

Logical Agents

Hendrik Thüs

22. Juni 2005

Über das 7. Kapitel des Buches „Artificial Intelligence - A modern approach“ von
Stuart Russel und Peter Norvig.

Inhaltsverzeichnis

1	Einleitung	3
2	Wissensbasierte Agenten	3
3	Die Aussagenlogik	7
3.1	Inferenz	8
3.2	Logische Äquivalenz	9
4	Inferenzschemata der Aussagenlogik	9
4.1	Resolution	10
5	Effektive aussagenlogische Inferenz	12
6	Agenten auf der Basis der Aussagenlogik	14
6.1	Schaltungsbasierte Agenten	15
7	Zusammenfassung	16

1 Einleitung

Logische Agenten sind Agenten, die die Welt repräsentieren können und Anhand von Schlussfolgerungen versuchen, neue Erkenntnisse über ihre Welt zu ziehen, damit sie wissen, was zu tun ist.

Es werden hier zwei zentrale Konzepte der künstlichen Intelligenz vorgestellt, *Wissensrepräsentation* und *Schlussfolgerungsprozesse*. Für ein erfolgreiches Verhalten von künstlichen Agenten sind diese beiden die zentralen Konzepte der KI. Wissen und Schließen ermöglicht den künstlichen Agenten ein erfolgreiches Verhalten, das auf andere Weise nur schwer zu realisieren wäre. Das Wissen problem-lösender Agenten ist jedoch nur auf bestimmte Probleme zugeschnitten und reicht nicht darüber hinaus. Das in einer allgemeinen Form ausgedrückte Wissen eines Agenten kann von ihm immer wieder kombiniert werden um Aufgaben zu lösen.

Dieses Wissen und Schließen ist auch im Umgang mit *partiell beobachtbaren* Umgebungen, in denen der Agent nur einen Teil seiner Umgebung betrachten kann, sehr wichtig. Der Agent entscheidet anhand von Regeln und von ihm vielleicht überhaupt nicht beschreibbaren Assoziationsmustern.

Diese Muster kommen zum Beispiel auch bei der natürlichen Sprache zum Einsatz: Bei einem Satz, in dem das Wort „es“ vorkommt, wissen wir, worauf sich dieses Wort bezieht. Durch das Schließen haben wir die Möglichkeit, trotz eines beschränkten Wissens, mit einer fast unendlichen Menge an Äußerungen zurecht zu kommen. Problemlösende Agenten haben hierbei Probleme.

2 Wissensbasierte Agenten

Ein Reflex-Agent nimmt über Sensoren seine Umgebung wahr und entscheidet anhand von einfachen Regeln, was nun zu tun ist. Diese Aktion hat nun wieder Auswirkungen auf seine Umwelt und der Prozess fängt von vorne an.

Modellbasierte Reflex-Agenten haben zusätzlich noch einen internen Zustand, der ihnen sagt, wie die Welt sich entwickelt hat, sie haben also ein *Modell der Welt*. In diesem Modell werden die Wahrnehmungen gespeichert, die der Agent über seine Sensoren aufnimmt. Was fehlt diesen beiden Agententypen nun noch?

Das wichtigste an einem wissensbasierten Agenten ist seine Wissensbasis (engl. *Knowledge Base, KB*), die dem Agenten Informationen über seine Umgebung liefert. Es muss dem Agenten möglich sein, der Wissensbasis Informationen hinzuzufügen und welche abzufragen. Bei beiden Operationen ist das Schließen (Inferenz) möglich, das bei logischen Agenten einem Grundsatz gehorchen muss: Eine Antwort kann nur aus dem folgen, was zu dem Zeitpunkt als Wissen in der Wissensbasis des Agenten vorhanden ist.

Ein einfaches Agenten-Programm nimmt Wahrnehmungen als Eingabe entgegen und gibt eine Aktion zurück. Der Agent gibt die Wahrnehmung an die Wissensbasis weiter und erhält dann die darauf folgende Aktion. Damit die Wissensbasis weiß, dass diese Aktion auch wirklich ausgeführt wurde, teilt der Agent ihr dieses

mit.

Einem solchen Agenten muss nur gesagt werden, was er wissen soll und was er für ein Ziel anstreben soll um sein Verhalten festzulegen. Ein Beispiel ist:

Ein automatisches Taxi soll einen Fahrgast nach Marin County bringen. Es weiß, dass dies in San Francisco ist und dass die Golden Gate Bridge die einzige Verbindung zwischen den zwei Orten ist. Wir können dann erwarten, dass das Taxi über die Golden Gate Bridge fährt, weil es weiß, dass es so ans Ziel kommt.

Um das anfängliche Hintergrundwissen des Agenten aufzubauen, wird ihm nach und nach Wissen über die Umgebung in Form von Sätzen gegeben. Hierfür werden im folgenden zwei Ansätze vorgestellt, der **deklarative** und der **prozedurale** Ansatz. Ein erfolgreicher Agent sollte beide Ansätze verfolgen.

Um einen Agenten zu testen wird eine Testumgebung entworfen. Die **Wumpus-Welt** ist eine solche Umgebung. Sie besteht aus Räumen, die wie in einer Matrix angeordnet sind. Die Räume sind jeweils mit Türen verbunden. In einem dieser Räume befindet sich das Wumpus, das für den Agenten tödlich ist. Gegen das Wumpus hat der Agent nur einen Pfeil, den er abschießen kann. In einem weiteren Raum befindet sich ein Berg Gold, den der Agent sucht. Desweiteren befinden sich noch Falltüren in einigen Räumen.

Die PEAS (**P**erformance, **E**nvironment, **A**ctuators, **S**ensors) - Beschreibung ist wie folgt:

- **Performance:** Berg Gold gefunden: +1000, Agent ist gestorben: -1000, Pfeil wird abgeschossen: -10 und -1 für jede andere Aktion
- **Environment:** Es ist ein 4x4 Raster an Räumen gegeben. Im Feld [1,1] beginnt der Agent mit Blick nach rechts, hier darf weder eine Falltür noch das Wumpus sein. Das Wumpus und das Gold werden zufällig auf die übrigen Quadrate verteilt. Mit einer Wahrscheinlichkeit von 0,2 kann jedes Quadrat eine Falltür haben.
- **Actuators:** Der Agent kann sich nach vorne bewegen und sich 90° nach links und nach rechts drehen. Das Bewegen nach vorne hat vor einer Wand natürlich keinen Effekt. Er stirbt, sobald er einen Raum mit einer Falltür oder dem lebenden Wumpus betritt. Der Agent kann mittels der Aktion *grab* einen Gegenstand nehmen, der sich im gleichen Raum wie er selbst befindet. Durch die Aktion *shoot* kann er einen Pfeil abschießen, der in die Blickrichtung des Agenten fliegt, bis er das Wumpus oder eine Mauer trifft.
- **Sensors:** Der Agent besitzt 5 Sensoren: In allen Quadraten direkt neben einem Objekt, die nicht diagonal davon liegen, nimmt der Agent etwas wahr: Um das Wumpus einen Gestank (*Stench*), um das Gold ein glitzern (*Glitter*) und um eine Falltür einen Luftzug (*Breeze*). Der Agent merkt, wenn er gegen eine Wand läuft (*Bump*) und er hört, wenn das Wumpus gestorben ist (*Scream*)

Der Agent bekommt in den meisten Instanzen der Wumpus-Welt sein Gold unverletzt. Es gibt auch Welten, in denen der Agent sich entscheiden muss, ob er lieber am leben bleiben will. 1/5 der Welten sind unfair, weil das Gold zum Beispiel von Falltüren umgeben ist.

Erkundet der Agent nun seine Welt und nimmt etwas wahr, so weiß er, auf welchen Quadraten ein Objekt sein kann. Nimmt er nichts wahr, so ist er sich sicher, dass er den nächsten Raum betreten kann. Diese Räume werden mit einem *OK* markiert. Nimmt er zum Beispiel auf Feld [2,1] einen Luftzug wahr, so weiß er, dass auf Feld [3,1] oder auf Feld [2,2] Falltüren sein können. Feld [1,1] ist hierbei ausgenommen, da es das Startfeld ist. So bekommt der Agent nach und nach ein Bild von seiner Umgebung, indem er nur Felder betritt, die für ihn eindeutig sicher sind. Geht der Agent nach dem Besuch von [2,1] auf das Feld [1,2], und nimmt nichts wahr, so weiß er, dass auf [2,2] keine Falltür war, sondern dass diese auf Feld [3,1] zu finden ist. Hierbei handelt es sich um ein typisches Schließen eines logischen Agenten, was ein recht schwieriger Schluss ist, da er Wissen kombinieren muss, das zu verschiedenen Zeiten aufgenommen wurde. So bewegt sich der Agent nun weiter, bis er das Gold gefunden hat, oder bis er merkt, dass es klüger ist, aufzuhören anstatt zu sterben. Wenn ein Agent einen Schluss aufgrund seines Wissens zieht, so ist dieser garantiert korrekt, wenn seine Informationen korrekt sind.

Mit Hilfe der **Logik** werden die Sätze in der Wissensbasis eines Agenten gebildet. Diese Sätze gehorchen einer **Syntax**. Durch die **Semantik** wird der Wahrheitsgehalt eines Satzes für jede mögliche Umgebung definiert. Jeder Satz muss entweder wahr oder falsch sein.

Als andere Bezeichnung für solche Umgebungen wird der Begriff **Modell** benutzt. Der Satz „ m ist ein Modell von α “ besagt, dass α in m wahr ist. Ein Modell beinhaltet alle möglichen Zuweisungen von Zahlen an die sämtliche Variablen eines Satzes. Jede dieser Zuweisungen entscheidet, ob der Satz wahr oder falsch ist.

Das Schließen beinhaltet auch die logische Konsequenz zwischen Sätzen. Folgt ein Satz logisch aus einem anderen, so schreibt man das folgendermaßen:

$$\alpha \models \beta$$

β ist also eine logische Konsequenz von α . Gilt dieser Satz, dann muss folgendes gelten: In jedem Modell, in dem α wahr ist, muss auch β wahr sein. Ein kleines Beispiel: Die logische Konsequenz eines mathematischen Satzes, wie etwa „ $x - y = 3$ “, wäre zum Beispiel der Satz „ $3 = x - y$ “. Die Wissensbasis wird auch oft als Aussage betrachtet und kann einen Satz als Konsequenz haben.

Zurück zur Wumpus-Welt. Der Agent hat auf Feld [1,1] nichts wahrgenommen. Auf Feld [2,1] hat er einen Luftzug bemerkt. Diese beiden Sätze zusammen mit den Regeln über diese Welt bilden in diesem Moment seine Wissensbasis. Die einzigen möglichen Räume, die er betreten könnte, sind [1,2], [2,2] und [3,1]. Es wäre für ihn durchaus von Vorteil, wenn er vorher weiß, in welchem Raum eine Falltür ist. Da jeder der drei Räume eine Falltür enthalten kann, existieren 2^3 mögliche Modelle. Ist die Wissensbasis in einem Modell falsch, so widerspricht dieses

Modell dem, was der Agent weiß. Besitzt ein Modell zum Beispiel eine Falltür in dem Raum [1,2], so widerspricht das dem Wissen des Agenten, weil dieser im Raum [1,1] keinen Luftzug bemerkt hat.

Von diesen 8 möglichen Modellen gibt es 3, in denen die Wissensbasis wahr ist. Betrachten wir zwei mögliche Schlussfolgerungen:

$$\alpha_1 = \text{„Es gibt keine Falltür in [1,2]“}$$
$$\alpha_2 = \text{„Es gibt keine Falltür in [2,2]“}$$

α_1 beinhaltet die Wissensbasis und das Modell, in dem überhaupt keine Falltüren vorkommen. Man kann also die Aussage treffen, dass α_1 in jedem Modell wahr ist, in dem auch die Wissensbasis (KB) wahr ist. Also gilt: $KB \models \alpha_1$

α_2 beinhaltet alle Modell, bei denen in Raum [2,2] keine Falltür vorkommt. Es gibt also Modelle, in denen die Wissensbasis (KB) wahr, α_2 aber falsch ist. Hier gilt also: $KB \not\models \alpha_2$. Der Agent kann keine Aussage darüber treffen, ob in [2,2] eine Falltür ist.

Dieser **Inferenz-Algorithmus**, bei dem alle möglichen Modelle aufgezählt werden, um dann zu überprüfen, ob α überall da wahr ist, wo KB wahr ist, nennt man **Model-Checking**. Man schreibt

$$KB \vdash_i \alpha$$

wenn ein Inferenz-Algorithmus i den Satz α von der KB ableitet. Man nennt einen Inferenz-Algorithmus, der nur folgerbare Sätze ableitet, **korrekt** oder **wahrheitshaltend**. Es ist natürlich sehr wichtig, dass eine gefolgerte Aussage auch korrekt ist. Model-Checking ist, wenn es angewendet werden kann, wahrheitserhaltend.

Des weiteren ist es wichtig, dass ein Inferenz-Algorithmus vollständig ist, das heißt, dass alle folgerbaren Sätze abgeleitet werden können. Wenn die Anzahl der Konsequenzen endlich ist, sollte dies keine großen Probleme darstellen. Ist die Anzahl der Konsequenzen jedoch unendlich groß, so wird die Vollständigkeit zu einem wichtigen Aspekt.

Es gilt also, dass das Schließen in allen Welten, in denen die Vorgaben (Wissensbasis) wahr ist, auch alle Schlussfolgerungen wahr sind. Ist die Wissensbasis wahr, so ist jeder Satz α , der durch einen wahrheitserhaltenden Inferenz-Algorithmus von der Wissensbasis abgeleitet wird, auch wahr.

Das letzte, was der logische Agent berücksichtigen muss, ist die **Fundierung**. Woher wissen wir, dass die Wissensbasis in der realen Welt wirklich wahr ist? Nimmt der Agent einen Luftzug wahr, so schreibt er das in seine Wissensbasis. Somit ist dieses auch in der realen Welt wahr. Die Sensoren des Agenten bilden also die Verbindung zwischen der Wissensbasis und der realen Welt. Andere Teile der Wissensbasis repräsentieren keine direkten Wahrnehmungen, sondern beruhen auf Erfahrungen des Agenten. Der Agent weiß, dass, wenn er einen Luftzug bemerkt, er in der Nähe einer Falltür ist. Aber es könnte ja durchaus auch sein, dass ein Raum einfach nur schlecht isoliert ist und deshalb ein Luftzug herrscht, der unabhängig von einer Falltür ist. Die Wissensbasis ist damit zwar in der realen Welt nicht wahr,

aber wenn der Agent über eine gute Lernprozedur verfügt, kann das nach wenigen Agenten schnell ausgeglichen werden.

3 Die Aussagenlogik

Die **Syntax** der Aussagenlogik definiert die erlaubten Sätze. Sätze, die nur aus einem einzigen Aussagensymbol bestehen nennt man **atomare Sätze**. Für die Symbole, die *wahr* oder *falsch* sein können, werden Großbuchstaben verwendet.

Komplexe Sätze sind atomare Sätze, die mit logischen Verknüpfungen verbunden sind:

\neg (nicht), \wedge (und), \vee (oder), \Rightarrow (impliziert) und \Leftrightarrow (genau dann, wenn)

Die Aussagenlogik genügt folgender Grammatik in BNF:

```

Satz    → Atomarer Satz | Komplexer Satz
Atomarer Satz → True | False | Symbol
Symbol  → P | Q | R | ...
Komplexer Satz → ¬ Satz
                | (Satz ∧ Satz)
                | (Satz ∨ Satz)
                | (Satz ⇒ Satz)
                | (Satz ⇔ Satz)
    
```

Die Prioritätsreihenfolge ist die folgende (von der höchsten zur niedrigsten): \neg , \wedge , \vee , \Rightarrow , \Leftrightarrow

Sätze, wie $A \vee B \vee C$ sind erlaubt, weil es hier egal wäre, welche Symbole geklammert werden. Bei Sätzen, wie $A \Rightarrow B \Rightarrow C$ wird eine Klammerung verlangt, weil der Satz unterschiedliche Bedeutungen haben kann.

Die **Semantik** definiert die Regeln, mit denen der Wahrheitsgehalt eines Satzes ermittelt wird. In einem Modell der Aussagenlogik wird jedem Aussagensymbol ein Wahrheitswert (*true* oder *false*) zugewiesen.

Die Semantik muss angeben, wie der Wahrheitswert beliebiger Sätze berechnet werden kann. Der Wahrheitswert atomarer Sätze ist der folgende: *true* ist immer wahr, *false* ist immer falsch, der Wahrheitswert anderer Symbole muss direkt angegeben werden.

Die Wahrheit eines komplexen Satzes kann auf die Wahrheit einfacherer Sätze reduziert werden. Die Regeln für jede der fünf Verknüpfungen ist in folgender Wahrheitstafel zusammengefasst:

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
<i>false</i>	<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>true</i>
<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>	<i>true</i>	<i>true</i>	<i>false</i>
<i>true</i>	<i>false</i>	<i>false</i>	<i>false</i>	<i>true</i>	<i>false</i>	<i>false</i>
<i>true</i>	<i>true</i>	<i>false</i>	<i>true</i>	<i>true</i>	<i>true</i>	<i>true</i>

Hiermit kann der Wahrheitswert jedes beliebigen Satzes festgestellt werden.

Ein Beispielsatz wäre zum Beispiel $S_{2,1} \Leftrightarrow (W_{2,2} \vee W_{3,1} \vee W_{1,1})$, wobei S für Stench (Gestank) und W für Wumpus steht. Auf Feld [1,1] kann hier jedoch kein Wumpus sein, da das von vorneherein ausgeschlossen wurde.

Jetzt folgt eine einfache Konstruktion einer Wissensbasis, die aber der Einfachheit halber nur das Wumpus selber behandelt:

- $W_{i,j}$ sei *true*, wenn sich auf [i,j] das Wumpus befindet.
- $S_{i,j}$ sei *true*, wenn es auf [i,j] nach dem Wumpus stinkt.

Die Wissensbasis des Agenten hat am Anfang folgende Sätze:

1. Auf Feld [1,1] befindet sich das Wumpus nicht:

$$R_1: \neg W_{1,1}$$

2. Auf einem Feld stinkt es genau dann, wenn auf einem der benachbarten Felder ein Wumpus wohnt. Der Agent hat sich von [1,1] nach Norden auf [1,2] bewegt:

$$R_2: S_{1,1} \Leftrightarrow (W_{1,2} \vee W_{2,1})$$

$$R_3: S_{1,2} \Leftrightarrow (W_{1,1} \vee W_{2,2} \vee W_{1,3})$$

3. Jetzt fehlen noch die Wahrnehmungen des Gestanks:

$$R_4: \neg S_{1,1}$$

$$R_5: S_{1,2}$$

Eine Wissensbasis ist nun eine Konjunktion der einzelnen Sätze. In diesem Fall sieht die Wissensbasis also folgendermaßen aus: $R_1 \wedge R_2 \wedge R_3 \wedge R_4 \wedge R_5$. Sie sagt also aus, dass jeder einzelne Satz wahr ist.

3.1 Inferenz

Mit der logischen Inferenz wird entschieden, ob ein Satz α von der Wissensbasis abgeleitet werden kann. Es wird überprüft, ob α genau da wahr ist, wo die Wissensbasis auch wahr ist. Kann man zum Beispiel eine Aussage darüber treffen, ob sich auf dem Feld [2,2] das Wumpus befindet?

Wir haben 7 Aussagensymbole: $S_{1,1}, S_{1,2}, W_{1,1}, W_{1,2}, W_{2,1}, W_{2,2}, W_{1,3}$. Das ergibt also $2^7 = 128$ verschiedene Modelle. In 2 dieser Modelle ist die Wissensbasis wahr, und zwar, wenn das Wumpus sich auf [1,3] oder auf [2,2] befindet. In beiden Modellen ist auf [1,2] kein Wumpus. Man kann aber nicht genau sagen, ob sich das Wumpus jetzt auf [2,2] oder auf [1,3] befindet.

Da das SAT-Problem (Erfüllbarkeitsproblem aussagenlogischer Ausdrücke) in NP liegt, ist es nicht in polynomialer Zeit zu lösen. Ein Inferenz-Algorithmus hat also im schlechtesten Fall eine zur Eingabegröße exponentielle Komplexität.

3.2 Logische Äquivalenz

Die logische Äquivalenz besagt, ob zwei Sätze α und β in genau derselben Modellmenge wahr sind. Man schreibt $\alpha \equiv \beta$. Dies ist recht einfach anhand von Wahrheitstabellen festzustellen. Es gelten dafür folgende Regeln:

$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$	Kommutativität von \wedge
$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$	Kommutativität von \vee
$((\alpha \wedge \beta) \vee \gamma) \equiv (\alpha \wedge (\beta \vee \gamma))$	Assoziativität von \wedge
$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$	Assoziativität von \vee
$\neg(\neg\alpha) \equiv \alpha$	Eliminierung der doppelten Negation
$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$	Kontraposition
$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$	Implikationseliminierung
$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$	Bikonditionaleliminierung
$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$	de Morgan
$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$	de Morgan
$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$	Distributivität von \wedge über \vee
$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$	Distributivität von \vee über \wedge

Die Symbole α , β und γ stehen für beliebige Symbole der Aussagenlogik.

Die **Gültigkeit** eines Satzes sagt aus, ob dieser Satz in jedem Modell wahr ist. Solch ein Satz wird auch als **Tautologie** bezeichnet.

Ein Satz ist **erfüllbar**, wenn er in mindestens einem Modell wahr ist. Um das zu überprüfen, können alle möglichen Modelle aufgelistet werden und dann deren Wahrheitswert ermittelt werden. Dieses Problem ist aber NP-vollständig.

4 Inferenzschemata der Aussagenlogik

Um Folgerungen abzuleiten gibt es sogenannte **Inferenzregeln**. Eine ist zum Beispiel der **Modus Ponens**:

$$\frac{\alpha \Rightarrow \beta, \alpha}{\beta}$$

Sind $\alpha \Rightarrow \beta$ und α gegeben, so kann β daraus geschlossen werden. Ein Beispiel: $(WumpusAhead \wedge WumpusAlive) \Rightarrow Shoot$ und $(WumpusAhead \wedge WumpusAlive)$ sind gegeben, dann kann $Shoot$ gefolgert werden.

Eine weitere Regel ist die **Und-Eliminierung**. Ist $(\alpha \wedge \beta)$ gegeben, so kann daraus sowohl α als auch β gefolgert werden.

Hier ein Beispiel für die Wumpus-Welt. Die Sätze R_1 bis R_5 sind gegeben:

$$1. R_6 : (S_{1,1} \Rightarrow (W_{1,2} \vee W_{2,1})) \wedge ((W_{1,2} \vee W_{2,1}) \Rightarrow S_{1,1})$$

Hier wurde die Bikonditional-Eliminierung auf den Satz R_2 angewendet.

$$2. R_7 : (W_{1,2} \vee W_{2,1}) \Rightarrow S_{1,1}$$

Und-Eliminierung auf Satz R_6 angewendet.

$$3. R_8 : (\neg S_{1,1} \Rightarrow \neg(W_{1,2} \vee W_{2,1}))$$

Die Kontraposition von Satz R_7 .

$$4. R_9 : \neg(W_{1,2} \vee W_{2,1})$$

Hier wurde der Modus-Ponens auf die Sätze R_4 und R_8 angewendet.

$$5. R_{10} : \neg W_{1,2} \wedge \neg W_{2,1}$$

Die de-Morgan-Regel angewendet auf Satz R_9 .

An dem Satz R_{10} kann der Agent nun erkennen, dass sich weder auf dem Feld [1,2], noch auf [2,1] ein Wumpus befindet.

Wie hier gezeigt, ist es auch möglich, einen Satz mit den Inferenzregeln und der Wissensbasis zu beweisen, anstatt alle möglichen Modelle aufzulisten. Leider ist aber auch die Inferenz NP-vollständig. Von Vorteil ist jedoch, dass sie sich nur auf die wichtigen Aussagensymbole, die in der Berechnung des Ergebnisses eine Rolle spielen, konzentrieren kann. Es wurden etwa alle Symbole aus dem Satz R_3 ignoriert, da diese keinen Einfluss auf das Ergebnis haben.

Eine Eigenschaft logischer Systeme ist die **Monotonie**, die besagt, dass bei einer festen Wissensbasis nur eine bestimmte Anzahl folgerbarer Sätze abgeleitet werden kann. Die Anzahl der folgerbaren Sätze wächst also nur, wenn der Wissensbasis Informationen hinzugefügt werden.

4.1 Resolution

Die Resolution ist ein Verfahren, das einen Satz in KNF auf Erfüllbarkeit testet.

Eine einfache Version der Resolutionsregel: Der Agent weiß, dass es auf dem Feld [3,4] genau dann stinkt, wenn sich das Wumpus auf einem der Felder [3,1], [2,2], [3,3] oder [4,2] befindet. Ausgedrückt durch folgenden Satz:

$$S_{3,4} \Leftrightarrow (W_{3,1} \vee W_{2,2} \vee W_{3,3} \vee W_{4,2})$$

Der Agent hat nun schon das Feld [2,1] besucht und hat keinen Gestank wahrgenommen. Er schließt also daraus, dass sich weder auf Feld [2,2] noch auf [3,1] ein Wumpus befindet:

$$\neg W_{2,2}$$

$$\neg W_{3,1}$$

Die Resolutionsregel besagt nun, dass das Literal $W_{2,2}$ mit dem Literal $\neg W_{2,2}$ resolviert (aufgelöst) werden kann. Das gleiche gilt natürlich auch für $W_{3,1}$. Es ergibt sich also folgende neue Regel:

$$S_{3,4} \Leftrightarrow (W_{3,3} \vee W_{4,2})$$

Würde der Agent zum Beispiel auf Feld [4,1] ebenfalls keinen Gestank wahrnehmen, so würde sich nach dem resolvidieren von $W_{4,2}$ herausstellen, dass sich das Wumpus auf [3,3] aufhalten muss.

$$\frac{l_1 \vee \dots \vee l_k, m}{l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k}$$

Hier ist jedes l ein Literal, wobei l_i und m **komplementäre Literale** sind. Zur vollständigen Resolutionsregel verallgemeinert lautet die Regel:

$$\frac{l_1 \vee \dots \vee l_k, m_1 \vee \dots \vee m_n}{l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n}$$

Hier sind die l_i und m_j jeweils komplementäre Literale.

Mit der Resolutionsregel kann jeder vollständige Suchalgorithmus jeden Satz, der logisch aus den Sätzen der Wissensbasis folgt, ableiten. Es gibt jedoch eine Einschränkung: Die Resolutionsregel kann nicht dazu benutzt werden, um wahre Sätze aufzulisten, sie kann einen Satz nur für wahr oder falsch erklären. Der Satz $A \vee B$ ist zum Beispiel wahr, wenn A wahr ist, er kann aber nicht mit Hilfe der Resolutionsregel aus $A = true$ erzeugt werden.

Wie kann jetzt ein Satz mit Hilfe der Resolution aus der Wissensbasis abgeleitet werden? Um zu zeigen, dass der Satz α aus der Wissensbasis folgt, wird bewiesen, dass der Satz $\neg\alpha$ zusammen mit der Wissensbasis zu einem Widerspruch führt. Aus diesem Widerspruch kann nun gefolgert werden, dass α zusammen mit der Wissensbasis wahr ist.

Jedes Paar komplementärer Literale wird resolviert. Die daraus entstehenden Sätze werden, sofern noch nicht vorhanden, in die Menge eingefügt. Das Ganze geschieht solange, bis entweder die leere Klausel abgeleitet wird oder bis keine neuen Klauseln mehr hinzugefügt werden können. Im ersten Fall folgt ein Satz β aus dem Satz α , im zweiten Fall nicht. Die leere Klausel ist hierbei äquivalent mit *false*, da nicht mindestens eines ihrer Disjunkte wahr ist.

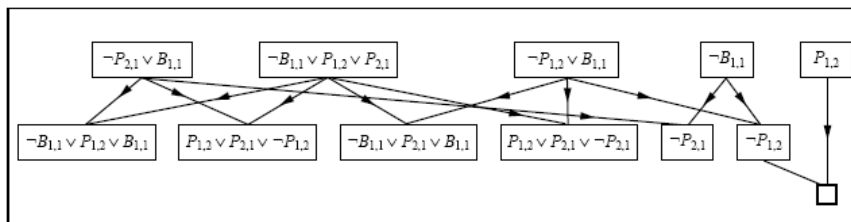


Abbildung 1: In dieser Abbildung wird gezeigt, dass $\alpha = \neg P_{1,2}$ wahr ist. $\neg\alpha$ wird zusammen mit der Wissensbasis resolviert. Viele Sätze sind hier unsinnig und können verworfen werden, weil die Erkenntnis, dass *true* wahr ist nicht sehr hilfreich ist.

5 Effektive aussagenlogische Inferenz

Hier werden zwei Familien effizienter, auf Model-Checking basierender Algorithmen für Aussagenlogische Inferenz vorgestellt: basierend auf der Backtracking-Suche und auf der Hillclimbing-Suche. Beide überprüfen die Erfüllbarkeit.

Der **DPLL-Algorithmus** ist eine Version des Davis-Putman-Algorithmus. DPLL erhält einen Satz in KNF und benutzt für seine Berechnung eine rekursive Tiefensuchauflistung möglicher Modelle. Vorteile:

- *Frühe Terminierung*: Der Algorithmus erkennt schon in einem früheren Stadium ob ein Satz wahr oder falsch ist, er weiß, dass eine Klausel wahr ist, wenn ein beliebiges Literal wahr ist. $(A \vee B) \wedge (A \vee C)$ ist wahr, wenn A wahr ist. Selbst dann, wenn die Wahrheitswerte von B und C noch nicht kontrolliert wurden. Ist eine ganze Klausel falsch (jedes seiner Literale ist falsch), so ist der komplette Satz auch falsch.
- *Reines-Symbol-Heuristik*: A ist zum Beispiel ein **reines Symbol**, wenn es in allen im Satz vorkommenden Klauseln immer dasselbe Vorzeichen besitzt. A ist also entweder nur positiv oder nur negativ. Gibt es reine Symbole, so müssen diese bei einem möglichen Modell immer *true* ergeben. Eine Klausel, in der ein solches Symbol vorkommt kann so nicht *false* sein, weil in der Disjunktion nun mindestens ein Element *true* ist. Weiß der Algorithmus vor der Überprüfung auf Reinheit eines Symbols, dass einige Klauseln schon wahr sind, so können diese bei der Überprüfung weggelassen werden. Ist $(A \vee \neg B)$ wahr und in den übrigen Klauseln kommt B ohne \neg vor, so ist B ein reines Symbol. Auf diese Weise kommen eventuell mehrere reine Symbole zustande.
- *Einheitsklausel-Heuristik*: Eine Klausel nennt man **Einheitsklausel**, wenn diese nur ein Literal enthält. Im DPLL-Algorithmus ist auch eine Klausel, dessen Literale alle, bis auf eins schon auf *false* gesetzt wurden, eine Einheitsklausel. $(\neg A \vee B)$ ist also ein Literal, wenn $B = false$ gilt. A wird also zwangsläufig auf *false* gesetzt, um die Klausel wahr zu machen. Damit können sich weitere Einheitsklauseln ergeben (zum Beispiel $(A \vee C)$), oder es kann zu einem Widerspruch führen.

Hier ein kleines Beispiel, wie der DPLL-Algorithmus arbeitet:

$$(A \vee B \vee D) \wedge (\neg B \vee D) \wedge (C \vee A) \wedge (\neg C \vee \neg B \vee \neg D) \wedge \neg D$$

A ist ein reines Symbol, also wird $A=true$ gesetzt:

$$(true \vee B \vee D) \wedge (\neg B \vee D) \wedge (C \vee true) \wedge (\neg C \vee \neg B \vee \neg D) \wedge \neg D$$

Die letzte Klausel beinhaltet nur $\neg D$, also ist dies eine Einheitsklausel. D wird also auf *false* gesetzt:

$$(true \vee B \vee false) \wedge (\neg B \vee false) \wedge (C \vee true) \wedge (\neg C \vee \neg B \vee true) \wedge true$$

Damit die zweite Klausel wahr wird, muss $B=false$ sein. Die zweite Klausel ist hier also eine Einheitsklausel:

$$(true \vee false \vee false) \wedge (true \vee false) \wedge (C \vee true) \wedge (\neg C \vee true \vee true) \wedge true$$

Das Symbol C wurde nicht betrachtet, weil es mit $A=true$ und $B=D=false$ egal ist, ob C *false* oder *true* ist.

Der **WalkSAT-Algorithmus**, ein lokaler Suchalgorithmus, nimmt sich eine Klausel, die nicht erfüllt wurde. Aus dieser wählt er ein Symbol, das nun geändert wird. Dieses Symbol wird mit einem der zwei folgenden Methoden ausgewählt: „min-conflicts“ minimiert die Anzahl der Klauseln, die durch die Änderung *false* werden, und der „zufällige Weg“, der einfach irgendein Symbol auswählt. Welche Methode genommen wird, ist Zufall. Das Problem bei WalkSAT ist, wie schon vorher beschrieben, dass das Erfüllbarkeitsproblem NP-Vollständig ist, und der Algorithmus, solange er kein positives Ergebnis liefert, keine Aussage über die Erfüllbarkeit des Satzes liefert. Ist der Satz nicht erfüllbar und es wird dem Algorithmus erlaubt, unendlich lange zu laufen, so terminiert er nicht. Liefert er ein Ergebnis, so ist dies ein Modell des Satzes.

Solche lokalen Suchalgorithmen sind meistens nur sinnvoll, wenn eine Lösung zu einem Satz existiert. Ob ein Satz eine Lösung hat, kann nicht herausgefunden werden. Wenn ein Suchalgorithmus in der Praxis jedoch immer recht schnell ein Ergebnis liefert, so kann man bei einer längeren Wartezeit davon ausgehen, dass keine Lösung existiert, was aber natürlich nicht beweist, dass keine Lösung existiert.

Um die Funktionsweise des Walk-SAT-Algorithmus zu verdeutlichen, hier auch ein kleines Beispiel:

$$(A \vee B \vee D) \wedge (\neg B \vee D) \wedge (C \vee A) \wedge (\neg C \vee \neg B \vee \neg D) \wedge \neg D$$

Den Symbolen A, B, C und D werden irgendwelche Wahrheitswerte zugewiesen, hier $A=B=C=D=true$:

$$(true \vee true \vee true) \wedge (false \vee true) \wedge (true \vee true) \wedge (false \vee false \vee false) \wedge false$$

Die ersten 3 Klauseln sind nun wahr. Die vierte Klausel nicht, also wird ein Symbol ausgesucht und dieses geändert, hier $C \rightarrow false$:

$$(true \vee true \vee true) \wedge (false \vee true) \wedge (false \vee true) \wedge (true \vee false \vee false) \wedge false$$

Jetzt muss die letzte Klausel noch wahr werden. Das erricht man durch ändern des Symbols D , also $D \rightarrow false$:

$$(true \vee true \vee false) \wedge (false \vee false) \wedge (false \vee true) \wedge (true \vee false \vee true) \wedge true$$

Dadurch ist nun aber die zweite Klausel *false*. Weil eine Änderung von B die wenigsten Konflikte verursacht, wird B auf *false* gesetzt:

$$(true \vee false \vee false) \wedge (true \vee false) \wedge (false \vee true) \wedge (true \vee true \vee true) \wedge true$$

Nun ist jede der fünf Klauseln *true* für $A=true$ und $B=C=D=false$.

Wie verhalten sich WalkSAT und DPLL in der Praxis: Probleme, wie zum Beispiel das n -Damen-Problem, bei dem es darum geht, 8 Damen auf einem Schachbrett so zu verteilen, dass keine eine andere schlagen kann, sind in der Praxis recht schnell mit lokalen Suchmethoden, wie 'min-conflicts', zu lösen. Mögliche Lösungen sind recht gut verteilt, so dass in der Nähe einer Zuweisung recht schnell eine Lösung gefunden werden kann. Das n -Damen-Problem ist so einfach, weil es **unterbeschränkt** ist. Ein unterbeschränktes Problem ist ein Problem, dessen KNF-Form nur wenige Klauseln hat und dadurch die Anzahl der Variablen beschränkt.

Wird die Anzahl der Klauseln bei gleichbleibender Symbolanzahl erhöht, so wird es schwieriger werden, eine Lösung zu finden. n sei die Anzahl der Symbole, m die Anzahl der Klauseln. Die Wahrscheinlichkeit der Erfüllbarkeit sinkt bei einem m/n -Verhältnis von 4,3 deutlich. An diesem **kritischen Punkt** nennt man die KNF-Sätze „fast erfüllbar“, beziehungsweise „fast unerfüllbar“. Nahe dieses kritischen Punktes steigt die Laufzeit stark an, diese Probleme sind also *sehr viel schwieriger* als andere.

6 Agenten auf der Basis der Aussagenlogik

Hier werden die Vor- und Nachteile von Agenten vorgestellt, die entweder Inferenz-Algorithmus und eine Wissensbasis verwenden oder solche, die logische Ausdrücke in Form von Schaltungen auswerten.

Wie findet ein aussagenlogischer Agent Falltüren oder das Wumpus? Er beginnt mit einer Wissensbasis, in der seine Welt beschrieben ist. Er weiß, dass er auf Feld [1,1] sicher ist, er weiß, wie Gestank und wie ein Luftzug entsteht und er weiß, dass es genau einen Wumpus gibt. Ein mögliches Agentenprogramm teilt nun seiner Wissensbasis einen möglichen Luftzug oder Gestank mit. Daraufhin wählt das Programm das nächste Feld aus. Am Besten wäre hier ein Feld, von dem der Agent sicher sagen kann, dass hier weder eine Falltür noch das Wumpus ist. Falls es kein beweisbar sicheres Feld gibt, kann sich der Agent auch auf ein Feld bewegen, über das er keine genaue Angabe machen kann, von dem er aber ausschließen kann, dass dort sicher eine Falltür oder das Wumpus ist.

In kleinen Welten funktioniert dieser Agent recht zufriedenstellend. In größeren Welten nimmt der Umfang der Wissensbasis jedoch drastisch zu. Für jedes Feld hat die Wissensbasis viele Gleichungen. Je größer die Welt des Agenten wird, umso mehr Gleichungen werden für jedes einzelne Feld benötigt.

Damit der Agent weiß, auf welchem Feld er sich gerade befindet und wie er sich weiterbewegen darf, wird ein Symbol L eingeführt, das aussagt, auf welchem Feld der Agent gerade ist. $L_{1,1}$ sagt zum Beispiel aus, dass der Agent sich auf Feld [1,1] befindet. Die Wissensbasis des Agenten sollte also Sätze beinhalten, die ihm sagen, auf welches Feld er sich bewegt, wenn er in eine bestimmte Richtung schaut und geradeaus geht. Solch ein Satz wäre zum Beispiel:

$$L_{1,1} \wedge FacingRight \wedge Forward \Rightarrow L_{2,1}$$

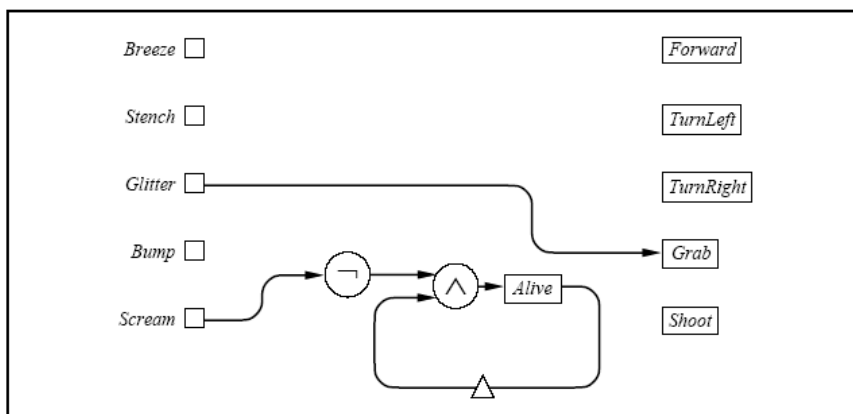


Abbildung 2: Nimmt der Agent ein Glitzern wahr, so ist laut der Schaltung Grab wahr. Der Agent weiß also, dass er das Gold nehmen darf. Solange der Agent keinen Schrei hört, lebt das Wumpus. Nimmt er einen Schrei wahr, so ist der Wahrheitswert in dem Register bis zum Ende false. Verzögerungen werden durch ein Dreieck dargestellt.

Das Problem hierbei ist aber, dass sowohl $L_{1,1}$ als auch $L_{2,1}$ wahr sind, wenn der Agent sich aufgrund dieses Satzes der Wissensbasis bewegt. Abhilfe schafft da eine Zeitkomponente t , die aussagt, dass ein $L_{i,j}^t$ nur zum Zeitpunkt t wahr ist. Korrekt heißt der obige Satz also:

$$L_{1,1}^1 \wedge FacingRight^1 \wedge Forward^1 \Rightarrow L_{2,1}^2$$

Allgemein gilt also der folgende Satz für jedes Feld (die rechten Randfelder ausgenommen):

$$L_{x,y}^t \wedge FacingRight^t \wedge Forward^t \Rightarrow L_{x+1,y}^{t+1}$$

Es ist leicht zu sehen, dass die Anzahl an Sätzen in der Wissensbasis stark ansteigt, wenn noch die Zeitkomponente dazukommt. Eine Grenze für aussagenlogische Programme liegt bei etwa 100 x 100 Feldern. Etwas Abhilfe schaffen hier die schaltungsbasierten Agenten.

6.1 Schaltungsbasierte Agenten

Ein **schaltungsbasierter Agent** ist eine Art **Reflexagent**. Seine Wahrnehmungen sind Eingaben für eine **sequentielle Schaltung**. Die Anweisungen an den Agenten entsprechen den Ausgaben der Schaltung, die in Registern gespeichert werden. Diese Agenten behandeln die Zeitkomponente besser als aussagenlogische Agenten, sie brauchen deswegen auch nicht eine solche Anzahl an Sätzen, sondern kommen mit wenigen Schaltungen aus.

Bei jedem Schritt, den der Agent macht, werden die Eingaben für die Schaltung gesetzt. Daraufhin wird mit den Ausgaben ein Ergebnis geliefert. Jeder Wert, der

ausgegeben wird, ist nur zu einer bestimmten Zeit gültig. Es wird hier also nicht für jeden Zeitpunkt t eine neue Instanz der Schaltung benötigt. Die Schaltung, ob das Wumpus noch lebt entspricht zum Beispiel folgendem aussagenlogischen Satz:

$$Alive^t \Leftrightarrow \neg Scream^t \wedge Alive^{t-1}$$

Das Problem bei diesem System ist jedoch, dass die Register immer einen Wert enthalten, entweder *true* oder *false*. Normalerweise sollte der Agent über ein Feld, das er noch nicht besucht hat, keine Aussage machen können. In der Aussagenlogik ist es möglich, dass der Agent keine Aussage über den Zustand eines Feldes machen kann. Ein schaltungsbasierter Agent kann aus einem Register jedoch immer nur *true* oder *false* lesen. Abhilfe bietet hier die Verwendung von zwei Bits statt einem Bit. Wird jede Anweisung an den Agenten durch zwei Register repräsentiert, so hat der Agent doppelt so viele Interpretationsmöglichkeiten. Sind beide Register *false*, so ist der Zustand des Feldes unbekannt. Ist eines der beiden *true*, so hängt es von der Beschriftung des Registers ab, ob der Agent die Belegung zum Beispiel als vorhandenen Luftzug oder als nicht vorhandenen Luftzug interpretiert. Sind beide Register *true*, so liegt ein Fehler vor.

Die Schaltungen, ob auf einem Feld $[x,y]$ eine Falltür ist, oder nicht, sind recht kompliziert, weil überprüft werden muss, ob ein Luftzug zu diesem Feld $[x,y]$ gehört oder zu einem anderen Feld gehört. Für jedes Feld gibt es jedoch nur eine konstante Anzahl von Gattern, schaltungsbasierte Agenten können also auch in größeren Systemen eingesetzt werden. Voraussetzung ist hier, dass das System **Lokalität** aufweist. Das bedeutet, dass es für ein beliebiges Feld eine obere Schranke an Feldern gibt, die betrachtet werden müssen, damit über dieses Feld eine Aussage gemacht werden kann. Das Spiel Minesweeper ist zum Beispiel nicht lokal, da man beliebig weit entfernte Felder betrachten muss, um eine Aussage über ein Feld machen zu können.

7 Zusammenfassung

Es gibt erhebliche Konflikte zwischen Effizienz, Kürze, Vollständigkeit und einfacher Konstruktion. Bei einfachen Zusammenhängen, wie bei *glitter* und *grab* ist wohl eine Schaltung optimal. Bei komplexeren Zusammenhängen, wie dem Erkennen von Falltüren aufgrund von einem Luftzug ist ein aussagenlogischer Agent eher geeignet. Bei einem inferenzbasierten Agenten nimmt die Anzahl an Sätzen in seiner Wissensbasis stark zu, wenn die Welt, in der er sich befindet, größer wird. Bei einem schaltungsbasierten Agenten kommen nur eine nahezu feste Anzahl an Schaltungen pro neuem Feld dazu. Diese Schaltungen können jedoch sehr kompliziert werden, die Sätze eines aussagenlogischen Agenten sind recht einfach. Beide Agenten haben ihre Vor- und Nachteile. Um eine einfache Intelligenz darzustellen würde diese Agenten reichen. Komplexere Agenten sollten aber die Vorteile beider Typen verwenden, dies wäre ein sogenannter **hybrider Agent**.