 aus.

5.8. $t += 1$

6. Falls 'π in den letzten Durchläufen verändert': zurück zu 4.

7. $\pi^* := \pi$

4.3.6 Praxisnahe Varianten des Sphinx-Learnings

Sphinx-Learning benötigt viel Arbeitsspeicher und eine durch das Abfragen und Revidieren von κ deutlich höhere Rechenzeit als das klassische $Q(\lambda)$-Learning. Daher ist es sinnvoll, das theoretische Modell leicht abzuwandeln und so möglichst praxistauglich zu gestalten.

Eine Möglichkeit, die sowohl den benötigten Speicherplatz vermindet als auch die Geschwindigkeit steigert, ist, wie bereits erwähnt, die Tiefe der Pattern-Frequency-Trees zu verringern. Dadurch werden nicht mehr alle möglichen Muster repräsentiert, sondern nur noch Muster bis zu einer bestimmten Länge. Dieser Optimierungsansatz ist sogar aus theoretischer Sicht interessant, denn oft sind die Zusammenhänge zwischen Wahrnehmung und geeigneter Aktion einfacher Natur, d. h. wenige Informationen über den Zustand der Umwelt reichen aus, um gewisse Aktionen auszuschließen oder gegenüber anderen zu bevorzugen. So schließt in der Gridworld mit Hindernissen (Beispiel 4.3.2.1) das Vorhandensein eines Hindernisses im Norden bereits die Aktion *GoNorth* aus. Die Ausprägungen der übrigen Variablen spielen dafür keine Rolle. Auch reicht im Wesentlichen die Wahrnehmung *TargetNorthSouth* = *NorthAhead* $\wedge$ *TargetEastWest* = *EastAside* aus, um sich für *GoNorth* oder *GoEast* zu entschließen. Selbst an kritischen Stellen (z. B. Feld $(6,4)$) stört diese einfache Regel nicht, sofern die Regel, dass bei einem Hindernis im Norden nicht nach Norden gegangen werden soll, epistemisch tiefer verwurzelt ist. Außerdem gibt es durch die Revision mit Information über eine schlechte Aktion bzw. beste Aktionen in einem bestimmten Zustand für Ausnahme-Situationen die Möglichkeit, spezielle Verhaltensweisen festzulegen.

Durch das Beschränken der maximalen Musterlänge benötigen die Pattern-Frequency-Trees weniger Speicher und die Rechenzeit wird vermindert, da es weniger verwaltete Muster gibt deutlich und dadurch weniger Revisionen durchgeführt werden. Außerdem lässt sich für die maximale Musterlänge für Revisionen mit Information über eine schlechte Aktion ein anderer Wert wählen als für die maximale Musterlänge für Revisionen mit Information über beste Aktionen, da die zugehörigen Algorithmen 4.3.5.1 bzw. 4.3.5.2 nicht auf den Pattern-Frequency-

Tree des jeweils anderen Algorithmus zugreifen.

Eine weitere Methode, die Rechenzeit zu vermindern, ist, die Policy dem menschlichen Verhalten anzugleichen. In vertrauten Zuständen handeln Menschen meistens intuitiv und denken bei der Aktionswahl nicht mehr bewusst nach [And1983]. Das Handeln wird demnach aus dem Bottom-Level gelenkt, ohne dass das Top-Level einbezogen wird.

Die Policy kann zur Steigerung der Geschwindigkeit so abgeändert werden, dass sie zunächst prüft, wie 'vertraut' der aktuelle Zustand *s* für den Agenten ist, d. h. ob er darin so geübt ist, dass er (vermutlich) bereits eine beste Aktion für *s* gelernt hat. Dies ist – wie bereits erläutert – erkennbar an einem hohen $age(a,s)$-Wert für mindestens eine Aktion *a*. Wenn es eine solche Aktion gibt, kann die Wahl der auszuführenden Aktion dem Bottom-Level überlassen werden. Gilt ein Zustand genau dann als vertraut, wenn der $age(a,s)$-Wert für mindestens eine Aktion größer oder gleich t_{age+} ist, und würde das Top-Level wie üblich mit einbezogen, würde dadurch i. d. R. die Wahl der Aktion nicht beeinflusst, weil dort durch den hohen $age(a,s)$-Wert bereits eine Revision mit $\left(\bigvee_{a \in bestActions} Action = a \middle| s_{sym} \right)$ durchgeführt wurde und das Top-Level somit bei der Wahrnehmung von *s* dieselben Aktionen für am plausibelsten/ besten halten würde wie das Bottom-Level.

4.3.7 Verwendung von Hintergrundwissen

Die Einbindung von Hintergrundwissen, d. h. Information über die Umwelt oder vermutlich günstige Verhaltensweisen, ist beim klassischen RL problematisch. Die erste Hürde ist, dass es a priori keine Struktur für semantische Beziehungen unter den Zuständen gibt und jeder Zustand isoliert betrachtet wird. Wissen, das man dem Agenten mitgeben möchte, müsste man zunächst von einer i. d. R. symbolischen Darstellung, die viele Zustände betrifft, in die codierten Darstellungen der betreffenden Zustände konvertieren. Dabei trifft man auf die zweite Schwierigkeit, denn das Wissen müsste in sinnvolle, möglichst sich nicht widersprechende, numerische Werte übersetzt werden – beim *Q*-Learning beispielsweise in Werte für die *Q*-Funktion.

Sphinx-Learning bietet die Möglichkeit, Hintergrundwissen ohne diese Probleme verwenden zu können, indem κ zu Beginn mit entsprechenden, symbolisch

dargestellten Regeln in Form von Konditionalen revidiert wird. Durch unterschiedliche Werte für α_{BR} ist es möglich, einige Regeln tiefer zu verwurzeln als andere. Dies ist insbesondere dann von Vorteil, wenn eine Regel das Verhalten in einer allgemeinen Situation beschreibt und eine weitere eine Ausnahme-Regel zur allgemeinen Regel darstellt.

Beispiel 4.3.7.1:

In der Gridworld mit Hindernissen (Beispiel 4.3.2.1) gilt i. A. die Regel: 'Wenn *TargetNorthSouth* = *NorthAhead* $\vee$ *TargetNorthSouth* = *NorthAside*, dann *Action* = *GoNorth*.' Befindet sich jedoch ein Hindernis im Norden, soll die Regel 'Wenn *ObstacleNorth* = *True*, dann $\overline{Action = GoNorth}$.' angewendet werden:

Tritt nun ein Zustandssignal auf, welches das Muster *TargetNorthSouth* = *NorthAhead* $\wedge$ *ObstacleNorth* = *True* enthält, soll letztere Regel gegenüber der anderen bevorzugt werden. Dazu ist es notwendig, dass sie tiefer verwurzelt ist als die erste Regel.

Auf Grund der Lernfähigkeit des Systems dürfen allgemein richtige Regeln als Hintergrundwissen verwendetet werden, die in einigen Situation falsch sind, ohne dass auch Regeln für diese Ausnahme-Situationen hinzugefügt werden müssen. Der Agent wird die allgemeinen Regeln berücksichtigen und selbstständig feststellen, in welchen Situationen andere Regeln gelten.

4.3.8 Implementierung

Das Sphinx-Learning wurde in der Entwicklungsumgebung 'Microsoft Visual Basic 2005' implementiert und getestet. Der Quellcode liegt dieser Arbeit bei. Die so genannte 'Express Edition' der Entwicklungsumgebung ist auf der Homepage von Microsoft kostenlos verfügbar (Stand: Mai 2007).

5. Analyse

5.1 Allgemeines

Im folgenden Kapitel werden an Hand von drei verschiedenen Testumgebungen die Lernleistungen des klassischen RL auf der einen und des Sphinx-Learnings auf der anderen Seite miteinander verglichen. Außerdem wird jeweils der Lernvorgang beim Sphinx-Learning detailiert geschildert. Für die $Q(\lambda)$-Agenten und die RL-Komponente der Sphinx-Agenten wurden in allen Beispielen folgende Parameter verwendet (vgl. Algorithmus 2.6.3.1):

- $\alpha_{RL} := \max\left\{\frac{1}{v_{(s_t,a_t)}}, \alpha_{min}\right\}$ mit $\alpha_{min} := 0.25$
- $\gamma := 1$
- $\lambda := 0$ oder $\lambda := 0.5$ für $Q(\lambda)$-Agenten; $\lambda := 0.5$ für Sphinx-Agenten
- $\varepsilon := 0.25$

5.2 Gridworld mit Hindernissen und Farben

5.2.1 Vorstellung der Umwelt

Diese Testumgebung entspricht der in Beispiel 4.3.2.1 geschilderten Lernaufgabe mit dem Unterschied, dass der Agent noch die Farbe des Feldes, auf dem er steht, wahrnimmt. Die Schwierigkeit besteht darin, dass die Farbe beim Betreten eines Feldes zufällig gewählt wird, also beim zweiten Besuch desselben Feldes sehr wahrscheinlich eine andere ist als beim ersten Besuch, und diese Information für das Erreichen des Ziels irrelevant ist.

Die Domains sind:

- *TargetNorthSouth* = {*Neither-Nor*, *NorthAhead*, *NorthAside*, *SouthAhead*, *SouthAside*}
- *TargetEastWest* = {*Neither-Nor*, *EastAhead*, *EastAside*, *WestAhead*, *WestAside*}
- *TargetDistance* = {*Far*, *Middle*, *Close*}
- *ObstacleNorth* = {*False*, *True*}
- *ObstacleEast* = {*False*, *True*}
- *ObstacleSouth* = {*False*, *True*}
- *ObstacleWest* = {*False*, *True*}

- *Color* = {*Black*, *White*, *Red*, *Green*, *Blue*, *Yellow*}
- *Action* = {*GoNorth*, *GoEast*, *GoSouth*, *GoWest*}

Sei Δx die Differenz zwischen der *x*-Koordinate des Zielfeldes x_T und der aktuellen *x*-Koordinate des Agenten x_A, also $\Delta x := x_T - x_A$, und sei Δy analog für die *y*-Koordinaten y_T bzw. y_A definiert durch $\Delta y := y_T - y_A$. Dann werden für s_{sym} die Variablen von der Umwelt wie folgt belegt:

- *TargetNorthSouth*:
  - falls $\Delta y = 0$, dann *TargetNorthSouth* = *Neither-Nor*
  - falls $\Delta y > 0$: falls $|\Delta x| < |\Delta y|$, dann *TargetNorthSouth* = *NorthAhead*, sonst *TargetNorthSouth* = *NorthAside*
  - falls $\Delta y < 0$: falls $|\Delta x| < |\Delta y|$, dann *TargetNorthSouth* = *SouthAhead*, sonst *TargetNorthSouth* = *SouthAside*
- *TargetEastWest*:
  - falls $\Delta x = 0$, dann *TargetEastWest* = *Neither-Nor*
  - falls $\Delta x > 0$: falls $|\Delta y| < |\Delta x|$, dann *TargetEastWest* = *EastAhead*, sonst *TargetEastWest* = *EastAside*
  - falls $\Delta x < 0$: falls $|\Delta y| < |\Delta x|$, dann *TargetEastWest* = *WestAhead*, sonst *TargetEastWest* = *WestAside*
- *TargetDistance*:
  - falls $\underbrace{|\Delta x| + |\Delta y|}_{\text{1-Norm}} \leq 1$, dann *TargetDistance* = *Close*
  - falls $2 \leq |\Delta x| + |\Delta y| \leq 4$, dann *TargetDistance* = *Middle*
  - falls $|\Delta x| + |\Delta y| > 4$, dann *TargetDistance* = *Far*
- *ObstacleNorth* = *True*, falls ein Hindernis im nördlich angrenzenden Feld ist, sonst *ObstacleNorth* = *False*
- *ObstacleEast* = *True*, falls ein Hindernis im östlich angrenzenden Feld ist, sonst *ObstacleEast* = *False*
- *ObstacleSouth* = *True*, falls ein Hindernis im südlich angrenzenden Feld ist, sonst *ObstacleSouth* = *False*
- *ObstacleWest* = *True*, falls ein Hindernis im westlich angrenzenden Feld ist, sonst *ObstacleWest* = *False*

- *Color* erhält einen zufälligen Wert.

Die Umwelt verwaltet intern die genaue Position (x_A, y_A) des Agenten und berechnet daraus das dem Agenten zur Verfügung gestellte Zustandssignal $s = (s_{ID}, s_{sym})$. Die Zustandsübergangsfunktion greift auf die Koordinaten (x_A, y_A) zu und ändert sie entsprechend der Aktion des Agenten. Hat dieser beispielsweise die Aktion *GoEast* gewählt, so wird *x* um 1 erhöht, sofern kein Hindernis im Osten ist. Befindet sich ein Hindernis in der vom Agenten gewählten Bewegungsrichtung, so lässt die Zustandsübergangsfunktion die Position unverändert.

Die Belohnung nach einem Schritt wird vergeben gemäß dieser Auflistung:

- falls die gewählte Aktion den Agenten dem Ziel näher gebracht hat: $r := 0.5$
- falls die gewählte Aktion den Agenten vom Ziel entfernt hat: $r := -1$
- falls der Agent durch die gewählte Aktion ein Hindernis angestoßen hat: $r := -10$
- falls der Agent durch die gewählte Aktion das Zielfeld erreicht hat: $r := 100$

Die maximale Dauer einer Episode sind 25 Schritte. Ist bis dahin das Zielfeld nicht erreicht worden, wird die Episode abgebrochen und als Fehlschlag gewertet.

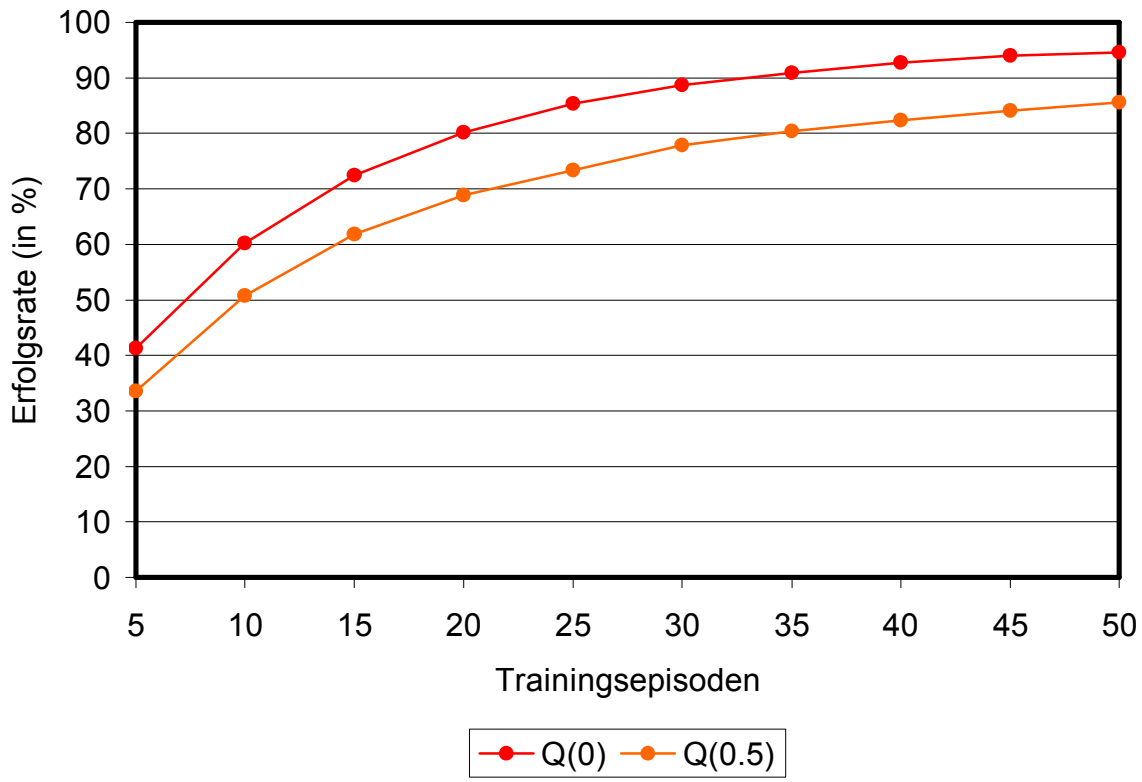
5.2.2 Ergebnisse

Die durchschnittlichen Erfolgsraten (ermittelt aus den Erfolgsraten von 1000 unabhängigen $Q(\lambda)$-Agenten) sind folgender Tabelle und nachfolgendem Diagramm zu entnehmen.

Tabelle 5.2.2.1:

Trainingsepisoden	Erfolgsrate (in %) ($Q(0)$)	Erfolgsrate (in %) ($Q(0.5)$)
5	41.3	33.6
10	60.2	50.8
15	72.4	61.8
20	80.2	68.9
25	85.3	73.4
30	88.7	77.8
35	90.9	80.4
40	92.7	82.4
45	94	84.1
50	94.6	85.6

Diagramm 5.2.2.1:



Bei den getesteten Sphinx-Agenten wurden gesetzt:

- *useEveryVisitForGeneralization* := *False*
- *badMaxPatternLength* := 1
- *bestMaxPatternLength* := 3
- $t_{-}(p) := 2^{badMaxPatternLength-|p|+1}$
- $t_{not-}(p) := 0$
- $t_{age-} := 1$
- $t_{+}(p) := 3$
- $t_{not+}(p) := \infty$
- $t_{max+}(p) := 2^{bestMaxPatternLength-|p|+2}$
- $t_{age+} := 2$

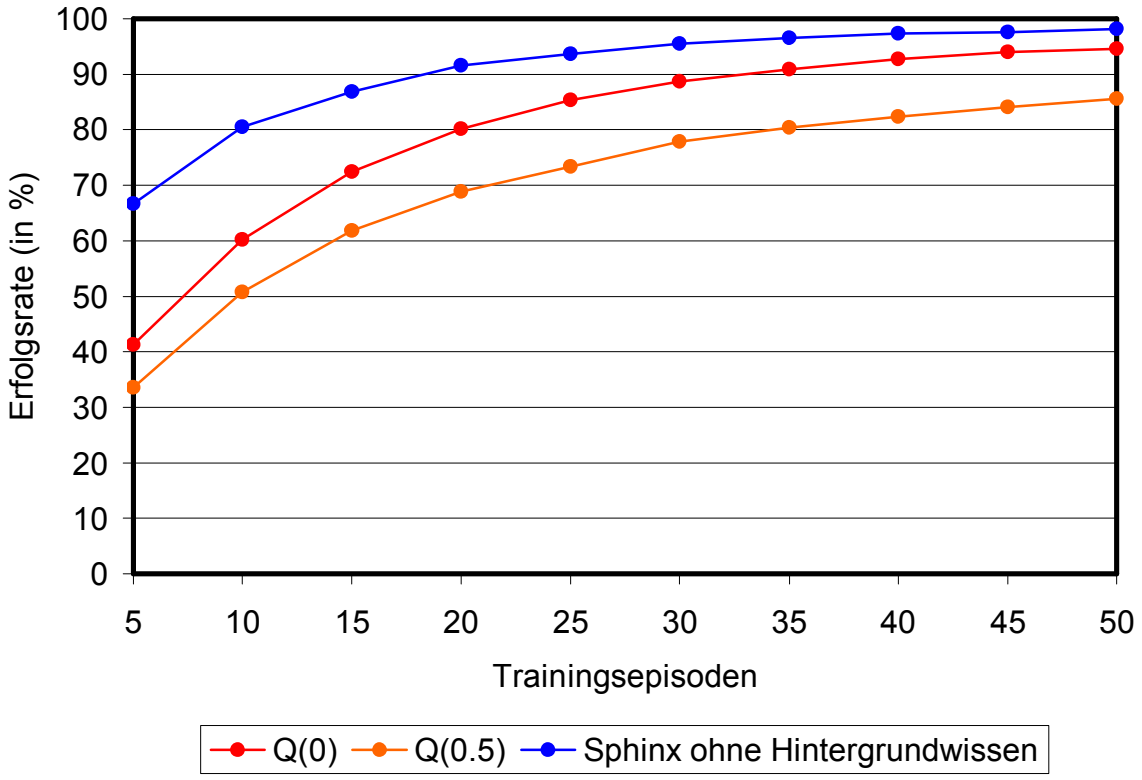
Diese Einstellungen führten zu folgenden, durchschnittlichen Erfolgsraten (berechnet aus 1000 unabhängigen Sphinx-Agenten).

Tabelle 5.2.2.2:

Trainingsepisoden	Erfolgsrate (in %) (Sphinx ohne Hintergrundwissen)
5	66.7
10	80.5
15	86.9
20	91.6
25	93.6
30	95.5
35	96.5
40	97.3
45	97.6
50	98.1

Verglichen mit den durchschnittlichen Erfolgsraten der $Q(\lambda)$-Agenten ergibt sich die durch das folgende Diagramm gegebene Übersicht.

Diagramm 5.2.2.2:



Im nächsten Analyseschritt wurde Hintergrundwissen hinzugefügt. In einem ersten Experiment wurde dem Agenten nur *etwas* Hintergrundwissen durch die Regel ‘Gehe in die Richtung, in der das Ziel liegt.’ mittels dieser Konditionale mitgegeben:

- (*Action* = *GoNorth* | *TargetNorthSouth* = *NorthAhead* $\vee$ *TargetNorthSouth* = *NorthAside*)
- (*Action* = *GoEast* | *TargetEastWest* = *EastAhead* $\vee$ *TargetEastWest* = *EastAside*)
- (*Action* = *GoSouth* | *TargetNorthSouth* = *SouthAhead* $\vee$ *TargetNorthSouth* = *SouthAside*)
- (*Action* = *GoWest* | *TargetEastWest* = *WestAhead* $\vee$ *TargetEastWest* = *WestAside*)

Um das Hintergrundwissen vor falschen, zu früh vermuteten Regeln zu schützen, wurden die Schwellwertfunktionen geändert zu:

- $t_{-}(p) := \left(\frac{1}{4} + \left(1 - \frac{1}{4}\right)\sqrt{\Pr(p)}\right) 2^{badMaxPatternLength - |p| + 4}$
- $t_{\max+}(p) := \left(\frac{1}{8} + \left(1 - \frac{1}{8}\right)\sqrt{\Pr(p)}\right) 2^{bestMaxPatternLength - |p| + 5}$
- $t_{age+} := 8$

In einem zweiten Test wurde *mehr* Hintergrundwissen durch die *zusätzliche* Regel ‘Wenn in einer Richtung ein Hindernis liegt, dann gehe nicht in diese Richtung.’ in Form dieser Konditionale hinzugefügt:

- $\left(\overline{Action = GoNorth} \middle| ObstacleNorth = True\right)$
- $\left(\overline{Action = GoEast} \middle| ObstacleEast = True\right)$
- $\left(\overline{Action = GoSouth} \middle| ObstacleSouth = True\right)$
- $\left(\overline{Action = GoWest} \middle| ObstacleWest = True\right)$

Diese Konditionale wurden mittels $\alpha_{BR} = 2$ epistemisch tiefer verwurzelt als die ersten, für welche $\alpha_{BR} = 1$ benutzt wurde, damit in Zuständen, in denen sowohl das Zielfeld als auch ein Hindernis in derselben Richtung liegen, die zweite Regel der ersten bevorzugt wird.

Die Schwellwerte waren gegeben durch:

- $t_{-}(p) := \left(\frac{1}{4} + \left(1 - \frac{1}{4}\right)\sqrt{\Pr(p)}\right) 2^{badMaxPatternLength - |p| + 4}$
- $t_{\max+}(p) := \left(\frac{1}{2} + \left(1 - \frac{1}{2}\right)\sqrt{\Pr(p)}\right) 2^{bestMaxPatternLength - |p| + 5}$

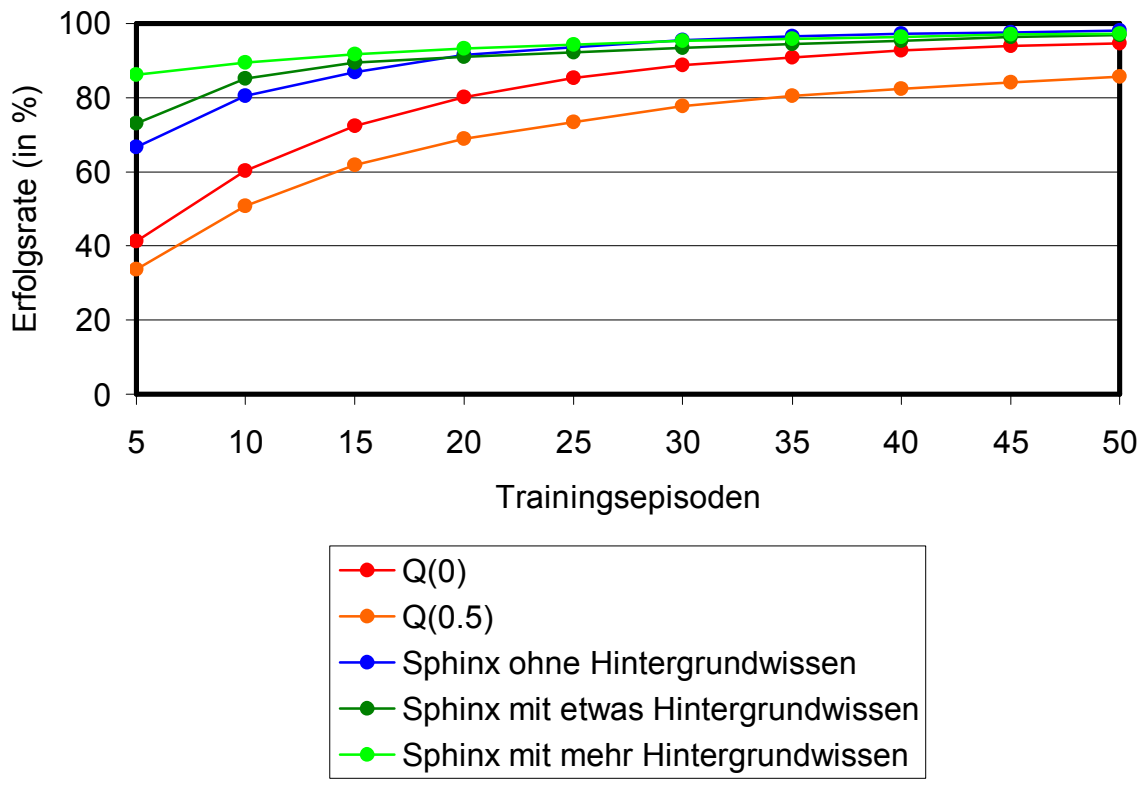
- $t_{age+} := 12$

Es ergaben sich die im Folgenden aufgelisteten Erfolgsraten.

Tabelle 5.2.2.3:

Trainingsepisoden	Erfolgsrate (in %) (etwas Hintergrundwissen)	Erfolgsrate (in %) (mehr Hintergrundwissen)
5	73.1	86.2
10	85.1	89.4
15	89.4	91.7
20	91.1	93.2
25	92.2	94.3
30	93.4	95.4
35	94.5	95.8
40	95.4	96.4
45	96.3	97
50	96.9	97.3

Diagramm 5.2.2.3:



Die in diesem Beispiel am Ende minimal schlechteren Erfolgsraten bei der Verwendung von Hintergrundwissen erklären sich durch die höheren Schwellwerte. Durch diese werden neue Regeln erst später hinzugefügt, wenn durch ausgiebige

Interaktion mit der Umwelt nahezu sicher davon ausgegangen werden kann, dass sie korrekt sind und so das Hintergrundwissen vor falschen Vermutungen geschützt ist. Ohne Verwendung von Hintergrundwissen ist es *im Durchschnitt* besser, niedrige Schwellwerte zu benutzen. Allerdings führt dies im Einzelfall dazu, dass bereits richtig Gelerntes durch falsche Regeln zerstört wird und der Agent die zuvor gelernten Zusammenhänge erneut erkennen muss. Zu niedriege Schwellwerte bei der Verwendung von Hintergundwissen machen den Vorteil zu Beginn des Lernens zumindest teilweise zunichte. Ein Vorteil der höheren Schwellwerte ist die stabilere Leistung der Agenten.

Zur Analyse wurde weiterhin untersucht, wie hoch für jedes Feld die Wahrscheinlichkeit ist, dass nach einer bestimmten Anzahl von Episoden das Zielfeld erreicht wird. Für klassische $Q(\lambda)$-Agenten bzw. Sphinx-Agenten nach nur fünf Trainingsepisoden wurden die folgenden Werte ermittelt.

Tabelle 5.2.2.4:

x	y	Q(0)	Q(0.5)	Sphinx (ohne Hintergrund-wissen)	Sphinx (etwas Hintergrund-wissen)	Sphinx (mehr Hintergrund-wissen)
0	0	0.086	0.044	0.516	0.532	0.677
0	1	0.108	0.056	0.525	0.546	0.707
0	2	0.142	0.074	0.558	0.587	0.751
0	3	0.154	0.096	0.546	0.584	0.774
0	4	0.186	0.122	0.566	0.587	0.750
0	5	0.178	0.136	0.521	0.618	0.794
0	6	0.236	0.176	0.559	0.642	0.814
0	7	0.219	0.162	0.578	0.658	0.828
0	8	0.233	0.194	0.741	0.678	0.809
0	9	0.244	0.210	0.752	0.682	0.824
1	0	0.113	0.053	0.546	0.595	0.707
1	1	0.170	0.101	0.534	0.571	0.726
1	2	0.206	0.123	0.572	0.588	0.776
1	3	0.216	0.154	0.541	0.613	0.763
1	4	0.248	0.146	0.569	0.639	0.801
1	5	0.254	0.182	0.511	0.625	0.814
1	6	0.299	0.196	0.548	0.654	0.837

1	7	0.270	0.230	0.572	0.670	0.829
1	8	0.319	0.234	0.716	0.686	0.831
1	9	0.341	0.271	0.755	0.723	0.857
2	0	0.130	0.091	0.507	0.540	0.708
2	1	0.185	0.119	0.524	0.585	0.762
2	2	0.204	0.124	0.540	0.620	0.782
2	3	0.239	0.154	0.553	0.615	0.812
2	4	0.282	0.173	0.552	0.640	0.792
2	5	0.297	0.215	0.516	0.651	0.830
2	6	0.308	0.204	0.545	0.659	0.831
2	7	0.343	0.246	0.596	0.668	0.875
2	8	0.367	0.296	0.720	0.728	0.849
2	9	0.420	0.326	0.802	0.766	0.882
3	0	0.171	0.093	0.519	0.596	0.774
3	1	0.223	0.139	0.547	0.573	0.774
3	2	0.223	0.166	0.546	0.616	0.801
3	3	0.256	0.190	0.550	0.605	0.803
3	4	0.279	0.187	0.565	0.656	0.863
3	5	0.278	0.210	0.500	0.654	0.839
3	6	0.336	0.242	0.531	0.658	0.873
3	7	0.398	0.307	0.567	0.693	0.844
3	8	0.442	0.369	0.735	0.726	0.883
3	9	0.540	0.407	0.814	0.827	0.914
4	0	0.191	0.126	0.533	0.571	0.767
4	1	0.233	0.157	0.567	0.605	0.817
4	2	0.261	0.169	0.526	0.617	0.841
4	3	0.276	0.177	0.533	0.643	0.848
4	4	0.283	0.195	0.558	0.657	0.827
4	5	0.317	0.208	0.490	0.682	0.862
4	6	0.347	0.270	0.498	0.675	0.853
4	7	0.401	0.335	0.561	0.702	0.883
4	8	0.530	0.506	0.779	0.854	0.939
4	9	0.627	0.519	0.835	0.930	0.955
5	0	0.203	0.145	0.522	0.630	0.805
5	1	0.277	0.174	0.497	0.625	0.807
5	2	0.260	0.174	0.567	0.646	0.823
5	3	0.292	0.238	0.494	0.635	0.817
5	4	0.322	0.196	0.499	0.635	0.853
5	5	n. a.	n. a.	n. a.	n. a.	n. a.

5	6	n. a.	n. a.	n. a.	n. a.	n. a.
5	7	n. a.	n. a.	n. a.	n. a.	n. a.
5	8	0.763	0.682	0.840	1.000	1.000
5	9	0.778	0.655	0.881	1.000	1.000
6	0	0.235	0.156	0.550	0.594	0.814
6	1	0.282	0.205	0.547	0.648	0.828
6	2	0.314	0.243	0.584	0.643	0.838
6	3	0.330	0.247	0.555	0.642	0.859
6	4	0.378	0.260	0.572	0.658	0.841
6	5	n. a.	n. a.	n. a.	n. a.	n. a.
6	6	0.942	0.910	0.803	1.000	1.000
6	7	0.954	0.915	0.879	1.000	1.000
6	8	0.857	0.825	0.905	1.000	1.000
6	9	0.825	0.773	0.891	1.000	1.000
7	0	0.229	0.166	0.581	0.632	0.820
7	1	0.263	0.186	0.571	0.667	0.815
7	2	0.329	0.258	0.585	0.658	0.849
7	3	0.391	0.294	0.561	0.681	0.864
7	4	0.434	0.330	0.591	0.662	0.876
7	5	n. a.	n. a.	n. a.	n. a.	n. a.
7	6	0.953	0.923	0.878	1.000	1.000
7	7	n. a.	n. a.	n. a.	n. a.	n. a.
7	8	0.913	0.876	0.912	1.000	1.000
7	9	0.896	0.827	0.903	1.000	1.000
8	0	0.262	0.174	0.690	0.649	0.812
8	1	0.302	0.240	0.741	0.653	0.840
8	2	0.355	0.265	0.728	0.685	0.867
8	3	0.468	0.360	0.744	0.765	0.886
8	4	0.555	0.484	0.773	0.857	0.923
8	5	0.816	0.689	0.867	0.999	1.000
8	6	0.892	0.836	0.898	1.000	1.000
8	7	0.934	0.875	0.865	0.998	0.998
8	8	0.888	0.827	0.906	0.999	0.997
8	9	0.862	0.822	0.882	0.998	0.995
9	0	0.269	0.175	0.756	0.664	0.846
9	1	0.382	0.255	0.778	0.686	0.849
9	2	0.416	0.301	0.785	0.746	0.885
9	3	0.527	0.377	0.782	0.849	0.923
9	4	0.644	0.532	0.820	0.921	0.947

9	5	0.777	0.653	0.872	1.000	0.998
9	6	0.859	0.765	0.896	1.000	0.999
9	7	0.881	0.801	0.865	0.999	0.997
9	8	0.856	0.804	0.855	1.000	0.996
9	9	0.852	0.741	0.877	1.000	0.996

Um diese Werte grafisch darzustellen, wurden die Felder in den nachfolgenden Abbildungen blau gefärbt. Je dunkler die Färbung eines Feldes ist, desto höher ist Wahrscheinlichkeit, dass das Ziel von dieser Zelle aus erreicht wird. Die erste Abbildung zeigt die Werte für $Q(0)$-Agenten.

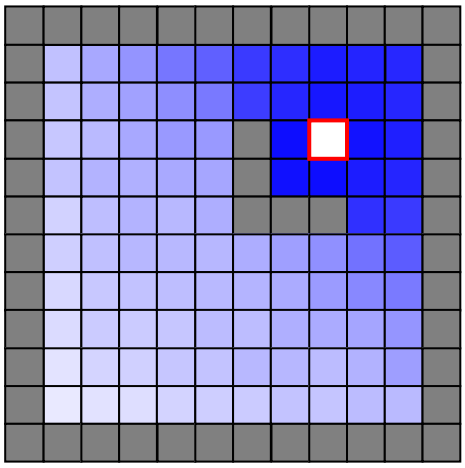


Abbildung 5.2.2.1

Die grafische Veranschaulichung der Werte für $Q(0.5)$-Agenten wird ersichtlich in Abbildung 5.2.2.2.

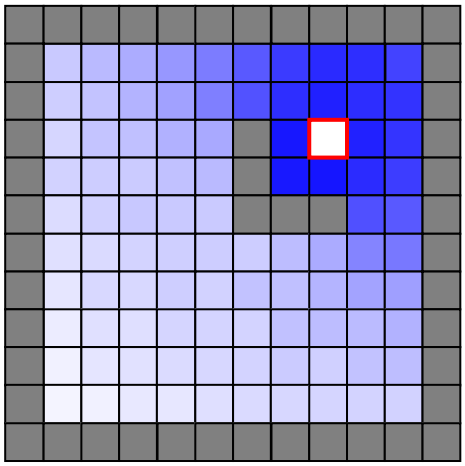


Abbildung 5.2.2.2

Für Sphinx-Agenten ohne Hintergrundwissen ergibt sich folgendes Bild.

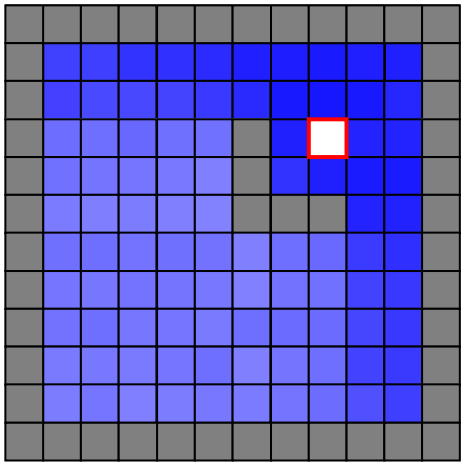


Abbildung 5.2.2.3

Neben der Erfolgsrate ist die Länge des gewählten Weges vom jeweiligen Startfeld zum Zielfeld ein weiteres Güte-Kriterium. Wünschenswert sind hohe Erfolgsraten *und* kurze Wege. Abbildung 5.2.2.4 zeigt die Mindestanzahlen von Schritten, die zum Erreichen des Ziels notwendig sind, für jedes Startfeld.

	9	8	7	6	5	4	3	2	3	4
	8	7	6	5	4	3	2	1	2	3
	9	8	7	6	5		1		1	2
	10	9	8	7	6		2	1	2	3
	11	10	9	8	7				3	4
	12	11	10	9	8	7	6	5	4	5
	13	12	11	10	9	8	7	6	5	6
	14	13	12	11	10	9	8	7	6	7
	15	14	13	12	11	10	9	8	7	8
	16	15	14	13	12	11	10	9	8	9

Abbildung 5.2.2.4

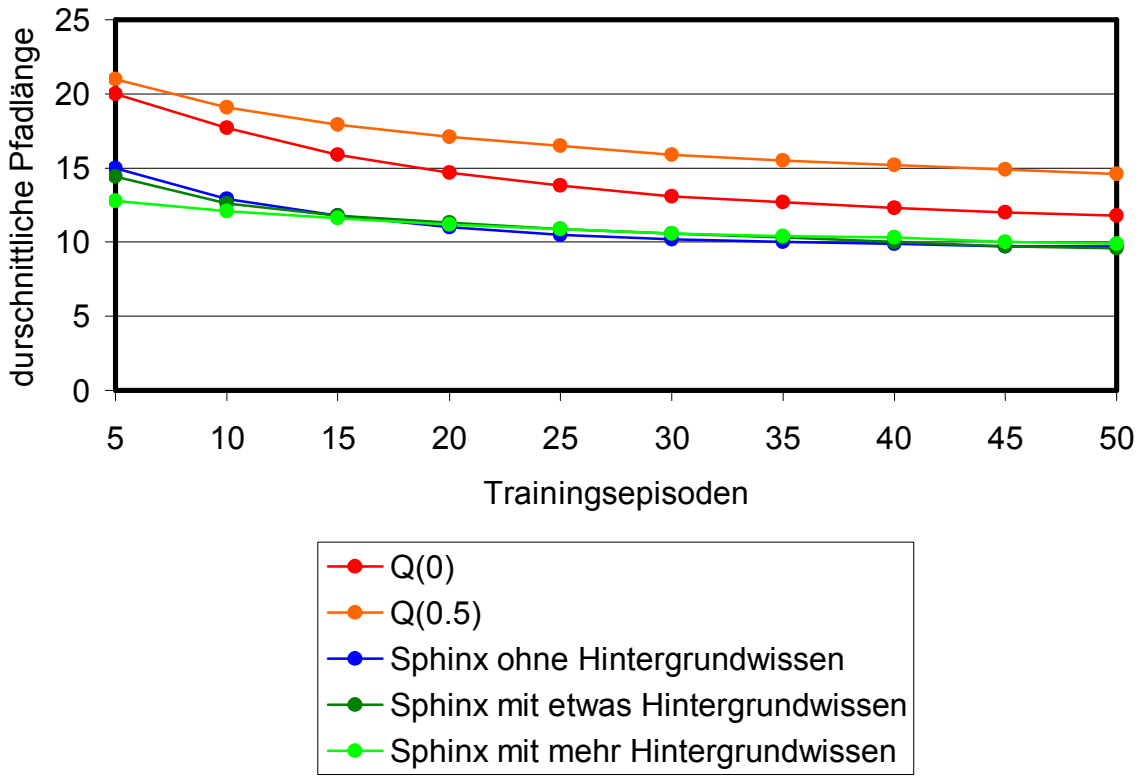
Rechnerisch ergibt sich ein Optimum von durchschnittlich $716:94 \approx 7.617$ Schritten.

Auch in diesem Bereich weist das Sphinx-Learning einen Vorsprung gegenüber dem klassischen RL auf. In der folgenden Tabelle ist aufgelistet, wie viele Schritte die Agenten durchschnittlich nach *n* Trainingsepisoden benötigen, um das Zielfeld zu erreichen. Bei den Werten ist zu berücksichtigen, dass Episoden, bei denen der Agent nach 25 Schritten das Ziel noch nicht erreicht hat, abgebrochen werden.

Tabelle 5.2.2.4:

Trainings-episoden	Q(0)	Q(0.5)	Sphinx (ohne Hintergrund-wissen)	Sphinx (etwas Hintergrund-wissen)	Sphinx (mehr Hintergrund-wissen)
5	20	21	15	14.4	12.8
10	17.7	19.1	12.9	12.6	12.1
15	15.9	17.9	11.8	11.8	11.6
20	14.7	17.1	11	11.3	11.2
25	13.8	16.5	10.5	10.9	10.9
30	13.1	15.9	10.2	10.6	10.6
35	12.7	15.5	10	10.3	10.4
40	12.3	15.2	9.9	10	10.3
45	12	14.9	9.7	9.7	10
50	11.8	14.6	9.7	9.6	9.9

Diagramm 5.2.2.4:



Schließlich wurde untersucht, wie viele und welche Regeln erstellt wurden, d. h. mit wie vielen und welchen Konditionalen κ revidiert wurde. In den nächsten drei Tabellen wird aufgeführt, wie viele Revisionen welchen Typs Sphinx-Agenten ohne, mit etwas bzw. mit mehr Hintergrundwissen durchschnittlich nach *n* Episoden durchgeführt haben. (Bemerkungen: Regeln können auch mehrfach hinzugefügt

werden, ggfs. mit einer anderen epistemischen Verwurzelung.)

Tabelle 5.2.2.5 (ohne Hintergrundwissen):

Trainings-episoden	schlechte Aktion, spezieller Zustand	schlechte Aktion, verallgemeinert	beste Aktionen, spezieller Zustand	beste Aktionen, verallgemeinert
5	16	15	61	2095
10	22	20	117	3419
15	27	24	172	4466
20	31	26	228	5342
25	34	29	283	6065
30	37	30	338	6725
35	39	32	394	7296
40	41	34	451	7845
45	44	35	509	8369
50	46	36	567	8849

Tabelle 5.2.2.6 (mit etwas Hintergrundwissen):

Trainings-episoden	schlechte Aktion, spezieller Zustand	schlechte Aktion, verallgemeinert	beste Aktionen, spezieller Zustand	beste Aktionen, verallgemeinert
5	18	1.1	0.4	0.2
10	25.2	2.8	2.5	9.5
15	29.3	4	6.1	44.8
20	32.5	5	11.7	111.7
25	35.3	6	18.6	204.8
30	37.8	6.9	26.4	318.9
35	40	7.8	35.4	444.9
40	42.2	8.7	45.7	581.2
45	44.3	9.7	56.8	720.7
50	46.5	10.7	69.1	865.8

Tabelle 5.2.2.7 (mit mehr Hintergrundwissen):

Trainings-episoden	schlechte Aktion, spezieller Zustand	schlechte Aktion, verallgemeinert	beste Aktionen, spezieller Zustand	beste Aktionen, verallgemeinert
5	3.4	0	0.1	0
10	6.3	0.1	0.3	1.7
15	9.2	0.2	0.8	10.6
20	11.8	0.5	1.8	28.1
25	14.2	1	3.5	51.5
30	16.5	1.5	5.8	78.8
35	18.8	2.2	9	107.5
40	21	2.9	12.8	138.3
45	23.3	3.6	17.8	170.6
50	25.4	4.4	23	202.7

Auf Grund dieser Fülle von Revisionen ist es unmöglich, sie alle aufzulisten. Daher werden an dieser Stelle nur einige der von einem durchschnittlichen Sphinx-Agenten (ohne Hintergrundwissen) gefundenen, generalisierten Regeln beispielhaft aufgeführt.

Regeln mit Information über eine schlechte Aktion:

- Wenn *ObstacleWest* = *True*, dann $\overline{Action = GoWest}$ (Verwurzelung: 2)
- Wenn *ObstacleEast* = *True*, dann $\overline{Action = GoEast}$ (Verwurzelung: 4)
- Wenn *ObstacleNorth* = *True*, dann $\overline{Action = GoNorth}$ (Verwurzelung: 4)
- Wenn *ObstacleSouth* = *True*, dann $\overline{Action = GoSouth}$ (Verwurzelung: 2)
- Wenn *TargetNorthSouth* = *SouthAhead*, dann $\overline{Action = GoNorth}$ (Verwurzelung: 1)

(Bemerkung zur letzten aufgeführten Regel: Diese Regel wurde in einem frühen Lernstadium (zwischen der ersten und fünften Trainingsepisode) erstellt. Sie ist entstanden, weil der Agent am oberen Rand der Gridworld mehrfach an Positionen, an denen *TargetNorthSouth* = *SouthAhead* gilt, die Aktion *GoNorth* ausprobiert hat und dabei immer eine schlechte Belohnung erhalten hat, da er vor die Begrenzung der Gridworld gestoßen ist. Zuvor hatte er noch nicht die Erfahrung gemacht, dass es auch Zustände gibt, in denen *TargetNorthSouth* = *SouthAhead* gilt, aber bei denen die Aktion *GoNorth* nicht schlecht belohnt wird.)

Nachfolgend werden nur exemplarisch einzelne gefundene Regeln mit Information über beste Aktionen aufgeführt. Eine ausführlichere Liste ausgewählter, gefundener

Zusammenhänge ist im Anfang A.1.1 zu finden.

Regeln mit Information über beste Aktionen:

- Wenn *TargetNorthSouth* = *NorthAhead* $\wedge$ *ObstacleNorth* = *False*, dann *Action* = *GoNorth* $\vee$ *Action* = *GoEast* $\vee$ *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *NorthAhead* $\wedge$ *TargetEastWest* = *WestAside*, dann *Action* = *GoNorth* $\vee$ *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *Neither-Nor* $\wedge$ *TargetDistance* = *Far* $\wedge$ *ObstacleEast* = *False*, dann *Action* = *GoEast* (Verwurzelung: 1)
- Wenn *TargetEastWest* = *NorthAhead* $\wedge$ *TargetDistance* = *Middle* $\wedge$ *ObstacleNorth* = *True*, dann *Action* = *GoEast* (Verwurzelung: 1)

5.2.3 Interpretation der Ergebnisse

In der Gridworld mit Hindernissen und Farben ist das Sphinx-Learning dem reinen RL selbst ohne Hintergrundwissen gerade zu Beginn deutlich überlegen. Die Erfolgsrate nach 5 Trainingsepisoden ist ca. 1.5-mal (gegenüber $Q(0)$-Agenten) bzw. 2-mal (gegenüber $Q(0.5)$-Agenten) so hoch, während die Pfadlängen kürzer sind. Dies ist dadurch zu erklären, dass Sphinx-Agenten durch Bildung von Regeln in noch nicht besuchten Zuständen auf Wissen zurückgreifen können, welches sie in ähnlichen Situationen gelernt haben. Für $Q(\lambda)$-Agenten liegt beispielsweise immer ein vollständig anderer Zustand vor, wenn ein Feld, auf dem sie sich bereits befunden haben, beim zweiten Besuch eine andere Farbe hat. Auch müssen sie für jeden Zustand einzeln lernen, dass sie nicht in die Richtung gehen sollten, in der ein Hindernis liegt. Sphinx-Agenten brauchen diese Erfahrung nur an wenigen Stellen machen und übertragen sie dann auf andere, so dass die jeweils beste Aktion nur noch aus einer kleineren Menge von Aktionen bestimmt werden muss.

Die Auswirkungen der Regelbildungen sind optisch erkennbar in den Abbildungen 5.2.2.1, 5.2.2.2 und 5.2.2.3. Während die Erfolgswahrscheinlichkeit, beim RL mit der Entfernung zum Ziel abnimmt, sind bei Abbildung 5.2.2.3 rechts und oben jeweils zwei Spalten bzw. Zeilen zu erkennen (siehe auch nachfolgende Abbildung), deren Felder annähernd die gleiche Färbung besitzen und sich sichtbar von den Feldern der Spalte links bzw. der Zeile darunter absetzen.

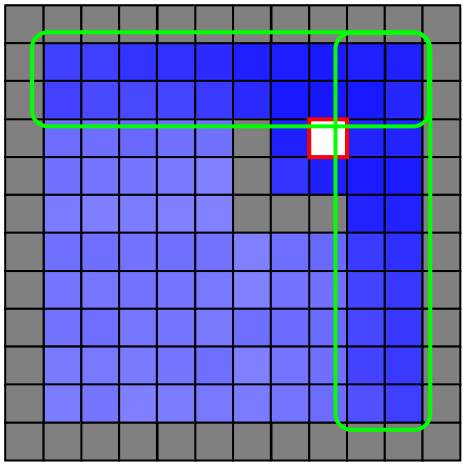


Abbildung 5.2.3.1

Eine Begründung hierfür ist, dass die in Abbildung 5.2.3.2 markierten Bereiche zu je einer Klasse zusammengefasst werden können, denn sie haben die Gemeinsamkeit *TargetNorthSouth* = *NorthAhead* $\wedge$ *TargetEastWest* = *WestAside* bzw. *TargetNorthSouth* = *SouthAside* $\wedge$ *TargetEastWest* = *EastAhead*.

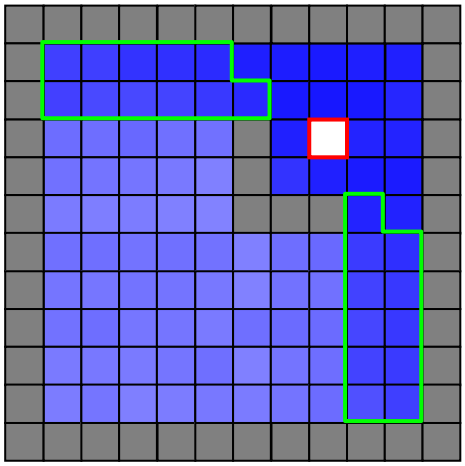


Abbildung 5.2.3.2

Ist auf Grund von Erfahrungen, die in einigen Feldern dieser Klassen gemacht wurden, der Zusammenhang mit *GoNorth* bzw. *GoEast* als einer besten Aktion erkannt worden, spielt die Entfernung zum Ziel, die momentane Farbe des Feldes oder das Vorhandensein oder Fehlen eines Hindernisses (Gridworld-Begrenzung) unmittelbar im Osten bzw. Norden keine Rolle mehr und auch in noch nicht besuchten Zuständen dieser Klassen wissen die Sphinx-Agenten, welche Aktion sie wählen sollten. Der Zusammenhang mit *GoNorth* bzw. *GoEast* ist leicht zu finden, weil der Agent auf dem direkten Weg zum Ziel keinem Hindernis begegnet und er daher keine Strategie entwickeln muss, ein solches zu umgehen.

Es werden außerdem Zusammenhänge entdeckt, die für Menschen weniger

offensichtlich sind, wie die Regel

‘Wenn	*ObstacleSouth*	= *True*	$\wedge$ *ObstacleWest*	= *False*,
dann	*Action*	= *GoNorth*	$\vee$ *Action*	= *GoEast*’

Man überzeugt sich leicht, dass sie in der gegebenen Testumgebung korrekt ist, da wenigstens eine der Aktionen aus {*GoNorth*, *GoEast*} eine beste Aktion in Zuständen, in denen die Prämisse zutrifft, ist.

Das Hinzufügen der Regel ‘Gehe in die Richtung, in der das Ziel liegt.’ zum Hintergrundwissen erhöht die Erfolgsrate besonders zu Anfang etwas. Allerdings weist man den Agenten durch diese Regel an, vor Hindernisse zu laufen, wenn diese auf dem Weg zum Ziel liegen. Dies bestätigen auch in die Anzahlen der Revisionen mit Information über eine schlechte Aktion in einem speziellen Zustand. Vergleicht man die entsprechenden Spalten von Tabelle 5.2.2.5 und Tabelle 5.2.2.6 miteinander, stellt man fest, dass die Werte in der letzteren Tabelle zu Anfang höher sind und sich nach und nach angleichen. Der Vorteil, der dem Agent durch diese Regel entsteht, wird beeinträchtigt, denn er muss den Glauben an diese Regel in Ausnahme-Situationen aufgeben und dort geltende Regeln finden. Dadurch erklärt sich der nur geringe Unterschied zum Agenten ohne Hintergrundwissen.

Wird dem Agenten für diese Ausnahme-Situationen mittels der Regel ‘Wenn in einer Richtung ein Hindernis liegt, dann gehe nicht in diese Richtung.’ zusätzlich mitgeteilt, wie er sich in ihnen *nicht* verhalten soll, ist das Hintergrundwissen in jeder Situation richtig, auch wenn der Agent in den Ausnahme-Situationen noch herausfinden muss, *wie* er sich am günstigsten verhält. In diesem Fall erhöht sich die Erfolgsrate zu Beginn des Lernen nochmals deutlich. Dass der Agent bei diesem Hintergrundwissen weniger häufig an ein Hindernis stößt, ist an den niedrigeren Werten der Spalte ‘schlechte Aktion, spezieller Zustand’ von Tabelle 5.2.2.7 erkennbar. Da der Agent durch sein Hintergrundwissen daran gehindert wird, an ein Hindernis zu stoßen, sind diese Werte auf explorative Aktionen des Agenten zurückzuführen.

Es ergibt sich der Schluss, dass Hintergrundwissen inbesondere dann einen Vorteil bietet, wenn es in jeder Situation korrekt ist. Hintergrundwissen, welches in Ausnahme-Situationen falsch ist, erhöht die Erfolgsrate auch, allerdings merklich weniger.

Ein Vergleich der Spalten über die Anzahlen der Revisionen mit Information über

beste Aktionen in den Tabellen 5.2.2.5, 5.2.2.6 und 5.2.2.7 zeigt, dass die bei den Experimenten mit Hintergrundwissen höher gewählten Schwellwerte erwartungsgemäß in den frühen Lernphasen die Anzahlen der Revision klein halten. Im späteren Verlauf steigen sie deutlich an, sobald nahezu sichere Regeln erstellt werden können.

5.3 Erkennung geometrischer Objekte

5.3.1 Vorstellung der Umwelt

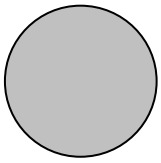
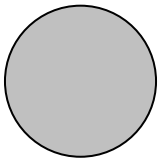
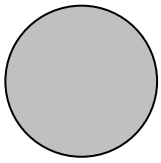
Die Aufgabe des Agenten in der nun betrachteten Umwelt ist es, geometrische Objekte zu erkennen. Dabei stehen folgende Objekte zur Verfügung:

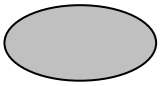
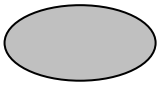
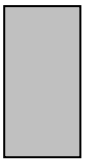
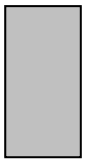
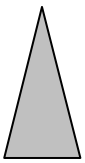
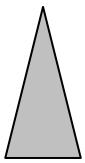
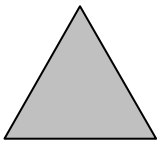
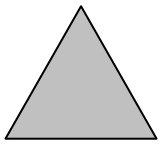
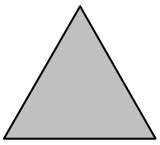
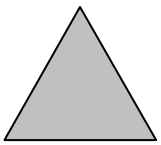
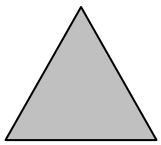
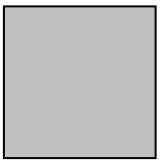
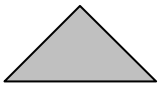
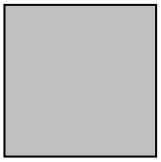
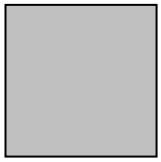
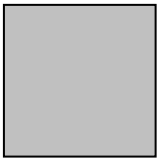
- Kugel
- Ellipsoid
- Zylinder
- Kegel
- Tetraeder
- Pyramide
- Prisma
- Würfel
- Quader

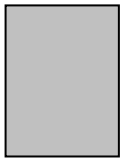
Der Agent kann sich das Objekt durch Interaktion mit der Umwelt von vorne, von der Seite oder von oben ansehen oder sich dazu entscheiden, dass Objekt zu erkennen.

Die Front-, Seiten- und Oberansichten der einzelnen Objekte sind wie folgt festgelegt.

Tabelle 5.3.1.1:

Objekt	Frontansicht	Seitenansicht	Oberansicht
Kugel			

Ellipsoid			
Zylinder			
Kegel			
Tetraeder			
Pyramide			
Prisma			
Würfel			

Quader			

Es gibt demnach fünf verschiedene Grundformen (unter Vernachlässigung der speziellen Proportionen):

- Kreis
- Ellipse (und nicht Kreis)
- Dreieck
- Quadrat
- Rechteck (und nicht Quadrat)

Diese Grundformen bilden die Grundlage der Domains für das von der Umwelt kommende Zustandssignal. Insgesamt werden folgende Domains festgelegt:

- *FrontView* = {*Unknown*, *Circle*, *Ellipse*, *Triangle*, *Square*, *Rectangle*}
- *SideView* = {*Unknown*, *Circle*, *Ellipse*, *Triangle*, *Square*, *Rectangle*}
- *TopView* = {*Unknown*, *Circle*, *Ellipse*, *Triangle*, *Square*, *Rectangle*}
- *Action* = {*LookAtFront*, *LookAtSide*, *LookAtTop, RecognizeUnknown*, *RecognizeSphere*, *RecognizeEllipsoid*, *RecognizeCylinder*, *RecognizeCone*, *RecognizeTetrahedron*, *RecognizePyramid*, *RecognizePrism*, *RecognizeCube*, *RecognizeCuboid*}

Zu Anfang wählt die Umwelt eines der neun geometrischen Objekte aus und erzeugt ein Zustandssignal mit der symbolischen Darstellung *FrontView* = *Unknown* $\wedge$ *SideView* = *Unknown* $\wedge$ *TopView* = *Unknown*. Wählt der Agent eine der Aktionen *LookAtFront*, *LookAtSide* bzw. *LookAtTop*, wird bei der symbolischen Darstellung des Folgezustands die Variable *FrontView*, *SideView* bzw. *TopView* auf die dem Objekt laut Tabelle 5.3.1.1 zugeordnete, entsprechende Grundform gesetzt.

Beispiel 5.3.1.1:

Zu erkennen sei das Objekt ‘Zylinder’. Die erste gewählte Aktion des Agenten sei *LookAtSide*. Dann ist die symbolische Darstellung des Folgezustands *FrontView* = *Unknown* $\wedge$ *SideView* = *Rectangle* $\wedge$ *TopView* = *Unknown*. Anschließend entscheidet sich der Agent für *LookAtTop*, was zum Zustand *FrontView* = *Unknown* $\wedge$ *SideView* =

Rectangle $\wedge$ *TopView* = *Circle* führt. Würde der Agent nun ein weiteres Mal *LookAtSide* oder *LookAtTop* wählen, bliebe das Zustandssignal unverändert.

Sobald der Agent eine der *Recognize...*-Aktionen wählt, endet die Episode, unabhängig davon, ob das Objekt korrekt erkannt wurde.

Die Belohnungsfunktion gibt eine Belohnung von -1, falls eine der Aktionen *LookAtFront*, *LookAtSide* oder *LookAtTop* ausgeführt wurde. Ansonsten wird mit 10 belohnt, falls das Objekt richtig erkannt wurde. Falls es falsch erkannt wurde, vergibt Belohnungsfunktion eine Belohnung von -10. Die Aktion *RecognizeUnknown* führt immer zu einer Belohnung von -10, da kein Objekt als 'unbekannt' erkannt werden soll.

Nach spätestens 10 Schritten wird die Episode zwangsweise beendet.

5.3.2 Ergebnisse

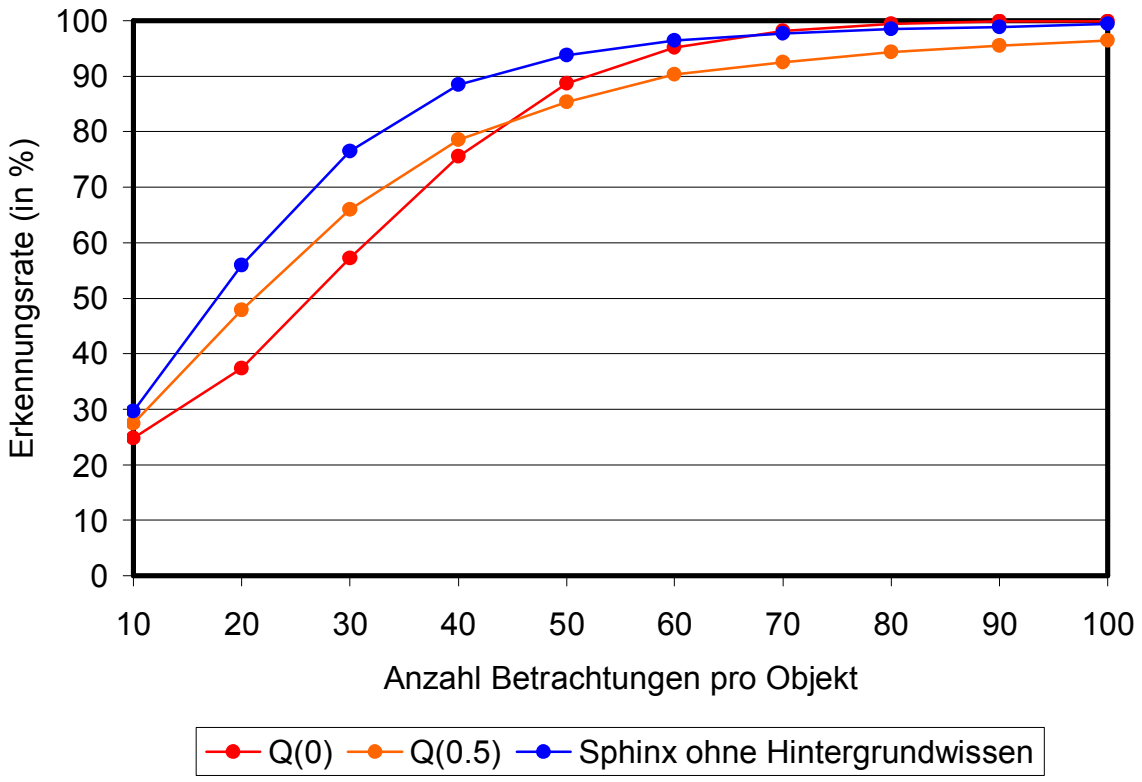
Tabelle 5.3.2.1 listet auf, wie hoch der Anteil korrekt erkannter Objekte nach jeder Trainingsphase durchschnittlich ist. In jeder Trainingsphase wird dem jeweiligen Agenten jedes Objekt zehnmal gezeigt. Die Werte ergeben sich aus jeweils 1000 unabhängigen Agenten. Die Sphinx-Agenten wurden mit folgenden Parametern gestartet:

- *useEveryVisitForGeneralization* := *True*
- *badMaxPatternLength* := 2
- *bestMaxPatternLength* := 2
- $t_{-}(p) := 2^{badMaxPatternLength-|p|+1}$
- $t_{\text{not}-}(p) := 0$
- $t_{age-} := 1$
- $t_{+}(p) := 1$
- $t_{\text{not}+}(p) := 0$
- $t_{\max+}(p) := 2^{bestMaxPatternLength-|p|+1}$
- $t_{age+} := 4$

Tabelle 5.3.2.1:

Anzahl Betrachtungen pro Objekt	Erkennungssrate (in %) ($Q(0)$)	Erkennungssrate (in %) ($Q(0.5)$)	Erkennungssrate (in %) (Sphinx ohne Hintergrundwissen)
10	24.8	27.5	29.7
20	37.4	47.9	55.9
30	57.2	66	76.5
40	75.5	78.6	88.5
50	88.7	85.4	93.8
60	95.1	90.3	96.4
70	98.2	92.5	97.7
80	99.4	94.4	98.5
90	99.9	95.5	98.9
100	99.9	96.4	99.4

Diagramm 5.3.2.1:



(Bemerkung: Die Erkennungsraten der Sphinx-Agenten sind gegenüber denen der $Q(0)$-Agenten zum Schluss hin minimal geringer. Wählt man auch für die Sphinx-Agenten $\lambda = 0$ und setzt die Schwellwerte höher an, so erreicht das Sphinx-Learning ebenfalls diese Erkennungsraten, allerdings auf Kosten der Erkennungsraten am Trainingsbeginn, welche dann zwar immer noch höher als die der $Q(0)$- und $Q(0.5)$-

Agenten sind, aber der Unterschied fällt weniger deutlich aus.)

Stellt man dem Agenten die Regel ‘Wenn noch keine Ansicht betrachtet wurde, dann betrachte das Objekt von vorne, von der Seite oder von oben.’ mittels des Konditionals:

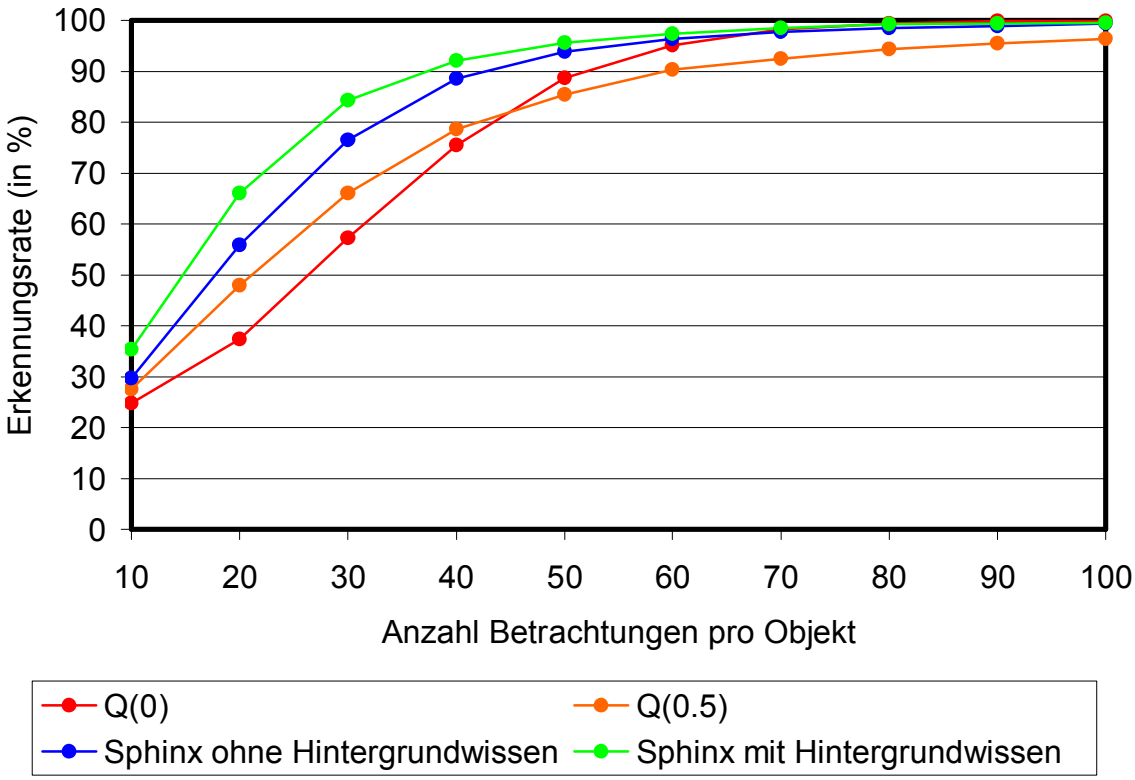
- $(Action = LookAtFront \vee Action = LookAtSide \vee Action = LookAtTop \mid FrontView = Unknown \wedge SideView = Unknown \wedge TopView = Unknown)$

als Hintergrundwissen zur Verfügung, ergeben sich die in folgender Tabelle aufgeführten Daten.

Tabelle 5.3.2.2:

Anzahl Betrachtungen pro Objekt	Erkennungssrate (in %) (Sphinx mit Hintergrundwissen)
10	35.4
20	66
30	84.3
40	92.1
50	95.6
60	97.4
70	98.5
80	99.2
90	99.4
100	99.5

Diagramm 5.3.2.2:



Das Hintergrundwissen wurde in dieser Testumgebung nicht durch allgemein höhere Schwellwerte geschützt, sondern indem Regeln der Form $\left(\bigvee_{a \in bestActions} Action = a \middle| s_{sym} \right)$ mit der Prämisse

$$s_{sym} = \\ FrontView = Unknown \wedge SideView = Unknown \wedge TopView = Unknown$$

nur dann zur Revision zugelassen wurden, wenn ihre Konsequenz mindestens eine Aktion aus der Menge {*LookAtFront, LookAtSide, LookAtTop*} enthielt. Das Hintergrundwissen konnte somit nicht durch Regeln mit völlig gegenteiliger Information überschrieben werden. Formal wurden die Schwellwerte wie oben definiert übernommen, nur der Schwellwert t_{age+} wurde geändert zu:

$$t_{age+} := \begin{cases} 4, \text{ falls } s_{sym} \neq FrontView = Unknown \wedge SideView = Unknown \\ \qquad \wedge TopView = Unknown \\ \quad \text{oder } bestActions \cap \{LookAtFront, LookAtSide, LookAtTop\} \neq \{\ \} \\ \infty, \text{ sonst} \end{cases}$$

Grund für dieses Vorgehen ist, dass das Hintergrundwissen sich nur auf einen

einzigen Zustand bezieht, nämlich den Startzustand. Allgemein höhere Schwellwerte verzögern das Lernen jedoch für alle Zustände, was in diesem Fall negative Konsequenzen hätte.

Die durchschnittlichen Episodenlängen nach *n* Trainingsepisoden allein sind für diese Umwelt kein Gütekriterium, weil Episoden enden, sobald der Agent das Objekt erkennt, unabhängig davon, ob richtig oder falsch. So können die Episoden anfangs kürzer sein als nach dem Lernvorgang, allerdings mit dem Unterschied, dass die Erkennungsrate zu Anfang niedrig ist und am Ende hoch. Daher wurde untersucht, wie viele Objektansichten die einzelnen Lernverfahren nach Beendigung des Lernens benötigen, um die Objekte zuverlässig zu erkennen.

Bei dieser Untersuchung konnten keine Unterschiede zwischen den Lernverfahren festgestellt werden. Alle benötigen durchschnittlich fast genau 1.9 Ansichten, bevor das Objekt korrekt erkannt wird. Die erste Aktion der meisten Agenten nach Beendigung des Lernens ist *LookAtTop*. Das Verhalten in Abhängigkeit von der nun bekannten *TopView* ist:

- Wenn *TopView* = *Circle*, dann L*ookAtFront*. Wenn anschließend *FrontView* = *Circle*, dann *RecognizeSphere*.
- Wenn *TopView* = *Ellipse*, dann *RecognizeEllipsoid*.
- Wenn *TopView* = *Circle*, dann *LookAtFront*. Wenn anschließend *FrontView* = *Rectangle*, dann *RecognizeCylinder*.
- Wenn *TopView* = *Circle*, dann *LookAtFront*. Wenn anschließend *FrontView* = *Triangle*, dann *RecognizeCone*.
- Wenn *TopView* = *Triangle*, dann *RecognizeTetrahedron*.
- Wenn *TopView* = *Square*, dann *LookAtLeft*. Wenn anschließend *LeftView* = *Triangle*, dann *RecognizePyramid*.
- Wenn *TopView* = *Rectangle*, dann *LookAtFront*. Wenn anschließend *FrontView* = *Triangle*, dann *RecognizePrism*.
- Wenn *TopView* = *Square*, dann *LookAtLeft*. Wenn anschließend *LeftView* = *Square*, dann *RecognizeCube*.
- Wenn *TopView* = *Rectangle*, dann *LookAtFront*. Wenn anschließend *FrontView* = *Rectangle*, dann *RecognizeCuboid*.

Die Anzahlen der vom jeweiligen Typ nach *n* Trainingsepisoden im Mittel durchgeführten Revisionen ohne bzw. mit Verwendung von Hintergrundwissen sind

den nächsten beiden Tabellen zu entnehmen.

Tabelle 5.3.2.3 (ohne Hintergrundwissen):

Anzahl Betrachtungen pro Objekt	schlechte Aktion, spezieller Zustand	schlechte Aktion, verallgemeinert	beste Aktionen, spezieller Zustand	beste Aktionen, verallgemeinert
10	77	164	118	419
20	142	374	281	705
30	195	559	469	918
40	242	728	671	1099
50	285	886	881	1264
60	327	1039	1094	1423
70	367	1190	1308	1577
80	408	1339	1523	1730
90	448	1488	1739	1881
100	487	1635	1956	2030

Tabelle 5.3.2.4 (mit Hintergrundwissen):

Anzahl Betrachtungen pro Objekt	schlechte Aktion, spezieller Zustand	schlechte Aktion, verallgemeinert	beste Aktionen, spezieller Zustand	beste Aktionen, verallgemeinert
10	75	146	127	471
20	135	333	292	794
30	185	503	484	1024
40	229	661	690	1217
50	271	812	901	1395
60	312	960	1114	1565
70	353	1108	1328	1730
80	393	1255	1543	1892
90	432	1402	1759	2054
100	472	1548	1975	2216

Einige der gelernten Zusammenhänge sind im Anhang A.1.2 aufgeführt. An dieser Stelle wird wiederum nur ein kleiner Auszug präsentiert.

Regeln mit Information über eine schlechte Aktion:

- Wenn $SideView = Unknown \wedge TopView = Unknown$, dann $\overline{Action = RecognizeTetrahedron}$ (Verwurzelung: 4)
- Wenn $FrontView = Triangle \wedge SideView = Unknown$, dann $\overline{Action = RecognizePyramid}$ (Verwurzelung: 1)
- Wenn $FrontView = Triangle \wedge SideView = Unknown$, dann

$\overline{Action = RecognizePrism}$ (Verwurzelung: 1)

- Wenn *FrontView* = *Triangle*, dann $\overline{Action = RecognizeCube}$ (Verwurzelung: 1)

Regeln mit Information über beste Aktionen:

- Wenn *FrontView* = *Circle*, dann *Action* = *RecognizeSphere* (Verwurzelung: 3)
- Wenn *FrontView* = *Ellipse*, dann *Action* = *RecognizeEllipsoid* (Verwurzelung: 3)
- Wenn *FrontView* = *Rectangle* $\wedge$ *TopView* = *Circle*, dann *Action* = *RecognizeCylinder* (Verwurzelung: 5)
- Wenn *FrontView* = *Unknown* $\wedge$ *SideView* = *Triangle*, dann *Action* = *LookAtFront* (Verwurzelung: 3)

5.3.3 Interpretation der Ergebnisse

Auch bei der Erkennung geometrischer Objekte ist Sphinx-Learning erfolgreicher als das klassische RL, wobei beide Verfahren vergleichbar starten und enden. In der Phase zwischen 30 und 50 Betrachtungen pro Objekt ist der Vorsprung am größten. Die Erkennungsraten der Sphinx-Agenten ist hier denen der $Q(\lambda)$-Agenten etwa 10 Betrachtungen voraus. Der Vorsprung resultiert daraus, dass falsche *Recognize*...-Aktionen durch Generalisierung bei ähnlichen Zuständen ausgeschlossen werden, insbesondere bei Zuständen, in denen viele Ansichten unbekannt sind. Somit werden dann die *LookAt*...-Aktionen bevorzugt, wodurch die Agenten mehr Information über das Objekt erlangen, bevor sie versuchen, es zu erkennen. Außerdem wird die Menge der plausiblen *Recognize*...-Aktionen auf weniger Objekte eingeschränkt, wenn ein Zustand ein Muster enthält, für das vermutet wird, dass es gewisse Objekte ausschließt, z. B. durch die Regel 'Wenn *FrontView* = *Triangle*, dann $\overline{Action = RecognizeCube}$'.

Parallel dazu wird durch Generalisierung erreicht, dass auch in unbekannten Zuständen, die jedoch ein bekanntes Muster enthalten, welches auf ein bestimmtes Objekt hinweist, eine korrekte Objekterkennung vorgenommen werden kann, beispielsweise durch die Regel 'Wenn *FrontView* = *Ellipse*, dann *Action* = *RecognizeEllipsoid*', oder, falls die bekannte Ansicht noch nicht eindeutig auf ein Objekt hinweist, als nächstes eine Ansicht betrachtet wird, welche die noch nötige Information liefert ('Wenn *FrontView* = *Unknown* $\wedge$ *TopView* = *Circle*, dann *Action* = *LookAtFront*').

Fügt man das Hintergrundwissen 'Wenn noch keine Ansicht betrachtet wurde,

dann betrachte das Objekt von vorne, von der Seite oder von oben.' hinzu, steigt die Erkennungsrate, weil die Agenten es nun unterlassen, 'blind' zu raten, und stattdessen zumindest im ersten Schritt Information über das Objekt sammeln. Dies spiegelt sich auch in den Tabellen 5.3.2.3 und 5.3.2.4 wider. Die Anzahlen der Revisionen über eine schlechte Aktion sind sowohl in der Spalte 'spezieller Zustand' als auch in der Spalte 'verallgemeinert' bei der Verwendung dieses Hintergrundwissens geringer, weil weniger Fehler gemacht werden. Aus demselben Grund sind die Anzahlen der Revisionen über beste Aktionen bei der Verwendung dieses Hintergrundwissens höher, denn weniger Fehler zu machen bedeutet, öfter erfolgreich zu sein.

5.4 Erkennung simulierter Objekte

5.4.1 Vorstellung der Umwelt

Diese Testumgebung dient dazu, die Fähigkeiten des Sphinx-Learnings unter realitätsnäheren Bedingungen, bei denen die Objekte nicht von vorn herein idealisierte Formen besitzen, zu analysieren. Hier wurden nicht zuerst Objekte gewählt und anschließend Formen gesucht, mit denen sich die Objekte beschreiben lassen, sondern es wurde die andere Richtung gewählt. D. h. zuerst wurde überlegt, durch welche Attribute man Objekte im Rahmen einer einfachen Objekterkennung darstellen kann und anschließend wurden zufällig gewählte Objekte mit den zuvor festgelegten Attributen beschrieben.

Der Agent kann die Objekte aus drei verschiedenen Blickwinkeln betrachten, nämlich frontal, von der Seite und aus einer Position zwischen diesen beiden Ansichten. Die Entscheidung für genau diese Blickwinkel, insbesondere für die Zwischenansicht, wurde auf Grund der Erkenntnis gefällt, dass Zwischenansichten bei der Objekterkennung durch Menschen von Bedeutung sind. Während Kleinkinder – besonders bei unbekannten Objekten – die Zwischenansicht bevorzugen, tendieren Erwachsene – besonders bei bekannten Objekten – eher zur frontalen oder die seitlichen Ansicht [PJJ2006].

Die Frontal- und die Seitenansicht werden beschrieben durch je drei Attribute:

- ungefähre (idealisierte) Form:
  - Kreis
  - Quadrat

    - Dreieck mit Spitze oben
    - Dreieck mit Spitze unten
- Proportionen der Form:
    - flach
    - normal
    - hoch
- Abweichung von der idealisierten Form:
    - wenig
    - mittel
    - viel

Außerdem wird das Objekt durch die Komplexität seiner Textur beschrieben. Mögliche Ausprägungen sind:

- einfach
- mittel
- komplex

Für die Experimente mit dieser Umwelt wurden die Attribute für jedes Objekt subjektiv durch den Anwender festgelegt, ließen sich aber in einer realen Anwendung formal durch das Verhältnis von Höhe zu Breite (Proportion der Form), Flächenabweichungen zwischen dem zweidimensional abgebildeten Objekt und der idealisierten Form (Abweichung von der idealisierten Form) und der Ausprägung von Farbunterschieden (Komplexität der Textur) spezifizieren.

Betrachtet der Agent das Objekt frontal oder von der Seite, so nimmt er die entsprechende idealisierte Form, die Proportionen der Form, die Abweichung und die Komplexität der Textur wahr. Aus der Zwischenansicht kann der Agent nur die idealisierten Formen der Front- und der Seitenansicht des Objekts und die Komplexität der Textur wahrnehmen, auf Grund des Betrachtungswinkels jedoch nicht die Proportionen oder die Abweichungen.

Beispiel 5.4.1.1:

Bei dem in folgender Abbildung aus der Zwischenansicht dargestellten Objekt ist erkennbar, dass Front- und Seitenform des Objekts rund sind, aber es ist kaum oder gar nicht zu erkennen, welche Proportionen sie haben, d. h. ob beide Formen Ellipsen (flache Kreise) sind, oder ob eine oder beide von ihnen annähernd normale

Kreise sind.

Abbildung 5.4.1.1

Erst, wenn das Objekt zur Front- (Abbildung 5.4.1.2) bzw. zur Seitenansicht (Abbildung 5.4.1.3) gedreht wird, werden die Proportionen erkennbar.

Abbildung 5.4.1.2

Abbildung 5.4.1.3

Die verwendeten Domains sind:

- *FrontViewShape* = {*Unknown*, *Circle*, *Square*, *TriangleUp*, *TriangleDown*}
- *FrontViewSize* = {*Unknown*, *Flat*, *Normal*, *Tall*}
- *FrontViewDeviation* = {*Unknown*, *Little*, *Medium*, *Much*}
- *SideViewShape* = {*Unknown*, *Circle*, *Square*, *TriangleUp*, *TriangleDown*}

- *SideViewSize* = {*Unknown*, *Flat*, *Normal*, *Tall*}
- *SideViewDeviation* = {*Unknown*, *Little*, *Medium*, *Much*}
- *Texture* = {*Unknown*, *Simple*, *Medium*, *Complex*}
- *Action* = {*RotateLeft*, *RotateRight*, *RecognizeUnkown*} $\cup$ *R*

R ist eine Menge von Aktionen, mit denen der Agent die erkannten Objekte benennt. Jedes Objekt ist durch eine *Recognize*...-Aktion in *R* vertreten.

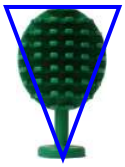
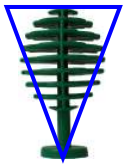
Die Zwischenansicht benötigt keine eigenen Domains, weil der Agent, wenn er das Objekt aus dieser Perspektive wahrnimmt, die dem Objekt entsprechenden Ausprägungen der Variablen *FrontViewShape*, *SideViewShape* und *Texture* durch die Umwelt erfährt.

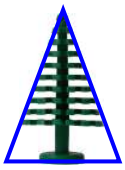
Zu Beginn jeder Episode nimmt der Agent das Objekt aus einer zufälligen Perspektive wahr und die Variablen werden entprechend dieser Perspektive und den tatsächlichen Objektdaten mit Werten belegt. Der Agent kann das Objekt nun, sofern möglich, nach links (von oben gesehen im Uhrzeigersinn) oder rechts rotieren oder es benennen. Wie bei der Erkennung geometrischer Objekte endet eine Episode, sobald der Agent eine der *Recognize*...-Aktionen gewählt hat, spätestens jedoch nach 10 Schritten.

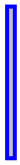
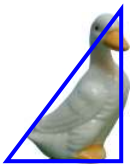
Die Belohnungsfunktion ist dieselbe wie bei der Erkennung geometrischer Objekte in Kapitel 5.3.

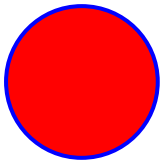
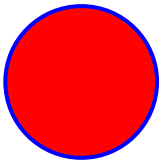
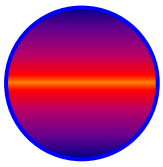
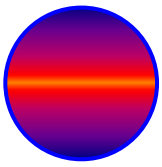
Als Beispiel-Objekte wurden die folgenden Objekte gewählt.

Tabelle 5.4.1.1:

Objektname	zu erkennen als	*FrontView*	*SideView*	*Texture*
cokeBottle	Bottle	 • *Shape*: *Triangle Up* • *Size*: *Tall* • *Deviation*: *Medium*	 • *Shape*: *Triangle Up* • *Size*: *Tall* • *Deviation*: *Medium*	Medium
vittelBottle	Bottle	 • *Shape*: *Triangle Up* • *Size*: *Tall* • *Deviation*: *Medium*	 • *Shape*: *Triangle Up* • *Size*: *Tall* • *Deviation*: *Medium*	Medium
normalTree	Tree	 • *Shape*: *Triangle Down* • *Size*: *Normal* • *Deviation*: *Much*	 • *Shape*: *Triangle Down* • *Size*: *Normal* • *Deviation*: *Much*	Simple

firTree	Tree	 • *Shape*: *Triangle Up* • *Size*: *Normal* • *Deviation*: *Little*	 • *Shape*: *Triangle Up* • *Size*: *Normal* • *Deviation*: *Little*	Simple
tetraPack	TetraPack	 • *Shape*: *Square* • *Size*: *Tall* • *Deviation*: *Little*	 • *Shape*: *Square* • *Size*: *Tall* • *Deviation*: *Little*	Complex
houseWithFrontEntrance	House	 • *Shape*: *Square* • *Size*: *Normal* • *Deviation*: *Little*	 • *Shape*: *Triangle Up* • *Size*: *Normal* • *Deviation*: *Medium*	Medium
houseWithSideEntrance	House	 • *Shape*: *Triangle Up* • *Size*: *Normal* • *Deviation*: *Medium*	 • *Shape*: *Square* • *Size*: *Normal* • *Deviation*: *Little*	Medium

pott	Pott	 • *Shape*: *Square* • *Size*: *Normal* • *Deviation*: *Little*	 • *Shape*: *Square* • *Size*: *Normal* • *Deviation*: *Little*	Simple
dvd	DVD	 • *Shape*: *Circle* • *Size*: *Normal* • *Deviation*: *Little*	 • *Shape*: *Square* • *Size*: *Tall* • *Deviation*: *Little*	Simple
flyingBird	Bird	 • *Shape*: *Square* • *Size*: *Flat* • *Deviation*: *Little*	 • *Shape*: *Square* • *Size*: *Flat* • *Deviation*: *Little*	Simple
walkingBird	Bird	 • *Shape*: *Triangle Up* • *Size*: *Normal* • *Deviation*: *Medium*	 • *Shape*: *Triangle Up* • *Size*: *Tall* • *Deviation*: *Medium*	Simple
football	Football	 • *Shape*: *Circle* • *Size*: *Normal* • *Deviation*: *Little*	 • *Shape*: *Circle* • *Size*: *Flat* • *Deviation*: *Little*	Medium

ball1	Ball	 • *Shape*: *Circle* • *Size*: *Normal* • *Deviation*: *Little*	 • *Shape*: *Circle* • *Size*: *Normal* • *Deviation*: *Little*	Simple
ball2	Ball	 • *Shape*: *Circle* • *Size*: *Normal* • *Deviation*: *Little*	 • *Shape*: *Circle* • *Size*: *Normal* • *Deviation*: *Little*	Medium
ball3	Ball	 • *Shape*: *Circle* • *Size*: *Normal* • *Deviation*: *Little*	 • *Shape*: *Circle* • *Size*: *Normal* • *Deviation*: *Little*	Complex

5.4.2 Ergebnisse

Die Parameter wurden wie folgt gesetzt:

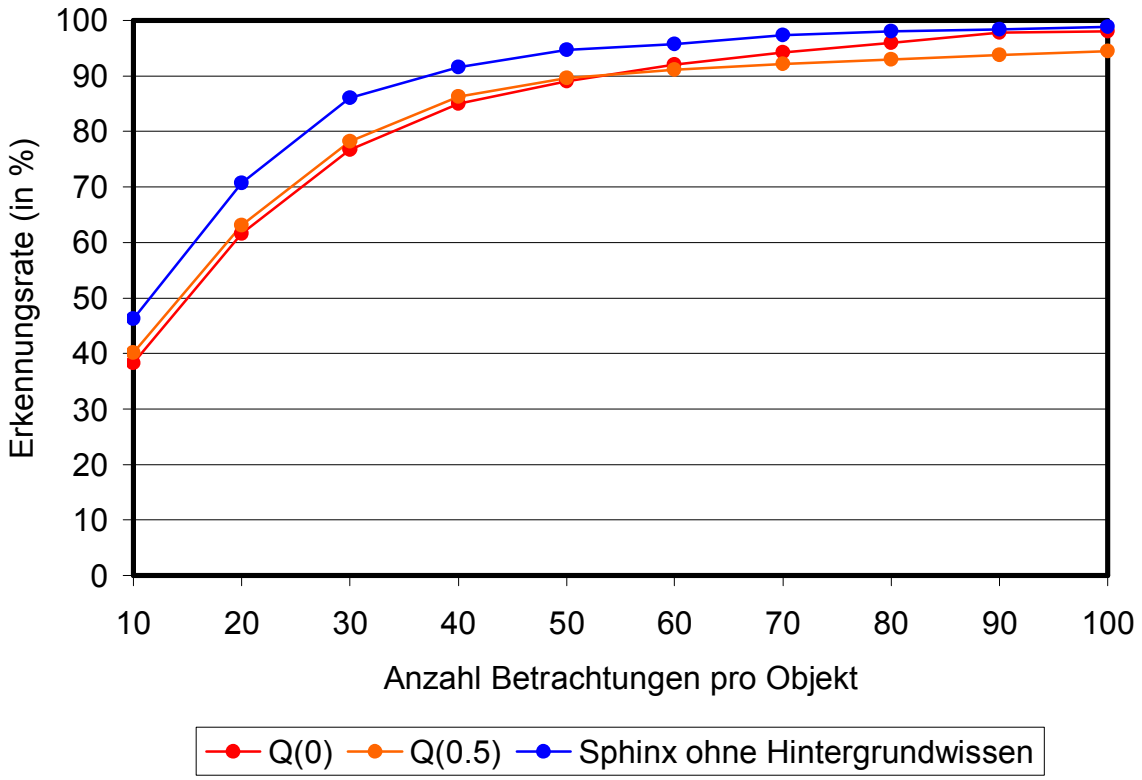
- *useEveryVisitForGeneralization* := *True*
- *badMaxPatternLength* := 2
- *bestMaxPatternLength* := 3
- $t_{-}(p) := 2^{badMaxPatternLength-|p|+1}$
- $t_{\text{not}-}(p) := 0$
- $t_{age-} := 1$
- $t_{+}(p) := 1$
- $t_{\text{not}+}(p) := 0$
- $t_{\text{max}+}(p) := 2^{bestMaxPatternLength-|p|+1}$
- $t_{age+} := 4$

Die aus 100 unabhängigen Agenten ermittelten, durchschnittlichen Erkennungsraten sind Tabelle 5.4.2.1 und Diagramm 5.4.2.1 zu entnehmen.

Tabelle 5.4.2.1:

Anzahl Betrachtungen pro Objekt	Erkennungssrate (in %) ($Q(0)$)	Erkennungssrate (in %) ($Q(0.5)$)	Erkennungssrate (in %) (Sphinx ohne Hintergrundwissen)
10	38.3	40.1	46.3
20	61.6	63.1	70.7
30	76.7	78.2	86
40	85	86.3	91.6
50	89	89.6	94.7
60	92	91.1	95.7
70	94.2	92.2	97.3
80	96	93	98
90	97.8	93.8	98.4
100	98	94.5	98.8

Diagramm 5.4.2.1:



Als Hintergrundwissen wurden im Folgenden Regeln zur Verfügung gestellt, durch die der Agent in die Lage versetzt wird, die Objekte richtig zu benennen, sofern er alle Ansichten wahrgenommen hat. Außerdem gehörten Regeln zum

Hintergrundwissen, die den Agenten anweisen, das Objekt zunächst von allen Seiten zu betrachten. Insgesamt wird dem Agenten also Wissen mitgegeben, welches ihm ermöglicht, von Beginn an alle Objekte richtig zu benennen. Allerdings ist dieser Lösungsweg nicht optimal, weil ggfs. unnötig viel vom Objekt betrachtet wird, bevor es erkannt wird. Die zu vollbringende Leistung des Agenten besteht darin zu lernen, wie er mit weniger Ansichten das Objekt benennen kann. Da der Agent die erste Ansicht nicht frei aussuchen kann, wie es bei der Erkennung geometrischer Objekte in Kapitel 5.3 der Fall ist, muss er die Wahl, ob er eine bzw. welche Ansicht er als nächstes erfahren möchte, davon abhängig machen, mit welcher Ansicht er beim Start der Episode konfrontiert wurde.

Die für das Hintergrundwissen verwendeten Konditionale waren:

- (*Action* = *RecognizeBottle* | *FrontViewShape* = *Triangle Up* ∧ *FrontViewSize* = *Tall* ∧ *FrontViewDeviation* = *Medium* ∧ *SideViewShape* = *Triangle Up* ∧ *SideViewSize* = *Tall* ∧ *SideViewDeviation* = *Medium* ∧ *Texture* = *Medium*)
- (*Action* = *RecognizeTree* | *FrontViewShape* = *Triangle Down* ∧ *FrontViewSize* = *Normal* ∧ *FrontViewDeviation* = *Much* ∧ *SideViewShape* = *Triangle Down* ∧ *SideViewSize* = *Normal* ∧ *SideViewDeviation* = *Much* ∧ *Texture* = *Simple*)
- (*Action* = *Tree* | *FrontViewShape* = *Triangle Up* ∧ *FrontViewSize* = *Normal* ∧ *FrontViewDeviation* = *Little* ∧ *SideViewShape* = *Triangle Up* ∧ *SideViewSize* = *Normal* ∧ *SideViewDeviation* = *Little* ∧ *Texture* = *Simple*)
- (*Action* = *TetraPack* | *FrontViewShape* = *Square* ∧ *FrontViewSize* = *Tall* ∧ *FrontViewDeviation* = *Little* ∧ *SideViewShape* = *Square* ∧ *SideViewSize* = *Tall* ∧ *SideViewDeviation* = *Little* ∧ *Texture* = *Complex*)
- (*Action* = *RecognizeHouse* | *FrontViewShape* = *Square* ∧ *FrontViewSize* = *Normal* ∧ *FrontViewDeviation* = *Little* ∧ *SideViewShape* = *Triangle Up* ∧ *SideViewSize* = *Normal* ∧ *SideViewDeviation* = *Medium* ∧ *Texture* = *Medium*)
- (*Action* = *RecognizeHouse* | *FrontViewShape* = *Triangle Up* ∧ *FrontViewSize* = *Normal* ∧ *FrontViewDeviation* = *Medium* ∧ *SideViewShape* = *Square* ∧ *SideViewSize* = *Normal* ∧ *SideViewDeviation* = *Little* ∧ *Texture* = *Medium*)
- (*Action* = *RecognizePott* | *FrontViewShape* = *Square* ∧ *FrontViewSize* = *Normal* ∧ *FrontViewDeviation* = *Little* ∧ *SideViewShape* = *Square* ∧ *SideViewSize* = *Normal* ∧ *SideViewDeviation* = *Little* ∧ *Texture* = *Simple*)
- (*Action* = *RecognizeDVD* | *FrontViewShape* = *Circle* ∧ *FrontViewSize* = *Normal* ∧

FrontViewDeviation = *Little* ∧ *SideViewShape* = *Square* ∧ *SideViewSize* = *Tall* ∧ *SideViewDeviation* = *Little* ∧ *Texture* = *Simple*)

- (*Action* = *RecognizeBird* | *FrontViewShape* = *Square* ∧ *FrontViewSize* = *Flat* ∧ *FrontViewDeviation* = *Little* ∧ *SideViewShape* = *Square* ∧ *SideViewSize* = *Flat* ∧ *SideViewDeviation* = *Little* ∧ *Texture* = *Simple*)
- (*Action* = *RecognizeBird* | *FrontViewShape* = *Triangle Up* ∧ *FrontViewSize* = *Normal* ∧ *FrontViewDeviation* = *Medium* ∧ *SideViewShape* = *Triangle Up* ∧ *SideViewSize* = *Tall* ∧ *SideViewDeviation* = *Medium* ∧ *Texture* = *Simple*)
- (*Action* = *RecognizeFootball* | *FrontViewShape* = *Circle* ∧ *FrontViewSize* = *Normal* ∧ *FrontViewDeviation* = *Little* ∧ *SideViewShape* = *Circle* ∧ *SideViewSize* = *Flat* ∧ *SideViewDeviation* = *Little* ∧ *Texture* = *Medium*)
- (*Action* = *RecognizeBall* | *FrontViewShape* = *Circle* ∧ *FrontViewSize* = *Normal* ∧ *FrontViewDeviation* = *Little* ∧ *SideViewShape* = *Circle* ∧ *SideViewSize* = *Normal* ∧ *SideViewDeviation* = *Little* ∧ *Texture* = *Little*)
- (*Action* = *RecognizeBall* | *FrontViewShape* = *Circle* ∧ *FrontViewSize* = *Normal* ∧ *FrontViewDeviation* = *Little* ∧ *SideViewShape* = *Circle* ∧ *SideViewSize* = *Normal* ∧ *SideViewDeviation* = *Little* ∧ *Texture* = *Medium*)
- (*Action* = *RecognizeBall* | *FrontViewShape* = *Circle* ∧ *FrontViewSize* = *Normal* ∧ *FrontViewDeviation* = *Little* ∧ *SideViewShape* = *Circle* ∧ *SideViewSize* = *Normal* ∧ *SideViewDeviation* = *Little* ∧ *Texture* = *Complex*)
- (*Action* = *RotateRight* | *FrontViewSize* = *Unknown*)
- (*Action* = *RotateLeft* | *SideViewSize* = *Unknown*)

Wie auch bei der Gridworld mit Hindernissen und Farben wurden die Schwellwerte erhöht. Hier jedoch aus zwei Gründen: Erstens um das Hintergrundwissen zu schützen und zweitens, damit es auch zur Anwendung kommt. Gerade diejenigen Regeln, die dem Agenten erlauben, das Objekt korrekt zu erkennen, können erst dann angewendet werden, wenn alle Ansichten bekannt sind. Bei zu niedrigen Schwellwerten würde der Agent auf Grund von unsicheren Mutmaßungen schon in einem frühen Lernstadium versuchen, das jeweilige Objekt nach wenigen Ansichten zu erkennen, so dass die *Recognize*...-Regeln des Hintergrundwissens nicht mehr zur Anwendung kommen würden, weil dazu alle Ansichten bekannt sein müssen. Da die zu Beginn des Lernens vermuteten Regeln mit einer höheren Wahrscheinlichkeit nicht richtig sind, wäre die Erkennungsrate wesentlich niedriger, als sie es bei

höheren Schwellwerten ist. Aus diesen Gründen wurde gesetzt:

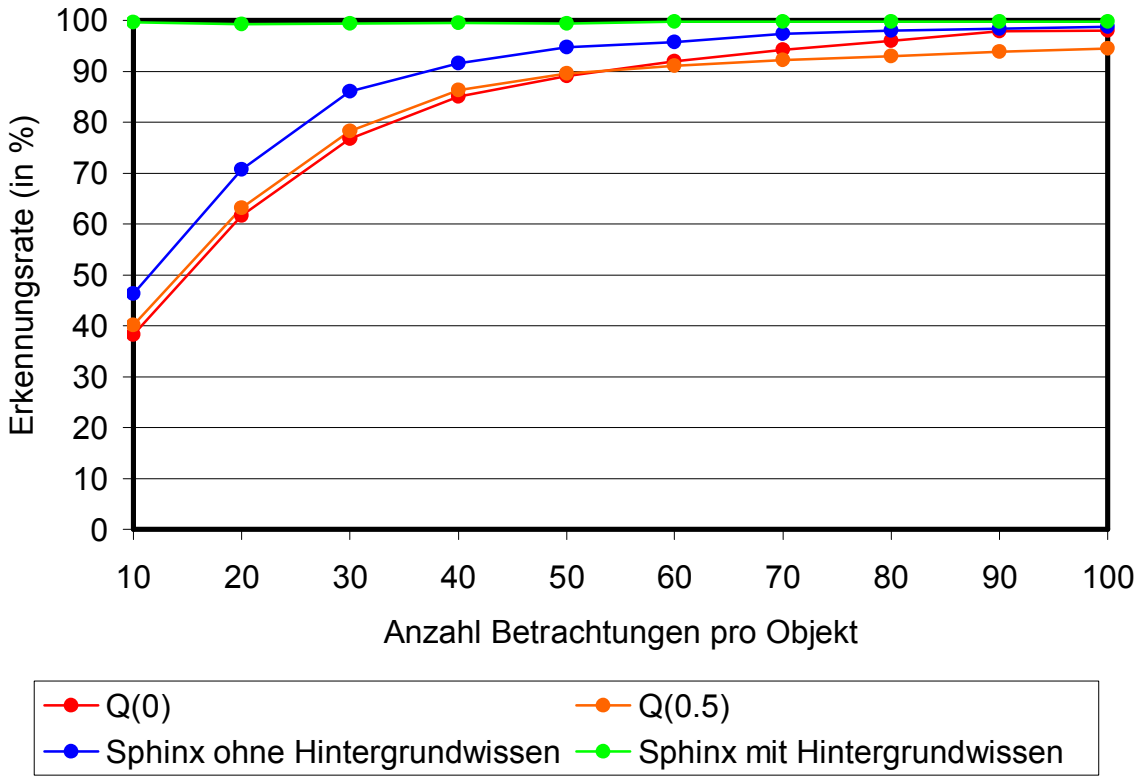
- $t_{-}(p) := \left(\frac{1}{4} + \left(1 - \frac{1}{4}\right)\sqrt{\Pr(p)}\right) 2^{badMaxPatternLength - |p| + 5}$
- $t_{\max +}(p) := \left(\frac{1}{4} + \left(1 - \frac{1}{4}\right)\sqrt{\Pr(p)}\right) 2^{bestMaxPatternLength - |p| + 6}$
- $t_{age+} := 8$

Es ergaben sich die im Folgenden aufgeführten Erkennungsraten.

Tabelle 5.4.2.2:

Anzahl Betrachtungen pro Objekt	Erkennungsrate (in %) (Sphinx mit Hintergrundwissen)
10	99.6
20	99.3
30	99.4
40	99.5
50	99.4
60	99.7
70	99.7
80	99.7
90	99.7
100	99.8

Diagramm 5.4.2.2:

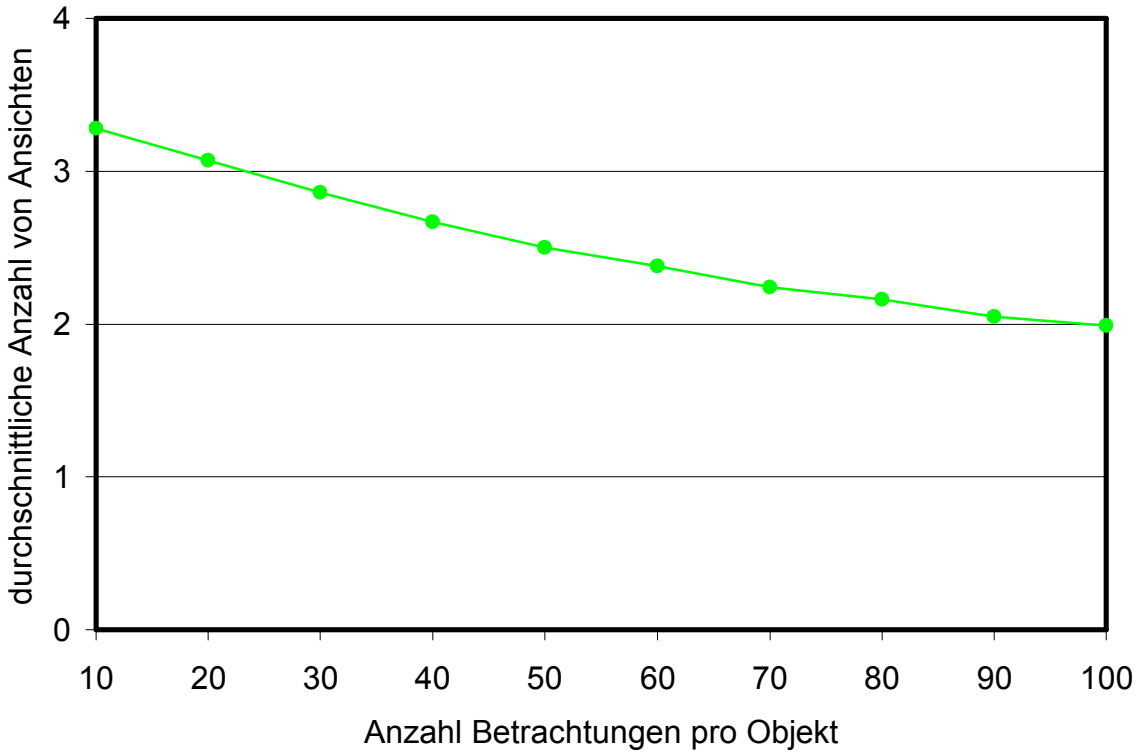


Da ein Lernziel bei der Verwendung von Hintergrundwissen die Minimierung der benötigten Ansichten ist, wurde in Erfahrung gebracht, nach wie vielen Ansichten ein Sphinx-Agent mit Hintergrundwissen die Objekte korrekt erkennen kann (siehe nachfolgende Tabelle).

Tabelle 5.4.2.3:

Anzahl Betrachtungen pro Objekt	durchschnittliche Anzahl von Ansichten
10	3.28
20	3.07
30	2.86
40	2.67
50	2.5
60	2.38
70	2.24
80	2.16
90	2.05
100	1.99

Diagramm 5.4.2.3:



(Bemerkung: Der anfängliche Wert von 3.28 Ansichten bei nur 3 zur Verfügung stehenden Ansichten insgesamt ergibt sich dadurch, dass, falls die von der Umwelt bestimmte erste Ansicht die Zwischenansicht ist, der Agent das Objekt gemäß seinen Regeln zunächst zur Front- bzw. Seitenansicht (zweite Ansicht) dreht und anschließend wieder zurück zur Zwischenansicht (dritte Ansicht), um zur Seiten- bzw. Frontansicht (vierte Ansicht) zu gelangen. Mit einer Wahrscheinlichkeit von $\frac{1}{3}$ werden demnach vier Ansichten wahrgenommen, dabei die Zwischenansicht doppelt. Ist die erste Ansicht die Front- oder Seitenansicht (jeweils ebenfalls mit einer Wahrscheinlichkeit von $\frac{1}{3}$), sind es nur drei Ansichten, da der Agent das Objekt dann nicht zurückdrehen muss. Die erwartete Anzahl von Ansichten liegt demnach rechnerisch bei $\frac{1}{3}\cdot 3+\frac{1}{3}\cdot 3+\frac{1}{3}\cdot 4=3.\overline{3}$. Der Wert von 3.28 nach der ersten Trainingsphase zeigt, dass der Agent sich zu diesem Zeitpunkt fast ausschließlich an die Regeln des Hintergrundwissens hält.)

Folgende Tabellen geben Auskunft darüber, wie viele Revisionen die Sphinx-Agenten (ohne bzw. mit Hintergrundwissen) im Schnitt nach *n* Trainingsepisoden

durchgeführt haben.

Tabelle 5.4.2.4 (ohne Hintergrundwissen):

Anzahl Betrachtungen pro Objekt	schlechte Aktion, spezieller Zustand	schlechte Aktion, verallgemeinert	beste Aktionen, spezieller Zustand	beste Aktionen, verallgemeinert
10	114	1043	56	4373
20	193	2282	198	6646
30	251	3189	356	8434
40	297	3880	525	10130
50	339	4519	700	11778
60	378	5120	880	13436
70	416	5704	1062	15102
80	455	6284	1245	16798
90	492	6861	1431	18430
100	529	7440	1618	20097

Tabelle 5.4.2.5 (mit Hintergrundwissen):

Anzahl Betrachtungen pro Objekt	schlechte Aktion, spezieller Zustand	schlechte Aktion, verallgemeinert	beste Aktionen, spezieller Zustand	beste Aktionen, verallgemeinert
10	74	0	24	2
20	146	0	214	146
30	214	0.1	485	499
40	279	1.4	780	1010
50	341	5.9	1071	1569
60	399	15.7	1354	2140
70	456	30	1628	2708
80	511	49.2	1893	3272
90	562	75.6	2149	3865
100	613	110.7	2399	4498

Einige der gefundenen Regel sind in Anhang A.1.3. Hier werden wie zuvor nur wenige, ausgesuchte aufgeführt:

Regeln mit Information über eine schlechte Aktion:

- Wenn *Texture* = *Complex*, dann $\overline{Action = RecognizeBottle}$ (Verwurzelung: 1)
- Wenn *FrontViewShape* = *Circle*, dann $\overline{Action = RecognizeBottle}$ (Verwurzelung: 1)
- Wenn *Texture* = *Medium*, dann $\overline{Action = RecognizePott}$ (Verwurzelung: 1)
- Wenn *RightViewShape* = *Square* $\wedge$ *Texture* = *Simple*, dann

$\overline{Action = RecognizeHouse}$ (Verwurzelung: 2)

Regeln mit Information über beste Aktionen:

- Wenn *FrontViewShape* = *TriangleUp* $\wedge$ *FrontViewSize* = *Tall*, dann *Action* = *RecognizeBottle* (Verwurzelung: 2)
- Wenn *FrontViewShape* = *TriangleDown*, dann *Action* = *RecognizeTree* (Verwurzelung: 1)
- Wenn *RightViewSize* = *Normal* $\wedge$ *RightViewDeviation* = *Medium*, dann *Action* = *RecognizeHouse* (Verwurzelung: 2)
- Wenn *FrontViewShape* = *Circle* $\wedge$ *RightViewShape* = *Unknown* $\wedge$ *Texture* = *Simple*, dann *Action* = *RotateLeft* (Verwurzelung: 4)

5.4.3 Interpretation der Ergebnisse

Ohne Verwendung von Hintergrundwissen ergibt sich bei der Erkennung simulierter Objekte ein ähnliches Bild wie bei der Erkennung geometrischer Objekte.

Das in dieser Testumgebung verwendete Hintergrundwissen, welches einen möglichen Lösungsweg für jedes Objekt vorgibt, sorgt für nahezu 100 %-ige Erkennungsraten. Die zunächst hohe Anzahl benötigter Ansichten sinkt im Laufe des Lernens von etwa 3.3 auf 2, allerdings nicht auf das Niveau ohne Verwendung von Hintergrundwissen von 1.3 notwendigen Ansichten nach 100 Betrachtungen pro Objekt. Die Ursache hierfür ist, dass das Hintergrundwissen durch die hohen Schwellwerte erst aufgegeben wird, wenn nahezu sicher davon ausgegangen werden kann, dass das Objekt sicher erkannt werden kann, ohne alle Ansichten wahrgenommen zu haben. Dies garantiert eine durchgängig zuverlässige Erkennungsrate, verlangsamt jedoch eine Änderung des Handelns.

Wie bei der Gridworld mit Hindernissen und Farben zögern die höheren Schwellwerte die Revisionen hinaus, wie aus den Tabellen 5.4.2.4 und 5.4.2.5 ersichtlich wird, anschließend steigen ihre Anzahlen aber ebenfalls deutlich an. Vergleicht man die Spalten ‘beste Aktionen, spezieller Zustand’, so stellt man fest, dass bereits nach 20 Betrachtungen pro Objekt bei der Verwendung des Hintergrundwissens mehr Revisionen dieses Typs gemacht wurden. Dies liegt daran, dass die Sphinx-Agenten mit Hintergrundwissen von Anfang an sehr erfolgreich sind und dementsprechend viele positive Erfahrungen machen, was zu vielen entsprechenden Revisionen führt.

Sphinx-Learning eignet sich – auf Grund der Lernkurve und der Möglichkeit, Hintergrundwissen einbringen zu können – prinzipiell besser als klassisches RL für die interaktive Objekterkennung, die auf schnelles Lernen und die Minimierung benötigter Ansichten angewiesen ist. Unabhängig davon, ob Sphinx-Learning oder RL verwendet wird, muss lediglich ein Modul vorgeschaltet werden, welches aus den Bild-Rohdaten gewisse Eigenschaften des Objekts extrahiert, wie hier die Grundform (incl. deren Proportionen), die Abweichung von dieser Grundform und die Komplexität der Textur. Welche und wie genau diese Eigenschaften durch das Modul bestimmt werden sollen (z. B. wann eine Ansicht eines Objekts als 'flach', 'normal' oder 'hoch' bezeichnet werden soll), muss der Anwender vorab nach seinen Wünschen festlegen. Die Eigenschaften können dann als Eingabedaten für das Sphinx-Learning verwendet werden. Die Bestimmung der Grundform und die Abweichung ist durch Matching möglich. Dabei wird für jede Grundform ermittelt, wie groß die Ähnlichkeit zwischen ihr und dem Objekt ist und diejenige als Beschreibung für das Objekt gewählt wird, bei der die Übereinstimmung am größten ist. Die 'Ähnlichkeit' kann i. A. durch den Flächenunterschied zwischen Grundform und Objekt definiert werden. Da hier nur einfache Grundformen für das Matching verwendet werden müssen, wird dieser Vergleich nur eine geringe Laufzeit in Anspruch nehmen. In dieser Testumgebung wurde aber die Dreiecksform ggfs. der Rechtecksform bevorzugt, wenn das Objekt (wie die Flaschen) oben spitz und unten breit oder oben breit und unten spitz ist, damit nicht zu viele Objekte durch die Rechtecksform beschrieben werden.

6. Ausblick und Resümee

6.1 Ausblick

Neben der primären Aufgabe, an die Umwelt angepasstes Verhalten zu lernen, kann Sphinx-Learning auch dazu verwendet werden, Regeln aus einer vorhandenen QTable eines trainierten $Q(\lambda)$-Agenten zu generieren, sofern das Zustandssignal um eine symbolische Komponente ergänzt werden kann. Dazu ist es lediglich notwendig, einen Sphinx-Agenten mit dieser QTable und einer uniformen OCF κ zu erschaffen. Die Schwellwertfunktionen werden hoch angesetzt, so dass Regeln nur gebildet werden, wenn ihre Korrektheit nahezu sicher angenommen werden kann. Durch die trainierte QTable wählt der Agent die jeweils günstigsten Aktionen. Das Top-Level kann das vom Bottom-Level gesteuerte Handeln 'beobachten' und nach einiger Zeit Zusammenhänge finden und Regeln erstellen.

Auch der umgekehrte Weg, eine QTable aus einer trainierten OCF zu generieren, ist möglich. Dabei bestimmt das Top-Level die Aktionen, während die QTable die entsprechenden *Q*-Werte annimmt.

Eine weitere mögliche Anwendung ist die Klassifikation im Generellen, ähnlich wie die in den Kapiteln 5.3 und 5.4 vorgestellte Klassifikation von Objekten an Hand von visuellen Eigenschaften. Für eine richtige Klassifikation erhält der Agent eine gute Belohnung, für eine falsche eine schlechte. Resultat sind Regeln, mit denen die Klassifikation mittels der zur Verfügung gestellten Attributen vorgenommen werden kann.

Schließlich wäre es interessant, in zukünftigen Arbeiten zu klären, wie Sphinx-Learning in Multi-Agenten-Systemen eingesetzt werden kann. Bei untereinander kommunizierenden Agenten könnten Agenten, wenn sie eine Regel gefunden zu haben glauben, den anderen Agenten die Regel mitteilen, woraufhin diese Gegenbeispiele in ihren Erfahrungen suchen könnten und sie, sofern es welche gibt, dem vorschlagenden Agenten als Antwort zurückgeben könnten. Ansonsten könnten alle Agenten diese Regel aufnehmen, auch wenn ihre eigene Erfahrung nicht ausreicht, um die nötigen Schwellwerte zu überschreiten und somit diese Regel selbstständig zu generieren.

6.2 Resümee

Der Wunsch, Unbekanntes zu entdecken und Bekanntes zu verbessern, bringt den Menschen auf immer neue Ideen und lässt ihn nach zuvor noch nie beschrittenen Wegen forschen, sie zu verwirklichen und somit die Grenzen des Bekannten zu überschreiten. Die Orientierung an den Lösungen der Natur ist ihm dabei oft eine Hilfe.

In der vorliegenden Arbeit konnte gezeigt werden, dass das vorgestellte, hybride Lernverfahren Sphinx, welches – ähnlich dem menschlichen Lernen – zwei Level kombiniert, einen Vorsprung in der Lerngeschwindigkeit gegenüber dem klassischen RL hat. Dieser Vorsprung existiert selbst dann, wenn dem Sphinx-Learning kein Hintergrundwissen zur Verfügung gestellt wird, wenn also beide Verfahren chancengleich gestartet werden. Dies wird erreicht, indem Sphinx-Learning generelle Zusammenhänge erkennt und so in unbekannten Situationen auf aus vorherigen, ähnlichen Situationen Erlerntes zugreifen kann. Dadurch verlieren auch Wahrnehmungen, die für die Lernaufgabe nicht von Bedeutung sind (wie die Farben der Zellen der Gridworld aus Kapitel 5.2), ihre Wirkung auf die Aktionswahl des Agenten.

Hintergrundwissen unterstützt den Lernprozess, so dass auch zu Beginn des Lernens hohe Erfolgsraten erzielt werden. Es kann mittels Konditionalen in symbolischer Form komfortabel eingegeben werden. Dies ist beim klassischen RL nicht möglich. Ferner kann Erlerntes für Menschen leicht verständlich in Form von Regeln ausgegeben werden; es ist keine Interpretation von numerischen Werten durch den Anwender notwendig.

Wie die Lernkurven in den Testumgebungen zeigen, sind Sphinx-Agenten i. A. schnell in der Lage, fehlendes Hintergrundwissen auszugleichen. Es gilt, dass Hintergrundwissen umso wertvoller ist, je schwieriger die Lernaufgabe ist und je informationsreicher und zuverlässiger es ist.

Außerdem eignet sich Sphinx-Learning in Umwelten, in denen das Zustandssignal auch unvollständige Information über den wahren Zustand liefern kann (wie es bei Aufgaben in der realen Welt üblich ist), weil es in diesem Fall auf Grund der Generalisierung gewisse Aktionen für plausibler hält als andere, während aus Sicht des klassischen RL eine völlig neue Situation vorliegt, wenn (z. B. bedingt durch die Hardware oder die Software-Subroutinen des Roboters) eine Komponente des

Zustandssignals nicht ermittelt werden kann.

Diese Eigenschaften qualifizieren Sphinx-Learning insbesondere für den Einsatz bei der interaktiven Objekterkennung. Durch die Möglichkeit, einen offensichtlichen (aber nicht notwendiger Weise optimalen) Lösungsweg mittels Regeln vorzugeben, der durch Interaktion mit der Umwelt laufend verbessert wird, ohne Zuverlässigkeit einzubüßen, werden hohe Erkennungsraten gewährleistet, während gleichzeitig die Menge benötigter Information sinkt. Zu erwähnen ist auch, dass die Erkennungsraten in der realitätsnäheren Umgebung 'Erkennung simulierter Objekte' (Kapitel 5.4) anfangs höher sind als bei der Erkennung geometrischer Objekte (Kapitel 5.3), und das, obwohl es deutlich mehr zu erkennende Objekte gibt.

Anhang

A.1 Auflistung einiger der gefundenen Regeln

A.1.1 Gridworld mit Hindernissen und Farben

Regeln mit Information über beste Aktionen:

- Wenn *TargetNorthSouth* = *NorthAhead* ∧ *TargetDistance* = *Middle* ∧ *ObstacleNorth* = *False*, dann *Action* = *GoNorth* (Verwurzelung: 2)
- Wenn *TargetNorthSouth* = *NorthAhead* ∧ *ObstacleNorth* = *False*, dann *Action* = *GoNorth* ∨ *Action* = *GoEast* ∨ *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *NorthAhead* ∧ *ObstacleNorth* = *False* ∧ *ObstacleSouth* = *False*, dann *Action* = *GoNorth* (Verwurzelung: 2)
- Wenn *TargetNorthSouth* = *NorthAhead* ∧ *TargetDistance* = *Middle*, dann *Action* = *GoNorth* ∨ *Action* = *GoEast* ∨ *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *NorthAhead* ∧ *TargetDistance* = *Middle* ∧ *ObstacleWest* = *False*, dann *Action* = *GoNorth* ∨ *Action* = *GoEast* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *NorthAhead* ∧ *TargetEastWest* = *WestAside* ∧ *ObstacleWest* = *False*, dann *Action* = *GoNorth* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *NorthAhead* ∧ *TargetEastWest* = *WestAside* ∧ *ObstacleSouth* = *False*, dann *Action* = *GoNorth* (Verwurzelung: 2)
- Wenn *TargetNorthSouth* = *NorthAhead* ∧ *TargetEastWest* = *WestAside* ∧ *ObstacleEast* = *False*, dann *Action* = *GoNorth* (Verwurzelung: 2)
- Wenn *TargetNorthSouth* = *NorthAhead* ∧ *TargetEastWest* = *WestAside* ∧ *ObstacleNorth* = *False*, dann *Action* = *GoNorth* ∨ *Action* = *GoWest* (Verwurzelung: 2)
- Wenn *TargetNorthSouth* = *NorthAhead* ∧ *TargetEastWest* = *WestAside* ∧ *TargetDistance* = *Middle*, dann *Action* = *GoNorth* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *NorthAhead* ∧ *TargetEastWest* = *WestAside*, dann *Action* = *GoNorth* ∨ *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetEastWest* = *EastAhead* ∧ *ObstacleWest* = *True*, dann *Action* = *GoEast* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *Neither-Nor* ∧ *TargetDistance* = *Far* ∧ *ObstacleEast* = *False*, dann *Action* = *GoEast* (Verwurzelung: 1)

- Wenn *TargetNorthSouth* = *Neither-Nor* ∧ *TargetEastWest* = *EastAhead* ∧ *TargetDistance* = *Far*, dann *Action* = *GoEast* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *SouthAside* ∧ *TargetEastWest* = *EastAside* ∧ *ObstacleNorth* = *False*, dann *Action* = *GoEast* (Verwurzelung: 1)
- Wenn *TargetDistance* = *Middle* ∧ *ObstacleSouth* = *True* ∧ *ObstacleWest* = *False*, dann *Action* = *GoEast* (Verwurzelung: 1)
- Wenn *TargetEastWest* = *EastAhead* ∧ *ObstacleEast* = *False* ∧ *ObstacleSouth* = *True*, dann *Action* = *GoEast* (Verwurzelung: 1)
- Wenn *TargetEastWest* = *EastAhead* ∧ *ObstacleNorth* = *False* ∧ *ObstacleSouth* = *True*, dann *Action* = *GoEast* (Verwurzelung: 1)
- Wenn *TargetEastWest* = *EastAhead* ∧ *TargetDistance* = *Middle* ∧ *ObstacleSouth* = *True*, dann *Action* = *GoEast* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *SouthAside* ∧ *ObstacleEast* = *False* ∧ *ObstacleSouth* = *True*, dann *Action* = *GoEast* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *SouthAside* ∧ *TargetEastWest* = *EastAhead* ∧ *ObstacleSouth* = *True*, dann *Action* = *GoEast* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *SouthAside* ∧ *TargetEastWest* = *EastAhead* ∧ *TargetDistance* = *Middle*, dann *Action* = *GoEast* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *Neither-Nor* ∧ *TargetDistance* = *Middle* ∧ *ObstacleEast* = *False*, dann *Action* = *GoEast* (Verwurzelung: 1)
- Wenn *TargetEastWest* = *EastAside* ∧ *TargetDistance* = *Far* ∧ *ObstacleNorth* = *True*, dann *Action* = *GoEast* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *NorthAhead* ∧ *TargetDistance* = *Far* ∧ *ObstacleNorth* = *True*, dann *Action* = *GoEast* (Verwurzelung: 1)
- Wenn *TargetEastWest* = *NorthAhead* ∧ *TargetDistance* = *Middle* ∧ *ObstacleNorth* = *True*, dann *Action* = *GoEast* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *NorthAhead* ∧ *TargetEastWest* = *Neither-Nor* ∧ *ObstacleNorth* = *True*, dann *Action* = *GoEast* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *NorthAside* ∧ *TargetEastWest* = *EastAside* ∧ *ObstacleNorth* = *False*, dann *Action* = *GoEast* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *NorthAside* ∧ *TargetEastWest* = *EastAside* ∧ *TargetDistance* = *Far*, dann *Action* = *GoEast* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *SouthAside* ∧ *TargetDistance* = *Far*, dann *Action* =

GoSouth (Verwurzelung: 1)

- Wenn *TargetEastWest* = *Neither-Nor* ∧ *TargetDistance* = *Close* ∧ *ObstacleSouth* = *False*, dann *Action* = *GoSouth* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *SouthAhead* ∧ *TargetEastWest* = *Neither-Nor*, dann *Action* = *GoSouth* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *SouthAside* ∧ *ObstacleNorth* = *True* ∧ *ObstacleEast* = *False*, dann *Action* = *GoSouth* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *SouthAside* ∧ *ObstacleNorth* = *True*, dann *Action* = *GoSouth* ∨ *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *SouthAside* ∧ *TargetEastWest* = *EastAside* ∧ *ObstacleNorth* = *True*, dann *Action* = *GoSouth* (Verwurzelung: 1)
- Wenn *TargetEastWest* = *EastAhead* ∧ *ObstacleNorth* = *True* ∧ *ObstacleSouth* = *False*, dann *Action* = *GoSouth* (Verwurzelung: 1)
- Wenn *TargetEastWest* = *EastAhead* ∧ *TargetDistance* = *Far* ∧ *ObstacleNorth* = *True*, dann *Action* = *GoSouth* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *SouthAside* ∧ *TargetEastWest* = *WestAside* ∧ *ObstacleWest* = *False*, dann *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetEastWest* = *WestAside* ∧ *ObstacleNorth* = *False* ∧ *ObstacleEast* = *True*, dann *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *Neither-Nor* ∧ *TargetEastWest* = *WestAhead*, dann *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetEastWest* = *WestAhead* ∧ *ObstacleEast* = *False* ∧ *ObstacleWest* = *False*, dann *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetEastWest* = *WestAhead* ∧ *TargetDistance* = *Close* ∧ *ObstacleWest* = *False*, dann *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *Neither-Nor* ∧ *TargetDistance* = *Close* ∧ *ObstacleWest* = *False*, dann *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *Neither-Nor* ∧ *TargetEastWest* = *EastAhead* ∧ *ObstacleNorth* = *False*, dann *Action* = *GoNorth* ∨ *Action* = *GoEast* (Verwurzelung: 2)
- Wenn *TargetNorthSouth* = *Neither-Nor* ∧ *TargetEastWest* = *EastAhead* ∧ *ObstacleEast* = *False*, dann *Action* = *GoNorth* ∨ *Action* = *GoEast* (Verwurzelung: 1)

- Wenn *TargetNorthSouth* = *Neither-Nor* $\wedge$ *TargetEastWest* = *EastAhead*, dann *Action* = *GoNorth* $\vee$ *Action* = *GoEast* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *NorthAhead* $\wedge$ *TargetEastWest* = *Neither-Nor*, dann *Action* = *GoNorth* $\vee$ *Action* = *GoEast* (Verwurzelung: 1)
- Wenn *TargetEastWest* = *EastAside* $\wedge$ *ObstacleNorth* = *False* $\wedge$ *ObstacleEast* = *False*, dann *Action* = *GoNorth* $\vee$ *Action* = *GoEast* (Verwurzelung: 2)
- Wenn *TargetNorthSouth* = *NorthAhead* $\wedge$ *TargetDistance* = *Far* $\wedge$ *ObstacleEast* = *False*, dann *Action* = *GoNorth* $\vee$ *Action* = *GoEast* (Verwurzelung: 2)
- Wenn *TargetNorthSouth* = *NorthAhead* $\wedge$ *TargetDistance* = *Far* $\wedge$ *ObstacleNorth* = *False*, dann *Action* = *GoNorth* $\vee$ *Action* = *GoEast* (Verwurzelung: 2)
- Wenn *TargetNorthSouth* = *NorthAhead* $\wedge$ *ObstacleNorth* = *False* $\wedge$ *ObstacleEast* = *False*, dann *Action* = *GoNorth* $\vee$ *Action* = *GoEast* (Verwurzelung: 2)
- Wenn *ObstacleSouth* = *False* $\wedge$ *ObstacleWest* = *True*, dann *Action* = *GoNorth* $\vee$ *Action* = *GoEast* (Verwurzelung: 1)
- Wenn *ObstacleSouth* = *True* $\wedge$ *ObstacleWest* = *False*, dann *Action* = *GoNorth* $\vee$ *Action* = *GoEast* (Verwurzelung: 1)
- Wenn *ObstacleEast* = *False* $\wedge$ *ObstacleSouth* = *True*, dann *Action* = *GoNorth* $\vee$ *Action* = *GoEast* (Verwurzelung: 1)
- Wenn *ObstacleNorth* = *False* $\wedge$ *ObstacleSouth* = *True*, dann *Action* = *GoNorth* $\vee$ *Action* = *GoEast* (Verwurzelung: 1)
- Wenn *TargetEastWest* = *EastAside* $\wedge$ *ObstacleNorth* = *False*, dann *Action* = *GoNorth* $\vee$ *Action* = *GoEast* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *SouthAside* $\wedge$ *TargetDistance* = *Far*, dann *Action* = *GoEast* $\vee$ *Action* = *GoSouth* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *SouthAside* $\wedge$ *TargetEastWest* = *EastAside* $\wedge$ *ObstacleSouth* = *False*, dann *Action* = *GoEast* $\vee$ *Action* = *GoSouth* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *SouthAside* $\wedge$ *TargetEastWest* = *EastAside* $\wedge$ *ObstacleEast* = *False*, dann *Action* = *GoEast* $\vee$ *Action* = *GoSouth* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *SouthAside* $\wedge$ *TargetEastWest* = *EastAhead* $\wedge$ *ObstacleEast* = *False*, dann *Action* = *GoEast* $\vee$ *Action* = *GoSouth* (Verwurzelung: 1)

- Wenn *TargetEastWest* = *EastAside* ∧ *TargetDistance* = *Middle*, dann *Action* = *GoEast* ∨ *Action* = *GoSouth* (Verwurzelung: 1)
- Wenn *TargetEastWest* = *Neither-Nor* ∧ *ObstacleNorth* = *True* ∧ *ObstacleSouth* = *False*, dann *Action* = *GoEast* ∨ *Action* = *GoSouth* (Verwurzelung: 1)
- Wenn *TargetEastWest* = *Neither-Nor* ∧ *ObstacleNorth* = *True* ∧ *ObstacleEast* = *False*, dann *Action* = *GoEast* ∨ *Action* = *GoSouth* (Verwurzelung: 1)
- Wenn *TargetEastWest* = *Neither-Nor* ∧ *TargetDistance* = *Middle* ∧ *ObstacleNorth* = *True*, dann *Action* = *GoEast* ∨ *Action* = *GoSouth* (Verwurzelung: 1)
- Wenn *TargetDistance* = *Far* ∧ *ObstacleNorth* = *True*, dann *Action* = *GoEast* ∨ *Action* = *GoSouth* (Verwurzelung: 1)
- Wenn *TargetDistance* = *Close* ∧ *ObstacleSouth* = *False* ∧ *ObstacleWest* = *False*, dann *Action* = *GoSouth* ∨ *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *SouthAside* ∧ *ObstacleNorth* = *True*, dann *Action* = *GoSouth* ∨ *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *SouthAside* ∧ *TargetDistance* = *Middle* ∧ *ObstacleEast* = *True*, dann *Action* = *GoSouth* ∨ *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *SouthAside* ∧ *TargetEastWest* = *WestAside* ∧ *TargetDistance* = *Middle*, dann *Action* = *GoSouth* ∨ *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetEastWest* = *WestAside* ∧ *ObstacleNorth* = *True* ∧ *ObstacleWest* = *False*, dann *Action* = *GoSouth* ∨ *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *SouthAhead* ∧ *TargetEastWest* = *WestAside* ∧ *TargetDistance* = *Middle*, dann *Action* = *GoSouth* ∨ *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetEastWest* = *WestAhead* ∧ *ObsatcleWest* = *False*, dann *Action* = *GoSouth* ∨ *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetEastWest* = *EastAhead* ∧ *TargetDistance* = *Middle* ∧ *ObstacleEast* = *True*, dann *Action* = *GoNorth* ∨ *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *Neither-Nor* ∧ *ObstacleEast* = *True* ∧ *ObstacleWest* = *False*, dann *Action* = *GoNorth* ∨ *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *Neither-Nor* ∧ *TargetDistance* = *Middle* ∧ *ObstacleEast* = *True*, dann *Action* = *GoNorth* ∨ *Action* = *GoWest* (Verwurzelung: 1)

- Wenn *TargetEastWest* = *WestAside* ∧ *ObstacleNorth* = *False*, dann *Action* = *GoNorth* ∨ *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetEastWest* = *WestAside* ∧ *TargetDistance* = *Middle* ∧ *ObstacleNorth* = *False*, dann *Action* = *GoNorth* ∨ *Action* = *GoWest* (Verwurzelung: 2)
- Wenn *TargetEastWest* = *WestAside* ∧ *TargetDistance* = *Middle* ∧ *ObstacleNorth* = *False*, dann *Action* = *GoNorth* ∨ *Action* = *GoWest* (Verwurzelung: 2)
- Wenn *TargetNorthSouth* = *NorthAhead* ∧ *TargetEastWest* = *WestAside* ∧ *ObstacleNorth* = *False*, dann *Action* = *GoNorth* ∨ *Action* = *GoWest* (Verwurzelung: 2)
- Wenn *TargetNorthSouth* = *NorthAhead* ∧ *TargetEastWest* = *WestAside*, dann *Action* = *GoNorth* ∨ *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *NorthAhead* ∧ *TargetEastWest* = *WestAside* ∧ *ObstacleWest* = *False*, dann *Action* = *GoNorth* ∨ *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *NorthAhead* ∧ *TargetEastWest* = *WestAside* ∧ *ObstacleEast* = *False*, dann *Action* = *GoNorth* ∨ *Action* = *GoWest* (Verwurzelung: 2)
- Wenn *TargetNorthSouth* = *SouthAside* ∧ *TargetDistance* = *Middle* ∧ *ObstacleNorth* = *False*, dann *Action* = *GoEast* ∨ *Action* = *GoWest* (Verwurzelung: 2)
- Wenn *TargetNorthSouth* = *Neither-Nor* ∧ *ObstacleEast* = *False*, dann *Action* = *GoEast* ∨ *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *Neither-Nor* ∧ *TargetDistance* = *Close* ∧ *ObstacleSouth* = *False*, dann *Action* = *GoEast* ∨ *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *Neither-Nor* ∧ *TargetDistance* = *Close* ∧ *ObstacleNorth* = *False*, dann *Action* = *GoEast* ∨ *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetNorthSouth* = *NorthAhead* ∧ *TargetDistance* = *Middle* ∧ *ObstacleNorth* = *True*, dann *Action* = *GoEast* ∨ *Action* = *GoWest* (Verwurzelung: 1)
- Wenn *TargetEastWest* = *Neither-Nor* ∧ *ObstacleNorth* = *True*, dann *Action* = *GoNorth* ∨ *Action* = *GoSouth* (Verwurzelung: 1)

(Bemerkung: Es wurden auch Regeln, welche Feldfarben beinhalten, erstellt, hier

jedoch nicht weiter beachtet.)

A.1.2 Erkennung geometrischer Objekte

Regeln mit Information über eine schlechte Aktion:

- Wenn *SideView* = *Unknown* $\wedge$ *TopView* = *Unknown*, dann $\overline{Action = RecognizeTetrahedron}$ (Verwurzelung: 4)
- Wenn *FrontView* = *Unknown* $\wedge$ *TopView* = *Unknown*, dann $\overline{Action = RecognizeTetrahedron}$ (Verwurzelung: 2)
- Wenn *FrontView* = *Unknown* $\wedge$ *SideView* = *Unknown*, dann $\overline{Action = RecognizeTetrahedron}$ (Verwurzelung: 1)
- Wenn *FrontView* = *Unknown*, dann $\overline{Action = RecognizeTetrahedron}$ (Verwurzelung: 1)
- Wenn *SideView* = *Unknown* $\wedge$ *TopView* = *Unknown*, dann $\overline{Action = RecognizeCone}$ (Verwurzelung: 4)
- Wenn *SideView* = *Unknown* $\wedge$ *TopView* = *Unknown*, dann $\overline{Action = RecognizePyramid}$ (Verwurzelung: 4)
- Wenn *FrontView* = *Triangle* $\wedge$ *SideView* = *Unknown*, dann $\overline{Action = RecognizePyramid}$ (Verwurzelung: 1)
- Wenn *SideView* = *Rectangle* $\wedge$ *TopView* = *Unknown*, dann $\overline{Action = RecognizeCube}$ (Verwurzelung: 1)
- Wenn *FrontView* = *Triangle* $\wedge$ *TopView* = *Unknown*, dann $\overline{Action = RecognizeCube}$ (Verwurzelung: 3)
- Wenn *FrontView* = *Triangle*, dann $\overline{Action = RecognizeCube}$ (Verwurzelung: 1)
- Wenn *FrontView* = *Triangle*, dann $\overline{Action = RecognizeCuboid}$ (Verwurzelung: 1)
- Wenn *SideView* = *Unknown* $\wedge$ *TopView* = *Circle*, dann $\overline{Action = RecognizeSphere}$ (Verwurzelung: 2)
- Wenn *FrontView* = *Rectangle* $\wedge$ *TopView* = *Circle*, dann $\overline{Action = RecognizeSphere}$ (Verwurzelung: 2)
- Wenn *FrontView* = *Rectangle* $\wedge$ *SideView* = *Unknown*, dann $\overline{Action = RecognizeSphere}$ (Verwurzelung: 3)

- Wenn *FrontView* = *Rectangle*, dann $\overline{Action = RecognizeSphere}$ (Verwurzelung: 1)
- Wenn *FrontView* = *Triangle* $\wedge$ *SideView* = *Unknown*, dann $\overline{Action = RecognizePrism}$ (Verwurzelung: 1)

Regeln mit Information über beste Aktionen:

- Wenn *FrontView* = *Circle*, dann *Action* = *RecognizeSphere* (Verwurzelung: 3)
- Wenn *FrontView* = *Circle* $\wedge$ *TopView* = *Unknown*, dann *Action* = *RecognizeSphere* (Verwurzelung: 6)
- Wenn *FrontView* = *Circle* $\wedge$ *SideView* = *Unknown*, dann *Action* = *RecognizeSphere* (Verwurzelung: 6)
- Wenn *FrontView* = *Circle* $\wedge$ *SideView* = *Circle*, dann *Action* = *RecognizeSphere* (Verwurzelung: 2)
- Wenn *SideView* = *Circle* $\wedge$ *TopView* = *Circle*, dann *Action* = *RecognizeSphere* (Verwurzelung: 1)
- Wenn *FrontView* = *Ellipse*, dann *Action* = *RecognizeEllipsoid* (Verwurzelung: 3)
- Wenn *TopView* = *Ellipse*, dann *Action* = *RecognizeEllipsoid* (Verwurzelung: 1)
- Wenn *FrontView* = *Rectangle* $\wedge$ *TopView* = *Circle*, dann *Action* = *RecognizeCylinder* (Verwurzelung: 5)
- Wenn *FrontView* = *Triangle* $\wedge$ *TopView* = *Circle*, dann *Action* = *RecognizeCone* (Verwurzelung: 5)
- Wenn *SideView* = *Triangle* $\wedge$ *TopView* = *Circle*, dann *Action* = *RecognizeCone* (Verwurzelung: 5)
- Wenn *TopView* = *Triangle*, dann *Action* = *RecognizeTetrahedron* (Verwurzelung: 2)
- Wenn *FrontView* = *Triangle* $\wedge$ *TopView* = *Triangle*, dann *Action* = *RecognizeTetrahedron* (Verwurzelung: 5)
- Wenn *SideView* = *Triangle* $\wedge$ *TopView* = *Triangle*, dann *Action* = *RecognizeTetrahedron* (Verwurzelung: 5)
- Wenn *FrontView* = *Triangle* $\wedge$ *TopView* = *Square*, dann *Action* = *RecognizePyramid* (Verwurzelung: 5)
- Wenn *SideView* = *Triangle* $\wedge$ *TopView* = *Square*, dann *Action* = *RecognizePyramid* (Verwurzelung: 5)
- Wenn *FrontView* = *Triangle* $\wedge$ *SideView* = *Rectangle*, dann *Action* = *RecognizePrism* (Verwurzelung: 5)

- Wenn *FrontView* = *Triangle* $\wedge$ *TopView* = *Rectangle*, dann *Action* = *RecognizePrism* (Verwurzelung: 2)
- Wenn *FrontView* = *Square*, dann *Action* = *RecognizeCube* (Verwurzelung: 3)
- Wenn *FrontView* = *Square* $\wedge$ *SideView* = *Unknown*, dann *Action* = *RecognizeCube* (Verwurzelung: 6)
- Wenn *FrontView* = *Square* $\wedge$ *TopView* = *Unknown*, dann *Action* = *RecognizeCube* (Verwurzelung: 6)
- Wenn *SideView* = *Square*, dann *Action* = *RecognizeCube* (Verwurzelung: 1)
- Wenn *SideView* = *Square* $\wedge$ *TopView* = *Unknown*, dann *Action* = *RecognizeCube* (Verwurzelung: 3)
- Wenn *FrontView* = *Rectangle* $\wedge$ *TopView* = *Rectangle*, dann *Action* = *RecognizeCube* (Verwurzelung: 5)
- Wenn *FrontView* = *Rectangle* $\wedge$ *SideView* = *Rectangle*, dann *Action* = *RecognizeCube* (Verwurzelung: 2)
- Wenn *FrontView* = *Unknown* $\wedge$ *SideView* = *Triangle*, dann *Action* = *LookAtFront* (Verwurzelung: 3)
- Wenn *FrontView* = *Rectangle* $\wedge$ *SideView* = *Unknown*, dann *Action* = *LookAtSide* (Verwurzelung: 4)
- Wenn *FrontView* = *Unknown* $\wedge$ *SideView* = *Triangle*, dann *Action* = *LookAtFront* (Verwurzelung: 4)
- Wenn *FrontView* = *Unknown* $\wedge$ *TopView* = *Circle*, dann *Action* = *LookAtFront* (Verwurzelung: 4)

A.1.3 Erkennung simulierter Objekte

Regeln mit Information über eine schlechte Aktion:

- Wenn *Texture* = *Complex*, dann $\overline{Action = RecognizeBottle}$ (Verwurzelung: 1)
- Wenn *FrontViewShape* = *Circle*, dann $\overline{Action = RecognizeBottle}$ (Verwurzelung: 1)
- Wenn *Texture* = *Middle*, dann $\overline{Action = RecognizeTetraPack}$ (Verwurzelung: 1)
- Wenn *RightViewShape* = *Circle*, dann $\overline{Action = RecognizeTetraPack}$ (Verwurzelung: 1)
- Wenn *FrontViewSize* = *Normal*, dann $\overline{Action = RecognizeTetraPack}$ (Verwurzelung: 1)

- Wenn *Texture* = *Simple*, dann $\overline{Action = RecognizeHouse}$ (Verwurzelung: 2)
- Wenn *FrontViewSize* = *Flat*, dann $\overline{Action = RecognizeHouse}$ (Verwurzelung: 1)
- Wenn *RightViewShape* = *Square* $\wedge$ *Texture* = *Simple*, dann $\overline{Action = RecognizeHouse}$ (Verwurzelung: 2)
- Wenn *RightViewShape* = *Square* $\wedge$ *Texture* = *Medium*, dann $\overline{Action = RecognizeBottle}$ (Verwurzelung: 1)
- Wenn *Texture* = *Medium*, dann $\overline{Action = RecognizePott}$ (Verwurzelung: 1)
- Wenn *RightViewDeviation* = *Unknown*, dann $\overline{Action = RecognizeFootball}$ (Verwurzelung: 2)
- Wenn *FrontViewSize* = *Tall*, dann $\overline{Action = RecognizeFootball}$ (Verwurzelung: 1)

Regeln mit Information über beste Aktionen:

- Wenn *FrontViewSize* = *Tall* $\wedge$ *Texture* = *Medium*, dann *Action* = *RecognizeBottle* (Verwurzelung: 2)
- Wenn *FrontViewSize* = *Tall* $\wedge$ *FrontViewDeviation* = *Medium*, dann *Action* = *RecognizeBottle* (Verwurzelung: 2)
- Wenn *FrontViewShape* = *TriangleUp* $\wedge$ *FrontViewSize* = *Tall*, dann *Action* = *RecognizeBottle* (Verwurzelung: 2)
- Wenn *FrontViewDeviation* = *Medium* $\wedge$ *RightViewShape* = *TriangleUp*, dann *Action* = *RecognizeBottle* (Verwurzelung: 2)
- Wenn *FrontViewSize* = *Tall* $\wedge$ *RightViewShape* = *TriangleUp*, dann *Action* = *RecognizeBottle* (Verwurzelung: 2)
- Wenn *RightViewSize* = *Tall* $\wedge$ *Texture* = *Medium*, dann *Action* = *RecognizeBottle* (Verwurzelung: 2)
- Wenn *FrontViewShape* = *TriangleUp* $\wedge$ *FrontViewDeviation* = *Little*, dann *Action* = *RecognizeTree* (Verwurzelung: 2)
- Wenn *FrontViewDeviation* = *Much*, dann *Action* = *RecognizeTree* (Verwurzelung: 1)
- Wenn *FrontViewShape* = *TriangleDown*, dann *Action* = *RecognizeTree* (Verwurzelung: 1)
- Wenn *RightViewSize* = *Tall* $\wedge$ *Texture* = *Complex*, dann *Action* = *RecognizeTetraPack* (Verwurzelung: 1)
- Wenn *RightViewShape* = *Square* $\wedge$ *Texture* = *Complex*, dann *Action* =

RecognizeTetraPack (Verwurzelung: 2)

- Wenn *FrontViewShape* = *Square* ∧ *Texture* = *Complex*, dann *Action* = *RecognizeTetraPack* (Verwurzelung: 2)
- Wenn *FrontViewSize* = *Normal* ∧ *RightViewDeviation* = *Little* ∧ *Texture* = *Medium*, dann *Action* = *RecognizeHouse* (Verwurzelung: 1)
- Wenn *RightViewShape* = *Square* ∧ *Texture* = *Medium*, dann *Action* = *RecognizeHouse* (Verwurzelung: 2)
- Wenn *FrontViewDeviation* = *Medium* ∧ *RightViewShape* = *Square*, dann *Action* = *RecognizeHouse* (Verwurzelung: 2)
- Wenn *RightViewShape* = *TriangleUp* ∧ *RightViewShape* = *Square*, dann *Action* = *RecognizeHouse* (Verwurzelung: 2)
- Wenn *RightViewSize* = *Normal* ∧ *RightViewDeviation* = *Medium*, dann *Action* = *RecognizeHouse* (Verwurzelung: 2)
- Wenn *FrontViewShape* = *Square* ∧ *RightViewSize* = *Normal*, dann *Action* = *RecognizePott* (Verwurzelung: 1)
- Wenn *FrontViewShape* = *Square* ∧ *FrontViewSize* = *Normal* ∧ *Texture* = *Simple*, dann *Action* = *RecognizePott* (Verwurzelung: 4)
- Wenn *FrontViewShape* = *Circle* ∧ *RightViewShape* = *Square*, dann *Action* = *RecognizeDVD* (Verwurzelung: 2)
- Wenn *FrontViewDeviation* = *Little* ∧ *RightViewSize* = *Tall*, dann *Action* = *RecognizeDVD* (Verwurzelung: 1)
- Wenn *FrontViewSize* = *Normal* ∧ *RightViewSize* = *Tall*, dann *Action* = *RecognizeDVD* (Verwurzelung: 1)
- Wenn *FrontViewDeviation* = *Medium* ∧ *Texture* = *Simple*, dann *Action* = *RecognizeBird* (Verwurzelung: 2)
- Wenn *RightViewSize* = *Flat* ∧ *Texture* = *Simple*, dann *Action* = *RecognizeBird* (Verwurzelung: 2)
- Wenn *RightViewShape* = *Square* ∧ *RightViewSize* = *Flat*, dann *Action* = *RecognizeBird* (Verwurzelung: 2)
- Wenn *FrontViewSize* = *Flat*, dann *Action* = *RecognizeBird* (Verwurzelung: 1)
- Wenn *FrontViewShape* = *Square* ∧ *FrontViewSize* = *Flat*, dann *Action* = *RecognizeBird* (Verwurzelung: 2)
- Wenn *FrontViewSize* = *Flat* ∧ *Texture* = *Simple*, dann *Action* = *RecognizeBird*

(Verwurzelung: 2)

- Wenn *RightViewSize* = *Flat* ∧ *Texture* = *Medium*, dann *Action* = *RecognizeFootball* (Verwurzelung: 2)
- Wenn *RightViewShape* = *Circle* ∧ *RightViewSize* = *Flat*, dann *Action* = *RecognizeFootball* (Verwurzelung: 2)
- Wenn *RightViewSize* = *Flat* ∧ *Texture* = *Medium*, dann *Action* = *RecognizeFootball* (Verwurzelung: 2)
- Wenn *RightViewShape* = *Circle* ∧ *Texture* = *Complex*, dann *Action* = *RecognizeBall* (Verwurzelung: 2)
- Wenn *FrontViewShape* = *Circle* ∧ *Texture* = *Complex*, dann *Action* = *RecognizeBall* (Verwurzelung: 2)
- Wenn *RightViewShape* = *Circle* ∧ *RightViewSize* = *Normal*, dann *Action* = *RecognizeBall* (Verwurzelung: 2)
- Wenn *RightViewShape* = *Circle* ∧ *Texture* = *Simple*, dann *Action* = *RecognizeBall* (Verwurzelung: 2)
- Wenn *FrontViewSize* = *Normal* ∧ *Texture* = *Complex*, dann *Action* = *RecognizeBall* (Verwurzelung: 2)
- Wenn *FrontViewDeviation* = *Little* ∧ *RightViewShape* = *Circle* ∧ *Texture* = *Medium*, dann *Action* = *RotateLeft* (Verwurzelung: 2)
- Wenn *FrontViewSize* = *Normal* ∧ *RightViewShape* = *Circle* ∧ *Texture* = *Medium*, dann *Action* = *RotateLeft* (Verwurzelung: 2)
- Wenn *FrontViewShape* = *Circle* ∧ *FrontViewDeviation* = *Little* ∧ *Texture* = *Medium*, dann *Action* = *RotateLeft* (Verwurzelung: 3)
- Wenn *FrontViewShape* = *Circle* ∧ *FrontViewSize* = *Normal* ∧ *Texture* = *Medium*, dann *Action* = *RotateLeft* (Verwurzelung: 3)
- Wenn *FrontViewShape* = *Circle* ∧ *RightViewShape* = *Unknown* ∧ *Texture* = *Simple*, dann *Action* = *RotateLeft* (Verwurzelung: 4)
- Wenn *FrontViewShape* = *Circle* ∧ *FrontViewDeviation* = *Unknown* ∧ *Texture* = *Medium*, dann *Action* = *RotateRight* (Verwurzelung: 4)
- Wenn *FrontViewShape* = *Circle* ∧ *FrontViewSize* = *Unknown* ∧ *Texture* = *Medium*, dann *Action* = *RotateRight* (Verwurzelung: 4)

Literaturverzeichnis

[AGM1985]: Alchourrón, Carlos, Gärdenfors, Peter und Makinson, David (1985). *On the logic of theory change: Partial meet contraction and revision functions* in: *Journal of Symbolic Logic, Vol. 50, No. 2, pp. 510 - 530*.

[And1983]: Anderson, John R. (1983). *The architecture of cognition*. Harvard University Press, Cambridge, MA.

[Bro1999]: Brokate, Martin (1999). *Analysis I*. http://www-m6.ma.tum.de/~brokate/an1_ws99.pdf (Zugriff vom: 2006-07-05).

[BuK2006]: Beierle, Christoph und Kern-Isberner, Gabriele (2006). *Methoden wissensbasierter Systeme, 3rd edition*. Vieweg-Verlag, Wiesbaden, Germany.

[CuD1998]: Clayton, Nicola S. und Dickinson, Anthony (1998). *Episodic-like memory during cache recovery by scrub-jays* in: *Nature, Vol. 395, pp. 272 - 274*. Nature Publishing, London, Großbritannien. http://www.indiana.edu/~animal/speakers/pdfs/clayton_dickenson_1998.pdf (Zugriff vom: 2007-01-17)

[Dal1988]: Dalal, Mukesh (1988). *Investigations Into a Theory of Knowledge Base Revision: Preliminary Report* in: *Proceedings of the Seventh National Conference on Artificial Intelligence, pp. 475 - 479*. ftp://ftp.cs.columbia.edu/pub/dalal/papers/aaai88.ps.gz (Zugriff vom: 2006-12-21).

[Day1992]: Dayan, Peter (1992). *The Convergence of TD(λ) for General λ* in: *Machne Learning, Vol. 8, 1992, pp. 341 - 362*. Springer Netherlands.

[DuP1996]: Darwiche, Adnan und Pearl, Judea (1996). *On the Logic of Iterated Belief Revision* in: *Artificial Intelligence, Vol 89, pp. 1 - 29*. http://ftp.cs.ucla.edu/pub/stat_ser/R202.ps (Zugriff vom: 2006-12-20).

[DuS1994]: Dayan, Peter und Sejnowski, Terry (1994). *TD(λ) Converges with Probability 1* in: *Machine Learning, Vol. 14, pp. 295 - 301*. Springer Netherlands. http://www.gatsby.ucl.ac.uk/~dayan/papers/ds94.pdf

(Zugriff vom: 2006-07-11).

[DuW2001]: Dayan, Peter und Watkins, Christopher J. C. H. (2001). *Reinforcement Learning* in: *Encyclopedia of Cognitive Science.* MacMillan Press. London, Großbritannien. http://www.gatsby.ucl.ac.uk/~dayan/papers/dw01.pdf (Zugriff vom: 2006-07-05).

[Fel1968]: Feller, W. (1968). *Laws of Large Numbers* in: *An Introduction to Probability Theory and Its Applications, Vol. 1, 3rd edition, pp. 243 - 248*. John Wiley and Sons, Inc., New York.

[GuM1988]: Gärdenfors, Peter und Makinson, David (1988). *Revisions of knowledge systems using epistemic entrenchment* in: *Proceedings of the Second Conference on Theoretical Aspects of Reasoning about Knowledge, pp. 83 - 95*. Morgan Kaufmann, California.

[GuP1996]: Goldszmidt, Moisés und Pearl, Judea. *Qualitative Probabilities for Default Reasoning, Belief Revision, and Causal Modeling* in: *Artificial Intelligence, Vol. 84, No. 1 - 2, pp. 57 - 112*. http://ftp.cs.ucla.edu/pub/stat_ser/R161-L.pdf (Zugriff vom: 2007-04-21).

[GuR1995]: Gärdenfors, Peter und Rott, Hans (1995). *Belief revision* in: Gabbay, D. M., Hogger, C. J. und Robinson, J. A.: *Handbook of Logic in Artificial Intelligence and Logic Programming, Vol. 4, pp. 35 - 132*. Clarendon Press, Oxford.

[Han1991]: Hansson, Sven Ove (1991). *Belief contraction without recovery* in: *Studia Logica, Vol. 50, pp. 251 – 260*.

[Han1996]: Hansson, Sven Ove (1996). *A Textbook of Belief Dymanics: Theory Change and Database Updating*. Uppsala University, Department of Philosophy, Uppsala, Sweden.

[Har1977]: Harper, William L. (1977). *Rational conceptual change* in: *Proceedings of the 1976 Meeting of the Philosophy of Science Association, Vol. 2, pp. 462 - 494*. University of Chicago Press,

Chicago.

[HuH2000]: Harmon, Mance E. und Harmon, Stephanie S. (2000). *Reinforcement Learning: A Tutorial*. http://www.nbu.bg/cogs/events/2000/Readings/Petrov/rltutorial.pdf (Zugriff vom 2006-07-06).

[JJS1994]: Jaakkola, Tommi, Jordan, Michael I. und Singh, Satinder P. (1994). *On the Convergence of Stochastic Iterative Dynamic Programming Algorithms* in: *Neural Computation, Vol. 6, No. 6, 1994*. The MIT Press. http://people.csail.mit.edu/tommi/papers/convergence.ps (Zugriff vom: 2006-07-05).

[K-I2004]: Kern-Isberner, Gabriele (2004/ 2005). *Darstellung, Verarbeitung und Erwerb von Wissen.*

[K-I2005]: Kern-Isberner, Gabriele (2005). *Wissensdynamik und Informationsfusion.*

[KLM1996]: Kaelbling, Leslie Pack, Littman, Michael L. und Moore, Andrew W. (1996). *Reinforcement Learning: A Survey* in: *Journal of Artificial Intelligence Research 4 (1996)*. AAAI Press. http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume4/kaelbling96a.pdf (Zugriff vom: 2006-07-06).

[KuM1991]: Katsuno, Hirofumi und Medelzon, Alberto (1991). *Propositional knowledge base revision and minimal change* in: *Artificial Intelligence, Vol. 52, No. 3, pp. 263 - 294*.

[Lev1977.1]: Levi, Isaac (1977). *Subjunctives, dispositions and chances* in: *Synthese, Vol. 34, pp. 423 - 455*.

[Lev1977.2]: Levi, Isaac (1977). *Direct Inference* in: *The Journal of Philosophy, Vol. 74, pp. 5 - 29*.

[PJJ2006]: Pereira, Alfredo, James, Kathrin H., Jones, Susan S. und Smith, Linda B. (2006). *Preferred Views in Children's Active Exploration of Objects* at: *International Conference on Development and Learning*

2006. Bloomington, Indiana.

[Reb1989]: Reber, Arthur S. (1989). *Implicit learning and tacit knowledge* in: *Journal of Experimental Psychology: General, Vol. 118, pp. 219 - 235*.

[Reb1993]: Reber, Arthur S. (1993). *Implicit learning and tacit knowledge: An Essay on the Cognitive Unconscious*. Oxford University Press, Oxford, Großbritannien.

[Rei1980]: Reiter, Raymond (1980). *A Logic for Default Reasoning* in: *Artificial Intelligence, Vol. 13, No. 1 - 2, pp. 81 - 132*.

[Rie2002]: Riedmiller, Martin (2002/ 2003). *Einführung in die Künstliche Intelligenz.*

[Rie2003]: Riedmiller, Martin (2003). *Optimierendes Lernen (Reinforcement Learning) – Adaptive Verfahren für dynamische Optimierungsprobleme.*

[RuN2003]: Russell, Stuart J. und Norvig, Peter (2003). *Artificial Intelligence: A Modern Approach (2nd Edition).* Prentice Hall.

[Sam1959]: Samuel, Arthur L. (1959). *Some studies in machine learning using the game of checkers* in: *IBM Journal of Research and Development, Vol. 3, No. 3, pp. 210 - 219.* http://www.research.ibm.com/journal/rd/441/samuel.pdf (Zugriff vom: 2006-12-15).

[SMB1989]: Stanley, William B., Mathews, Robert C., Buss, Ray R. und Kotler-Cope, Susan (1989). *Insight without awareness: on the interaction of verbalization, instruction and practice in a simulated process control task* in: *Quarterly Journal of Experimental Psychology, Vol. 41A, No. 3, pp. 553 - 577*.

[SMP2001]: Sun, Ron, Merrill, Edward und Peterson, Todd (2001). *From implicit skills to explicit knowledge: a bottom-up model of skill learning* in: *Cognitive Science, Vol. 25, No. 2, pp. 203 - 244*.

http://www.cogsci.rpi.edu/~rsun/sun.cs01.pdf (Zugriff vom: 2007-01-17).

[SPM1996] Sun, Ron, Peterson, Todd und Merrill, Edward (1996). *Bottom-up skill learning in reactive sequential decision tasks* in: *Proceedings of 18th Cognitive Science Society Conference, pp. 684 - 690*. Lawrence Erlbaum Associates, Hillsdale, NJ. http://www.cogsci.rpi.edu/~rsun/sun.cog96.ps (Zugriff vom: 2007-01-17).

[Spo1988]: Spohn, Wolfgang (1988). *Ordinal conditional functions: A dynamic theory of epistemic states* in: Harper, William L. und Skyrms, Brian: *Causation in Decision: Belief, Change and Statistics: Proceedings of the Irvine Conference on Probability and Causation, Vol. 2, pp. 105 - 134*. Reidel, Dordrecht, Netherlands.

[SuB1998]: Sutton, Richard S. und Barto, Andrew G. (1998). *Reinforcement Learning: An Introduction.* Bradford Book, The MIT Press.

[Sun1997]: Sun, Ron (1997). *Learning, action, and consciousness: A hybrid approach towards modeling consciousness* in: *Neural Networks, special issue on consciousness, Vol. 10, No. 7, pp.1317 - 1331*. http://www.cogsci.rpi.edu/~rsun/sun.nn97.pdf (Zugriff vom: 2007-01-17).

[SuP1997]: Sun, Ron und Peterson, Todd (1997). *A hybrid model for learning sequential navigation* in: *Proceedings of IEEE International Symposium on Computational Intelligence in Robotics and Automation*. Monterey, CA. IEEE Press. http://www.cogsci.rpi.edu/~rsun/sun.cira97.ps (Zugriff vom: 2007-01-17).

[SuP1998]: Sun, Ron und Peterson, Todd (1998). *Some experiments with a hybrid model for learning sequential decision making* in: *Information Sciences, Vol. 111, pp. 83 - 107*. http://www.cogsci.rpi.edu/~rsun/sun.info-sci2.ps (Zugriff vom: 2007-01-17).

[Wat1989]: Watkins, Christopher J. C. H. (1989). *Learning from Delayed Rewards*. Ph. D. Thesis, Cambridge University, Cambridge, Großbritannien.

[Whe2004]: Wheeler, Gregory R. (2004). *Belief Revision and Belief Updating*. http://ssdi.di.fct.unl.pt/mei/taia/modulos.html#brbu (Zugriff vom: 2006-12-21).

[WNB1989]: Willingham, D., Nissen, M., und Bullemer, P. (1989). *On the development of procedural knowledge* in: *Journal of Experimental Psychology: Learning, Memory, and Cognition, Vol. 15, pp. 1047 - 1060*.

[WuD1992]: Watkins, Christopher J. C. H. und Dayan, Peter (1992). *Q-Learning* in *Machine Learning, Vol. 8, pp. 279 - 292*.