Kapitel 1 - Einführung

Was ist GAI

Definition ist abhängig von der Dimension:

1) Denkprozesse und logisches Schließen
   Systeme die wie Menschen denken, Systeme die rational denken
2) Verhalten
   Systeme die wie Menschen agieren, Systeme die rational agieren

Ansätze der verschiedenen Definitionen

Systeme, die wie Menschen denken – Cognitive Science

Damit man sagen kann, dass ein Programm wie ein Mensch denkt, muss man festlegen können wie Menschen denken → man muss verstehen wie das menschliche Gehirn arbeitet:
durch Introspektion und psychologische Experimente möglich.

Validierung:

- top-down
  (das Verhalten von Menschen testen und vorhersagen) → = cognitive science
- button-up
  (neurologische Daten verwenden) → = cognitive neuroscience

Beide Vorgehensweisen werden von Artificial Intelligence unterschieden. Sie teilen aber folgende **Charakteristik:**

- die Theorien erklären nicht, wie man menschliche Intelligenz erreicht.

Systeme die wie Menschen agieren – Turing Test

1950 von Allan Turing entwickelt
Ziel: operationale Definition der Intelligenz
Inhalt: ein Computer besteht den Test, wenn eine menschliche Person nicht feststellen kann, ob auf der anderen Seite ein Mensch oder eine Maschine sitzt.

Computer muss folgende Fähigkeiten besitzen:

- Verarbeitung natürlicher Sprache (ermöglicht sinnvolle Kommunikation)
- Wissensrepräsentation (Wissen speichern)
- automatisches logisches Schließen (Wissen nutzen um Fragen zu beantworten)
- Lernen (Muster erkennen und an neue Umstände anpassen)

zusätzlich beim totalen Turing-Test (Fragen per Videosignal):

- Computervision (Objekte wahrnehmen)
- Robotik (Objekte manipulieren bzw. bewegen)

Probleme

- Test ist nicht reproduzierbar
- nicht aufbauend
- nicht für mathematische Analyse geeignet

Systeme die rational denken - Logik
Bereits Aristoteles versuchte „richtiges Denken“ zu kodifizieren → unwiderlegbare Prozesse für logisches Schließen festzulegen → Logik mit Notation und Ableitungsregeln war geboren

2 Probleme
- nicht jede Form von intelligentem Verhalten kann mit logischen Überlegungen vermittelt werden
- Ziel des Denkens kann mit Logik nicht beantwortet werden (Welche Gedanken sollte ich aus allen möglichen haben?)

Systeme die rational agieren – rationale Agenten
rationales Verhalten = das richtige Tun (das, was die Möglichkeit der Zielerreichung mit der gegebenen Information maximiert) → involviert nicht notwendigerweise Denken

Rationale Agenten
Ein Agent ist etwas das agiert/handelt. Zusätzliche Attribute bei rationalen Agenten:

- selbstständige Steuerung
- Wahrnehmung der Umgebung
- Persistenz über einen längeren Zeitraum
- Anpassung an Änderungen
- Möglichkeit das Ziel eines anderen weiterzuführen

→ ein rationaler Agent verhält sich so, dass er das beste Ergebnis erzielt oder das beste zu erwartende Ergebnis (bei Unsicherheiten)

State of the Art
AI ist in sehr vielen Teilbereichen aktiv. **Beispiele**:
autonomes Planen und Zeitplanen, Spiele (SW schlägt Schachweltmeister), autonome Kontrolle (Autosteuerung), Diagnostik (Medizin), logistische Planung (Militär), Robotik (Medizin), Spracherkennung und Problemlösung

Kapitel 2 – Intelligente Agenten

Agenten und Umwelt
Agenten sind Menschen, Roboter, Software-Bots, Thermostate, etc. → alles, was seine Umgebung über Sensoren wahrnehmen kann und in dieser Umgebung durch Aktuatoren handelt.

Sensor
ermöglichen die Wahrnehmung. zB Mensch: Augen, Ohren – Roboter: Kameras, Infrarotsensoren – Software: Netzwerkpakete, Dateiinhalte

Aktuatoren
ermöglichen das Handeln. zB Mensch: Hände, Füße – Roboter: Motoren – Software: Bildschirm zur Interaktion

Wahrnehmungsfolge
vollständiger Verlauf von allem, was der Agent je wahrgenommen hat. Bisherige Wahrnehmungsfolge beeinflusst die Auswahl einer Aktion des Agenten zu jedem Zeitpunkt

Agentenfunktion vs. Agentenprogramm
Die Agentenfunktion ist eine abstrakte mathematische Beschreibung. Das Agentenprogramm ist eine konkrete Implementierung der Agentenfunktion für einen künstlichen Agenten.

Typen von Agentenprogrammen: Reflex-Agenten mit/ohne Zustand), zielorientierte Agenten, nutzenorientierte Agenten, alle können in lernende Agenten umgewandelt werden

Ein Agent besteht aus der Architektur (Sensoren, Aktuatoren) und dem Programm

Rationalität
… benötigt ein Leistungsmaß für den Grad an Rationalität

… ein rationaler Agent wählt die Aktion die den erwarteten Grad der Rationalität maximiert

Rational != allwissend
es gibt in der Realität keinen allwissenden Agenten, zB liefern die Sensoren nicht sämtliche relevante Information

Rational != hellseherisch
zB Resultat einer Aktion ist nicht so wie erwartet → daher: rational != erfolgreich

Erfolgreiche Rationalität erreichbar durch:
- Exploration (neue Information aus der Umwelt gewinnen)
- Lernen (durch Erfahrung lernen)
- Autonomie (Erweiterung oder Austausch von vorhandenem Wissen und gelerntem Wissen)

PEAS
= die Arbeitsumgebung eines Agenten = Performance Measure, Environment, Actuators, Sensors

… muss immer so vollständig wie möglich spezifiziert werden um einen rationalen Agenten designen zu können

Beispiel: Automatisiertes Taxi

Leistungsmaß	Environment	Actuators	Sensors
Sicherheit, Schnelligkeit, Komfort, der StVo entsprechend fahren, Profit	Straßen, Verkehr, Fußgänger, Wetter, Fahrgäste	Gas, Lenkrad, Bremse, Hupe, Blinker, Display	Tachometer, GPS, Kilometerzähler, Kameras, Tastatur, Motorsensoren, Bremssensoren

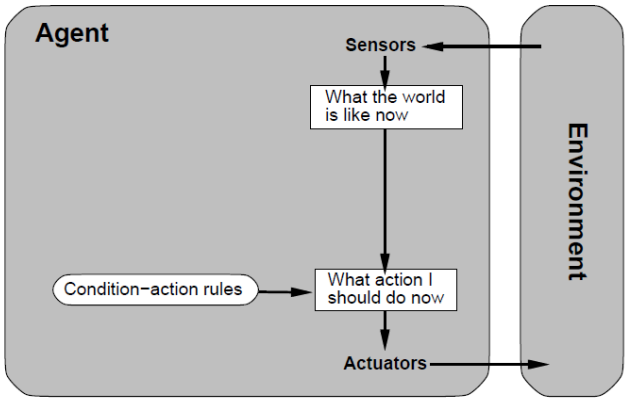
Eigenschaften der Arbeitsumgebung

…. der Typ der Arbeitsumgebung bestimmt das Design des Agenten

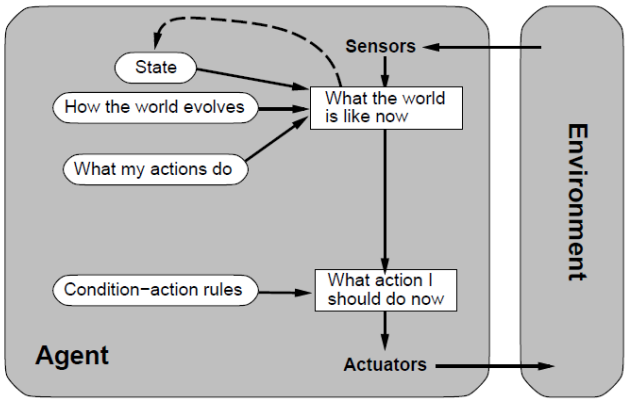
- **beobachtbar / teilweise beobachtbar**
  beobachtbar, wenn die Sensoren alle relevanten Eigenschaften erfassen können
- **deterministisch / stochastisch**
  deterministisch, wenn der nächste Zustand der Umgebung vollständig durch den aktuellen Zustand und die durch den Agenten ausgeführten Aktionen festgelegt werden kann
- **episodisch / nicht episodisch**
  episodisch, wenn die Erfahrung der Agenten ist in atomare Agenten unterteilt werden kann , die unabhängig voneinander sind
- **statisch / dynamisch**
  statisch, wenn sich die Umgebung nicht ändert während der Agent eine Entscheidung trifft
- **diskret / stetig**
- **single-agent / multi-agent**

Unsere Welt ist teilweise beobachtbar, stochastisch, nicht episodisch, dynamisch, stetig, multi-agent

Typen von Agenten Agenten



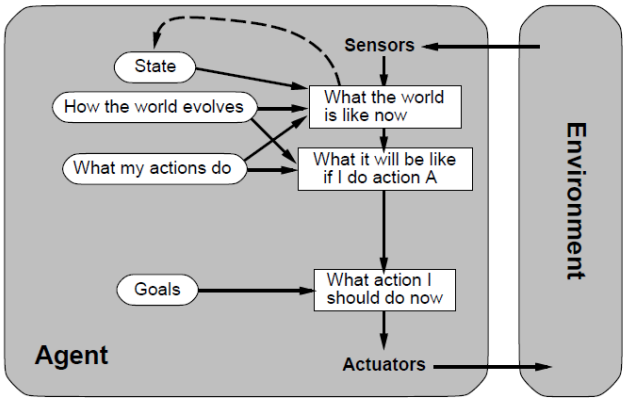
Einfache Reflex-Agenten
am einfachsten, wählt Aktion auf Grundlage der aktuellen Wahrnehmung und ignoriert den Verlauf. → begrenzt intelligent, funktioniert am besten in einer vollständig beobachtbaren Umgebung ohne Unsicherheit



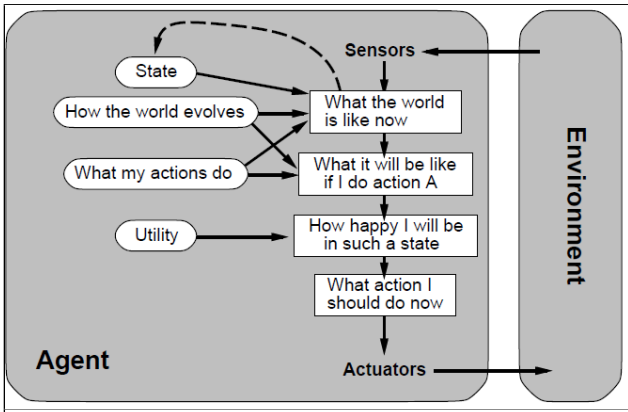
Modellbasierte Agenten / Reflex-Agenten mit Zustand
Agent führt einen internen Zustand, der vom Wahrnehmungsverlauf abhängig ist. → reflektiert dadurch zumindest einen Teil der nicht beobachtbaren Aspekte des aktuellen Zustands. Anschließend wählt er eine Aktion wie der Reflex-Agent aus.
2 Arten von Wissen sind notwendig um die interne Zustandsinformation aktualisieren zu können:

- Information wie sich die Welt unabhängig vom Agenten weiterentwickelt
- Information wie sich die Aktionen des Agenten selbst auswirken

→ Wissen wie die Welt funktioniert wird als Modell der Welt bezeichnet
→ Agent der ein solches Modell verwendet = modellbasierter Agent

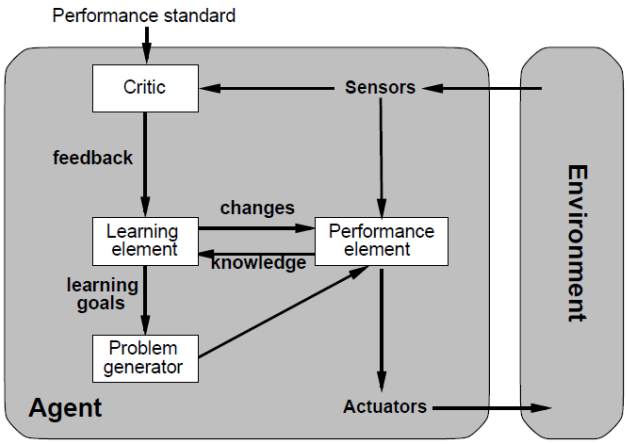


Zielorientierte Agenten
Zusätzliche Zielinformation um eine Aktion zu Ende zu bringen
Diese Art der Entscheidungsfindung unterscheidet sich von den Bedingung/Aktion-Regeln (der vorigen), weil dabei die Zukunft wesentlich berücksichtigt wird.
→ erscheinen ineffizient (weil Zielerreichung komplex ist) sind jedoch sehr flexibel, weil das Wissen, das seine Entscheidung unterstutzt, explizit dargestellt wird und verändert werden kann.



Nutzenorientierte Agenten
Zusätzliche Nutzenfunktion, eine vollständige Spezifikation dieser Funktion erlaubt rationale Entscheidungen in 2 Arten von Fällen wo Ziele nicht ausreichend sind:

- Ziele die sich gegenseitig widersprechen (Nutzenfunktion gibt Abwägung)
- Mehrere Ziele die der Agent anstreben kann



Lernende Agenten
Um If-Then Statements zu vermeiden sollte ein Agent selbst lernen wie er sich verhalten soll.
4 Komponenten:

- Kritiker (vergleicht erreichte Leistung mit der gewünschten Leistung, gibt Feedback zur Steigerung der Leistung)
- Lernelement (verantwortlich für Verbesserung durch Veränderung – kann Veränderungen an jeder Komponente vornehmen)
- Problemgenerator (bietet Aktionen die zu neuen Erfahrungen führen können)
- Leistungselement (wählt Aktion aus abhängig von Zustand, Wahrnehmung, etc.)

Kapitel 3 – Problemlösung und Suche

Problemformulierung

... Ist der Prozess zu entscheiden, welche Aktionen und Zustände berücksichtigt werden sollen um ein bestimmtes Ziel zu erreichen.

Daher wird eine **Zielformulierung** benötigt.
Zielformulierung ist der erste Schritt zur Problemlösung und basiert auf der aktuellen Situation und dem Leistungsmaß des Agenten.

Komponenten eines Problems

- **Anfangszustand**
  der Zustand des Agenten am Beginn

- **Aktionen / Nachfolgefunktion**
  S(x) … eine Menge an sortierten Aktion-Zustands Paaren zB S(Arad) = {(Arad→Zerind, Zerind),…}
  Aktionen sind die Möglichkeiten, die dem Agenten zur Verfügung stehen. → mit Hilfe der möglichen Aktionen erhalten wir einen Zustandsraum
  mathematische Formulierung ist dieNachfolgefunktion
  Zusammen bilden der Ausgangszustand und die Nachfolgefunktion implizit den Zustandsraum des Problems.
  Ein Pfad in einem Zustandsraum ist eine Abfolge von Zuständen die durch eine Aktionsfolge verbunden sind.

- **Zieltest**
  überprüft ob ein Ziel erreicht ist
  explizit: zB x="at Bukarest"
  implizit: zB NoDirt(x) = die Zielvorstellung ist gegeben

- **Pfadkosten**
  Kosten eines Pfades ist die Summe der Kosten der einzelnen Aktionen
  zB Summe der Entfernungen, Anzahl der Aktionen, etc.
  um zB festzustellen welcher Pfad der günstigste oder kürzeste ist (abhängig vom Ziel)

Weitere Begriffe

- **Lösung**
  ist die Sequenz an Aktionen vom Anfangszustand zum Zielzustand. Qualität der Lösung wird mit den Pfadkosten bewertet.

- **Abstraktion**
  Unwichtige Details werden durch Abstraktion aus der Repräsentation entfernt (aufgrund der Komplexität der realen Welt)
  gute Abstraktion: wenn so viele Details wie möglich entfernt wurden und die Repräsentation dennoch valide ist.

Probleme aus der realen Welt

TSP – Traveling Salesman Problem
= kombinatorisches Optimierungsproblem, ist ein NP-äquivalentes Problem

eine Reihenfolge für den Besuch mehrerer Orte soll so gewählt werden, dass die gesamte Reisestrecke des Handlungsreisenden nach der Rückkehr zum Ausgangsort möglichst kurz ist.

Weitere: Roboternavigation, Internetsuche, VLSI-Layout, Touring,Probleme, etc.

Suchbäume

… eine Datenstruktur, **Elemente**: Knoten (Elternknoten, Kindknoten), Pfadkosten, Tiefe

In der AI ist der Graph **implizit** durch den **Ausgangszustand** und die **Nachfolgefunktion** gegeben.

Eine „Expand"-Funktion kreiert neue Knoten (die Successor-/Nachfolgefunktion kreiert die entsprechenden Zustände)

Zustände vs. Knoten

ein Zustand ist eine Repräsentation einer physischen Konfiguration, ein Knoten ist ein Teil eines Suchbaums (also einer Datenstruktur)
→ Zustände haben keine Elternknoten, Kindknoten, Pfadkosten, Tiefe

Suchstrategien

… die Strategie definiert die Reihenfolge der „Knoten-Expansion"

Qualität der Strategie abhängig von

Zeit, Vollständigkeit, benötigter Speicher, Optimalität (wird immer die beste Lösung gefunden?)

Zeit und Speicher wird folgendermaßen gemessen:

- b …maximaler Verzweigungsfaktor des Suchbaums
- d … Tiefe des flachsten Knotens
- m … maximale Tiefe eines beliebigen Pfades im Zustandsraum (kann auch unendlich sein)

Die Effektivität wird mit den Suchkosten oder den Gesamtkosten bewertet.

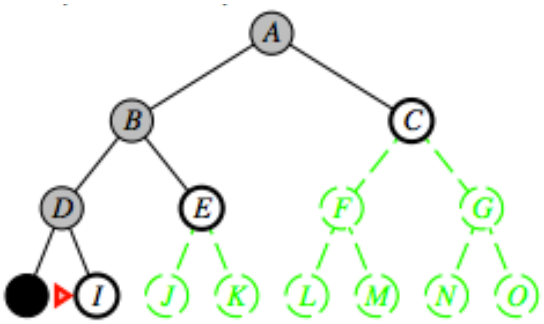
Uninformierte/Blinde Strategien

… haben keine zusätzlichen Informationen über Zustände außer der in der Problemdefinition

- **Breitensuche / Breadth-First Search**
  Wurzelknoten wird als erstes expandiert, dann alle Nachfolger und dann deren Nachfolger, usw.
  Implementierbar als FIFO-Queue (first in first out, dadurch werden Knoten die als erstes besucht werden auch als erstes expandiert)
  Optimal? Ja, wenn jede Aktion die gleichen Kosten verursacht -> also nicht generell optimal!
  <u>PROBLEME:</u>
  - benötigter Speicher (jeder Knoten muss im Speicher gehalten werden)

- **Suche mit einheitlichen Kosten / Uniform-Cost Search**
  Erweiterung der Breitensuche (da diese nur optimal ist, wenn alle Schrittkosten gleich sind)
  Statt dem flachsten Knoten wird der Knoten mit den geringsten Pfadkosten expandiert.

- **Tiefensuche / Depth-First Search**
  es wird immer der tiefste Knoten im aktuellen Radbereich des Suchbaums expandiert. Wenn ein Knoten expandiert wurde wird er aus dem Randbereich entfernt -> Vorteil: lineare Speicherkomplexität
  Implementierbar mit LIFO-Queue (last in first out)
  BEISPIEL:



  PROBLEME:

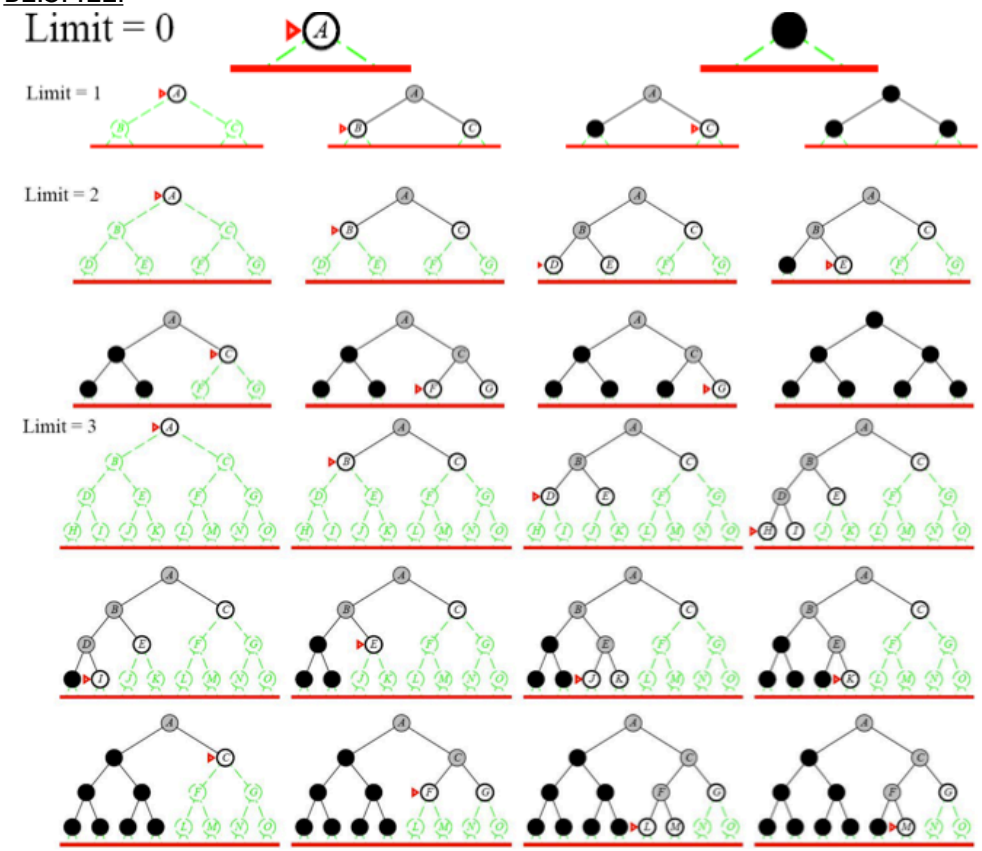
  - Zeit (wenn b > c -> wenn c die Lösung wäre -> es muss zuerst die gesamte linke Seite durchsucht werden)
  - Nicht komplett (wenn zB der linke Teil unendlich Tief ist)

- **Tiefenbeschränkte Suche / Depth-Limited Search**
  erweitert die Tiefensuche und behebt das Problem mit unendlich tiefen Bäumen (Tiefenbeschränkung l wird eingeführt).
  PROBLEME:

  - bei l<d (d=Lösung) wird keine Lösung gefunden

- **Iterativ vertiefte Tiefensuche / Iterativ Deepning Depth-First Search**
  ... häufig in Kombination mit der Tiefensuche verwendet, um die beste Tiefenbegrenzung zu ermitteln
  Das Limit wird schrittweise erhöht bis ein Ziel gefunden ist (dies ist der Fall, wenn die Tiefenbegrenzung d erreicht ist = die die Tiefe des flachsten Zielknotens)
  -> ist dadurch schneller als die Breitensuche
  Die Iterativ vertiefte Tiefensuche wird bei großen Suchräumen mit unbekannter Tiefe bevorzugt

BEISPIEL:



Vergleich der Strategien

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes*	Yes*	No	Yes, if $l \geq d$	Yes
Time	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	b^m	b^l	b^d
Space	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	bm	bl	bd
Optimal?	Yes*	Yes	No	No	Yes*

Wiederholte Zustände

Wenn wiederholte Zustände nicht erkannt werden, kann ein lineares Problem zu einem exponentiellen werden!

-> Die Möglichkeit Zeit zu verschwenden, indem man Zustände expandiert, die bereits besucht und expandiert wurden. Es soll daher der Tree-Search-Algorithmus so abgeändert werden, dass er eine Datenstruktur beinhaltet, die als **geschlossene Liste** bezeichnet wird. In dieser Liste wird jeder weitere Knoten gespeichert. Wenn der aktuelle Knoten in dieser Liste ist, wird er verworfen.

-> Dieser neue Algorithmus wird als **Graph-Such-Algorithmus** bezeichnet.

Tree Search vs. Graph Search

Durch die Verwendung einer geschlossenen Liste, sind die Speicheranforderungen der Tiefensuche und der iterativ vertiefendenen Tiefensuche nicht mehr linear! Außerdem sind einige Suchen auf Grund von Speicherbeschränkungen nicht mehr durchführbar. (weil jeder Knoten im Speicher behalten wird)

Graph-Suche ist effizienter als Tree-Suche, wenn es viele wiederholte Zustände gibt, weil die Graph-Suche jeden Knoten im Speicher behält. Die Worst-Case Zeit und die Speicheranforderungen sind proportional zur Größe des Zustandsraums.

Kapitel 4 - Informierte Suche und Exploration

... nutzt neben der Definition des Problems auch problemspezifisches Wissen.
-> findet daher Lösungen effizienter als uninformierte Strategien.

Best-First Suche / Bestensuche

... allgemeiner Ansatz, Beispiel für den allgemeinen Tree-Search und Graph-Search Algorithmus.
... Knoten werden abhängig von einer **Evaluierungsfunktion f(n)** ausgewählt. Im Normalfall wird der Knoten mit der geringsten Bewertung ausgewählt, weil als Bewertung die Distanz zum Ziel ermittelt wird.

Implementierung

...mit Hilfe einer Prioritätschlange = eine Datenstruktur, die den Randbereich in aufsteigender Reihenfolge von den f-Werten (die Bewertungen) verwaltet.

Spezialfälle

Es gibt eine ganze Familie von Best-First-Suchalgorithmen mit unterschiedlichen Evaluierungsfunktionien. Eine Schlüsselkomponente dieser Algorithmen ist eine **Heuristikfunktion h(n)**, die die Kosten für den billigsten Pfad vom Knoten n zu einem Zielknoten schätzt.

Beispiele: Greedy-Best-First-Search, A*-Suche

Greedy-Best-First Suche

... versucht jenen Knoten zu expandieren, der dem Ziel am nächsten liegt, mit der Begründung, dass dies wahrscheinlich schnell zu einer Lösung führt.
-> es wird der Knoten expandiert, der am nächsten zum Ziel zu sein scheint.
-> damit ist die **Heuristikfunktion gleich** der **Evaluierungsfunktion** -> h(n) = f(n)

Die Heuristik kann bewirken, dass man unnötige Knoten expandiert. -> erinnert an die Tiefensuche. -> Bewertung ist daher gleich schlecht wie die Tiefensuche:
Komplett: Nein.
Zeit: $O(b^m)$ -> eine gute Heuristik kann die Laufzeit stark verbessern
Speicher: $O(b^m)$ m = maximale Tiefe des Raums
Optimal: Nein.

A* Suche

... Minimierung der geschätzten Gesamtkosten für die Lösung

... **Idee**: das Expandieren von Pfaden die bereits teuer sind vermeiden

-> Es werden die Kosten zur **Erreichung des Ziels g(n)** mit den Kosten, um vom aktuellen Knoten zum Ziel zu gelangen h(n) verknüpft: **f(n)** = **g(n)** + **h(n)**.

Weil g(n) die Pfadkosten vom Ausgangsknoten angibt und h(n) die geschätzten Kosten des billigsten Pfades von n zum Ziel sind, erhalten wir f(n) als geschätzte Kosten für die billigste Lösung durch n.

Optimalität von A*

- Eine Heuristik ist akzeptabel, wenn:
  h(n) <= h*(n), h*(n) sind die wahren Kosten von n

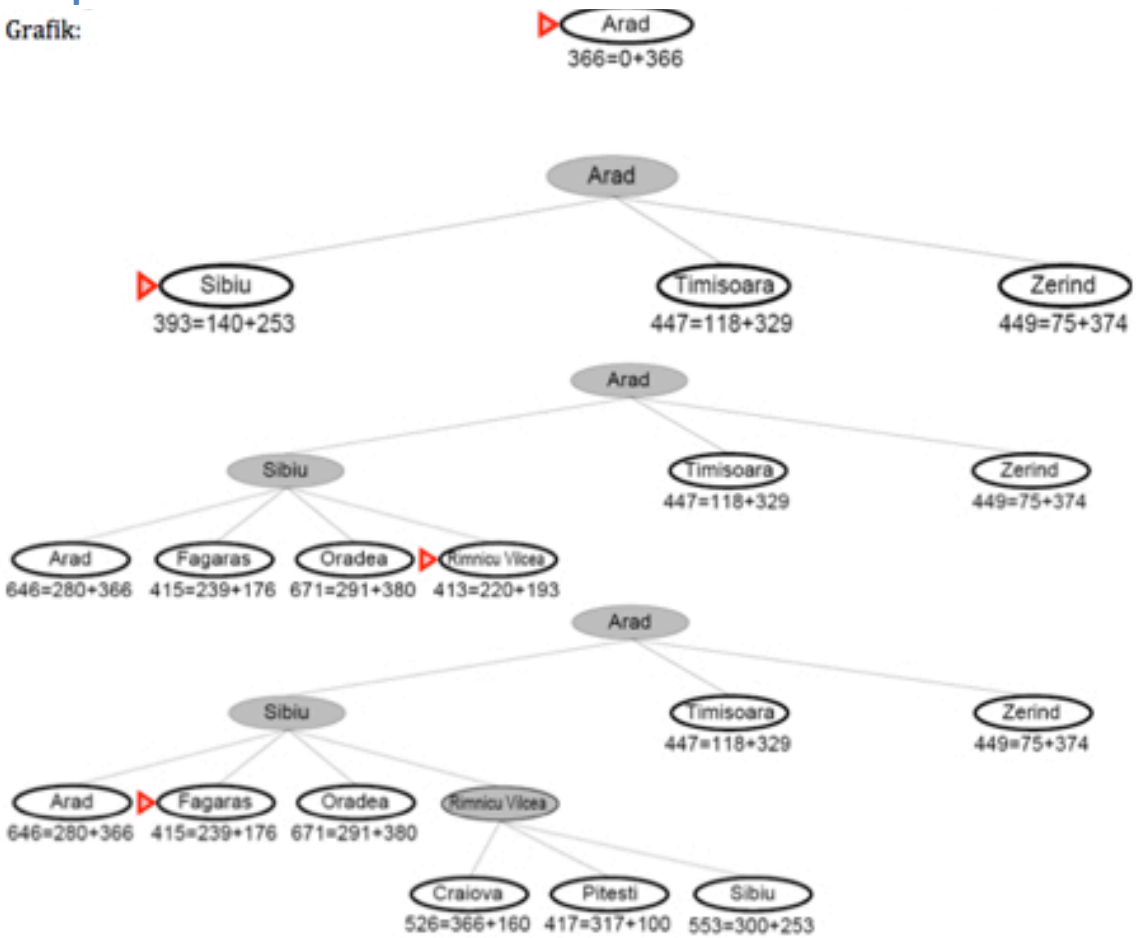
- **bei Tree-Search:**
  A* ist optimal, wenn es sich bei h(n) um eine **zulässige Heuristik** handelt, dh h(n) überschätzt nie die Kosten, das Ziel zu erreichen h()n <= h*(n).
  Da g(n) die exakten Kosten für das Erreichen von n darstellen, folgt dass f(n) die tatsächlichen Kosten einer Lösung durch n nicht überschätzt.

- **bei Graph-Search:**
  A* ist optimal, wenn h(n) **konsistent** ist, dh die Werte von f(n) sind entlang eines beliebigen Pfades nicht fallend (f-Kosten sind auf einem beliebigen Pfad nicht fallend). -> es können **Konturen** im Zustandsraum gezeichnet werden.

- da A* den **Randbereichsknoten** mit den kleinsten f-Kosten expandiert, sehen wir, dass sich eine A*-Suche vom Ausgangsknoten aus ausbreitet und Knoten in konzentrischen Bändern steigender f-Kosten hinzufügt
  -> Algorithmen, die Suchpfade von der Wurzel aus erweitern, sind A* optimal effizient für jede beliebige Heuristikfunktion.

Beispiel

Grafik:



usw.

Konsistenz - Proof of lemma

-> wenn h(n) konsistent ist, dann sind die Werte von f(n) entlang eines beliebigen Pfades nicht fallend.

Eine Heuristik ist Konsistent, wenn:

$h(n) <= c(n,a,n') + h(n')$

Beweis: Annahme n' ist ein Nachfolger von n, dann gilt für alle a:

$$
\begin{aligned}
f(n') &= g(n') + h(n') \\
&= g(n) + c(n,a,n') + h(n') \\
&>= g(n) + h(n) \\
&= f(n)
\end{aligned}
$$

-> Die Knotenfolge liegt in einer nicht fallenden Reihenfolge von f(n).

Bewertung A*

Komplett: ja, außer es gibt unendlich viele Knoten mit f<=f(g)
Zeit: exponentiell in: (relative Fehler in h * Lösungslänge)
Speicher: braucht alle Knoten im Speicher (größter Nachteil von A*) -> Speicher geht aus lange bevor Rechenzeit aus geht
Optimal: Ja.

Weitere akzeptable Heuristiken

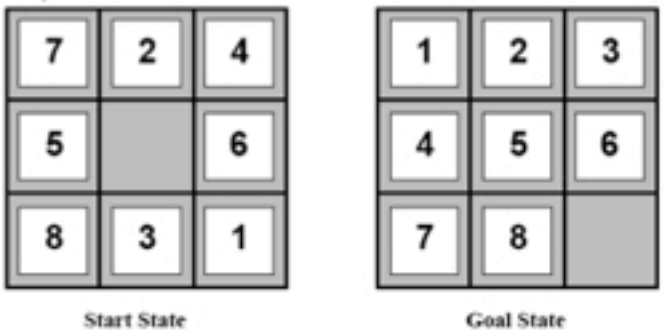
Speicherbegrenzte heuristische Suche

... um die Speicheranforderungen von A* zu reduzieren

... iterative Vertiefung wird in den heuristischen Suchkontext aufgenommen, die Kürzung erfolgt jedoch an den f-Kosten und nicht an der Tiefe

Nachteile: siehe iterative Version der Suche mit einheitlichen Kosten

Heuristiken für das 8-Felder Puzzle



h1 = Anzahl der falsch platzierten Felder
h2 = Summe der Distanz der Felder von ihren Zielpositionen = Manhatten Distanz

h1 des Spiels = 6
h2 des Spiels = 4+0+3+3+1+0+2+1 = 14

Dominanz

wenn h2 >= h1 ist, dann dominiert h2 h1 und ist besser für die Suche geeignet.

typische Suchkosten:

d=14	d=24
IDS = 3479141 Konten	IDS ≈ 54000000000 Konten
A*(h1) = 539 Knoten	A*(h1) = 39135 Konten
A*(h2) = 113 Knoten	A*(h2) = 1641 Konten

-> jede gegebene Heuristik ha, hb, für die gilt:
h = max(ha,hb) ist auch akzeptabel und dominiert ha, hb

Relaxed Problems / Gelockerte Probleme

... Problem welches weniger Beschränkung im Hinblick auf die Aktionen aufweist

... Kosten einer optimalen Lösung für ein gelockertes Problem sind eine zulässige Heuristik für das ursprüngliche Problem.

Beispiel 8-Felder Puzzle

Würden die Regeln für das Puzzle so geändert, dass ein Feld sich an eine beliebige Position bewegen könnte, würde h1 die genaue Anzahl der Felder für die kürzeste Lösung enthalten.
Könnte sich vergleichbar dazu ein Feld um ein Quadrat in jeder beliebigen Richtung bewegen, würde h2 die genaue Anzahl der Schritte für die kürzeste Lösung angeben.
-> die optimalen Lösungskosten eines entspannten Problems sind nicht größer als die optimalen Lösungskosten eines realen Problems!

Hillclimbing Algorithmus

... ist die grundlegendste lokale Suchtechnik. Bei jedem Schritt wird der aktuelle Knoten durch den besten Nachbarn ersetzt. Sie terminiert, wenn der „Gipfel“ erreicht ist.

```
function hillClimbing(problem) returns lokalesMaximum
current <- makeNode(problem)
loop do
      neighbor <- höchstwertiger Nachfolger von current
      if Value [neighbor] <= Value[currentNode] then return State[current]
      current <- neighbor
```

Kapitel 6 - Adversariale Suche / Game Playing

Im Gegensatz zur Spieltheorie haben in der AI Spiele in der Regeln eine spezielle Natur:
Spieltheoretiker sprechen von deterministischer, abwechselnder Reihenfolge unterliegender Zwei-Spieler-Nullsummen-Spiele mit vollständiger Information. In der AI sind das deterministische, vollständig beobachtbare Umgebungen, in denen sich zwei Agenten befinden, die abwechselnd handeln. Die Nutzwerte sind dabei am Ende des Spiels immer gleich groß und entgegengesetzt. Diese Gegnerschaft der Nutzenfunktion macht die Situation **adversial**.

Der Zustand eines Spiels ist einfach darzustellen:

- Agenten sind auf eine Anzahl von Aktionen begrenzt
- Ergebnisse sind durch Regeln definiert.

Spiele sind daher interessant, da sie schwierig zu lösen sind und in der realen Welt Spieler fordern eine Entscheidung zu treffen, selbst wenn die Entscheidung nicht möglich ist (Suchbaum bei Schach hat etwa 10^{154} Knoten).

Spieltypen

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble nuclear war

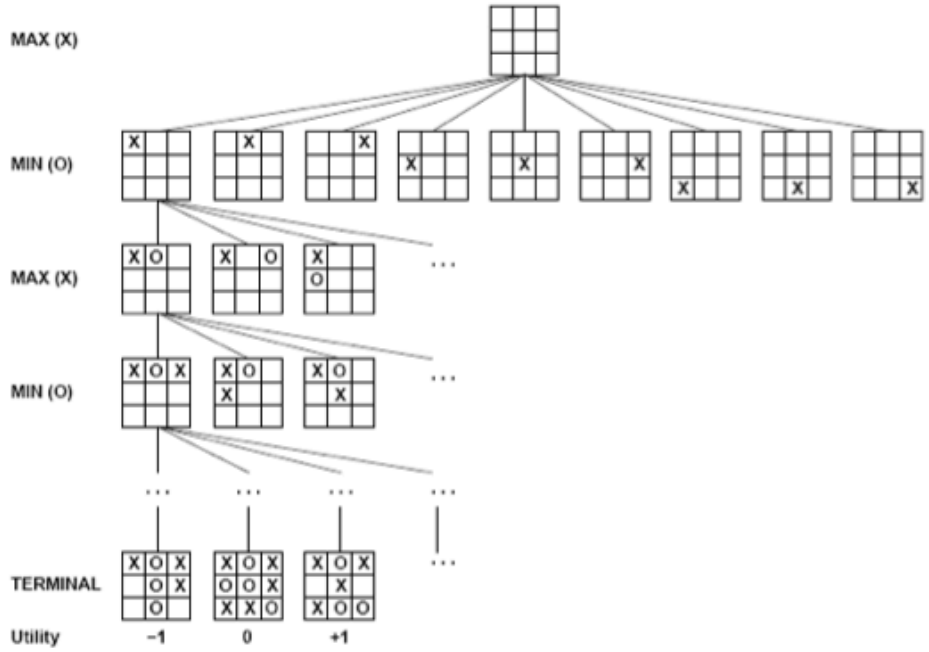
Formale Definition

= Suchproblem mit folgenden Komponenten

- Ausgangszustand
- Nachfolgerfunktion
- Endzustände
- Nutzenfunktion
- Spielbaum (definiert durch Ausgangszustand und erlaubte Züge)

Ziel: Eine optimale Strategie zu wählen, deren Ergebnis mindestens so gut ist wie jede andere Strategie, wenn man gegen einen unfehlbaren Gegner spielt.

Beispiel Spielbaum „Tic-Tac-Toe"



-> kompletter Spielbaum wäre sehr komplex selbst für ein einfaches Spiel wie Tic-Tac-Toe

Minimax

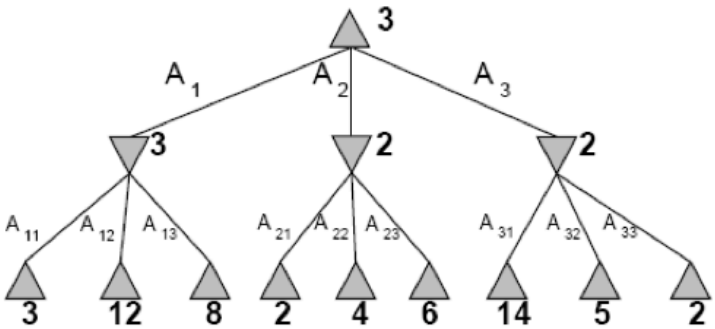
... Minimax Value n um die optimale Strategie eines Spielbaums zu errechnen

... für deterministische Spiele mit perfekter Information

Idee

wähle die Aktion mit dem höchsten n = der beste zu erreichende Payoff gegen das beste Spiel

Beispiel



∇ ... MIN-Knoten -> wählt minimales n (perfektes Spiel des Gegners wird simuliert)
Δ ... MAX-Knoten -> wählt maximales n (wir wollen unseren Nutzen maximieren)

-> MAX wählt als erstes A1, MIN wählt A11

Algorithmus

... berechnet die Minimax-Entscheidung aus dem aktuellen Zustand

... verwendet eine einfache rekursive Berechnung der Minimax-Werte jedes Nachfolgerzustandes
... führt eine vollständige Tiefensuche für den Spielbaum durch.

Komplett: Ja, bei einem endlichen Baum
Optimal: Ja, gegen einen optimalen Gegner (es wird der worst-case von MAX maximiert). Gegen einen nicht optimalen Gegner, wird MAX noch besser
Zeit: $O(b^m)$
Speicher: O(bm)
Anzahl der auszuwertenden Spielzustände ist exponentiell zur Anzahl der Züge, PROBLEM!!

Beispiel Schach: b=35, m=100 -> exakte Lösung unmöglich, aber muss wirklich jeder Pfad analysiert werden?? -> Alpha-Beta Kürzung

Alpha-Beta Kürzung / Pruning

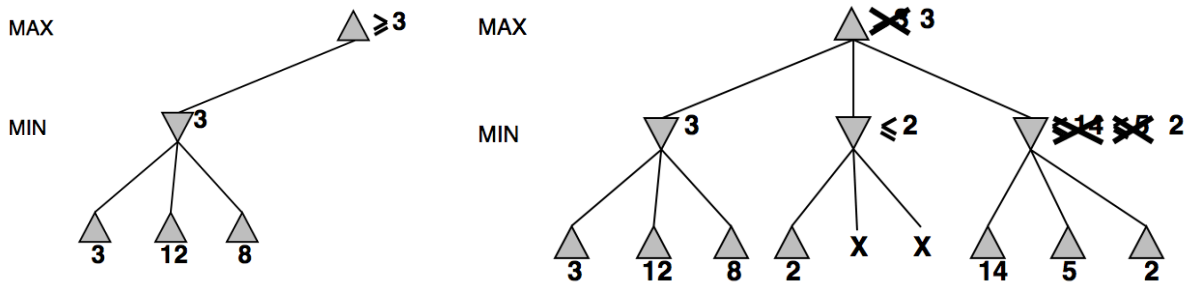
... Exponenten können nicht gänzlich eliminiert werden, aber man kann sie halbieren.
... anwendbar auf Bäume beliebiger Tiefe, häufig möglich nicht nur einzelne Blätter sondern Unterbäume weg zu kürzen.

Prinzip

Knoten n wird betrachtet, Spieler hat die Wahl diesen Knoten zu ziehen Hat Spieler bessere Wahl m am Elternknoten von n oder an einem anderen Punkt weiter oben, wird n niemals erreicht. Es werden also die Nachfolger von n betrachtet, um dieses Fazit zu erzielen. Danach kann n gekürzt werden.

alpha: der Wert der höchsten Wahl, die wir bisher an jedem Auswahlpunkt entlang des Pfades für MAX ermittelt haben.
beta: der Wert der niedrigsten, die wir bisher an jedem Auswahlpunkt entlang des Pfades für MIN ermittelt haben.

Beispiel



1. Nachfolgerknoten der Wurzel = B
MAX findet als Werte 3, 12 und 8 -> B ist genau 3 + Wurzel ist mindestens 3, weil MAX eine Auswahlmöglichkeit von 3 an der Wurzel hat.

2. Nachfolgerknoten der Wurzel = C
1. Blatt hat Wert 2 -> C ist mindestens 2 (MIN-Knoten)
B = 3 -> Wurzel würde nie C wählen -> keine weitere Nachfolger werden betrachtet

Analyse

- ...sinnvoll zuerst die Nachfolger zu testen, die wahrscheinlich die besten Werte liefern -> wesentlich ist die Reihenfolge -> Sortierung der besten Werte erhöht die Effektivität -> $O(b^{m/2})$ statt $O(b^m)$ = perfektes Pruning
- Schach: 35^{50} ist immer noch unmöglich
- Pruning verändert nicht das Endergebnis

Unvollständige Entscheidungen

Motivation

Minimax erzeugt einen ganzen Spielsuchbaum, Pruning erlaubt es zu kürzen, trotzdem muss der gesamte Weg bis zu den Endzuständen für zumindest einen Teil des Suchraumes durchsucht werden.

Idee

- Programm soll Suche in die Tiefe abbrechen und eine heuristische Bewertungsfunktion für Zustände anwenden (E val- statt Nutzenfunktion)
- Cutoff-Test statt Endtest zB Tiefenlimit

Bewertungsfunktionen - E val

...Schätzung des erwarteten Nutzens von einer bestimmten Position aus; genau so wie die Heuristikfunktion aus Kapitel 4 eine Schätzung der Distanz zum Ziel zurück gibt.

- Sollte Endzustände auf die gleiche Weise sortieren wie die echte Nutzenfunktion
- Sollte nicht zu lange für die Berechnung benötigen
- Sollte stark mit den tatsächlichen Wahrscheinlichkeiten zu gewinnen zusammen hängen

Die meisten Bewertungsfunktionen berechnen verschiedene Funktionsmerkmale des Zustandes. Zustände kann man in Kategorien mit gleichen Funktionsmerkmalen einteilen. Jede Kategorie kann Zustände enthalten, die zum Gewinn, Verlust oder Remis (zB bei Schach) führen. Die Bewertungsfunktion selbst weiß nicht welche wozu führen, aber sie kann einen Wert zurückgeben, der die Proportion der Zustände zu jedem Ergebnis reflektiert.

Beispiel für sinnvolle Bewertung: gewichteter Durchschnitt, Erwartungswert

Mathematisch: gewichtete Linearfunktion

Annahme: jedes Funktionsmerkmal ist unabhängig von den Werten der anderen Funktionsmerkmale

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

wi ... Gewichtung, fi ... Funktionsmerkmal

bei Schach: fi zB die Anzahl der Figurenarten auf dem Brett
wi zB Werte für die Figuren

Abbrechen der Suche - Cutoff
...E val soll Suche abbrechen sobald es sinnvoll ist

einfachster Ansatz:
Vorgabe einer festen Tiefe -> kann zu Fehlern führen

komplexerer Ansatz – Erweiterung der festen Tiefe
Bewertungsfunktion nur bei Positionen die ruhend sind (die in naher Zukunft keine wilden Wertänderungen aufweisen werden) -> nichtruhende Positionen sollen expandiert werden = **Ruhensuche**

zB bei Schach für ruhende Positionen: Positionen in denen günstige Figuren geschlagen werden können

Problem des Horizonteffekts: wenn das Programm einen Zug des Gegners erwartet, der ernsthafte Schäden verursacht, aber nicht mehr zu vermeiden ist.
Lösung: singuläre Erweiterungen nach Hardwareverbesserungen möglich (geht über normale Tiefenbeschränkuhng hinaus)

komplexerer Ansatz – Vorabkürzung
bestimmte Züge werden sofort gekürzt -> sehr gefährlich (es könnte der beste Zug gekürzt werden) -> nur in gewissen Situationen einsetzbar (zB bei 2 symmetrischen Zügen)

Spiele mit Zufallskomponente / nicht-deterministische Spiele
typisches Beispiel: **Backgammon** -> Zufall und Können wird kombiniert
Vor jedem Zug wird gewürfelt, um die erlaubten Züge festzulegen
-> Spielbaum muss auch **Zufallsknoten** (neben MIN und MAX) haben

Algorithmus
wie MINIMAX nur müssen auch die Zufallsknoten behandelt werden.
-> Zufallsknoten müssen bewertet werden (zB mit gewichtetem Durchschnitt der Werte)

```
EXPECTIMINIMAX (n) =
     - Utility (n)                       wenn n = Endzustand
     - max s EXPECTIMINIMAX (s)          wenn n = Max-Knoten
     - min s EXPECTIMINIMAX (s)          wenn n = Min-Knoten
     - Sum (P(s)*EXPECTIMINIMAX (s))     wenn n = Zufallsknoten
```

Komplexität
nur MINIMAX: O (b^m)
mit Zufall: O ($b^m n^m$) n ... Anzahl der Würfel
-> maximale 3 Schritte können voraus gesehen werden
-> alpha-beta pruning ist nicht mehr so effektiv

Spiele mit unvollkommener Information / Kartenspiele
typisches Beispiel: Kartenspiele, Karten des Gegner sind unbekannt

Ansatz / Idee

berechnen der Wahrscheinlichkeiten der Karten der anderen Spieler
-> berechnen des MINIMAX-Werts jeder Aktion mit jeder möglichen Karte -> Aktion auswählen die den höchsten Erwartungswert hat

Spiele in der Praxis

- **Dame**
  1952 erstes Programm, lernte seine eigene Bewertungsfunktion indem es gegen sich selbst spielte
  Programm „Chinook“ (mit alpha beta Suche) gegen Weltmeister Tinsley (seit 40 Jahren Weltmeister) w.o. von Tinsley

- **Schach**
  Programm „Deep Blue“ schlug Weltmeister Kasparov, es sucht 200 Millionen Positionen pro Sekunde

- **Othello / Reversi**
  Menschen haben bei Othello keine Chance gegen den Computer

- **Go**
  Computer sind noch zu schlecht, da bei Go der Verzweigungsfaktor bereits bei 361 beginnt

- **Backgammon**
  Programm TD-Gammon befindet sich unter den 3 besten Spielern der Welt

Kapitel 10 - Wissensrepräsentation

Was ist Wissensrepräsentation?

...sind gemeinsam mit Reasoning (Schlussfolgerungsprozesse) ein wesentlicher Teil der AI

Beispiele

- spielen eine wesentliche Rolle im Zusammenhang mit partiellen beobachtbaren Umgebungen
  -> ein wissensbasierter Agent kann allgemeines Wissen mit aktuellen Wahrnehmungen kombinieren, um verborgene Aspekte des Zustands abzuleiten, bevor Aktionen ausgewählt werden
  Beispiel: Art diagnostiziert Erkrankung, indem er Patienten nach Zustand befragt und auf sein Wissen zurückgreift (zB von Büchern oder der Ausbildung). Zusätzlich aber auch auf Wissen, das er nicht bewusst beschreiben kann (zB Erfahrung)

- Verständnis natürlicher Sprache
  -> Ableitung verborgener Zustände ist notwendig, verborgener Zustand ist zB die Absicht des Sprechers

Aufgabe / Ziel

Wissen so zu aufzuarbeiten, dass dieses Wissen auch von einem Computer verstanden werden kann.
-> mithilfe von Logik (formale Strukturen), Ontologie (definiert Objekte) und Informatik (unterstützt mit Applikationen)
-> Wissen wird in passende Datenstrukturen verwandelt

deklarativ vs. prozedural

- **deklarativ**
  Wissen wird in Sätzen in passender formaler Sprache ausgedrückt, Zugriff erfolgt durch Prozeduren die dieses Wissen benutzen
  -> explizite Repräsentation und Processing werden separiert
  Vorteil: Modularität (einfachere Änderungen), Flexibilität bei komplexen Aufgaben

- **prozedural**
  Wissen wird implizit als Sequenz von Operationen gespeichert, direkt im Programm-Code
  Vorteil: kann evtl. effizienter sein
  Beispiel: viele if-Schleifen Konstrukte

-> erfolgreiche Agenten kombinieren prozeduralen und deklarativen Ansatz

Wissensbasierte Agenten

Zentrale Komponente: **Wissensbasis:**

- = Menge an Sätzen in formaler Sprache
- Sätze in formaler Sprache sind Informationen über die Welt.

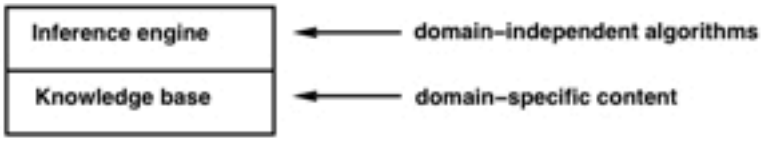
- Hinzufügen von neuen Sätzen zur Wissensbasis:
  - **Tell**-Funktion teilt mit was wahrgenommen wird
  - **Ask**-Funktion fragt Wissensbasis welche Aktion auszuführen ist (Schließen was gemacht werden soll)
  - Wahl wird mit **Tell** aufgezeichnet (damit Wissensbasis weiß ob hypothetische Aktion wirklich ausgeführt wurde)

- Weitere Funktionen
  - **Make-Percept-Sentence:** Input: Wahrnehmung und Zeit, gibt einen Satz zurück
  - **Make-Action-Query**: Zeit als Input, gibt einen Satz zurück der fragt welche der gewählten Aktionen ausgeführt werden soll
  - **Make-Action-Sentence**: Input: Aktion und Zeit, erklärt warum eine Aktion ausgeführt wurde

Zusammenfassung der benötigten Funktionen eines Agenten

Präsentation des Status und der Aktionen, neue Wahrnehmung repräsentieren, interne Darstellung der Welt verändern, versteckte Eigenschaften der Welt ableiten, geeignete Maßnahmen ableiten

Architektur



Logik

... ist eine formale Sprache um Information zu repräsentieren wie zB die Schlüsse die gezogen werden können

- Syntax definiert die Sätze der formalen Sprache
- Semantik definiert die Bedeutung der Sätze zB ob ein bestimmter Satz in einer Welt wahr ist.

Beispiel Arithmetik:

- x+2>y ist ein Satz, x+2y > ist kein Satz (Syntax)
- x+2>y ist wahr in einer Welt wo x=7 und y=1 ist (Semantik)

Nachteil von Datenstrukturen / Warum Logik?

- Programmiersprachen fehlt Mechanismus um aus Fakten andere Fakten abzuleiten. Jede Aktualisierung einer Datenstruktur erfolgt durch eine spezifische Prozedur (Details vom Programmierer und seinem Wissen abgeleitet)

- Datenstrukturen in Programmen fehlt eine Möglichkeit zB zu sagen „Es gibt einen freien Parkplatz in [1,12] ([Stock, Parkplatz])“ oder „Wenn Uwe Egly eine VO im Raum EI3 haltet ist er nicht im Raum HS13“.

- Aussagenlogik hat die Eigenschaft der Kompositionalität. In einer kompositionalen Sprache ist die Bedeutung eines Satzes eine Funktion der Bedeutung seiner Bestandteile. Dies gilt nicht für Datenstrukturen.

Logische Konsequenz (Entailment)

... um mithilfe von Logik Konsequenzen zu ziehen
-> logische Konsequenzen zwischen Sätzen = Idee, dass ein Satz logisch aus einem anderen folgt.

β ist die logische Konsequenz von α (α ≡ β oder α |= β), wenn β wahr in allen Welten wahr ist in denen auch α wahr ist

Beispiel: logische Konsequenz von x+y=4: 4=x+y

Modelle

Semantik kann auch mithilfe von Modellen (= formal strukturierte Welten) definiert sein.

Beispiel: m ist ein Modell eines Satzes α, wenn α in m wahr ist, M(α) ist dabei ein Set von allen Modellen von α.
-> zB KB |= α, wenn M(KB) ⊆ M(α) ist.

Weitere wichtige Begriffe

- **logische Äquivalenz** (siehe logische Konsequenz)
  zwei Sätze sind logisch äquivalent, wenn sie in der selben Modellmenge wahr sind

- **Gültigkeit**
  ein Satz ist gültig wenn er in allen Modellen wahr ist, gültige Sätze werden als Tautologie bezeichnet (sind immer wahr und damit semantisch ohne Aussagekraft)

- **Erfüllbarkeit**
  ein Satz ist erfüllbar, wenn er in irgendeinem Modell wahr ist

- **Unerfüllbarkeit**
  ein Satz ist unerfüllbar, wenn er in keinem Modell wahr ist

- **logische Inferenz**
  logische Konsequenz ist wie die Nadel im Heuhaufen zu finden; die Inferenz ist der Prozess sie zu finden
  wenn ein Inferenz-Algorithmus i α von KB ableiten kann:
  wenn KB |- α (α wird durch i von KB abgeleitet oder i leitet KB von α ab), dann ist auch KB |= α wahr.
  I**nferenz ist vollständig**, wenn jeder folgerbare Satz abgeleitet werden kann (wenn KB |= α imm KB |- α wahr ist)

- **Widerlegung / Widerspruch**

-> α ist gültig, wenn ¬α unerfüllbar ist.
-> KB |= α wenn KB ∪ {¬α} unerfüllbar ist (= reductio ad absurdum)

Wahrheitstabelle

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

Aussagenlogik

... nimmt an, dass die Welt aus Fakten besteht, die durch **Konnektoren** in atomaren Formeln ausgedrückt werden können

Konnektoren

Negation (¬), Konjunktion (∧), Disjunktion (∨), Implikation (⇒), Bikonditional (⇔)

Problem

... wie natürliche Sprache hat auch die Aussagenlogik nur sehr eingeschränkte Aussagefähigkeit

Beispiel: Folgendes kann nicht ausgedrückt werden:
Alle Superhelden sind tapfer. Superman ist ein Superheld. -> Superman ist tapfer.
In Aussagenlogik 3 atomare Sätze A, B, C. Aber: A, B |= C hält nicht!

-> FOL ist notwendig!

FOL - First Order Logic / Prädikatenlogik

... nimmt an, dass die Welt aus **Objekten** (Menschen, Häuser, Theorien, Superman, ...), **Relationen** (rot, Bruder von, größer als, ...) und **Funktionen** (Vater von, bester Freund, einer mehr als, ...) besteht.

Bestandteile

- Konstanten, (Superman)
- Prädikate (Freund, >)
- Funktionen (Wurzel)
- Variablen (x, y)
- Konnektoren (⇒, ⇔, ∧, ∨, ¬)
- Gleichheit (=)
- Quantoren
  ∀ = Allquantor – „für alle“
  ∃ = Existenzquantor – „für ein, es existiert ein“

Atomare Sätze

... Verweis auf Objekte und Relationen werden kombiniert.
`Prädikat(term1,...,termn) oder term1 = term2`
Term = `Funktion(term1,...,termn)` oder `Konstante` oder `Variable`

-> Ein atomarer Satz ist in einem bestimmten Modell unter einer bestimmten Interpretation wahr, wenn die durch das Prädikatensymbol angegebene Relation für die von den Argumenten angegebenen Objekte gilt.

Komplexe Sätze
... bestehen aus atomaren Sätzen die mit Konnektoren und Quantoren verbunden sind

Modelle
... Sätze sind wahr bezüglich einem Modell und einer Interpretation
... ein Modell besteht aus ≥ 1 Objekt und Relationen rund um dieses

für die **Interpretation** benötigt man 3 Elemente:
- Konstantensymbole (stehen für Objekte)
- Prädikatensymbole (stehen für Relationen)
- Funktionssymbole (stehen für Funktionen)

Übersetzung von natürlicher Sprache in Logik / Wissensengineering
= aufbauen einer Wissensbasis

... ist keine einfache Aufgabe und erfordert Erfahrung:
- Welt die übersetzt werden muss, muss verstanden werden
- implizites Wissen in der natürlichen Sprache muss explizit gemacht werden
- Doppeldeutigkeiten müssen aufgelöst werden

-> Statements müssen oftmals umschrieben werden um eine passende logische Struktur zu erreichen. Dies involviert:
- direkte Übersetzung von adäquaten Wörtern in logische Symbole
- Neuformulierung von Teilen um Doppeldeutigkeiten zu überlisten – damit kein Term mit mehr als einer Bedeutung falsch verwendet wird (= **fallacy of equivocation**)
- Gruppierungen von umschriebenen Statements ermitteln

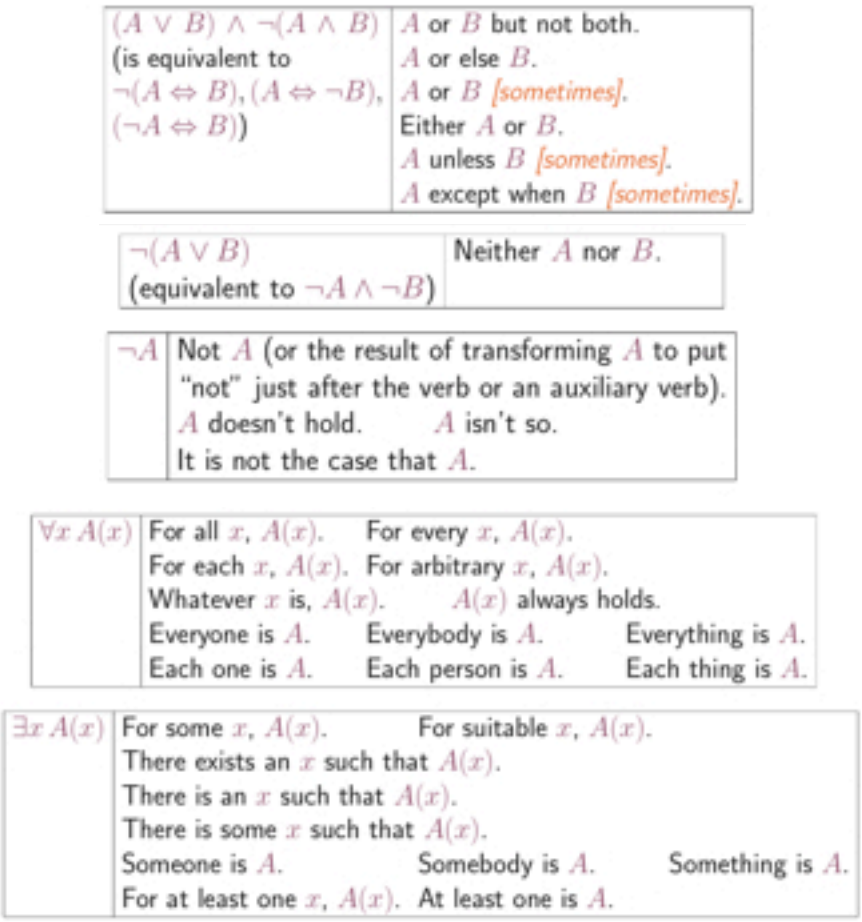
Übersetzung von Ausdrücke in Symbole und Quantoren

$A \Rightarrow B$	If A, then B.	B if A.	A only if B.
	When A, then B.	B when A.	A only when B.
	In case A, B.	B in case A.	A only in case B.
	B provided that A.		
	A is a sufficient condition for B.		
	B is a necessary condition for A.		
	A (materially) implies B.		

$A \Leftrightarrow B$	A if and only if B. (Abbreviation: A iff B.)
	A if B, and B if A.
	If A then B, and conversely.
	A exactly if B. A exactly when B. A just in case B.
	A is a necessary and sufficient condition for B.
	A is (materially) equivalent to B.

$A \wedge B$	A and B.	Both A and B.	
	A but B.	Not only A but B.	
	A although B.	A despite B.	A yet B. A while B.

$A \vee B$	A or B.	A and/or B *[in legal documents]*.	A or B or both.
	A unless B.	A except when B.	



$(A \vee B) \wedge \neg(A \wedge B)$ (is equivalent to $\neg(A \Leftrightarrow B), (A \Leftrightarrow \neg B), (\neg A \Leftrightarrow B)$)	A or B but not both. A or else B. A or B *[sometimes]*. Either A or B. A unless B *[sometimes]*. A except when B *[sometimes]*.
$\neg(A \vee B)$ (equivalent to $\neg A \wedge \neg B$)	Neither A nor B.
$\neg A$	Not A (or the result of transforming A to put "not" just after the verb or an auxiliary verb). A doesn't hold. A isn't so. It is not the case that A.
$\forall x\, A(x)$	For all x, $A(x)$. For every x, $A(x)$. For each x, $A(x)$. For arbitrary x, $A(x)$. Whatever x is, $A(x)$. $A(x)$ always holds. Everyone is A. Everybody is A. Everything is A. Each one is A. Each person is A. Each thing is A.
$\exists x\, A(x)$	For some x, $A(x)$. For suitable x, $A(x)$. There exists an x such that $A(x)$. There is an x such that $A(x)$. There is some x such that $A(x)$. Someone is A. Somebody is A. Something is A. For at least one x, $A(x)$. At least one is A.

Typische Fehler:

- **⇒ ist der Hauptkonnektor in Verbindung mit ∀ (nicht ∧!)**

  Beispiel: jeder in Berkeley ist intelligent: ∀ x (At(x, Berkeley) ⇒ Smart(x))

  falsch: ∀x At(x, Berkeley) ∧ Smart (x) – bedeutet, dass jeder in Berkeley ist und jeder intelligent ist

- **∧ ist der Hauptkonnektor in Verbindung mit ∃ (nicht ⇒!)**

  Beispiel: für ein S gilt P: ∃(x) (S(x) ∧ P(x))

Probleme während der Übersetzung

Einige Nuancen der Bedeutung können während der Übersetzung verloren gehen.

Beispiel: In der Aussagenlogik ist A ∧ B äquivalent zu B ∧ A. Die folgenden Sätze sind jedoch unterschiedlich zu interpretieren:
„Mary had a baby and got married." – „Mary got married and had a baby"
-> die vorgeschlagene zeitliche Folge geht bei der Übersetzung verloren.

Beispiel: Fallacy of Equivocation

„A feather is light. What is light cannot be dark. Therefore, a feather cannot be dark."

-> light wurde einmal als hell und einmal als leicht benutzt.
-> die Konjunktion beider Sätze ist inkonsistent!

Weitere Doppeldeutigkeiten

- die Verwendung des Artikels „**a**“ hängt vom Kontext ab
  „A child needs affection“ -> $\forall$ x (C(x) $\Rightarrow$ A(x))

  „A man climbed the Everest“ -> $\exists$ x (M(x) $\wedge$ E(x))

- die Verwendung von „**any**“ hängt vom Kontext ab
  -> in der Bedeutung „all“ und in der Bedeutung „some“ bei Negationen
  „I would do that for anyone“ -> $\forall$x A(x)
  „I would not do that for anyone“ -> $\neg\exists$ A(x)

Das Gruppierungsproblem bei komplexen Sätzen

... man sollte zuerst die äußerste Struktur analysieren und sich dann nach innen Schritt für Schritt vor arbeiten

Beispiel:
The guard searched all who entered the building except those who were accompanied by member of the firm.
Some of Fioreccio‘s men entered the building unaccompanied by anyone else.
The guard searched none of Fioreccio‘s men.
-> Conclusion: Some of Fioreccio‘s men were members of the firm

- **1. Klausel, 1. Satz**
  $\forall$x (x entered the building $\Rightarrow$ (x was searched by the guard except x was accompanied by some member of the firm))
- **Implikation miteinbeziehen**
  $\forall$x (x entered the building $\Rightarrow$ (x was searched by the guard $\vee$ x was accompanied by some member of the firm))
- **2. Klausel, 1. Satz**
  $\forall$x (x entered the building $\Rightarrow$ (x was searched by the guard $\vee$ $\exists$y (x was accompanied by y $\wedge$ y was a member of the firm)))
- **formal**
  $\forall$x (B(x) $\Rightarrow$ (S(x) $\vee$ $\exists$y (A(x,y) $\wedge$ M(y))))
- **2. Satz**
  $\exists$x (F(x) $\wedge$ B(x) $\wedge$ x was unaccompanied by anyone else)
- „x was unaccompanied by anyone else“ = „anyone accompanying x as one of Fioreccio‘s men“
  $\forall$y (A(x,y) $\Rightarrow$ F(y))
- **3. Satz**
  $\forall$x (F(x) $\Rightarrow$ $\neg$S(x)) und die Conclusion: $\exists$x (F(x) $\wedge$ M(x))
- **Man kann also zeigen:**
  $\forall$x (B(x) $\Rightarrow$ (S(x) $\vee$ $\exists$y (A(x,y) $\wedge$ M(y))))

  $\exists$x (F(x) $\wedge$ B(x) $\wedge$ $\forall$y (A(x,y) $\Rightarrow$ F(y)))

  $\forall$x (F(x) $\Rightarrow$ $\neg$S(x)) |= $\exists$x (F(x) $\wedge$ M(x))

Ontologisches Engineering

... verwandt zu Wissensengineering, bezieht sich aber auf einen größeren Gültigkeitsbereich -> allgemeinere Konzepte zB Aktionen, Zeit, physische Objekte, Glauben = allgemeines Gerüst = **obere Ontologie**

Motivation

FOL hat Probleme bestimmte Dinge der realen Welt abzubilden. Es gibt für fast alle Verallgemeinerungen Ausnahmen oder sie gelten nur zu einem gewissen Grad.

Beispiel: „Tomaten sind rot“ -> sinnvolle Regel, aber es gibt auch grüne, gelbe oder orange Tomaten.

Ziel einer Ontologie ist es auf jede Domäne angewendet werden zu können. Außerdem muss es in jeder anspruchsvollen Domäne möglich sein, verschiedene Wissensbereiche zu vereinheitlichen.

Kategorien und Objekte

Obwohl die Interaktion mit der Welt auf der Ebene einzelner Objekte stattfindet, wird häufig auf der Ebene von Kategorien geschlossen.

Beispiel: Einkäufer kann Ziel haben einen Basketball zu kaufen und nicht einen bestimmten Basketball „BB9“.

In FOL gibt es 2 Möglichkeiten Kategorien zu repräsentieren:

1) **Prädikat** Basketball(b)
2) oder Kategorie wird als **Objekt** reifiziert – Basketbälle
   -> Element (b, Basketbälle) – also: b $\in$ Basketbälle
   oder auch als Untermenge: Basketbälle $\subset$ Bälle

-> Kategorien kann man sich also als Menge ihrer Elemente oder als komplexeres Objekt vorstellen.

Vererbung

... um durch Kategorien die Wissensbasis organisieren zu können.

Unterklassenrelationen ordnen die Kategorien in einer **Taxonomie** oder taxonomischen Hierarchie an.

Beispiel: Bibliothekswesen hat eine Taxonomie aller Wissensbereiche entwickelt.

Kategorien in FOL

- Objekt ist Teil einer Kategorie: BB9 $\in$ Basketbälle
- Kategorie ist Unterklasse einer anderen Kategorie: Basketballs $\subset$ Balls
- Alle Elemente einer Kategorie haben die gleichen Eigenschaften: x $\in$ Basketbälle $\Rightarrow$ Round(x)
- Elemente einer Kategorie können durch einige Eigenschaften erkannt werden Orange (x) $\wedge$ Round (x) $\wedge$ Diameter (x)=9.5“ $\wedge$ x $\in$ Balls $\Rightarrow$ x $\in$ Basketballs
- Eine ganze Kategorie erbt Eigenschaften: Dogs $\in$ DomesticatedSpecies

Weitere Beziehungen zwischen Kategorien

- **Disjunktion**
  wenn 2 oder mehr Kategorien keine gemeinsamen Elemente haben
  Disjunkt(s) $\Leftrightarrow (\forall c1,c2 ((c1 \in s \wedge c2 \in s \wedge c1 \neq c2) \Rightarrow$ Schnittmenge(c1,c,2)={}))

  man kann Disjunktion in 2 Teile unterteilen:

- **Erschöpfende Zerlegung**
  zB Amerikaner, Kanadier sind Nordamerikaner -> Doppelstaats-bürgerschaften möglich (daher keine Partition)
  ErschöpfendeZerlegung (s, c) $\Leftrightarrow \forall$ i (i $\in$ c $\Leftrightarrow \exists$ c2 c2 $\in$ s $\wedge$ i $\in$ c2)

- **Partition**
  zB essbare Objekte, nicht essbare Objekte sind Objekte
  Partition (s, c) $\Leftrightarrow$ Disjunkt(s) $\wedge$ Erschöpfende Zerlegung (s, c)

Physische Zusammensetzung / TeilVon-Beziehung

... Objekt kann Teil eines anderen sein -> Hierarchien entstehen

BSP: TeilVon(Ried, Österreich), TeilVon(Österreich, Europa), TeilVon (Europa, Erde)

TeileVon Beziehung ist **transitiv** und **reflexiv**:
TeilVon (x,y) $\wedge$ TeilVon(y,z) $\Rightarrow$ TeilVon(x,z) – zB TeilVon(Ried, Erde)

Außerdem ist es möglich mit der TeilVon Beziehung **zusammengesetzte Objekte** zu definieren:
Konzept des **Bündels** (Objekte werden unstrukturiert zusammengesetzt). Beispiel: BündelVon({Apfel1, Apfel2, Apfel3}). Die 3 Äpfel sind dabei Teile und keine Elemente

formal: $\forall$x x $\in$ s $\Rightarrow$ TeilVon(x, BündelVon(s))

BündelVon(s) ist das kleinste Objekt, das diese Bedingung erfüllt. Also muss BündelVon(s) Teil jedes Objekts sein, das alle Elemente von s als seine Teile enthält:

$\forall$ y ($\forall$x x $\in$ s $\Rightarrow$ TeilVon(x, y)) $\Rightarrow$ TeilVon(BündelVon(s), y)

-> dies ist ein Beispiel für **logische Minimierung**, dh. es wird ein Objekt als kleinstes definiert, das eine bestimmte Bedingung erfüllt

Substanz und Objekte

... Teil der Realität der Individualisierung trotzt = Stoffe zB Butter (wenn man sie teilt erhält man 2 Stück Butter, nicht so zB bei Schnitzel)

Unterscheidung

- **intrinsisch (Stoffe)**
  gehört zu der eigentlichen Substanz des Objekts und nicht zu dem Objekt als Ganzen. Wenn man eine Stoffe teilt behalten die beiden Stücke die selben immanenten Eigenschaften (zB Dichte, Siedpunkt, etc.).
  = **Substanz** oder Massensubstantiv

- **extrinsisch (Dinge)**
  Eigenschaften wie Gewicht, Länge, Form, Funktion werden bei Unterteilung nicht beibehalten
  = **zählbares Substantiv**

-> alle physischen Objekte gehören zu beiden Kategorien, die Kategorien sind koexistent.

Situationskalkül

... um mehrere Kopien von Axiomen zu vermeiden
... statt mit expliziten Zeiten wie t+1 zu arbeiten, konzentriert man sich auf Situationen, die die Zustände angeben, die aus der Ausführung von Aktionen resultieren = Situationskalkül.

Situationskalkül **beinhaltet** folgende Ontologien:

- **Aktionen**
  logische Terme, zB Drehen(Rechts)

- **Situationen**
  logische Terme, die aus der Ausgangssituation und allen Situationen, die durch Anwendung der Aktion erzeugt werden, bestehen

- **Fluents**
  Funktionen und Prädikate, die von einer Situation zur nächsten variieren, zB Postition des Agenten, Alter(Superman, S0)

- **Atemporäre / ewige Prädikate**
  zB Prädikat Bananen(B1), Funktion LinkesBeinVon(Max)

Aktionsfolgen

... Ergebnisse von Aktionsfolgen können mithilfe der Ergebnisse der einzelnen Aktionen definiert werden:

- Ausführung einer leeren Folge belässt die Situation unverändert:
  Ergebnis([], s) = s

- Ausführen einer nicht-leeren Folge ist das gleiche wie die Ausführung der ersten Aktion und dann den Rest in der resultierenden Situation:
  Ergebnis([a|Folge], s) = Ergebnis (Folge, Ergebnis(a,s))

-> einfache Agenten sollten in der Lage sein:
- bestimmte Aktionsfolgen ableiten zu können = **Projektionsaufgabe**
- eine Folge von Aktionen zu finden, welche einen gewünschten Effekt erzielen = **Planungsaufgabe**

Beispiel: Bananen und Affen in 2-dimensionaler Welt
(x,y) = Position/Koordinaten
Affe: (1,1), Banane B1 (1,2) -> Ziel des Affen ist es, die Bananen auf Position (1,1) zu haben.

Fluent: Bei(o,x,s) and Halten(o,s)
Aktionen: Gehen(x,y), Nehmen(b), Loslassen(b)

Aussagen zu Beginn:
-> Bei(Affe, (1,1), S0) ∧ Bei(B1, (1,2), S0)
-> Bei(o,x,S0) ⇔ ((o=Affe ∧ x=(1,1)) ∨ (o=B1 ∧ x=(1,2))). ¬ Halten(o,S0)

-> Bananen(B1) ∧ Angrenzend((1,1),(1,2)) ∧ Angrenzend((1,2),(1,1)).

Ziel: beweisen, dass Affe sein Ziel erreichen wird indem er zu (1,2) geht, die Bananen nimmt und zu (1,1) zurück kehrt:
Bei(B1, (1,1), Ergebnis((Go((1,1),(1,2)), Grab(B1), Gehen((1,2),(1,1))), S0))

Interessanter ist aber die Frage „Welche Folge von Aktionen ergibt, dass die Bananen auf (1,1) sind?“
∃ Folge Bei(B1, (1,1), Ergebnis(Folge, S0)) -> wir benötigen mehr Informationen!

Beschreibung von Aktionen

jede Aktion kann mit 2 Axiomen beschrieben werden:

- **Möglichkeitsaxiom**: wann ist es möglich eine Aktion auszuführen?
  Vorbedingung ⇒ Möglichkeit(a,s)
- **Effektaxiom**: was passiert, wenn eine mögliche Aktion ausgeführt wird?
  Möglichkeit(a,s) ⇒ Änderungen

Beispiel: Bananen und Affen in 2-dimensionaler Welt
Annahmen:
s = Situationen über die Zeit, a = über Aktionen, o = über Objekte, b = über Bananen, x,y = Koordinaten

Folgende Möglichkeitsaxiome können aufgestellt werden:
Bei(Affe, x,y) ∧ Angrenzend(x,y) ⇒ Möglichkeit(Gehen(x,y), s).

Bananen(b) ∧ Bei(Affe, x,y) ∧ Bei(b,x,s) ⇒ Möglichkeit(Nehmen(b), s).

Halten(b,s) ⇒ Möglichkeit(Loslassen(b), s).

Folgende Effektaxiome können aufgestellt werden:
Möglichkeit(Gehen(x,y),s) ⇒ Bei(Affe, y, Ergebnis(Gehen(x,y), s)).

Möglichkeit(Nehmen(b),s) ⇒ Halten(b, Ergebnis(Nehmen(b),s)).

Möglichkeit(Loslassen(b), s) ⇒ ¬Halten(b, Ergebnis(Loslassen(b), s)).

Frame-Problem

= die Darstellung aller Dinge die gleich bleiben.

Statt die Effekte jeder Aktion zu formulieren, werden die einzelnen Fluent-Prädikate über die Zeit betrachtet = **Nachfolge-Zustand-Axiome**.

Allgemeine Form:
Aktion ist möglich ⇒ (Fluent ist im Ergebniszustand wahr ⇔ Effekt der Aktion hat Fluent wahr gemacht ∨ Fluent war zuvor wahr und Fluent hat ihn nicht verändert)

-> der nächste Zustand kann komplett über den Wert jedes Fluent spezifiziert werden (es sind keine zusätzlichen Frame Axiome notwendig)

Gesamtgröße der Axiome:O(AE) A...Aktion, E...Effekte

Beispiel für die Position des Affen
Möglichkeit(a,s) ⇒ (Bei(Affe,y,Ergebnis(a,s)) ⇔ a = Gehen(x,y) ∨ (Bei(Affe,y,s) ∧ a ≠ Gehen(y,z)))

-> Axiom ist genau dann wahr, wenn die rechte Seite gilt.

Ramifikationsproblem

... um das Affe-Bananen Beispiel zu lösen brauchen wir noch mehr Axiome:
Es muss ausgedrückt werden, dass ein **impliziter Effekt** der Bewegung des Affen von x nach y ist, dass sich die Banane die er trägt, ebenfalls bewegt.

-> Das Vorhandensein von impliziten Effekten wird als Ramifikaitonsproblem bezeichnet

Beispiel: Lösung durch ein allgemeines Nachfolge-Zustand Axiom für „Bei“
Möglichkeit(a,s) ⇒

Bei(o,y,Ergebnis(a,s)) ⇔

(a = Gehen(x,y) ∧ (o = Affe ∨ Halten(o,s))) ∨
(Bei(o,y,s) ∧ ¬(∃z y ≠ z ∧ a = Gehen (y,z) ∧ (o = Affe ∨ Halten(o,s))))

-> ein Objekt o ist auf Position y wenn o auf y gegangen ist und o ein Affe oder etwas das der Affe gehalten hat ist. Oder wenn o schon auf y ist und o nirgends hin gegangen ist, mit o dem Affen oder etwas das der Affe gehalten hat.

Eindeutige Namen

... FOL erlaubt, dass unterschiedliche Konstanten auf dasselbe Objekt verweisen
-> Wissensbasis muss ein Axiom enthalten, das dies verhindert.
-> Axiom der eindeutigen Namen legt die Ungleichheit für jedes Konstantenpaar in der Wissensbasis fest

Beispiel:
Ungleichheit zwischen Gehen((1,1),(1,2)) und Gehen((1,2),(1,1)) muss festgehalten werden:
für jedes Paar an Aktionsnamen soll gelten:
A(x1,...,xm) ≠ B(y1,...,ym).
Die 2 Aktionsterme mit demselben Aktionsnamen dürfen nur dann auf dieselbe Aktion verweisen, wenn sie alle die selben Objekte beinhalten:
A(x1,...,xm) = A(y1,...,ym) ⇔ x1=y1 ∧ ... ∧ xm=ym.

-> diese beiden Formeln sind die **Axiomen eindeutiger Aktionen**.

Kapitel 11 - Planen
= eine Folge von Aktionen finden, die zu einem Ziel führen

Klassische Planungsumgebung
...ist überschaubar, deterministisch, endlich, statisch (Änderungen nur durch den Agenten) und diskret (in Zeit, Aktion, Objekten, Effekten).

Suche vs. Planen
Suchalgorithmen sind nicht in der Lage große Probleme aus der realen Welt zu lösen. (würde zu enormen Suchraum führen – Agent wird mit irrelevanten Aktionen überhäuft)

Beispiel: Kauf eines Buches mit 10-stelliger ISBN über einen Online-Shop
-> 10^{10} mögliche Aktionen

Ein sensibler Planungsagent sollte aber in der Lage sein, von einer **expliziten Zielschreibung** (zB Haben(ISBN1234567891)) auszugehen und daraus rückwärts direkt die richtige Aktion Kaufen (1234567891) zu erzeugen. Beispielsweise mit dem allgemeinen Wissen des Agenten: Kaufen(x) führt zu Haben(x)
-> es ist nur ein Unifikationsschritt notwendig

Nächste Schwierigkeit: gute Heuristikfunktion finden
zB Agent hat Ziel 4 verschiedene Bücher zu kaufen -> 10^{40} Pläne -> Suche ohne Heuristik nicht möglich

Sprache der Planungsprobleme
... Sprache muss ausdrucksstark genug sein um die Probleme beschreiben zu können und einschränkend genug um effiziente Algorithmen zuzulassen.

STRIPS (Standford Research Institute Problem Solver) - Syntax
- **Repräsentation von Zuständen**
  Planer zerlegen Welt in logische Bedingungen, Zustand = Konjunktion positiver Literale (zB Reich $\wedge$ imGefängnis)
  - können in FOL (aber: muss funktionsfrei und nur Grundliterale) oder in Aussagenlogik sein. zB Bei(x,y) ist nicht erlaubt
  - Welt ist geschlossen (alles was nicht definiert ist, ist falsch)

- **Repräsentation von Zielen**
  Ziel ist ein partiell spezifizierter Zustand als Konjunktion positiver Grundliterale (zB Reich $\wedge$ Berühmt oder Bei(P2, Tahiti))
  - ein Zustand s erfüllt ein Ziel g wenn s alle Atome in g enthält. (zB Reich $\wedge$ Berühmt $\wedge$ imGefängnis erfüllt das Ziel Reich $\wedge$ Berühmt)

- **Repräsentation von Aktionen**
  Aktion wird im Hinblick auf die Vorbedingungen und Effekte (die bei der Ausführung entstehen) spezifiziert. Vorbedingungen müssen gelten damit Aktion ausgeführt werden kann.

- Beispiel: Aktion(Fliegen(p, von, nach))
  PRECOND: Bei(p,von) ∧ Flugzeug(p) ∧ Flugzeug(nach)
  EFFECT: ¬Bei(p,von) ∧ Bei(p, nach)

- **Aktionsschema**
  genauer genommen ist obiges Beispiel ein Aktionsschema (mehrere Aktionen entstehen durch das Instantiieren der Variablen p, von, nach).
  Aktionsschema besteht aus 3 Teilen:
  - Aktionsname und Parameterliste
  - Vorbedingung als Konjunktion funktionsfreier positiver Literale, die wahr sein müssen bevor die Aktion ausgeführt werden kann
  - Effekt als Konjunktion funktionsfreier Literale, die die Zustandsveränderungen beschreiben

STRIPS (Standford Research Institute Problem Solver) - Semantik

- eine Aktion ist in jedem Zustand anwendbar, in dem die Vorbedingung erfüllt ist, sonst hat die Aktion keinen Effekt.

- Beginnend mit dem Zustand s ist das Ergebnis der Ausführung eine Aktionsfolge, die bei Ausführen im Ausgangszustand zu einem Zustand führt, der das Ziel erfüllt.

- STRIPS Annahme: jedes Literal das nicht angeführt ist, bleibt unverändert

- Wenn ein positiver Effekt bereits im Zustand s ist, wird er nicht ein zweites Mal hinzugefügt. Das gleiche bei einem negativen Effekt: Wenn ein negativer Effekt nicht in s ist, wird dieser Teil des Effekts ignoriert.

Probleme von STRIPS

Ausdruckskraft ist für einige reale Domänen nicht ausreichend
-> viele Varianten der Sprache wurden entwickelt zB **ADL (Action Description Language)**

Weitere Möglichkeiten in ADL: positive und negative Literale in Zuständen, offene Welt Annahme, Ziele erlauben auch Disjunktionen, Effekte mit Bedingung sind erlaubt, Gleichheit ist möglich, Quantoren sind erlaubt,...

Probleme von STRIPS und ADL
Ramifikationen können nicht natürlich abgebildet werden, Qualifikationsproblem wird nicht gelöst

Beispiel für STRIPS - Luftfrachttransport

3 Aktionen: Laden, Entladen, Fliegen
Aktionen betreffen 2 Prädikate: In(plane, cargo), Bei(x,airport)

$Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK) \land Cargo(C_1) \land$
$Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \land Airport(SFO) \land Airport(JFK))$

$Goal(At(C_1, JFK) \land At(C_2, SFO))$

$Action(Load(c, p, a),$
PRECOND: $At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a),$
EFFECT: $\neg At(c, a) \land In(c, p))$

$Action(Unload(c, p, a),$
PRECOND: $In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a),$
EFFECT: $At(c, a) \land \neg In(c, p))$

$Action(Fly(p, from, to),$
PRECOND: $At(p, from) \land Plane(p) \land Airport(from) \land Airport(to),$
EFFECT: $\neg At(p, from) \land At(p, to))$

Eine Lösung:

$[Load(C_1, P_1, SFO), Fly(P_1, SFO, JFK), Unload(C_1, P_1, JFK),$
$Load(C_2, P_2, JFK), Fly(P_2, JFK, SFO), Unload(C_2, P_2, SFO)].$

Init	$At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK) \land$ $Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \land Airport(SFO) \land Airport(JFK))$
$\Downarrow$	$Action(Load(C_1, P_1, SFO),$ PRECOND: $At(C_1, SFO) \land At(P_1, SFO) \land Cargo(C_1) \land Plane(P_1) \land Airport(SFO),$ EFFECT: $\neg At(C_1, SFO) \land In(C_1, P_1))$
s_1	$In(C_1, P_1) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK) \land$ $Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \land Airport(SFO) \land Airport(JFK))$
$\Downarrow$	$Action(Fly(P_1, SFO, JFK),$ PRECOND: $At(P_1, SFO) \land Plane(P_1) \land Airport(SFO) \land Airport(JFK),$ EFFECT: $\neg At(P_1, SFO) \land At(P_1, JFK))$
s_2	$In(C_1, P_1) \land At(C_2, JFK) \land At(P_1, JFK) \land At(P_2, JFK) \land$ $Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \land Airport(SFO) \land Airport(JFK))$
$\Downarrow$	$Action(Unload(C_1, P_1, JFK),$ PRECOND: $In(C_1, P_1) \land At(P_1, JFK) \land Cargo(C_1) \land Plane(P_1) \land Airport(JFK),$ EFFECT: $At(C_1, JFK) \land \neg In(C_1, P_1))$
s_3	$At(C_1, JFK) \land At(C_2, JFK) \land At(P_1, JFK) \land At(P_2, JFK) \land$ $Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \land Airport(SFO) \land Airport(JFK))$

s_3	$At(C_1, JFK) \land At(C_2, JFK) \land At(P_1, JFK) \land At(P_2, JFK) \land$ $Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \land Airport(SFO) \land Airport(JFK))$
$\Downarrow$	$Action(Load(C_2, P_2, JFK),$ PRECOND: $At(C_2, JFK) \land At(P_2, JFK) \land Cargo(C_2) \land Plane(P_2) \land Airport(JFK),$ EFFECT: $\neg At(C_2, JFK) \land In(C_2, P_2))$
s_4	$At(C_1, JFK) \land In(C_2, P_2) \land At(P_1, JFK) \land At(P_2, JFK) \land$ $Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \land Airport(SFO) \land Airport(JFK))$
$\Downarrow$	$Action(Fly(P_2, JFK, SFO),$ PRECOND: $At(P_2, JFK) \land Plane(P_2) \land Airport(JFK) \land Airport(SFO),$ EFFECT: $\neg At(P_2, JFK) \land At(P_2, SFO))$
s_5	$At(C_1, JFK) \land In(C_2, P_2) \land At(P_1, JFK) \land At(P_2, SFO) \land$ $Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \land Airport(SFO) \land Airport(JFK))$
$\Downarrow$	$Action(Unload(C_2, P_2, SFO),$ PRECOND: $In(C_2, P_2) \land At(P_2, SFO) \land Cargo(C_2) \land Plane(P_2) \land Airport(SFO),$ EFFECT: $At(C_2, SFO) \land \neg In(C_2, P_2))$
s_6	$At(C_1, JFK) \land At(C_2, SFO) \land At(P_1, JFK) \land At(P_2, SFO) \land$ $Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \land Airport(SFO) \land Airport(JFK))$

$\Longrightarrow$ s_5 satisfies the goal $At(C_1, JFK) \land At(C_2, SFO)$.

Planungsalgorithmen

Planen mit Zustansraumsuche

... einfachster Ansatz. Man unterscheidet:

- **Ausgangszustand** des Planungsproblems
  jeder Zustand wird mit positiven Grundliteralen gesetzt, Literale die nicht vorkommen sind falsch

- **Aktionen**
  alle Aktionen können angewendet werden, deren Vorbedingung erfüllt ist. Nachfolgezustand wird erzeugt indem alle positiven Effektliterale hinzugefügt und alle negativen entfernt werden.

- **Zieltest** überprüft ob der Zustand das Ziel des Planungsproblems erfüllt

- **Schrittkosten** sind normalerweise für jede Aktion = 1 (≠ 1 möglich aber selten in STRIPS)

- Beim Fehlen von Funktionssymbolen: **Zustandsraum ist endlich** -> jeder Graphsuchalgorithmus der vollständig ist ist ein vollständiger Planungsalgorithmus

2 Möglichkeiten der Suche:

- **Vorwärts-Zustandsraumsuche / Progressionsplanen**
  beginnend vom Ausgangszustand werde die Aktionsfolgen betrachtet bis eine Folge gefunden ist, die den Zielzustand erreicht.

- **Rückwärts-Zustandsraumsuche / Regressionsplanen**
  Vorteil: es werden nur relevante Aktionen berücksichtigt
  Eine Aktion ist dann für ein konjunktives Ziel relevant, wenn sie eines der Konjunkte des Ziels enthält.
  - **Beispiel: Luftfrachttransport mit 10 Flughäfen, 20 Frachtstücken zum Flughafen B bringen**
    Bei(C1,B) ∧ ... ∧ Bei(C20,B)
    - Bei(C1,B) wird betrachtet
    - Aktionen suchen, die diesen Effekt haben: nur Entladen(C1,p,B) wird gefunden (Flugzeug p ist nicht spezifiziert)
    - die Aktion geht nur wenn die Vorbedingungen erfüllt sind: In(C1,p) ∧ Bei(p, B).
    - Außerdem sollte das Unterziel Bei(C1,B) in jedem Vorgängerzustand nicht wahr sein (sonst wäre die Aktion irrelevant)
    - -> Vorgängerbeschreibung: = Definition der Relevanz
      In(C1, p) ∧ Bei(p,B) ∧ Bei(C2,B) ∧ ... ∧ Bei(C20,B)
    - Es muss sichergestellt werden, dass keine die Aktionen keine gewünschten Literale rückgängig machen. zB Laden(C2,p) ist inkonsistent mit dem aktuellen Ziel, weil sie Bei(C2,B) negiert.
      = Definition der Konsistenz

- Nun kann der Prozess beschrieben werden, wie die Rückwärtssuche Vorgänger erzeugt:
  -> alle positiven Effekte von A in G werden gelöscht
  -> jede Vorbedingung wird zu A hinzugefügt (außer schon da)
  G ... Beschreibung, A...Aktion (relevant und konsistent)

Partiell Ordnendes Planen

... Planen mit Zustansraumsuche ist eine Form der vollständigen ordnenen Plansuche (es werden nur streng linearen Aktionsfolgen betrachtet)

Jeder Planungsalgorithmus der zwei Aktionen in einem Plan setzen kann **ohne die Reihenfolge** festzulegen, ist ein partiell ordnender Plan.

Beispiel

$Init()$

$Goal(RightShoeOn \wedge LeftShoeOn)$

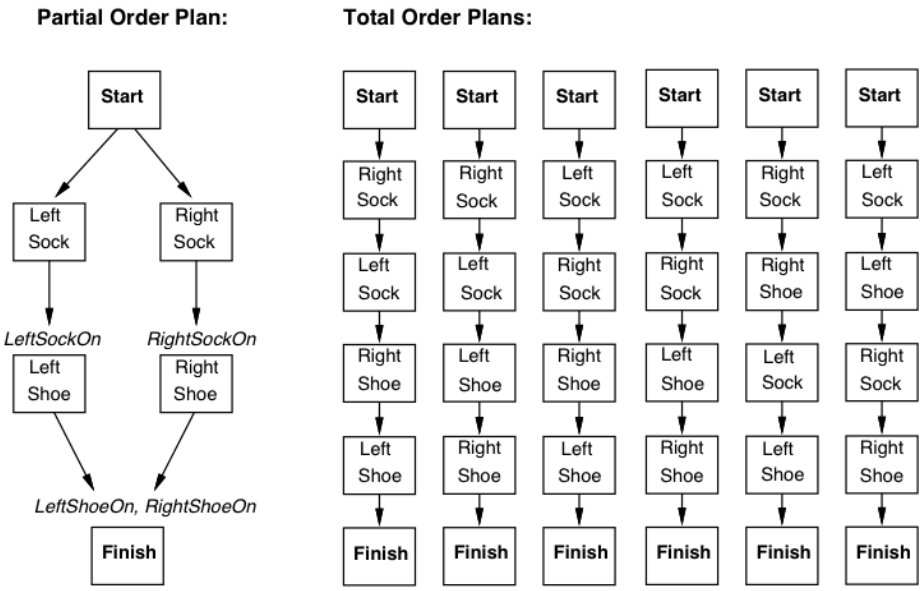
$Action(RightShoe, \text{PRECOND} : RightSockOn, \text{EFFECT} : RightShoeOn)$

$Action(RightSock, \text{EFFECT} : RightSockOn)$

$Action(LeftShoe, \text{PRECOND} : LeftSockOn, \text{EFFECT} : LeftShoeOn)$

$Action(LeftSock, \text{EFFECT} : LeftSockOn)$

Ein partielle ordnender Plan sollte nur 2 Aktionssequenzen betrachten:
(RightSock, RightShoe) um den ersten Teil des Konjunkts zu erreichen
(LeftSock, LeftShoe) für den zweiten Teil
-> Diese 2 Sequenzen können dann kombiniert werden um den finalen Plan zu erreichen. Dabei werden die beiden Subsequenzen **unabhängig** manipuliert.



Implementierung

... als Suche im Raum geordneter Pläne möglich

Bestandteile

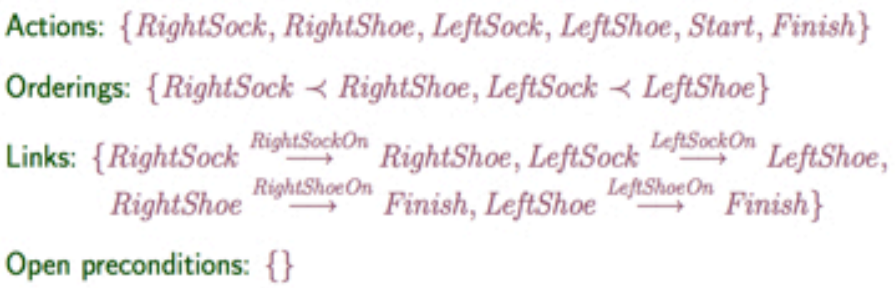
- Menge von **Aktionen**
  leerer Plan: Start (keine Vorbedingungen) und Ende (keine Effekte, Vorbedingung sind die Zielliterale)

- Menge von **Ordnungsbedingungen**
  A ∠ B (A vor B) bedeutet, dass Aktion A vor der Aktion B stattfindet. Jeder Zyklus (zB A ∠ B, B ∠ A) wäre ein Widerspruch

- Menge von **kausalen Verknüpfungen**
  A $^{p}\rightarrow$ B (A erzielt p für B) bedeutet, dass in der Verknüpfung zwischen A und B p immer wahr sein muss.

- Menge von **offenen Vorbedingungen**
  Vorbedingung ist offen, wenn sie nicht durch eine Aktion im Plan erzielt wurde -> es wird versucht offene Vorbedingungen auf die leere Menge zu reduzieren.

Beispiel für die Komponenten: Schuhe und Socken



Lösung
ein konsistenter Plan wird als Plan definiert der keine Zyklen in den Ordnungsbedingungen und keine Konflikte in den kausalen Bedingungen enthält.
-> ein partiell ordnender Plan wird durch wiederholtes aussuchen einer der möglichen nachfolgenden Aktionen ausgeführt.

POP-Algorithmus

- **Ausgangszustand** enthält
  Start/Ende (Aktionen), Start ∠ Ende (Ordnungsbedingungen), keine kausalen Verknüpfungen, Vorbedingungen von Ende sind offen

- **Nachfolgefunktion** wählt zufällig
  - eine offene Vorbedingung p für eine Aktion B und
  - generiert einen Nachfolgerplan für jede mögliche konsistente Weise eine Aktion A auszuwählen, die p erzielt.

- **Konsistenz** wird dabei erzwungen:
  - kausale Verknüpfung A $^{p}\rightarrow$ B und Ordnungsbedingung A ∠ B werden hinzugefügt. A kann eine vorhandene oder eine neue Aktion sein (bei einer neuen wird sie in den Plan hinzugefügt: Start ∠ A, A ∠ Ende)
  - mögliche entstehende Konflikte zB zwischen A $^{p}\rightarrow$ B und C (C ist neue Aktion) muss aufgelöst werden, indemC irgendwann außerhalb des

Schutzintervalls ausgeführt wird (durch Hinzufügen von B ∠ C oder C ∠ A, je nachdem welche(s) zu konsistenten Plänen führt)

- **Zieltest** prüft ob ein Plan eine Lösung für das Problem ist, indem er lediglich überprüft ob es noch offene Vorbedingungen gibt (möglich weil nur konsistente Pläne generiert werden)

GraphPlan-System

... basiert auf einem Graph als Datenstruktur (=Planungsgraph) um Pläne zu extrahieren

Planungsgraph beinhaltet eine Folge von Layern die den einzelnen Schritten in der Zeit entsprechen. Jeder Layer hat ein Superset an Literalen und Aktionen, die zu einer bestimmten Zeit eintreffen können. Außerdem implementiert jeder Layer gegenseitig ausschließende Verknüpfungen (= Mutex-Verknüpfungen) unter den Literalen und Aktionen.
Der Graph-Plan-Algortithmus erlaubt es aus einem Plan einen Palnungsgraphen zu extrahieren.

Kapitel 13/14 - Unsicherheit

Derzeitiges Problem / Motivation

Aussagen können nur wahr, falsch oder unbekannt sein
-> wenn Agent genug über die Umwelt weiß, kann er garantiert funktionierende Pläne ableiten
-> Annahme von klassischer Logik ist, alles über die Domäne zu wissen

Agenten wissen jedoch nie die ganze Wahrheit über die Umwelt
-> Agenten müssen unter Unsicherheit handeln

Für einen logischen Agenten kann es unmöglich werden eine komplette und korrekte Lösung zu generieren.
(zB A90 ist Aktion um 90 Minuten von zu Hause zum Flughafen einplanen zu müssen um einen Flieger zu erwischen. Auf der Fahrt kann es jedoch zB zu einem Stau kommen)
= **Qualifikationsproblem** (Agent kann nicht schließen und somit ist die Auflistung aller Vorbedingungen, damit eine Aktion in der realen Welt einen gewünschten Effekt erzielt hat unmöglich)

-> die richtige Aktion ist sowohl von der relativen Wichtigkeit der verschiedenen Ziele als auch von der Wahrscheinlichkeit, in welchem Maße sie erreicht werden können, abhängig

Umgang mit unsicherem Wissen

Beispiel FOL: Modellieren eines medizinischen Diagnoseproblems

Regel: $\forall$ p Symptom(p,Zahnschmerzen) $\Rightarrow$ Krankheit(p,Loch)

Problem ist, dass die Regel falsch ist, da nicht alle mit Zahnschmerzen auch ein Loch haben:
$\forall$ p Symptom(p,Zahnschmerzen) $\Rightarrow$ Krankheit(p,Loch) $\vee$ Krankheit (p, Abszess) $\vee$ ...

-> damit die Regel vollständig ist brauchen war fast unendlich viele Gründe

Erster Lösungsansatz: Regel in eine kausale Regel verwandeln
$\forall$ p Krankheit(p,Loch) $\Rightarrow$ Symptom(p,Zahnschmerzen)

-> wieder falsch, da nicht alle Löcher Schmerzen verursachen

Lösung: die einzige Möglichkeit ist es, sie logisch erschöpfend zu formulieren: Die linke Seite muss mit allen Qualifikationen, die gelten müssen, erweitert werden, damit ein Loch Zahnschmerzen verursacht:
$\forall$ p Krankheit(p,Loch) $\wedge$ Q1 $\wedge$ Q2 $\wedge$ ... $\Rightarrow$ Symptom(p,Zahnschmerzen)

-> Diese Qualifikationsliste ist jedoch wieder eine Form des Qualifikationsproblems

Gründe für das Fehlschlagen / für nicht komplettes Wissen

- **Faulheit**
  zu viel Arbeit, die vollständige Menge an Antezedenzen oder Konsequenzen aufzulisten

- **Theoretisches Unwissen**
  zB Medizin hat kein vollständiges Wissen über die Domäne

- **Praktisches Unwissen**
  selbst wenn alle Regeln bekannt sind, kann man bei zB einem bestimmten Patienten noch unsicher sein, weil zB nicht alle erforderlichen Tests ausgeführt wurden (werden können)

-> mit Wahrscheinlichkeiten können Faulheit und Unwissen abgedeckt werden

Grundlegende Notation für Wahrscheinlichkeiten

= Erweiterung der Aussagenlogik

Grundlegendes Element = Zufallsvariable

- **Namen** von Zufallsvariablen werden groß geschrieben, zB Loch

- jede Zufallsvariable hat eine **Domäne** (klein geschrieben)
  -> kann die Zufallsvariable annehmen
  zB Loch = true

Zufallsvariablen können in 3 Arten unterteilt werden:

- **Boolsche Zufallsvariablen**
  haben die Domäne (true, false), zB Loch = true oder auch als loch bzw. ¬loch geschrieben

- **Diskrete Zufallsvariablen**
  Werte sind abzählbare, sich wechselseitig ausschließende und erschöpfende Domänen, zB Wetter: (sonnig, regnerisch, wolkig,...)

- **Stetige Zufallsvariablen**
  Werte sind reelle Zahlen, entweder gesamte Menge oder auch einzelne Intervalle

Atomare Ereignisse

= vollständige Beschreibung des Zustands der Welt, über den der Agent unsicher ist.

Beispiel: Welt besteht aus nur 2 boolschen Variablen Loch und Zahschmerzen. Es gibt daher nur 4 verschiedene atomare Ereignisse. Eines davon ist:

loch ∧ zahnschmerzen

Eigenschaften von atomaren Ereignissen

- schließen sich wechselseitig aus -> es kann nur max. 1 Ereignis der Fall sein

- Menge aller atomaren ist erschöpfend -> es muss mind. 1 der Fall sein -> Disjunktion aller atomaren Ereignisse = true

- aus jedem atomaren Ereignis folgt true oder false für jede Aussage

- jede Aussage ist logisch äquivalent mit der Disjunktion aller atomaren Ereignisse, aus denen true/false logisch folgt, zB loch ist äquivalent mit loch ∧ zahnschmerzen und mit loch ∧ ¬zahnschmerzen

Unbedingte Wahrscheinlichkeit / A-priori Wahrscheinlichkeit

einer Aussage a wird eine Wahrscheinlichkeit zugeordnet, formal: P(a)

Beispiel:
P(Wetter=sonnig) = 0.7, P(Wetter=regnerisch) = 0.2 ,...
oder in Vektorform: P(Wetter) = ⟨0.7, 0.2, ...⟩

Gemeinsame Wahrscheinlichkeitsverteilung

Vektor V1,...,Vn ist ein Mapping P, wobei jedem ein Wert P(w) zugewiesen wird:
w = (V1=v1,...,Vn=vn)
-> $0 \leq P(w) \leq 1$, $\sum P(w) = 1$.

Notationen

- w |= A bedeutet: Aussage A ist wahr im atomaren Ereignis w
- P(V) bedeutet: P(V1,...,Vn)
- Vektoren V=v sind atomare Ereignisse: V1=v1,...Vn=vn

Marginalisierung

= Aufsummierung einer Untermenge einer gegebenen Menge an Variablen
gegeben: P(V,W), zu berechnen: P(V)

$P(V) = \sum P(V,w)$

Beispiel: P(loch) = P(loch,¬zahnschmerzen) + P(loch,zahnschmerzen)
= 0.108 + 0.072 = 0.18

Wahrscheinlichkeiten komplexer Aussagen

aus der gemeinsamen Wahrscheinlichkeitsverteilung kann man ableiten:
P(true)=1, P(false)=0, $P(a \vee b)=P(a)+P(b)-P(a \wedge b)$, $P(a \wedge b)=P(a,b)$, $P(\neg a)=1-P(a)$

Bedingte Wahrscheinlichkeit

P(a|b) ... Wahrscheinlichkeit von a unter der Bedingung b (b zu wissen)

Kann folgendermaßen berechnet werden:
$P(a|b) = P(a,b) / P(b)$

Dies gilt auch für mehrere Variablen zB boolsche Variable B, T, A, W:
$P(b,\neg t \mid \neg a,w) = P(b,\neg t,\neg a,w) / P(\neg a,w)$
-> ergibt die Wahrscheinlichkeit, dass B=true und T=false ist falls A=false und W=true bekannt ist.

Produktregel

$P(V1,...,Vn) = \prod P(Vi \mid Vi-1,...,V1)$
-> Anmerkung: Faktorisierung ist unabhängig von der Reihenfolge!

Bayessche Regel

2 Instanzen der Produktregel:
P(Vi,Vj) = P(Vi|Vj)*P(Vj)
P(Vi,Vj) = P(Vj|Vi)*P(Vi)

Setzt man die beiden rechten Seiten gleich und dividiert durch P(Vj), erhält man:

$$P(V_i|V_j) = \frac{P(V_j|V_i)P(V_i)}{P(V_j)}$$

-> ist in allen KI-Systemen Grundlage für probabilistisches Schließen

Abstraktes Beispiel: Diagnose in medizinischen Domänen:
P(Ursache|Wirkung) = (P(Wirkung|Ursache) * P(Ursache)) / P(Wirkung)

Konkretes Beispiel: Diagnose von Meningitis
M...Meningitis, G...Genickstarre, P(g|m)=0.5, P(m)=1/50000, P(g)=1/20

P(m|g) = (P(g|m)*P(m)) / P(s) = (0.5*(1/50000)) / (1/20)
= 0.0002 Personen mit Genickstarre haben Meningitis

Probabilistisches Schließen

gegeben: boolsche Variablen V = V1,...,Vn, Subset E ⊆ V, Aussage E=e (true/false)
gesucht: P(Vi=true|E=e) oder P(Vi=false|E=e) -> eines davon reicht aus, da P(Vi=true|E=e) + P(Vi=false|E=e) = 1

Aufgrund der bedingten Wahrscheinlichkeitsregel gilt:
P(Vi=true | E=e) = P(Vi=true, E=e) / P(E=e)

Die rechte Seite wird durch Marginalisierung der gemeinsamen Wahrscheinlichkeitsverteilung von P(V1,...,Vn) gelöst:

$$P(V_i = true, \mathbf{E} = \mathbf{e}) = \sum_{V_i = true, \mathbf{E} = \mathbf{e}} P(V_1, \ldots, V_n)$$

$$P(\mathbf{E} = \mathbf{e}) = \sum_{\mathbf{E} = \mathbf{e}} P(V_1, \ldots, V_n)$$

Beispiel: boolsche Variablen P,Q,R in gemeinsamer Verteilung P(P,Q,R):

$P(p, q, r)$	0.3	$P(\neg p, q, r)$	0.05
$P(p, q, \neg r)$	0.2	$P(\neg p, q, \neg r)$	0.1
$P(p, \neg q, r)$	0.2	$P(\neg p, \neg q, r)$	0.05
$P(p, \neg q, \neg r)$	0.1	$P(\neg p, \neg q, \neg r)$	0

Gesucht: P(q|¬r)

$$\begin{aligned} P(q|\neg r) &= \frac{P(q, \neg r)}{P(\neg r)} \\ &= \frac{P(p, q, \neg r) + P(\neg p, q, \neg r)}{P(\neg r)} \\ &= \frac{0.2 + 0.1}{P(\neg r)} = \frac{0.3}{P(\neg r)}. \end{aligned}$$

P(¬r) = ?, **wir verwenden P(¬q|¬r)**

$$P(\neg q|\neg r) = \frac{P(\neg q, \neg r)}{P(\neg r)} = \frac{P(p, \neg q, \neg r) + P(\neg p, \neg q, \neg r)}{P(\neg r)}$$
$$= \frac{0.1 + 0}{P(\neg r)} = \frac{0.1}{P(\neg r)}.$$

Es gilt P(q|¬r) + (¬q|¬r) = 1:

P(q|¬r) = 1 – P(¬q|¬r)
P(q|¬r) = 1 – 0.1/P(¬r)
P(q|¬r) = 1 – 0.1/0.3 * P(q|¬r) // Umformen von P(q|¬r)=0.3/P(¬r) // zu P(¬r) = 0.3/P(q|¬r)

-> 1 = P(q|¬r) + 0.1/0.3 * P(q|¬r)
1 = P(q|¬r) * (1 + 0.1/0.3)
P(q|¬r) = 1/(1 + 0.1/0.3) = 0.75

Probleme dieses Ansatzes

exponentiell mit der Anzahl der Variablen und gemeinsame Wahrscheinlichkeitsverteilung sind in der Praxis nicht vollständig bekannt.
-> bedingte Unabhängigkeit ist notwendig!

Bedingte Unabhängigkeit

Variable V ist bedingt unabhängig von Variable Vi unter einer gegebenen Aussage E, wenn:

P(V|Vi,E) = P(V|E)
-> zusätzliche Info Vi bei Aussage E bedeutet kein zusätzliche Wissen über E

Generell: V1,...Vn sind gegenseitig bedingt unabhängig von Aussage E, wenn:

P(Vi|V1,...,Vi,...,Vn,E) = P(Vi|E)

Für gegenseitig bedingt unabhängige Variablen **gilt**:

P(V1,...,Vn|E) = ∏P(Vi|E)

Spezialfall: E ist leer:

P(V1,...,Vn) = P(V1)*...*P(Vn) = Produktregel für geg. bedingt unab. Variablen

Bayessche Netzwerke

= eine Graph-Datenstruktur, die die Abhängigkeiten zwischen Variablen darstellt und eine Spezifikation beliebiger vollständiger gemeinsamer Wahrscheinlichkeitsverteilungen erlaubt.
Ein BNW ist ein gerichteter Graph, in dem jeder Knoten mit quantitativen Wahrscheinlichkeitsinfos beschriftet ist.

Bestandteile

- Menge an **Zufallsvariablen** (= **Knoten** des Netzwerks)
- Menge an **gerichteten Kanten** zwischen den Knoten (V1 -> V2 = V1 ist der Elternknoten von V2)
- Jeder Knoten x hat eine **bedingte Wahrscheinlichkeitsverteilung**: P(x | Eltern(x)) -> quantifiziert den Effekt, den die Elternknoten verursachen
- Graph ist **azyklisch**

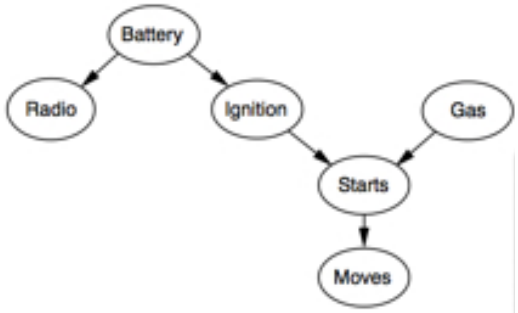
Topologie des Netzwerkes (= Knoten und Kanten) spezifiziert die geltende **Unabhängigkeitsbeziehung**:

- jeder Knoten ist bedingt unabhängig von seinen Eltern:
  P(V|W, Eltern(V)) = P(V|Eltern(V))
  für eine beliebige Menge W an Knoten, die weder Eltern noch Kinder von V sind

Netzwerk bietet vollständige Info über die gemeinsame Wahrscheinlichkeitsverteilung:

$$P(V_1, \ldots, V_n) = \prod_{i=1}^{n} P(V_i | Parents(V_i))$$

Beispiel 1



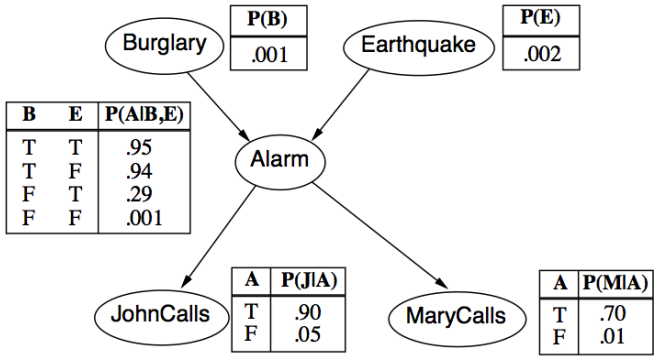
Der Graph zeigt folgende Unabhängigkeitsbeziehungen:

$$P(Starts|Battery, Radio, Ignition, Gas) = P(Starts|Ignition, Gas)$$
$$P(Gas|Battery, Radio, Ignition) = P(Gas)$$
$$P(Ignition|Radio, Gas, Battery) = P(Ignition|Battery)$$
$$P(Battery|Gas) = P(Battery)$$
$$P(Radio|Gas, Battery) = P(Radio|Battery)$$

Beispiel 2: Szenario Einbruchssicherung

Einbruchssicherung erkennt Einbrüche relativ zuverlässig, reagiert aber auch auf kleinere Erdbeben

- Nachbarn John und Mary rufen bei Alarm an
- John ruft immer an (verwechselt auch manchmal das Läuten des Telefons mit Alarm), Mary überhört manchmal Alarm
- -> Mit der Aussage wer angerufen hat, soll die Wahrscheinlichkeit eines Einbruchs abgeschätzt werden
- B ... Einbruch (Burglary), E ... Erdbeben, A ... Alarm, J ... John ruftan, M ... Mary



-> Einbruch und Erdbeben wirken sich direkt auf die Wahrscheinlichkeit aus, dass Alarm ausgelöst wird, während ein Anruf nur vom Alarm abhängig ist.

Gesucht: P(b|¬j,m)
Es gilt: P(b|¬j,m) = P(b,¬j,m) / P(¬j,m)

Wir berechnen P(b,¬j,m) und P(¬j,m) durch Marginalisierung:

$$
\begin{aligned}
P(b,\neg j,m) &= P(b,\neg j,m,a,e) + P(b,\neg j,m,a,\neg e) + \\
&\quad P(b,\neg j,m,\neg a,e) + P(b,\neg j,m,\neg a,\neg e) \\
&= P(b)P(\neg j|a)P(m|a)[P(a|b,e)P(e) + P(a|b,\neg e)P(\neg e)] + \\
&\quad P(b)P(\neg j|\neg a)P(m|\neg a)[P(\neg a|b,e)P(e) + P(\neg a|b,\neg e)P(\neg e)] \\
&= 0.001 \cdot (1-0.9) \cdot 0.7 \cdot [0.95 \cdot 0.002 + 0.94 \cdot (1-0.002)] + \\
&\quad 0.001 \cdot (1-0.05) \cdot 0.01 \cdot [(1-0.95) \cdot 0.002 + \\
&\quad (1-0.94) \cdot (1-0.002)] \\
&= 6.637121 \cdot 10^{-5}
\end{aligned}
$$

d-Seperation

= allgemeines topologisches Kriterium, um zu entscheiden, ob eine Menge von Knoten X unabhängig von einer anderen Menge Y ist, wenn man eine dritte Menge Z kennt.

Komplexität von Schließen

generell exponentiell (auch mit d-Seperation), zur Verbesserung gibt es Algorithmen in Polynomialzeit für spezielle Netz-Datenstrukturen (zB Poly-Bäume)