Skriptum zur Vorlesung:

Künstliche Intelligenz

WS ’96/’97

Dr. Jürgen Dix

Universität Koblenz-Landau
Institut für Informatik

Februar 1997

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Vorwort

Dieses Skriptum faßt den Stoff der jeweils im Wintersemester ’95/’96 und ’96/’97 von mir gehaltenen sechs-stündigen Vorlesung (davon 2 Stunden Übung) *Künstliche Intelligenz* zusammen. Die Übungen wurden jeweils von Herrn Frieder Stolzenburg gehalten, der auch für die genaue Aufgabenstellungen zuständig war. Das Skriptum besteht, wie die Vorlesungen, aus 12 Kapiteln. Außer Kapitel 5, 6, 8 und 12, stammen alle anderen Kapitel (mit einigen Modifikationen) aus dem Buch [RN95]. Die Autoren dieses Buches haben freundlicherweise die Abbildungen als postscript Dateien frei zugänglich gemacht.

1 Einleitung

1.1 Was ist KI?

1. Kognitionswissenschaft,

2. Sokrates ist ein Mensch. Menschen sind sterblich. Also ist Sokrates sterblich. *Syllogismen* (Aristoteles).

Informale Beschreibung $\leadsto$ * *Formale Beschreibung*
Formale Beschreibung $\leadsto$ ** *Lösung des Problems*.

* ist problematisch, da oft nicht genügend spezifiziert,

** ist Deduktion (korrekte Inferenzen): nur aufzählbar, nicht rekursiv.

3. Turing Test.

- Üblicher Turing Test
- Totaler Turing Test

"The exciting new effort to make computers think ... *machines with minds*, in the full and literal sense" (Haugeland, 1985) "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning ..." (Bellman, 1978)	"The study of mental faculties through the use of computational models" (Charniak and McDermott, 1985) "The study of the computations that make it possible to perceive, reason, and act" (Winston, 1992)
"The art of creating machines that perform functions that require intelligence when performed by people" (Kurzweil, 1990) "The study of how to make computers do things at which, at the moment, people are better" (Rich and Knight, 1991)	"A field of study that seeks to explain and emulate intelligent behavior in terms of computational processes" (Schalkoff, 1990) "The branch of computer science that is concerned with the automation of intelligent behavior" (Luger and Stubblefield, 1993)

Tabelle 1.1: Einige Definitionen von künstlicher Intelligenz

Turing glaubte um 1950, daß im Jahr 2000 ein Computer mit 10^9 Einheiten Speicher so programmiert werden kann, daß er sich 5 Minuten mit einem Menschen unterhalten kann und zu 30 % den Turing-Test besteht.

4. In Punkt **2.** wurde von *korrekten* Inferenzen gesprochen. Oft liegt nicht genügend Information vor um sinnvolle Handlungen (beweisbar korrekte) auszuführen ($\rightsquigarrow$ Kapitel 8).

Die Welt ist i.A. nicht genügend spezifiziert. Außerdem kann man rational agieren, ohne *korrekte* Inferenzen machen: *Reflexe*.

Dieser Ansatz wird in der gesamten Vorlesung verfolgt.

Beim *totalen Turing-Test* kommt noch eine physikalische Interaktion hinzu. Also ein Video-Signal. Man braucht daher noch *Computer-Vision* und *Robotics*.

Die Frage *Können Maschinen denken?* führt zu 2 KI-Thesen:

Schwache KI-These: Man kann Maschinen bauen, die sich so verhalten als ob sie intelligent sind.

Starke KI-These: Maschinen die sich intelligent verhalten besitzen kognitive Zustände, d.h. *Geist*.

Man beachte, daß aus der starken keineswegs die schwache KI-These folgt.

Das chinesische Zimmer

Searle hat dies 1980 als Beispiel gebracht, daß ein System den Turing-Test bestehen kann, aber die starke KI-These widerlegt. In einem Zimmer befinden sich

CPU: ein Mensch, der nur Englisch nicht aber Chinesisch spricht,

Programm: ein Buch, das Regeln enthält die in Englisch formuliert sind und ihm sagen wie chinesische Texte übersetzt werden.

Speicher: genügend Papier und Schreibmaterial.

Es werden ihm chinesisch beschriebene Blätter gereicht, die er mit Hilfe des Buches übersetzt. *Kann man sagen, daß das Zimmer Chinesisch versteht?*

1.2 Grundlagen

450 v. Chr. : **Plato**, **Sokrates**, **Aristoteles**
Sokr.: *Was ist charakteristisch für das Gute?*
Aris.: *Welche Gesetze regeln den Verstand?*
800 n. Chr. : **Al Chwarizmi**, Arabien
1300 : **Raymundus Lullus**: *Ars Magna*

1596–1650 : **R. Descartes**:
Verstand = physikalisches System
Freier Wille, Dualismus
1623–1662 : **B. Pascal, W. Schickard**
Additions-Maschinen
1646–1716 : **G. W. Leibniz**
Materialismus, greift Idee der Ars Magna auf
um mechanisches Gerät zur Simulation des
menschlichen Verstandes zu bauen
1561–1626 : **F. Bacon**, *Empirismus*
1632–1704 : **J. Locke**, *Empirismus*
“Nihil est in intellectu quod non antefuerat in sensu.”
1711–1776 : **D. Hume**, *Induktion*
1724–1804 : **I. Kant**,
“Der Verstand schöpft seine Gesetze
nicht aus der Natur, sondern schreibt sie dieser vor.”
1805 : **Jacquard**-Webstuhl
1815–1864 : **G. Boole**
Formale Sprache,
Logik als **mathematische** Disziplin
1792–1871 : **Ch. Babbage**
Difference Engine: Ber. von Log.-Tafeln
Analytical Engine: besitzt schon (1) adr. Speicher,
(2) bed. Sprünge und (3) Programmspeicher
1848–1925 : **G. Frege**,
Begriffsschrift: 2-dim. Notation für PL1
1862–1943 : **D. Hilbert**,
Vortrag 1900 in Paris: 23 Probleme
23. Problem: *Entscheidungsproblem*
1872–1970 : **B. Russell**,
1910: Principia Mathematica
Logischer Positivismus, Wiener Kreis (1920–40)
1902–1983 : **A. Tarski**
Wahrheitsbegriff in den formalisierten Sprachen, 1936
1906–1978 : **K. Gödel**,
Vollst/Unvollst. Sätze, 1930/31
1912–1954 : **A. Turing**
Turing-Maschine 1936
Berechenbarkeit
1903–1995 : **A. Church**,
λ-Kalkül, Church’sche These

1940 : Erster Computer “Heath Robinson” zum
Dechiffrieren deutscher Codes (Turing),
1943 Collossus
1941 : Erster frei programmierbarer Computer:
Z 3 von **K. Zuse** ($\rightsquigarrow$ Deutsches Museum)
Gleitkomma-Arithmetik,
1945 Plankalkül: erste Progammier-Hochsprache
1940-1945 : **H. Aiken** entwickelt MARK I, II, III.
ENIAC: erster elektronischer Computer
1952 : IBM 701, erster Computer der Profit abwirft

1.3 Historisches zur KI

1943

McCulloch und W. Pitts bringen 3 Dinge miteinander in Verbindung

1. Physiologie und Funktion der Neuronen im Gehirn.
2. Aussagenlogik nach Russell/Whitehead.
3. Turing’s Theorie der Berechenbarkeit.

Sie konstruieren ein abstraktes Modell bestehend aus vernetzten Neuronen und beweisen

- *Berechenbare Funktionen können durch solche Modelle berechnet werden.*
- *Aussagenlogische Junktoren werden durch sehr einfache Netze implementiert.*

1951

Minsky und Edwards bauen den ersten Computer der auf Neuronalen Netzen basiert (Princeton).

1952

A. Samuel entwickelt Mühle-Programme die lernen ihre Spielstärke zu verbessern

1956

2 monatiger Workshop in Dartmouth (McCarthy, Minsky, Shannon, Rochester). Es werden die *Zusammenhänge von Automatentheorie, Neuronalen Netzen und Intelligenzforschung* diskutiert. (10 Teilnehmer)

Newell und Simon treten mit einem lauffähigen Programm auf: *LT, Logical Theorist*. LT kann viele Theoreme aus Kap. 2 der Principia Mathematica beweisen. Sogar eines mit einem kürzeren Beweis: das renommierte *Journal of Symbolic Logic* lehnt jedoch das eingereichte Papier der Autoren *Newell, Simon und LT* ab. Newell/Simon behaupten das *Leib-Seele-Problem* gelöst zu haben.

Der Begriff **Artificial Intelligence** wird als Name der neuen Disziplin vorgeschlagen und etabliert sich.

GPS: General Problem Solver von Newell/Simon ist die Weiterentwicklung von LT. Die Idee ist nun, menschliches Problem-Lösen zu imitieren.

1958

Geburtsjahr der KI. McCarthy geht ans MIT und entwickelt

1. **Lisp**,
2. **Time-Sharing**,
3. **Programs with Common-Sense**.
   Advice-Taker: hypothetisches Programm, das als erstes vollständiges KI-SYSTEM angesehen werden kann. Im Unterschied zu LT und GPS wird nicht nur spezifisches Domänen-Wissen benutzt, sondern auch allgemeines Wissen über die Welt.

1959

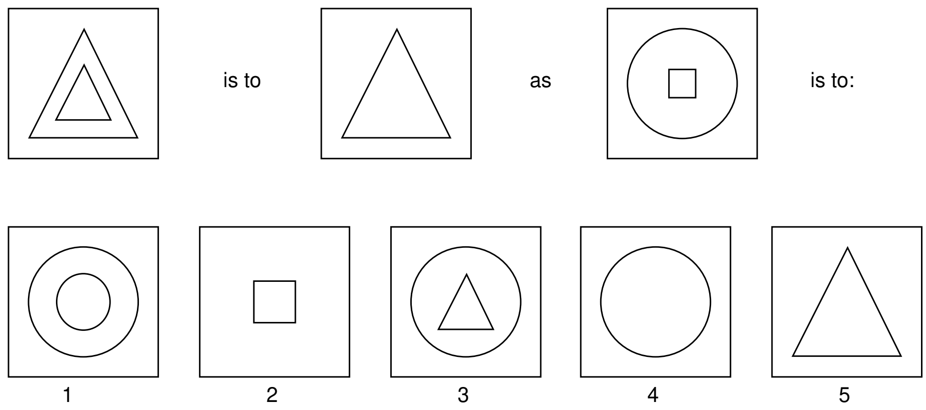
H. Gelernter: Geometrie Beweiser

1960-1966

McCarthy konzentriert sich auf *Wissensrepräsentation und Schließen in formaler Logik* ($\leadsto$ *Robinson's Resolution* (Kapitel 5),
$\leadsto$ *Green's Planner*, $\leadsto$ *Shakey* (Kapitel 7)).

Minsky konzentriert sich auf lauffähige Programme für spezielle Welten: *Mikrowelten*.

- SAINT, Integrations-Probleme des 1/2 Semesters
- ANALOGY, Einfache Analogien in IQ-Tests



Eine der wichtigsten Mikrowelten ist die *Blocksworld*. Weiterentwicklung der Ideen von McCulloch und Pitts. **Perceptron** von Rosenblatt und Konvergenz-Theorem für Perceptronen:

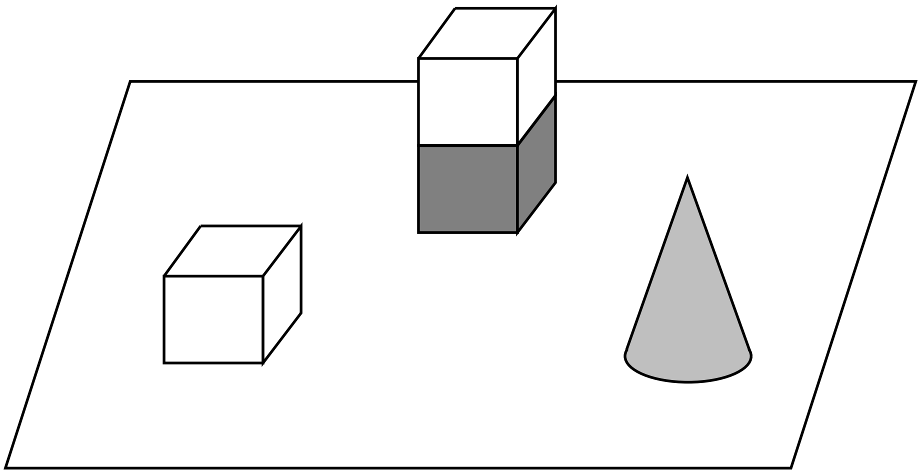


Abbildung 1.1: Die Block-Welt

- *Es gibt einen Lern-Algorithmus der ein Perceptron so aufbaut, daß es mit den Input-Daten übereinstimmt.*

Insgesamt gab es große Versprechungen für die Zukunft. Zunächst auch Anfangserfolge, aber nur miserable weiterführende Ergebnisse.

1966: Einstellen aller US-Funds für Übersetzungs-Syst. Angeblich wurde der Satz

The spirit is willing but the flesh is weak.

mit

The vodka is good but the meat is rotten.

übersetzt.

1966–1974

Simon 1958: *"In 10 Jahren wird der Schachweltmeister ein Computer sein."* Einfache Probleme deshalb lösbar, weil Suchraum klein. Ernsthafte Probleme bleiben unlösbar. *Hoffnung:* Schnellere Hardware, größerer Speicher werden alles lösen. $\leadsto$ NP-vollständige Probleme in Kapitel5, (S. Cook/R. Karp 1971/72), $P \neq NP$?

1973

Lighthill-Report führt zur Einstellung der Forschungsgelder für KI in Großbritannien. Erste Resultate über grundsätzliche Grenzen einiger Ansätze (Minsky's *"Perceptrons"*).

1969–79 *Wissensbasierte Systeme*

General Purpose-Mechanismen benutzen keine oder nur sehr wenig Wissen über ein Anwendungsgebiet. Für komplexe Gebiete führt dies zur kombinatorischen Explosion. *Man benutze typisches Anwendungswissen und trenne dies von der deduktiven Komponente.*

Beispiel: 1969 DENDRAL. Dies führt zu Expertensystemen wie MYCIN (Diagnose von Blut-Infektionen).

1973

PROLOG: "Programmation en Logique"

DENDRAL bestimmt die Molekül-Struktur aus vorliegenden Massenspektren. Die naive Version funktioniert nicht (zähle alle Möglichkeiten auf und sondere jene aus, die den Messungen widersprechen) da es zuviele sind. Man betrachtet *special purpose rules* die, falls anwendbar schon eine gewisse Struktur vorgeben und damit die Möglichkeiten einschränken. Z.B. falls dort und dort ein Peak ist, dann enthält das Molekül eine **C=O** Bindung.

Ein Schmankerl

1902 wurde ein deutsches Gedicht ins Japanische übersetzt. Die japanische Fassung wurde dann 1911 ins Französische übersetzt. Schließlich wurde diese Fassung (in der Annahme es handele sich um ein japanisches Gedicht) ins Deutsche übertragen.
Heraus kam schließlich folgendes:

> *Stille ist im Pavillon aus Jade,*
> *Krähen fliegen stumm*
> *Zu beschneiten Kirschbäumen im Mondlicht.*
> *Ich sitze*
> *und weine.*

Wie lautet das ursprüngliche Gedicht?

1.4 Aufgaben

Aufgabe 1.1 (Was ist Intelligenz?)
Stimmen die folgenden Aussagen und Begründungen Ihrer Meinung nach?

Tiere *können niemals Intelligenz besitzen; denn sie tun nur das, was ihre Gene ihnen vorgeben.*

Computer *können niemals Intelligenz besitzen, denn Sie tun nur das, was ein Programmierer in sie eingegeben hat.*

Aufgabe 1.2 (Was ist KI?)
Wir wollen in der Übungsstunde eine Diskussion führen. Die folgenden Aufgaben sollen dazu als Grundlage dienen.

1. *Was ist „Künstliche Intelligenz“ (abgekürzt KI, engl. „Artificial Intelligence“)? Überlegen Sie sich eine Definition! Konsultieren Sie Lexika, Lehrbücher, Zeitschriften etc.*

2. *Stellen Sie (wenige) Thesen auf, welches Ziel, welche möglichen Anwendungsgebiete und welchen Sinn die Forschung im Bereich „Künstliche Intelligenz“ hat bzw. haben soll – Ihrer Meinung nach!*

2 Intelligente Agenten

2.1 Definitionen von Agenten

Definition 2.1.1 (Agent)
Ein Agent ist etwas, das seine Umgebung mittels Sensoren wahrnimmt und in dieser Umgebung mittels Werkzeugen agiert.

Definition 2.1.2 (Rationaler, allwissender Agent)
Rationale Agenten sind solche, die das **Richtige** *tun. (Ein Maß für die Performanz wird benötigt.) Allwissende Agenten sind solche, die die Resultate ihrer Aktionen im voraus wissen.*

Rationale Agenten sind i.a. nicht allwissend! Aphorismus von Karl Kraus: *Im Zweifelsfall entscheide man sich für das richtige.*

Wie ist das **Richtige** definiert und wovon hängt es ab?

1. Performanz-Maß (möglichst objektiv),
2. Percept-Sequenz: was bisher wahrgenommen wurde,
3. Wissen des Agenten über die Umgebung,
4. wie der Agent agieren kann.

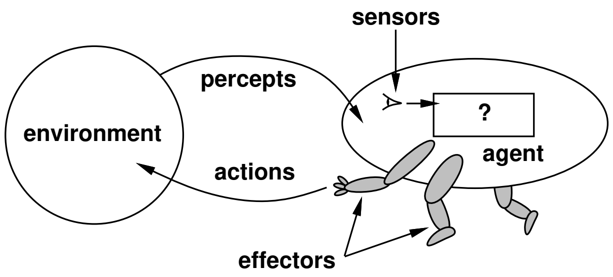


Abbildung 2.1: Ein Agent.

Agent Type	Percepts	Actions	Goals	Environment
Medical diagnosis system	Symptoms, findings, patient's answers	Questions, tests, treatments	Healthy patient, minimize costs	Patient, hospital
Satellite image analysis system	Pixels of varying intensity, color	Print a categorization of scene	Correct categorization	Images from orbiting satellite
Part-picking robot	Pixels of varying intensity	Pick up parts and sort into bins	Place parts in correct bins	Conveyor belt with parts
Refinery controller	Temperature, pressure readings	Open, close valves; adjust temperature	Maximize purity, yield, safety	Refinery
Interactive English tutor	Typed words	Print exercises, suggestions, corrections	Maximize student's score on test	Set of students

Tabelle 2.1: PAGE-Beschreibung

Definition 2.1.3 (Idealer rationaler Agent)
Ein idealer rationaler Agent wählt für jede Percept-Sequenz genau die Aktion aus, die sein Performanz-Maß maximiert (bei gegebenem Wissen über die Umgebung).

Agenten können durch Abbildungen

$$\text{Menge der Percept-Sequenzen} \mapsto \text{Menge der Aktionen}$$

mathematisch beschrieben werden. Die interne Struktur eines Agenten ist

Agent = Architektur + Programm

Die KI beschäftigt sich damit, Agenten-Programme zu entwerfen.

Agenten und ihre PAGE-Beschreibung

Einfachste Form von Agenten-Programmen:

```
function SKELETON-AGENT( percept) returns action
   static: memory, the agent's memory of the world

   memory ← UPDATE-MEMORY(memory, percept)
   action ← CHOOSE-BEST-ACTION(memory)
   memory ← UPDATE-MEMORY(memory, action)
   return action
```

Theoretisch ist alles trivial, da man ja in einer Tabelle nachschauen kann:

```
function TABLE-DRIVEN-AGENT( percept) returns action
   static: percepts, a sequence, initially empty
           table, a table, indexed by percept sequences, initially fully specified

   append percept to the end of percepts
   action ← LOOKUP( percepts, table)
   return action
```

2.2 Beispiele von Agenten

Beispiel 2.2.1 (Taxi-Fahrer)

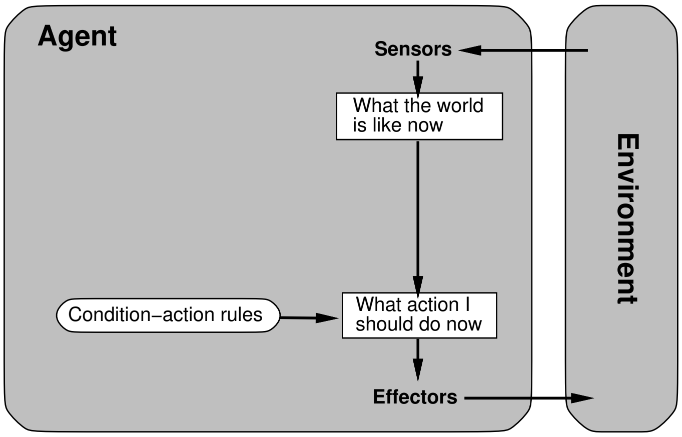
Agent Type	Percepts	Actions	Goals	Environment
Taxi driver	Cameras, speedometer, GPS, sonar, microphone	Steer, accelerate, brake, talk to passenger	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers

Wie können wir ein Programm entwerfen? Wir können das Problem unter verschiedenen Gesichtspunkten sehen:

1. Agenten, die auf einfache Reflexe reagieren,
2. Agenten, die berücksichtigen wie sich die Welt durch ihre Aktionen ändert,
3. Agenten, die zielgerichtet agieren,
4. Agenten, die so agieren, daß Zustände bevorzugt werden, die besonders nützlich sind (bezüglich eines vorgegebenen Performanz-Maßes).

Hier einige Beispiele:

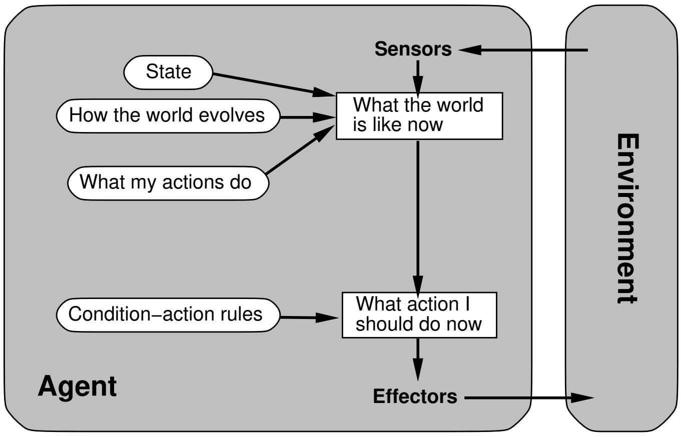
1. *Produktions-Regeln:* Falls der Vorgänger bremst, dann bremse auch.



function SIMPLE-REFLEX-AGENT(*percept*) **returns** *action*
  static: *rules*, a set of condition-action rules

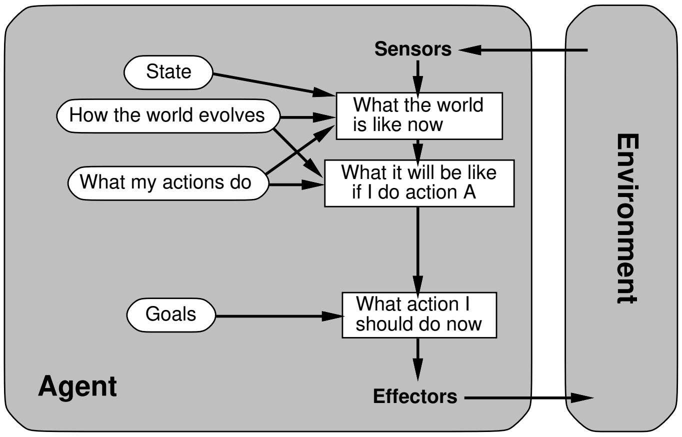
  state ← INTERPRET-INPUT(*percept*)
  rule ← RULE-MATCH(*state*, *rules*)
  action ← RULE-ACTION[*rule*]
  return *action*

2. *Zusätzlich interne Zustände:* was der Taxifahrer im Spiegel gesehen hat, sollte er sich merken.

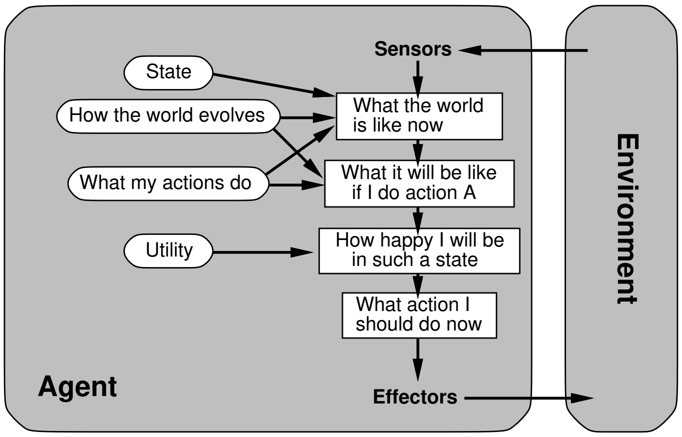


```
function REFLEX-AGENT-WITH-STATE( percept) returns action
   static: state, a description of the current world state
           rules, a set of condition-action rules

   state ← UPDATE-STATE(state, percept)
   rule ← RULE-MATCH(state, rules)
   action ← RULE-ACTION[rule]
   state ← UPDATE-STATE(state, action)
   return action
```

3. führt später zum *Planen*. Verfolgen von Zielen, Erkennen von Folgen.



4. z.B.: schnell, billig, sicher.



Wie beeinflussen Eigenschaften der Umgebung das Design eines dazu passenden Agenten? Hier einige grundlegende Eigenschaften:

Zugänglich/Unzugänglich: Falls nicht vollst. zugängl. braucht man interne Zustände.

Determinist./Indeterm.: Falls die Umgebung unzugänglich ist, kann sie indeterministisch scheinen, obwohl sie es nicht ist.

Episodisch/Nichtepisodisch: Percept-Aktion-Sequenzen sind voneinander unabhängig. Abgeschlossene Episoden.

Statisch/Dynamisch: Dynamisch heißt: während der Agent noch überlegt, ändert sich schon die Welt. Semi-dynamisch heißt: Zwar ändert sich nicht die Welt, aber das Performanz-Maß des Agenten.

Diskret/Stetig: bezieht sich auf die Menge der Wahrnehmungen und Aktionen.

Environment	Accessible	Deterministic	Episodic	Static	Discrete
Chess with a clock	Yes	Yes	No	Semi	Yes
Chess without a clock	Yes	Yes	No	Yes	Yes
Poker	No	No	No	Yes	Yes
Backgammon	Yes	No	No	Yes	Yes
Taxi driving	No	No	No	No	No
Medical diagnosis system	No	No	No	No	No
Image-analysis system	Yes	Yes	Yes	Semi	No
Part-picking robot	No	No	Yes	No	No
Refinery controller	No	No	No	No	No
Interactive English tutor	No	No	No	No	Yes

2.3 Aufgaben

Aufgabe 2.1 (Rationale Agenten)
Betrachten Sie eine „Staubsauger-Welt“, d.h. einen Roboter, der eine Wohnung ganz sauber staubsaugt.

1. *Schreiben Sie eine ausführliche PAGE-Beschreibung (percepts, actions, goals, environment) für einen intelligenten Agenten, d.h. den Roboter, in dieser Umgebung (environment).*

2. *Charakterisieren Sie die Umgebung hinsichtlich der folgenden Eigenschaften:*

   - *zugänglich / unzugänglich,*
   - *deterministisch / indeterministisch,*
   - *episodisch / nicht episodisch,*
   - *statisch / dynamisch,*
   - *diskret / stetig*

3 Problem-Lösen durch Suchen

Gesucht: *Agenten, die Probleme lösen.* Diese sind offensichtlich eine Teilklasse der Ziel-basierten Agenten.

Struktur: *Formuliere, Suche, Führe aus.*

Formuliere:

- **Ziel:** eine Menge von Zuständen,
- **Problem:** Zustände, Aktionen, die Zustande ineinander überführen, Transitionen

Suche: Welche Folge von Aktionen ist hilfreich?

Führe aus: Man wende die Lösungsfolge von Aktionen auf den Anfangszustand an.

3.1 Problem-Formulierung

Wir unterscheiden zwischen 4 verschiedenen Typen:

1. **1-Zustand-Probleme:** Aktionen sind vollständig beschrieben. Vollständige Information durch Sensoren zur eindeutigen Bestimmung des gegenwärtigen Zustands.

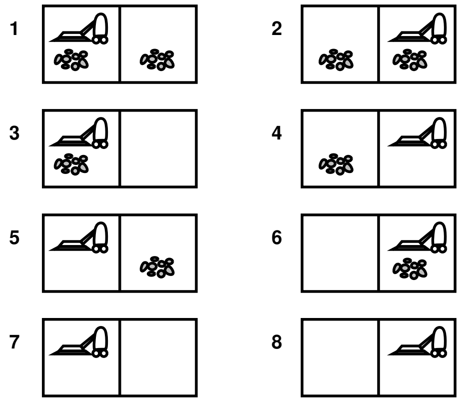
2. **Multiple-Zustand-Probleme:** Aktionen sind vollständig beschrieben, aber der Anfangszustand liegt nicht fest.

3. **Kontingenz-Probleme:** Manchmal gibt es keine fixe Sequenz als Lösung, sondern man muss den ganzen Baum betrachten.
   ($\rightsquigarrow$ Übung: Murphy's Gesetz, $\rightsquigarrow$ Kapitel 7)

4. **Explorations-Probleme:** Nicht einmal die Effekte der Aktionen sind bekannt. Man muss in der Welt suchen, nicht im abstrakten Modell.

```
function SIMPLE-PROBLEM-SOLVING-AGENT(p) returns an action
  inputs: p, a percept
  static: s, an action sequence, initially empty
          state, some description of the current world state
          g, a goal, initially null
          problem, a problem formulation

  state ← UPDATE-STATE(state, p)
  if s is empty then
     g ← FORMULATE-GOAL(state)
     problem ← FORMULATE-PROBLEM(state, g)
     s ← SEARCH(problem)
  action ← RECOMMENDATION(s, state)
  s ← REMAINDER(s, state)
  return action
```

Tabelle 3.1: Einfacher Problem-lösender Agent.



Definition 3.1.1 (1-Zustand-Problem)
Ein 1-Zustand-Problem besteht aus

- *einer Menge von Zuständen, darunter einer der als* Anfangszustand *ausgezeichnet ist,*
- *einer Menge von n* Aktionen *(wir nennen sie auch* Operatoren*), die, in einem Zustand ausgeführt, zu einem anderen Zustand überleiten:*

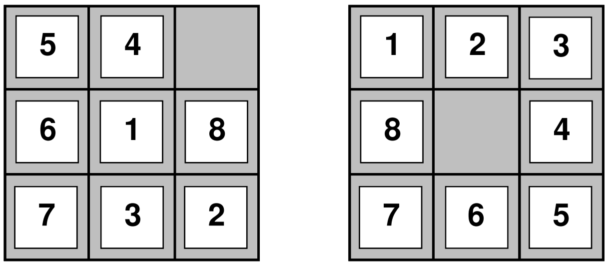
$$\textit{Operator}_i\textit{: Zustände} \mapsto \textit{Zustände}, \ i = 1, \ldots, n$$

- *einer Menge von* Ziel-Zuständen*, oder einem Ziel-Test, der, angewandt auf einen Zustand, bestimmt ob es sich um einen Ziel-Zustand handelt oder nicht,*
- *einer Kosten Funktion g, die jedem Pfad im Zustandsraum (Menge der erreichbaren Zustände) gewisse Kosten zuordnet. Üblicherweise ist g additiv.*

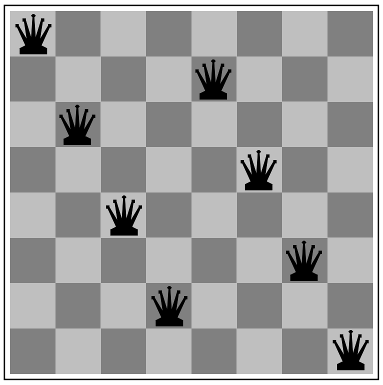
Wie geht das für Multiple Zustand-Probleme? Wie wählt man Aktionen und Zustände aus? Durch geeignete *Abstraktion*.

3.2 Einige Beispiele

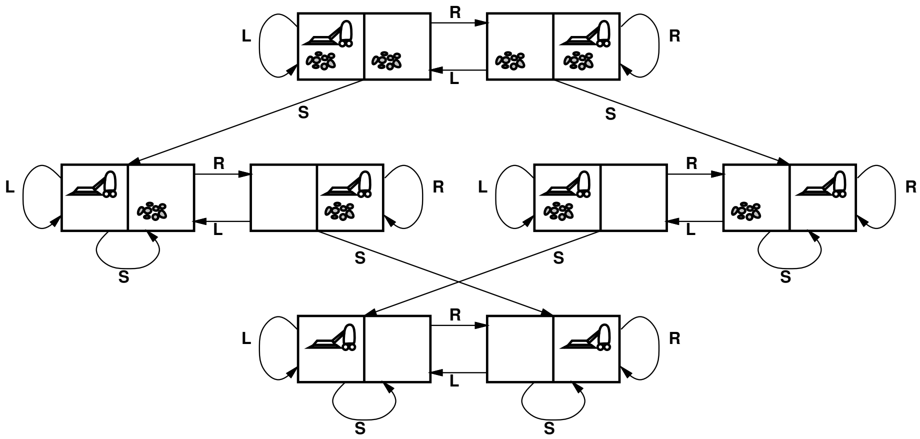
Spiel-Beispiele:



Start State **Goal State**



Staubsauger Welt mit Murphy's Gesetz



```
function GENERAL-SEARCH( problem, strategy) returns a solution, or failure
  initialize the search tree using the initial state of problem
  loop do
     if there are no candidates for expansion then return failure
     choose a leaf node for expansion according to strategy
     if the node contains a goal state then return the corresponding solution
     else expand the node and add the resulting nodes to the search tree
  end
```

Tabelle 3.2: Allgemeine Suche.

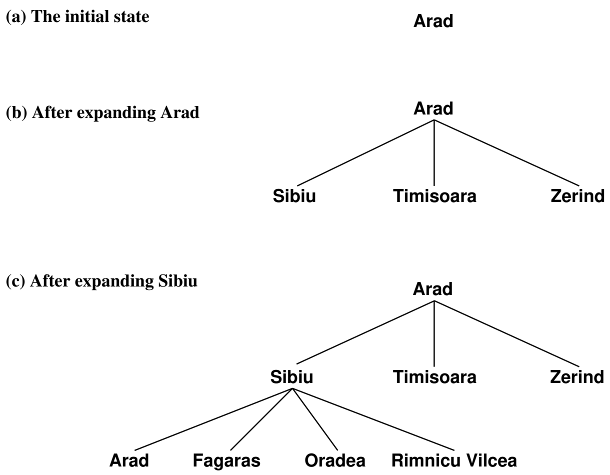
- Missionare vs. Kannibalen

Real-World-Probleme:

- Travelling-Salesman-Problem
- VLSI-Layout
- Landkarten beschriften
- Roboter-Navigation

3.3 Such-Strategien (allgemein)

Prinzip: Auswählen, Testen, Expandieren. ($\rightsquigarrow$ Such-Baum)



Zustands-Raum versus Such-Baum: Während der Zustandsraum üblicherweise endlich ist, ist der Such-Raum meist abzählbar unendlich.

Definition 3.3.1 (Datentyp Knoten)
Der Datentyp Knoten *ist durch die Komponenten* Zustand, Elter-Knoten, Operator, Tiefe *und* Pfad-Kosten *definiert.*

- Ein Knoten ist ein Buchhalter bezüglich einer Problem-Instanz und bezüglich eines bestimmten Algorithmus.
- Ein Zustand ist ein Schnappschuss der Welt.

Beachten Sie die *rekursive* Abhängigkeit (Knoten/Elter-Knoten). Falls wir die *Tiefe* weglassen, müssen wir einen ausgezeichneten Knoten $root$ einführen, die *Wurzel*. Dann ist die Tiefe *definierbar*. Umgekehrt ist $root$ durch die Tiefe *definierbar*: der Knoten, der sich selbst als Elter hat und die Tiefe 0.

Wir wollen versuchen, die Funktion GENERAL-SEARCH etwas konkreter zu machen und treffen folgende *Design-Entscheidung:* Die zu expandierenden Knoten stellen wir nicht als Menge sondern als **Queue** dar.

Make-Queue(Elements)	**Remove-Front(Queue)**
Empty?(Queue)	**Queing-Fn(Queue,Elements)**

```
function GENERAL-SEARCH( problem, QUEUING-FN) returns a solution, or failure

  nodes ← MAKE-QUEUE(MAKE-NODE(INITIAL-STATE[problem]))
  loop do
      if nodes is empty then return failure
      node ← REMOVE-FRONT(nodes)
      if GOAL-TEST[problem] applied to STATE(node) succeeds then return node
      nodes ← QUEUING-FN(nodes, EXPAND(node, OPERATORS[problem]))
  end
```

Was sind interessante Anforderungen an Such-Strategien?

- *Vollständigkeit*,
- *Zeit-Komplexität*,
- *Speicher-Komplexität*,
- *Optimalität (bez. Pfad-Kosten)*

Wir unterscheiden zwischen **Uninformierter** und **informierter Suche**.

3.4 Uninformierte Suche

Breadth-First-Suche: "Knoten der kleinsten Tiefe werden zuerst expandiert",

Queing-Fn($Queue, Expand(\dots)$) : neue Knoten ans Ende

Vollständig: Ja.
Optimal: Ja, falls alle Operatoren dieselben Kosten haben.

Für einen konstanten Verzweigungsfaktor b gilt: In der Tiefe d gibt es maximal

$$1 + b + b^2 + \ldots + b^{d-1} + b^d\text{-viele Knoten.}$$

Depth	Nodes	Time	Memory
0	1	1 millisecond	100 bytes
2	111	.1 seconds	11 kilobytes
4	11,111	11 seconds	1 megabyte
6	10^6	18 minutes	111 megabytes
8	10^8	31 hours	11 gigabytes
10	10^{10}	128 days	1 terabyte
12	10^{12}	35 years	111 terabytes
14	10^{14}	3500 years	11,111 terabytes

Uniform-Cost-Suche: "Knoten n mit kleinsten Pfad-Kosten $g(n)$ werden zuerst expandiert",

Queing-Fn($Queue, Expand(\ldots)$) : neue Knoten werden gemäss ihrer Pfad-Kosten mit denen in der Queue verglichen und entsprechend eingefügt

Vollständig: Ja, falls jeder Operator die Pfadkosten um mindestens einen Betrag $\delta > 0$ anwachsen lässt (siehe unten).

Falls alle Operatoren die gleichen Kosten haben (insbesondere wenn $g(n) =$ Tiefe(n)) gilt: Uniform-Cost=Breadth-First.

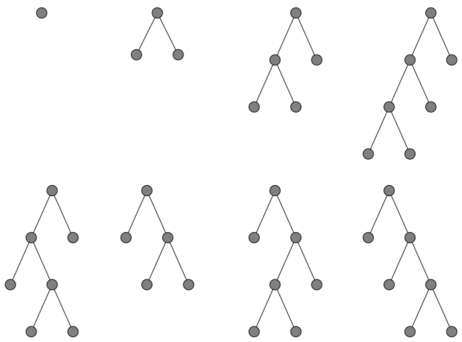
Theorem 3.4.1 (Optimalität von Uniform-Cost)
Gilt $\exists\delta > 0 : \ g(succ(n)) \geq g(n) + \delta$ *so ist Uniform-Cost optimal.*

Depth-First-Suche: "Knoten der grössten Tiefe werden zuerst expandiert".

Queing-Fn($Queue, Expand(\ldots)$) : neue Knoten werden am Anfang eingefügt

Falls Verzweigungsfaktor b konstant und max. Tiefe m:

- **Speicherkomplexität:** $b \times m$,
- **Zeitkomplexität:** b^m.

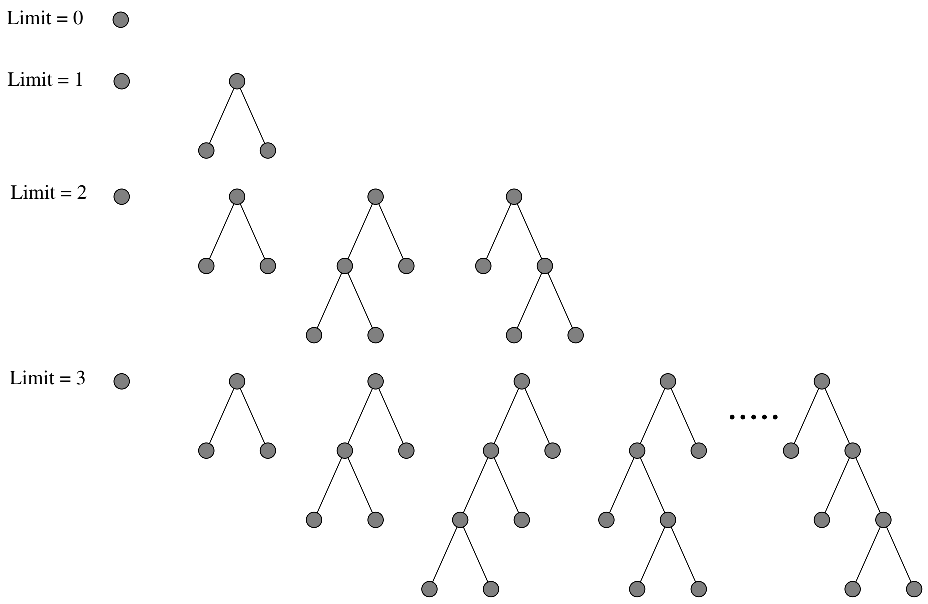


Depth-First mit Tiefenbeschränkung d: “gehe höchstens in Tiefe d”.

Insgesamt gibt es $1 + b + b^2 + \ldots + b^{d-1} + b^d$-viele Expansionsschritte.

Iterative Deepening: “Tiefe wird immer grösser”

```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution sequence
  inputs: problem, a problem

  for depth ← 0 to ∞ do
    if DEPTH-LIMITED-SEARCH( problem, depth) succeeds then return its result
  end
  return failure
```



Wieviele Expansionsschritte gibt es insgesamt? Genau

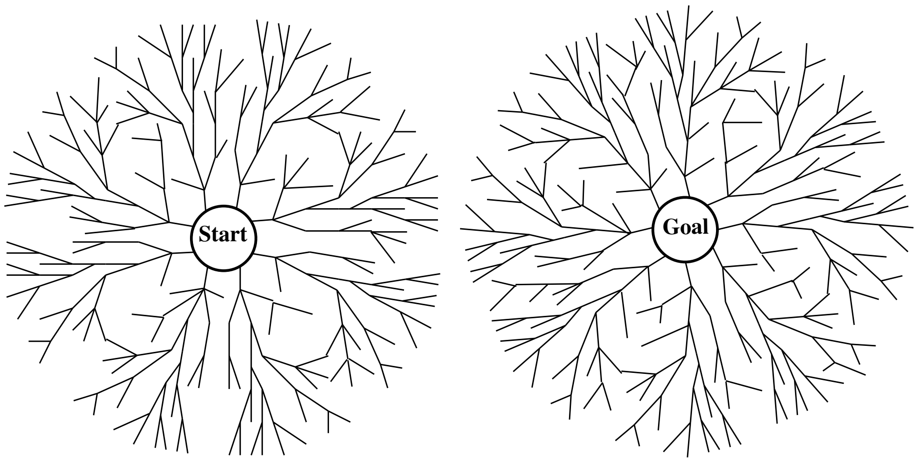
$$(d + 1) \times 1 + d \times b + (d - 1) \times b^2 + \ldots + 2 \times b^{d-1} + 1 \times b^d\text{-viele.}$$

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes

Tabelle 3.3: Eigenschaften von Such-Algorithmen

Man beachte, daß nur etwa doppelt soviele Schritte im Vergleich zu Depth-First mit Tiefenbeschränkung ausgeführt werden ($\leadsto$ geometrische Reihe). Der Aufwand für mehrfach berechnete Knoten hält sich also in Grenzen.

Bidirektionale Suche:



3.5 Aufgaben

Aufgabe 3.1 (General Search)
Der General-Search-Algorithmus umfaßt drei Schritte: (i) Test auf Zielzustand, (ii) Generierungsphase und (iii) Anordnungsfunktion, in dieser Reihenfolge. Ein gewisser Nachteil dieser Formulierung besteht darin, daß nicht sofort erkannt wird, wenn generierte Knoten Zielzustände darstellen, weil die Anordnungsfunktion sie nicht an die erste Stelle plaziert. Schaffen Sie Abhilfe, indem sie den Algorithmus auf zwei Arten neu formulieren:

1. *Jeder Knoten wird sofort nach seiner Generierung getestet.*
2. *Die Anordnungsfunktion wird entsprechend angepaßt.*

Aufgabe 3.2 (Chain-Problem)
Das Chain-Problem *besteht darin, Ketten verschiedener Länge umzubauen. Operatoren in diesem Zusammenhang sind* ÖFFNEN *und* SCHLIESSEN *eines Gliedes. In der Standard-Formulierung des Problems besteht der Anfangszustand aus vier Ketten mit jeweils drei Gliedern; der Zielzustand ist eine geschlossene Kette aus allen zwölf Gliedern. Definieren Sie das Problem formal und bestimmen Sie eine kürzeste Lösung!*

Aufgabe 3.3 (Suche mit Prolog)
Gegeben sei der Graph $G(n)$ für $n \in \mathbb{N}$, bestehend aus einem Startknoten $start$ und den folgenden Knoten: $incest(i, j)$ für $i \in 0..1$ und $j \in 1..n$; $cycle(k)$ für $k \in 1..2$; $tree(l)$ für $l \in 1..2^n - 1$. Zwischen den Knoten bestehen folgende Verbindungen:

$start \to incest(0, 1)$, $start \to incest(1, 1)$
$incest(i, j) \to incest(i', j + 1)$ *für* $i, i' \in 0..1$ *und* $j < n$
$incest(1, n) \to cycle(1)$
$cycle(1) \to cycle(2)$, $cycle(2) \to cycle(1)$
$cycle(2) \to tree(1)$
$tree(l) \to tree(2 * l)$, $tree(l) \to tree(2 * l + 1)$ *für* $l < 2^{n-1}$

Dabei bedeutet die Schreibweise $u \to v$, daß es eine Verbindungskante von Knoten u nach v gibt (aber nicht notwendigerweise umgekehrt).

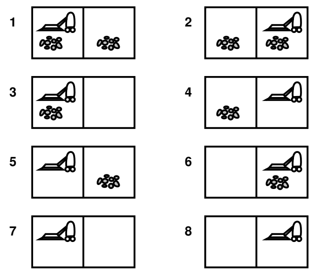
1. *Zeichnen Sie den Graphen $G(3)$! Beschreiben Sie, welche Probleme sich ergeben können bei der Suche eines Weges vom Startknoten aus! In welcher Reihenfolge werden bei Tiefen- und Breitensuche die Knoten im Graphen $G(3)$ besucht?*

2. *Programmieren Sie Tiefen- und Breitensuche in Graphen explizit in Prolog. Der zu untersuchende Graph sei dabei gegeben durch Fakten zum Prädikat $edge/2$. Der Startknoten S sei durch das Faktum $init(S)$. ausgezeichnet.*

   Definieren Sie ein Prädikat $search/2$, welches im zweiten Argument alle vom Startknoten aus erreichbaren Knoten genau einmal nacheinander durch Backtracking berechnet, und zwar in der durch die Suchstrategie festgelegten Reihenfolge. Jeder Knoten soll nur genau einmal berechnet werden.

Aufgabe 3.4 (Multiple-Zustand-Probleme)
In der Vorlesung haben Sie 1-Zustand-Probleme kennengelernt. Verallgemeinern Sie die Definition auf Multiple-Zustand-Probleme!

Aufgabe 3.5 (Murphy's Law)
Betrachten Sie die folgende sehr eingeschränkte und zugängliche Staubsauger-Welt mit acht Zuständen.



Zeigen Sie, daß zu jedem Anfangszustand eine Sequenz von Aktionen existiert, die einen Zielstand erreicht, d.h. beide Gebiete sind sauber, selbst wenn Murphy's Law *gilt:*

> *Die Aktion* SAUGEN *verursacht manchmal Dreck, wenn vorher kein Schmutz dagewesen ist.*

Zeichnen Sie auch ein Zustandübergangsdiagramm!

4 Informierte Suche

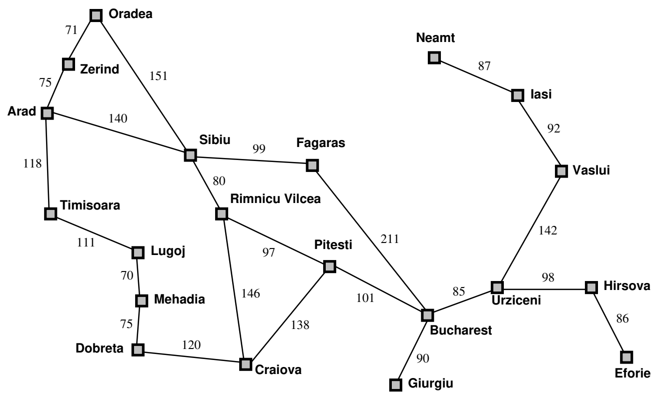
Nutze Problem-spezifisches Wissen um die Suche intelligenter zu machen. **General-Search** in Tabelle 3.2 ist schon recht weit instantiiert. Einzige Freiheit: Queing-Fn.

4.1 Best-First-Suche

Angenommen wir haben eine Evaluierungsfunktion f die jedem Knoten n einen Wert $f(n)$ zuordnet. Wir ändern nun die Queing-Fn so ab, daß die Knoten mit kleinstem f-Wert ganz vorne in der Queue liegen: die Queue wird also gemäss f geordnet.

```
function BEST-FIRST-SEARCH(problem, EVAL-FN) returns a solution sequence
  inputs: problem, a problem
          Eval-Fn, an evaluation function

  Queueing-Fn ← a function that orders nodes by EVAL-FN
  return GENERAL-SEARCH(problem, Queueing-Fn)
```

Beispiel 4.1.1 (Weg-Suche)



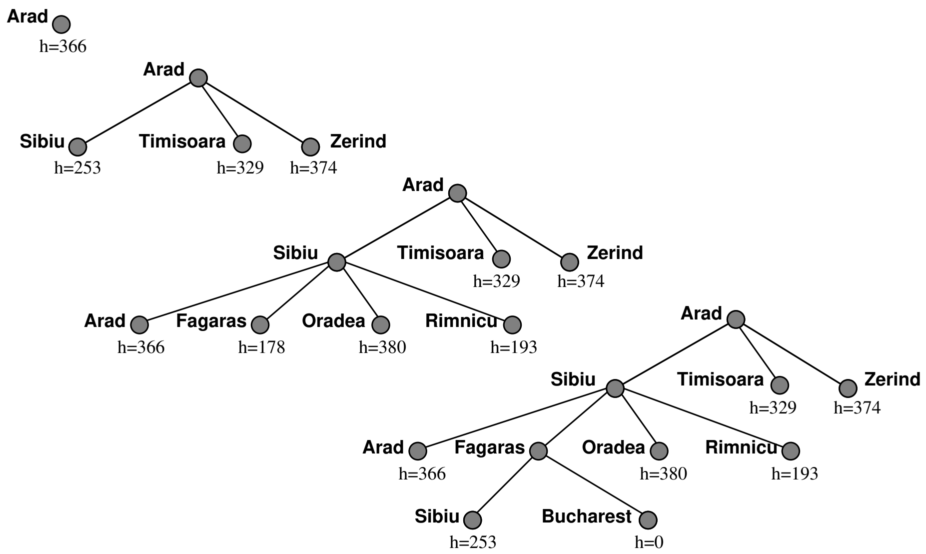
Straight−line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

$h(n)$ kann als Luftlinien-Entfernung von Budapest bis zu der zu n gehörenden Stadt definiert werden.

Greedy-Suche: Hier benutzt man als Evaluierungsfunktion Eval_Fn(n):= die erwarteten Kosten des *optimalen* Pfades vom Zustand in n zum Ziel-Zustand. Optimal bedeutet dabei bezüglich der gegebenen Kostenfunktion g.

Diese Evaluierungsfunktion bezeichnet man auch als Heuristikfunktion und schreibt sie als $h(n)$.



Ist die Greedy[1]-Suche optimal? Ist sie vollständig?

Greedy ist nicht optimal (in Beispiel wird ein nicht optimaler Weg gefunden). Greedy ist auch nicht vollständig, denn es ist Oszillation möglich: betrachte die Aufgabe einen Weg von Iasi nach Fagaras zu finden.

A*-Suche: Hier benutzt man als Evaluierungsfunktion Eval_Fn(n) die Summe aus einer Heuristikfunktion $h(n)$ und den tatsächlichen Pfadkosten $g(n)$:

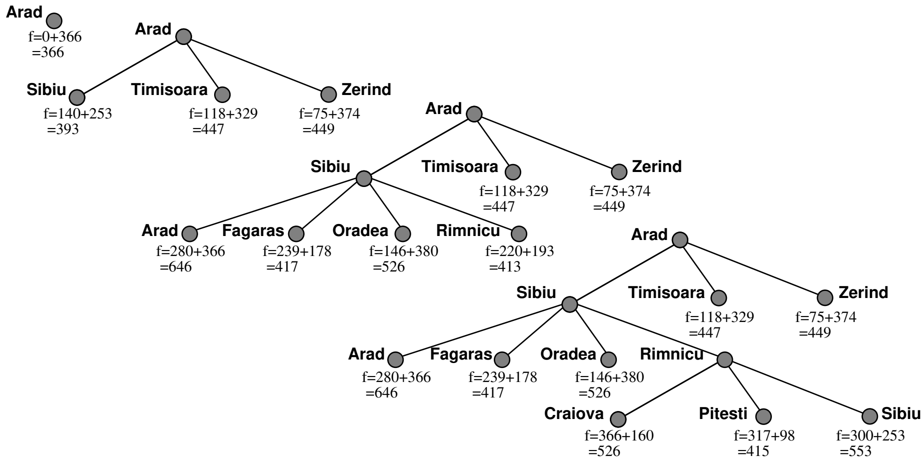
$$\text{Eval\_Fn}(n := h(n) + g(n)$$

Diese Evaluierungsfunktion schreiben wir auch einfach als $f(n)$.

A*-Suche ist also “Greedy + Uniform-Cost”, denn für Zielzustände n_z gilt ja $h(n_z) = 0$, also Eval_Fn$(n_z) = g(n_z)$.

Wir stellen allerdings eine Bedingung an die Heuristikfunktion h: $h(n)$ soll immer kleiner sein als die optimalen Kosten von n zu einem Ziel-Knoten ($h(n) < g(n_z) - g(n)$)

[1] “Greedy” heißt gierig und ist eine der sieben Todsünden.



In unserem Beispiel ist f := Eval_Fn monoton. Im allgemeinen gilt das nicht. Allerdings können wir f leicht monoton wachsend machen:

Definition 4.1.2 ($f \rightsquigarrow$ monotones$f$)
Wir definieren eine modifizierte Funktion f^{mon} rekursiv nach der Knotentiefe

$$f^{mon}(n) := max\{f^{mon}(n'), f(n)\}$$

wobei n' der Elter von n ist.

Theorem 4.1.3 (Optimalität von A*)
A^* ist optimal.

Hier ein Beispiel für nichtmonotones f. Sei n' der Elter von n. Sei $g(n') = 3$ und $g(n) = 4$, und $h(n') = 4$ und $h(n) = 2$. Damit ist $f(n') = 7$ und $f(n) = 6$.

Beweis: Sei G ein optimaler Zielzustand, und G' ein suboptimaler, d.h. $g(G') > g(G)$. Angenommen, A* hat G' aus der Queue ausgewählt. Dann hat A* irgendwann einmal einen Knoten n_0 der zum optimalen Pfad nach G führt "verschmäht" (also nicht expandiert), d.h. n_0 kam in die Queue. Nun gilt aber

$$g(G) \geq f(n_0)$$

wegen der Bedingung an h, und es gilt auch

$$f(n_0) \geq f(G')$$

da ja G' und nicht n_0 (der noch in der Queue sein muss) ausgewählt wurde.

Nimmt man beide Gleichungen zusammen erhält man

$$g(G) \geq f(G') = g(G')$$

was ein Widerspruch zu unserer Annahme ist. ■

Für die Vollständigkeit müssen wir 2 Fälle ausschliessen:

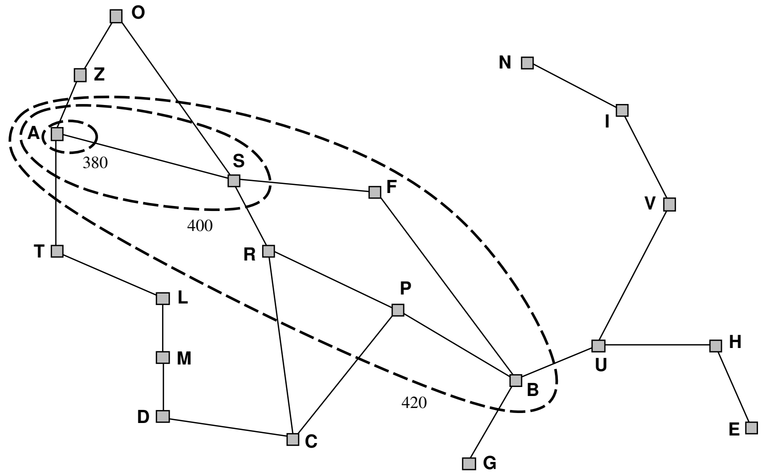


Abbildung 4.1: Der A*-Algorithmus

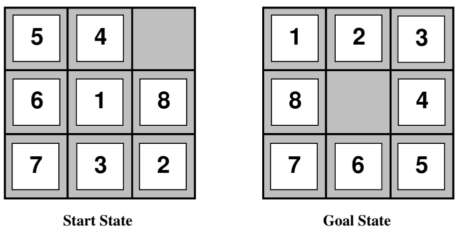
1. Es werden auf einen Schlag unendlich viele Knoten erzeugt.
2. Es werden Knoten erzeugt deren Evaluierungsfunktion zwar wächst, aber unter einem bestimmten Wert ($g(Ziel)$) bleibt.

Theorem 4.1.4 (Vollständigkeit von A*)
A* ist vollständig für lokal endliche Graphen mit: $\exists \delta > 0$: *jeder Operator erhöht* $g(n)$ *um mindestens* δ.

A* zieht Konturen die zielgerichtet sind. Was macht Uniform-Cost? Uniform-Cost zieht konzentrische Kreise!
Die Anzahl der Knoten im Speicher ist exponentiell in der Länge der Lösung. Auch bei der A*-Suche ist der Speicher das Problem, nicht die Zeit.
Dies führt zu

- **IDA*:** A* + Iterative Deepening,
- **SMA*:** Simplified Memory Bounded A* ($\rightsquigarrow$ Abschnitt 4.3)

4.2 Beispiele für Heuristiken



Was ist eigentlich der Verzweigungsfaktor in diesem Problem? Ungefähr 3. Wieviele Zustände müssen schlimmstenfalls betrachtet werden? Ungefähr

$$3^g \approx 10^{10},$$

wobei g die Anzahl der Züge ist, um zu einer Lösung zu gelangen. g ist ungefähr 20. Aber es gibt ja nur $9! \approx 10^5$ Zustände! Mit anderen Worten: Testen auf Schleifen hilft sehr viel.

Welche Heuristikfunktionen bieten sich an?

Hamming-Distanz: h_1 ist Anzahl der Zahlen in falscher Position: $h_1(Anfang) = 8$.

Manhattan-Distanz: Wir rechnen für jedes Teil die Entfernung zur richtigen Position aus und summieren diese:

$$h_2 := \sum_{i=1}^{8} \text{Entfernung von } i \text{ zur richtigen Position}$$

Z.B. $h_2(Anfang) = 2 + 3 + 3 + 2 + 4 + 2 + 0 + 2 = 18$.

Strenggenommen gibt es keinen einheitlichen Verzweigungsfaktor. 1 Zustand hat VF 4 (das Kästchen in der Mitte), die 4 Eckzustände haben VF 2, und die 4 übrigen Zustände haben 3.

Wie kann man die Qualität einer Heuristik messen? Möglichst in einem einzigen Wert!

Definition 4.2.1 (Effektiver Verzweigungsfaktor b^*)
Angenommen, A^* findet die optimale Lösung für eine Problem Instanz in der Tiefe d und musste N Knoten expandieren. Dann heißt b^* definiert durch

$$N = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$$

der effektive Verzweigungsfaktor von A^*.

Man beachte, daß b^* nicht nur von h sondern auch von der speziellen Problem-Instanz abhängt. Allerdings stellt man fest, daß b^* ziemlich konstant für grosse Klassen von Problem-Instanzen ist.

	Search Cost			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

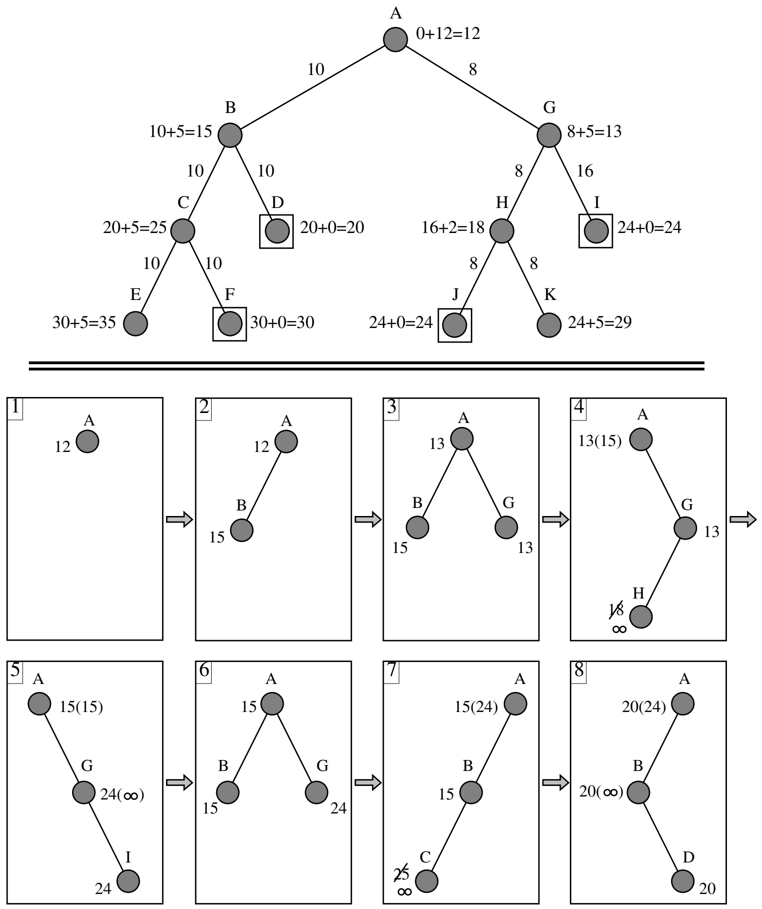
4.3 SMA*-Suche

A* braucht viel Speicher. Was, wenn nur beschränkter Speicher vorhanden ist? Wenn also die Queue nur eine maximale Größe haben kann?

SMA* ist eine Erweiterung von A* die mit beschränktem Speicher auskommt. Falls kein Platz mehr da ist, aber Knoten expandiert werden müssen, werden Knoten aus der Queue gestrichen:

- und zwar solche mit möglichst großen f-Kosten (*forgotten nodes*). Deren f-Kosten merkt man sich aber. Später werden diese Knoten nur dann wieder betrachtet, wenn alle anderen Pfade zu höheren Kosten führen.

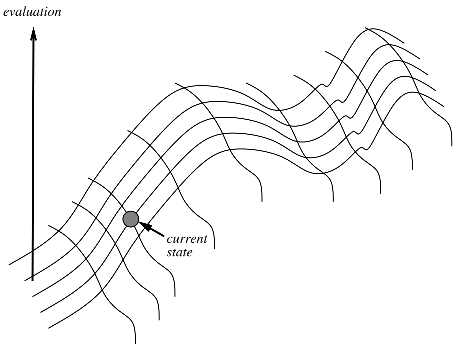
Theorem 4.3.1 (Eigenschaften von SMA*)
SMA* ist vollständig, falls genügend Speicher da ist um den **kürzesten** *Lösungspfad zu speichern. SMA* ist optimal, falls genügend Speicher da ist um den* **optimalen** *Lösungspfad zu speichern.*



```
function SMA*(problem) returns a solution sequence
  inputs: problem, a problem
  static: Queue, a queue of nodes ordered by f-cost

  Queue ← MAKE-QUEUE({MAKE-NODE(INITIAL-STATE[problem])})
  loop do
    if Queue is empty then return failure
    n ← deepest least-f-cost node in Queue
    if GOAL-TEST(n) then return success
    s ← NEXT-SUCCESSOR(n)
    if s is not a goal and is at maximum depth then
      f(s) ← ∞
    else
      f(s) ← MAX(f(n), g(s)+h(s))
    if all of n's successors have been generated then
      update n's f-cost and those of its ancestors if necessary
    if SUCCESSORS(n) all in memory then remove n from Queue
    if memory is full then
      delete shallowest, highest-f-cost node in Queue
      remove it from its parent's successor list
      insert its parent on Queue if necessary
    insert s on Queue
  end
```

4.4 Iterative Verbesserungen

Bei manchen Problemen ist nur der jeweilige Zustand wichtig, auf den Pfad der dorthin führt kommt es nicht an.



Das Problem ist natürlich beliebig schwierig!

Hill-Climbing: Gradienten-Abstieg (bzw. Aufstieg). Gehe zufällig in irgendeine Richtung. Vergleiche die neue Evaluierung mit der alten. Gehe vom neuen Punkt aus, falls die Evaluierung besser ist.

Probleme:

- Lokale Maxima (verlaufen),
- Plateaux (umherirren),
- Grat (Umwege).

```
function HILL-CLIMBING(problem) returns a solution state
   inputs: problem, a problem
   static: current, a node
           next, a node

   current ← MAKE-NODE(INITIAL-STATE[problem])
   loop do
       next ← a highest-valued successor of current
       if VALUE[next] < VALUE[current] then return current
       current ← next
   end
```

Simulated Annealing: "Abkühlung", modifiziertes Hill-Climbing: auch schlechte Züge (Eval. ist klein) werden mit einer gewissen Wahrscheinlichkeit zugelassen, und zwar mit $e^{-\frac{|\Delta Eval|}{T}}$. Diese hängt ab von der "Temperatur" T, die von ∞ nach 0 geht.

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to "temperature"
   static: current, a node
           next, a node
           T, a "temperature" controlling the probability of downward steps

   current ← MAKE-NODE(INITIAL-STATE[problem])
   for t ← 1 to ∞ do
       T ← schedule[t]
       if T=0 then return current
       next ← a randomly selected successor of current
       ΔE ← VALUE[next] – VALUE[current]
       if ΔE > 0 then current ← next
       else current ← next only with probability e^(ΔE/T)
```

Man finde den Gipfel des Mount Everest, und zwar bei dichtem Nebel und Gedächtnisverlust!

4.5 Aufgaben

Aufgabe 4.1 (Heuristische Suche)
Beim 8-Schiebepuzzle haben Sie zwei Heuristiken kennengelernt: (1) Hamming-Distanz, d.h. Anzahl der falschen Positionen; (2) Manhattan-Distanz.

1. *Welche Heuristik ist besser? Oder sind sie gar nicht miteinander vergleichbar?*
2. *Überlegen Sie sich ein formales Kriterium zum Vergleich der Güte von Heuristiken!*

Aufgabe 4.2 (Rundreisen)
Ein Geschäftsmann will eine möglichst kurze Rundreise durch die vier Orte A, B, C und D machen. Beginn und Ende der Reise ist der Ort A. Die Entfernungen seien durch die folgende Matrix gegeben. Der letzte Eintrag in der ersten Zeile besagt z.B., daß die Entfernung von A nach D (in dieser Richtung) 7 Einheiten beträgt.

	A	B	C	D
A	-	4	5	7
B	3	-	4	9
C	5	4	-	6
D	6	8	7	-

Bestimmen Sie die kürzeste Rundreise unter Anwendung des A-Algorithmus' und zeichnen Sie den Suchbaum! Als heuristische Funktion h verwenden die direkte Entfernung vom erreichten Ort zurück nach A als Abschätzung für den restlichen Weg.*

Aufgabe 4.3 (Greedy-Algorithmus)

1. *Wie verhält sich der Greedy-Algorithmus, falls als heuristische Funktion $h(n) = -g(n)$ angesetzt wird? Zur Erinnerung: $g(n)$ sind die Kosten des Pfades bis zum Knoten n.*

2. *Beim Beispiel aus der Vorlesung (Reise durch Rumänien) läuft der Greedy-Algorithmus mit Luftlinien-Heuristk in die Irre beim Suchen eines Weges von Iasi nach Fagaras. In Gegenrichtung passiert das jedoch nicht. Gibt es Fälle, wo dies in beide Richtungen geschehen kann?*

Aufgabe 4.4 (SMA*)

1. *SMA* sieht von Pfaden ab, die allein den ganzen verfügbaren Speicher für sich beanspruchen, aber keinen Zielzustand erreichen. Zeigen Sie, daß das Verfahren ohne diesen Test in eine Endlosschleife gerät, falls nicht genügend Speicher da ist, um den kürzesten Lösungspfad abzulegen.*

2. *Zeigen Sie, daß kein Baum zweimal generiert wird im Verlaufe der Suche. Tip: Der Beweis kann per Induktion über die Baumtiefe geführt werden, beginnend mit der maximalen Tiefe.*

Aufgabe 4.5 (Tiefensuche versus Iterative-Deepening)
Betrachten Sie exponentielle Suchbäume mit einem Verzweigungsfaktor $b > 1$ und einer Tiefe $d \in \mathbb{N}$.

1. *Berechnen Sie, wie viele Knoten (i) bei Tiefensuche und (ii) bei Anwendung von Iterative-Deepening expandiert werden!*

2. *Zeigen Sie, daß für $d \to \infty$ der Quotient von (ii) und (i) gegen $b/(b-1)$ geht!*

5 Wissensrepräsentation I

5.1 Aussagenlogik (AL)

Definition 5.1.1 (Aussagenlogik $\mathcal{L}_{AL}$, Sprache $\mathcal{L} \subseteq \mathcal{L}_{AL}$)
Die Sprache $\mathcal{L}_{AL}$ der Aussagenlogik besteht aus

- $\Box$ *und* $\top$*: dem falsum und dem verum,*
- $p, q, r, x_1, x_2, \ldots x_n, \ldots$*: einer abzählbaren Menge* $\mathcal{AT}$ *von AL- Konstanten,*
- $\neg$, $\wedge$, $\vee$, $\rightarrow$*: den Junktoren (*$\neg$ *ist einstellig, alle anderen zweistellig),*
- (,)*: den Klammern, die der eindeutigen Lesbarkeit dienen.*

Üblicherweise betrachten wir nur eine endliche Menge von aussagenlogischen Konstanten. Allgemein legt eine Teilmenge von $\mathcal{AT}$ *eine Sprache* $\mathcal{L} \subseteq \mathcal{L}_{AL}$ *fest. Daraus ergibt sich induktiv in natürlicher Weise die Menge aller* $\mathcal{L}$*-Formeln* $Fml_{\mathcal{L}}$.

Definition 5.1.2 (Semantik, Bewertung, Modell)
Eine Bewertung v *für eine Sprache* $\mathcal{L} \subseteq \mathcal{L}_{AL}$ *ist eine Abbildung auf der Menge der durch* $\mathcal{L}$ *festgelegten aussagenlogischen Konstanten und* $\{\top, \Box\}$ *in die Menge* $\{\mathbf{t}, \mathbf{f}\}$ *mit* $v(\Box) = \mathbf{f}$, $v(\top) = \mathbf{t}$.

Jede Bewertung v *kann zu einer eindeutigen Funktion* $\bar{v} : Fml_{\mathcal{L}} \rightarrow \{\mathbf{t}, \mathbf{f}\}$ *so fortgesetzt werden, daß gilt:*

- $\bar{v}(\neg p) = \begin{cases} \mathbf{t}, & \textit{falls } \bar{v}(p) = \mathbf{f}, \\ \mathbf{f}, & \textit{falls } \bar{v}(p) = \mathbf{t}. \end{cases}$
- $\bar{v}(\varphi \wedge \gamma) = \begin{cases} \mathbf{t}, & \textit{falls } \bar{v}(\varphi) = \mathbf{t} \textit{ und } \bar{v}(\gamma) = \mathbf{t}, \\ \mathbf{f}, & \textit{sonst.} \end{cases}$
- $\bar{v}(\varphi \vee \gamma) = \begin{cases} \mathbf{t}, & \textit{falls } \bar{v}(\varphi) = \mathbf{t} \textit{ oder } \bar{v}(\gamma) = \mathbf{t}, \\ \mathbf{f}, & \textit{sonst.} \end{cases}$
- $\bar{v}(\varphi \rightarrow \gamma) = \begin{cases} \mathbf{t}, & \textit{falls } \bar{v}(\varphi) = \mathbf{f} \textit{ oder } (\bar{v}(\varphi) = \mathbf{t} \textit{ und } \bar{v}(\gamma) = \mathbf{t}), \\ \mathbf{f}, & \textit{sonst.} \end{cases}$

Eine Bewertung v *legt daher in eindeutiger Weise ein* $\bar{v}$ *fest. Wir nennen* $\bar{v}$ *auch* $\mathcal{L}$*-Struktur oder Modell* $\mathcal{A}_v$*. Ein Modell legt also für jede Formel fest, ob sie wahr ist oder falsch. Im folgenden werden wir sowohl von Modellen als auch von Bewertungen sprechen.*

Definition 5.1.3 (Gültigkeit einer Formel, Tautologie)

1. *Eine Formel $\varphi \in Fml_{\mathcal{L}}$ gilt unter der Belegung v, falls $\bar{v}(\varphi) = \mathbf{t}$. Wir schreiben auch $\bar{v} \models \varphi$ oder einfach $v \models \varphi$.*
2. *Eine Theorie ist eine Menge von Formeln : $T \subseteq Fml_{\mathcal{L}}$. v erfüllt T falls $\bar{v}(\varphi) = \mathbf{t}$ für alle $\varphi \subseteq T$. Wir schreiben $v \models T$.*
3. *Eine $\mathcal{L}$-Formel φ heißt $\mathcal{L}$-Tautologie, falls für alle möglichen Bewertungen v auf $\mathcal{L}$ gilt: $v \models \varphi$.*

Im folgenden lassen wir die zugrundeliegende Sprache $\mathcal{L}$ meistens weg, da sie aus dem Zusammenhang klar ersichtlich ist. Trotzdem muß sie natürlich immer festgelegt sein.

Definition 5.1.4 (Folgerungsmenge $Cn(T)$)
Eine Formel φ folgt aus T, falls für alle Bewertungen v mit $v \models T$ auch $v \models \varphi$ gilt. Dafür schreiben wir: $T \models \varphi$.

Wir bezeichnen mit $Cn_{\mathcal{L}}(T) :=_{def} \{\varphi \in Fml_{\mathcal{L}} : T \models \varphi\}$, oder einfach $Cn(T)$, den semantischen Konsequenzoperator.

Lemma 5.1.5 (Eigenschaften von $Cn(T)$)
Der semantische Konsequenzoperator hat folgende Eigenschaften:

1. T-Erweiterung: $T \subseteq Cn(T)$,
2. Monotonie: $T \subseteq T' \Rightarrow Cn(T) \subseteq Cn(T')$,
3. Abgeschlossenheit: $Cn(Cn(T)) = Cn(T)$.

Lemma 5.1.6 ($\varphi \notin$ Cn(T))
$\varphi \notin Cn(T)$ gdw *es gibt eine Bewertung v mit $v \models T$ und $\bar{v}(\varphi) = \mathbf{f}$.*

Definition 5.1.7 (MOD(T), $Cn(\mathcal{U})$)
Ist $T \subseteq Fml_{\mathcal{L}}$, so bezeichnen wir mit MOD(T) die Menge aller $\mathcal{L}$-Strukturen $\mathcal{A}$ die Modelle von T sind:

$$\mathit{MOD}(T) :=_{def} \{\mathcal{A} : \mathcal{A} \models T\}.$$

Ist $\mathcal{U}$ eine Menge von Modellen (also Bewertungen), so können wir alle jene Sätze betrachten, die in allen Modellen gelten. Diese Menge bezeichnen wir ebenfalls mit $Cn(\mathcal{U})$:

$$Cn(\mathcal{U}) :=_{def} \{\varphi \in Fml_{\mathcal{L}} : \forall v \in \mathcal{U} : \bar{v}(\varphi) = \mathbf{t}\}.$$

Offenbar ist MOD der zu Cn duale Begriff:

$$Cn(\mathrm{MOD}(T)) = Cn(T), \;\; \mathrm{MOD}(Cn(T)) = \mathrm{MOD}(T).$$

Definition 5.1.8 (Vollständigkeit einer Theorie T)
T heißt vollständig, falls für jede Formel $\varphi \in Fml$: $T \models \varphi$ oder $T \models \neg\varphi$.

Diese letzte Bedingung ist nicht zu verwechseln mit der entsprechenden Eigenschaft einer Bewertung v bzw. in einem Modell: Jede Bewertung hat natürlich diese Eigenschaft.

Definition 5.1.9 (Konsistenz einer Theorie)
T heißt konsistent, falls es eine Bewertung v mit $\bar{v}(\varphi) = \mathbf{t}$ für alle $\varphi \in T$ gibt.

Lemma 5.1.10 (Ex Falso Quodlibet)
T ist konsistent gdw $Cn(T) \neq Fml_{\mathcal{L}}$.

5.2 Kalküle für die AL

Definition 5.2.1 (Axiom, Inferenzregel)
Axiome der Aussagenlogik sind die folgenden Formeln:

- $p \to \top$, $\Box \to p$, $\neg\top \to \Box$, $\Box \to \neg\top$,
- $(p \land q) \to p$, $(p \land q) \to q$,
- $p \to (p \lor q)$, $q \to (p \lor q)$,
- $\neg\neg p \to p$, $(p \to q) \to ((p \to \neg q) \to \neg p)$,
- $p \to (q \to p)$, $p \to (q \to (p \land q))$.

Einzige Inferenzregel der Aussagenlogik ist die folgende Abbildung Modus Ponens

$$MP : Fml \times Fml \to Fml : (\varphi, \varphi \to \psi) \mapsto \psi.$$

Wir schreiben dafür auch kurz

$$\textbf{(MP)} \ \ \frac{\varphi,\ \varphi \to \psi}{\psi}.$$

Die Inferenzregel ist so zu verstehen, daß φ, ψ beliebige zusammengesetzte Formeln sein können.

Definition 5.2.2 (Beweis)
Ein Beweis einer Formel φ aus einer Theorie $T \subseteq Fml_{\mathcal{L}}$ ist eine Folge $\varphi_1, \ldots, \varphi_n$ von Formeln so, daß $\varphi_n = \varphi$ und für alle i mit $1 \leq i \leq n$ gilt eine der folgenden Bedingungen:

- *φ_i ist Substitutions-Instanz eines Axioms,*
- *$\varphi_i \in T$,*
- *Es gibt $\varphi_l, \varphi_k = (\varphi_l \to \varphi_i)$ mit $l, k < i$. φ_i ist also das Resultat der Anwendung des Modus Ponens auf Vorgänger-Formeln von φ_i.*

Wir schreiben: $T \vdash \varphi$.

Theorem 5.2.3 (Vollständigkeit)
Eine Formel folgt semantisch aus einer Theorie T genau dann, wenn sie abgeleitet werden kann:

$$T \vdash \varphi \ \underline{\text{gdw}} \ T \models \varphi$$

Theorem 5.2.4 (Kompaktheit)
Eine Formel folgt aus einer Theorie T genau dann, wenn sie schon aus einer endlichen Teilmenge von T folgt:

$$Cn(T) = \bigcup \{Cn(T') : T' \subseteq T, T' \textit{ endlich}\}.$$

Obwohl wir in der Aussagenlogik mit den obengenannten Axiomen und dem Modus Ponens auskommen, ist es sinnvoll, allgemeinere Systeme zu betrachten. Dazu führen wir den folgenden Begriff ein:

Definition 5.2.5 (Regelsystem MP + D)
Sei D eine Menge von allgemeinen Inferenzregeln, d.h. Abbildungen, die einer endlichen Menge von Formeln $\varphi_1, \varphi_2, \dots, \varphi_n$ eine Formel ψ zuordnen. Wir schreiben dies als

$$\frac{\varphi_1, \varphi_2, \dots, \varphi_n}{\psi}.$$

Unter MP + D verstehen wir das Regelsystem, das entsteht, wenn die Regeln aus D zur Regel Modus Ponens hinzugenommen werden.

Für $W \subseteq Fml$ sei im folgenden

$$Cn^D(W)$$

die Menge aller Formeln φ, die man aus W und den Inferenzregeln aus MP + D (im Sinne der Definition 5.12, nun dürfen aber zusätzlich die Regeln aus D angewandt werden) ableiten kann.

Man beachte, daß $Cn(W)$ semantisch, $Cn^D(W)$ aber syntaktisch (über den Begriff des Beweises) definiert sind. Im Spezialfall $D = \emptyset$ stimmen beide Mengen nach dem Vollständigkeitssatz überein.

Lemma 5.2.6 (Eigenschaften von Cn^D)
Sei D eine Menge von allgemeinen Inferenzregeln und $W \subseteq Fml$. Dann gilt:

1. $Cn(W) \subseteq Cn^D(W)$.
2. $Cn^D(Cn^D(W)) = Cn^D(W)$.
3. $Cn^D(W)$ *ist die kleinste unter D abgeschlossene Menge die W enthält.*

Was ist der Unterschied zwischen einer Inferenzregel $\frac{\varphi}{\psi}$ und der Implikation $\varphi \to \psi$? Angenommen, wir haben eine Formelmenge T gegeben und wir wählen zwei Konstanten $p, q \in \mathcal{L}$. Man könnte doch einerseits

(1) T zusammen mit MP und $\{\frac{p}{q}\}$

und andererseits

(2) $T \cup \{p \to q\}$ zusammen mit MP

betrachten. Im ersten Fall bedeutet dies

$$Cn^{\{\frac{p}{q}\}}(T)$$

und im zweiten Fall

$$Cn(T \cup \{p \to q\}).$$

Ein Resolutionskalkül für AL

Gegeben sei eine Menge M von Klauseln der Form

$$A \vee \neg B \vee C \vee \ldots \vee \neg E$$

Solch eine Klausel schreiben wir auch als Menge

$$\{A, \neg B, C, \ldots, \neg E\}.$$

Wir definieren die folgende Inferenzregel:

Definition 5.2.7 (AL-Resolution)
Leite aus $C_1 \cup \{A\}$ und $C_2 \cup \{\neg A\}$ die Klausel $C_1 \cup C_2$ ab.

Ist dieser Kalkül korrekt und vollständig?
Jedes Problem der Form “$T \vdash \phi''$ ist äquivalent zu

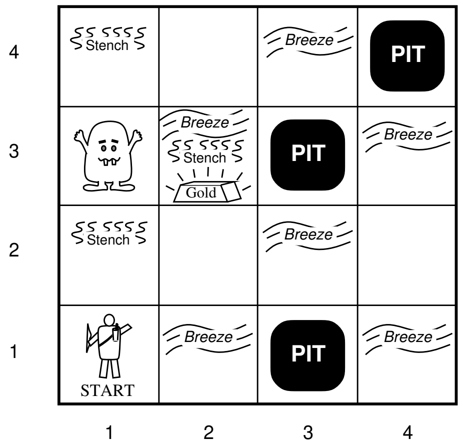
$$T \cup \{\neg\phi\} \text{ ist unerfüllbar”}$$

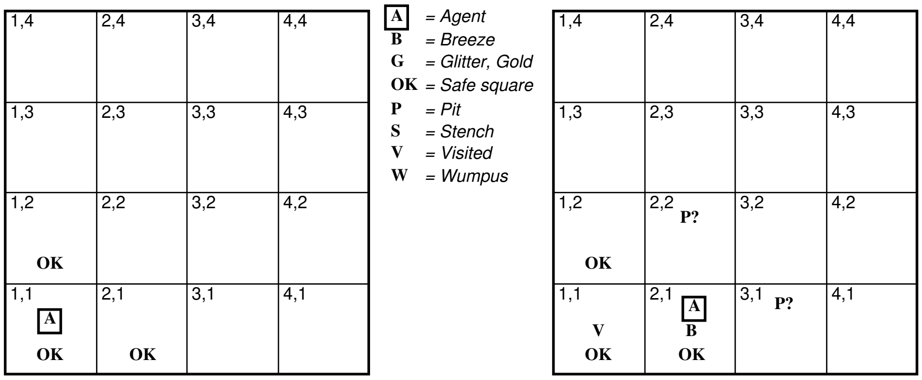
bzw. zu

$$T \cup \{\neg\phi\} \vdash \Box.$$

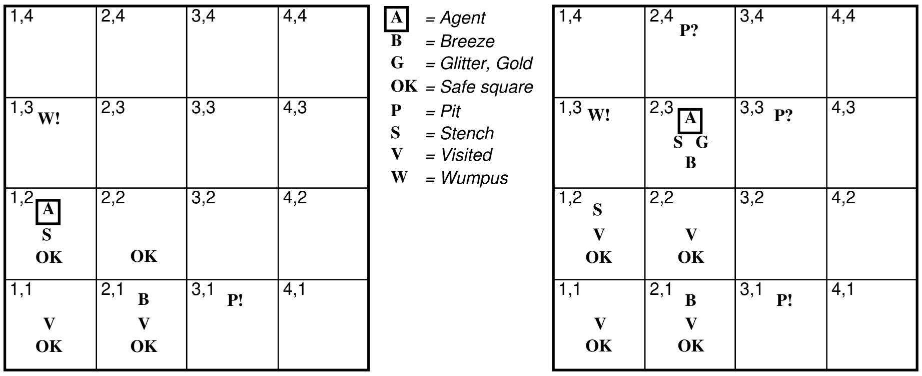
Theorem 5.2.8 (Widerlegungsvollständigkeit)
Ist M eine unerfüllbare Menge von Klauseln, so kann aus M allein mit der Resolutionsregel die leere Klausel $\Box$ abgeleitet werden.

5.3 Die Wumpus-Welt in AL





(a) (b)



(a) (b)

Festlegung der Sprache:

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

Allgemeines Wissen:

$$\begin{array}{lcl} \neg S_{1,1} & \longrightarrow & (\neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,1}) \\ \neg S_{2,1} & \longrightarrow & (\neg W_{1,1} \wedge \neg W_{2,1} \wedge \neg W_{2,2} \wedge \neg W_{3,1}) \\ \neg S_{1,2} & \longrightarrow & (\neg W_{1,1} \wedge \neg W_{1,2} \wedge \neg W_{2,2} \wedge \neg W_{1,3}) \\ & & \\ S_{1,2} & \longrightarrow & (W_{1,3} \wedge W_{1,2} \wedge W_{2,2} \wedge W_{1,1}) \end{array}$$

Wissen nach dem 3. Zug:

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

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

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

Zusätzliche Axiome:

$$\begin{array}{lcl} A_{1,1} \wedge East \wedge W_{2,1} & \longrightarrow & \neg Forward \\ A_{1,1} \wedge East \wedge Grube_{2,1} & \longrightarrow & \neg Forward \\ A_{i,j} \wedge Gl_{i,j} & \longrightarrow & Take_{Gold} \end{array}$$

Nachteile:

- Aktionen können nur geraten werden,
- Datenbasis muss fortlaufend geändert werden,
- Regelmenge werden sehr gross, da keine Variablen vorhanden.

Bei geeigneter Formalisierung (zusätzliche Axiome) kann man jedenfalls prüfen ob

$$\text{KB} \vdash \neg \text{ aktion} \quad \text{bzw. KB} \vdash \text{aktion}$$

Es kann auch sein, dass weder das Eine noch das Andere ableitbar ist.

5.4 Das P=NP Problem

Beobachtung: Polynomiale Algorithmen sind schnell, exponentielle dagegen langsam.

Um eine AL-Formel mit der Wahrheitstafel-Methode als unerfüllbar nachzuweisen, braucht man im schlimmsten Fall 2^n Belegungen (bei n-Konstanten). *Kann man dieses Problem vielleicht mit anderen Methoden in polynomialer Zeit lösen?*

Wir betrachten Ja-Nein Probleme. Für jede Instanz $\mathcal{I}$ eines Problems können wir fragen ob $\mathcal{I}$ die entsprechende Eigenschaft hat oder nicht: $\mathcal{I}$ ist Ja oder Nein-Instanz.

Definition 5.4.1 (Klasse P)
Ein Problem liegt in der Klasse **P** *der in polynomialer deterministischer Zeit erkennbaren Probleme, falls es einen Algorithmus* Alg. *und ein Polynom* $p(n)$ *gibt, so dass für jede Instanz* $\mathcal{I}$ *gilt:*

$\mathcal{I}$ *ist Ja-Instanz*	*gdw*	Alg. *antwortet "Ja" auf Eingabe von* $\mathcal{I}$ *nach höchstens* $p(\textit{länge}(\mathcal{I}))$*-Schritten.*

Definition 5.4.2 (Klasse NP)
Ein Problem liegt in der Klasse **NP** *der in polynomialer nicht-deterministischer Zeit erkennbaren Probleme, falls es einen Algorithmus* Check *und ein Polynom* $p(n)$ *gibt so dass für jede Instanz* $\mathcal{I}$ *gilt:*

$\mathcal{I}$ *ist Ja-Instanz*	*gdw*	*es gibt eine Instanz* $\mathcal{I}'$ *eines anderen Problems mit* *(1)* $\textit{länge}(\mathcal{I}') < p(\textit{länge}(\mathcal{I}))$, *(2)* Check *antwortet "Ja" auf Eingabe von* $\mathcal{I}'$ *nach höchstens* $p(\textit{länge}(\mathcal{I}))$*-Schritten.*

Man sieht leicht, dass die Erfüllbarkeit einer AL-Formel in **NP** liegt, und dass **NP** jedenfalls **P** umfasst. Was sind die schwierigsten Probleme in **NP**?

Offenbar sind Probleme die man polynomial aufeinander reduzieren kann gleichwertig. Ein Problem Prob_1 ist polynomial transformierbar zu einem Problem Prob_2, falls man für jede Instanz $\mathcal{I}_1$ von Prob_1 in polynomialer Zeit eine Instanz $\mathcal{I}_2$ von Prob_2 berechnen können, so dass gilt:

$\mathcal{I}_1$ ist Ja-Instanz von Prob_1 **genau dann wenn** $\mathcal{I}_2$ ist Ja-Instanz von Prob_2

Definition 5.4.3 (NP-Vollständigkeit)
Ein Problem aus **NP** *heißt NP-vollständig, wenn sich alle anderen Probleme aus* **NP** *in polynomialer Weise auf es reduzieren lassen.*

Folgende Probleme sind NP-vollständig:

- Gegeben ein endlicher Graph und $k \in \mathbb{N}$. Kann der Graph mit k-Farben gefärbt werden?
- Gegeben ein endlicher Graph. Gibt es einen Hamilton-Kreis?
- Gegeben eine AL-Formel ϕ. Ist ϕ erfüllbar?

- Gegeben eine Menge von Klauseln in den Konstanten $c_1, \ldots, c_n$. Ist die Menge erfüllbar?
- Gegeben eine Menge von Klauseln in den Konstanten $c_1, \ldots, c_n$. Jede Klausel enthält höchstens 3 Literale. Ist die Menge erfüllbar?

5.5 Prädikatenlogik erster Stufe (PL1)

Definition 5.5.1 (Prädikatenlogik 1. Stufe $\mathcal{L}_{PL1}$, Sprache $\mathcal{L} \subseteq \mathcal{L}_{PL1}$)
Die Sprache $\mathcal{L}_{PL1}$ der Prädikatenlogik erster Stufe besteht aus

- $x, y, z, x_1, x_2, \ldots, x_n, \ldots$*: einer abzählbaren Menge* $\mathcal{VAR}$ *von Variablen,*
- *für jedes* $k \in \mathbb{N}$*:* $P_1^k, P_2^k, \ldots, P_n^k, \ldots$ *einer abzählbaren Menge* $\mathcal{PRED}^k$ *von k-stelligen Prädikatssymbolen*[1]*,*
- *für jedes* $k \in \mathbb{N}$*:* $f_1^k, f_2^k, \ldots, f_n^k, \ldots$ *einer abzählbaren Menge* $\mathcal{FUNCT}^k$ *von k-stelligen Funktionssymbolen,*
- $\neg$, $\wedge$, $\vee$*: den Junktoren (*$\neg$ *ist einstellig, alle anderen zweistellig),*
- (,)*: den Klammern,*
- $\forall$, $\exists$*: den Quantoren.*

0-stellige Funktionssymbole heißen Individuen-Konstanten, wobei wir die Klammern weglassen. Im allgemeinen werden wir, wie in der Aussagenlogik auch, nur eine gewisse Teilmenge der Prädikats- bzw. Funktionssymbole benötigen. Diese legen eine Sprache $\mathcal{L} \subseteq \mathcal{L}_{PL1}$ *fest (analog zu Definition 5.1.1). Die zugrundeliegende Menge der Funktions- und Prädikatssymbole bezeichnet man auch als Signatur* Σ*. Daraus ergeben sich induktiv der Begriff des* $\mathcal{L}$*-Termes* t *und der* $\mathcal{L}$*-Formel* φ*:*

Term: $\mathcal{L}$*-Terme* t *sind induktiv wie folgt definiert:*

1. *Jede Variable ist ein* $\mathcal{L}$*-Term.*
2. *Falls* f^k *ein Funktionssymbol mit Stelligkeit* k *aus* $\mathcal{L}$ *ist und* $t_1, \ldots, t_k$ $\mathcal{L}$*-Terme, so ist auch* $f^k(t_1, \ldots, t_k)$ *ein* $\mathcal{L}$*-Term.*

Die Menge aller $\mathcal{L}$*-Terme die man mit einer Menge* $X \subseteq \mathcal{VAR}$ *bilden kann bezeichnen wir mit* $Term_{\mathcal{L}}(X)$ *oder mit* $Term_{\Sigma}(X)$*. Für* $X = \emptyset$ *erhalten wir die Menge der Grundterme* $Term_{\mathcal{L}}(\emptyset)$ *oder kurz* $Term_{\mathcal{L}}$*.*

Formel: $\mathcal{L}$*-Formeln* φ *sind ebenfalls induktiv definiert:*

1. *Falls* P^k *ein Prädikatssymbol mit Stelligkeit* k *aus* $\mathcal{L}$ *ist und* $t_1, \ldots, t_k$ $\mathcal{L}$*-Terme, dann ist* $P^k(t_1, \ldots, t_k)$ *eine* $\mathcal{L}$*-Formel.*
2. *Für jede* $\mathcal{L}$*-Formel* φ *ist auch* $(\neg\varphi)$ *eine* $\mathcal{L}$*-Formel.*

[1] Die 0-stelligen Prädikatssymbole sind gerade die aussagenlogischen Konstanten aus $\mathcal{AT}$ von $\mathcal{L}_{AL}$. Wir setzen voraus, daß $\Box$ und $\top$ vorhanden sind.

3. *Für alle $\mathcal{L}$-Formeln φ und ψ sind auch $(\varphi \wedge \psi)$ und $(\varphi \vee \psi)$ $\mathcal{L}$-Formeln.*
4. *Falls x eine Variable und φ eine $\mathcal{L}$-Formel, so sind auch $(\exists\, x\, \varphi)$ und $(\forall\, x\, \varphi)$ $\mathcal{L}$-Formeln.*

Atomare $\mathcal{L}$-Formeln sind die, die gemäß 1. aufgebaut sind, wir bezeichnen sie mit $At_{\mathcal{L}}(X)$ ($X \subseteq \mathcal{VAR}$). Die Menge aller $\mathcal{L}$-Formeln bezüglich X bezeichnen wir mit $Fml_{\mathcal{L}}(X)$.

Positive Formeln, $Fml^+_{\mathcal{L}}(X)$ sind solche, die nur mittels 1., 3. *und* 4. *gebildet werden.*

Falls φ eine $\mathcal{L}$-Formel ist, die als Teil einer $\mathcal{L}$-Formel ψ auftritt, so heißt φ Teilformel von ψ.

Ein kleines Beispiel zur Illustration

Beispiel 5.5.2 (Von Halbgruppen zu Ringen)
Wir betrachten $\mathcal{L} = \{0, 1, +, \cdot, \leq, =\}$, wobei $0, 1$ Konstanten, $+, \cdot$ zweistellige Funktionszeichen und $\leq, =$ zweistellige Relationszeichen sind. Was kann man in dieser Sprache aufschreiben?

Ax 1:	$\forall x \forall y \forall z$	$x + (y + z) = (x + y) + z$
Ax 2:	$\forall x$	$x + 0 = 0 + x$
Ax 3:	$\forall x \forall y$	$x + y = 0 \wedge y + x = 0$
Ax 4:	$\forall x \forall y$	$x + y = y + x$
Ax 5:	$\forall x \forall y \forall z$	$x \cdot (y \cdot z) = (x \cdot y) \cdot z$
Ax 6:	$\forall x \forall y \forall z$	$x \cdot (y + z) = x \cdot y + x \cdot z$
Ax 7:	$\forall x \forall y \forall z$	$x \cdot (y + z) \cdot x = y \cdot x + z \cdot x$

Ax 1 beschreibt eine Halbgruppe*, Ax 1-2 ein* Monoid*, Ax 1-3 eine* Gruppe*, Ax 1-4 eine* abelsche Gruppe*, Ax 1-7 einen* Ring.

Definition 5.5.3 ($\mathcal{L}$-Struktur $\mathcal{A} = (U_{\mathcal{A}}, I_{\mathcal{A}})$)
Eine $\mathcal{L}$-Struktur oder $\mathcal{L}$-Interpretation ist ein Paar $\mathcal{A} :=_{def} (U_{\mathcal{A}}, I_{\mathcal{A}})$, wobei $U_{\mathcal{A}}$ eine beliebige nicht leere Menge ist, die die Grundmenge (oder das Universum bzw. der Individuenbereich) von $\mathcal{A}$ genannt wird. Ferner ist $I_{\mathcal{A}}$ eine Abbildung, die:

- *jedem k-stelligen Prädikatssymbol P^k aus $\mathcal{L}$ ein k-stelliges Prädikat über $U_{\mathcal{A}}$ zuordnet,*
- *jedem k-stelligen Funktionssymbol f^k aus $\mathcal{L}$ eine k-stellige Funktion auf $U_{\mathcal{A}}$ zuordnet.*

Mit anderen Worten: Der Definitionsbereich von $I_{\mathcal{A}}$ ist gerade die Menge der Prädikats- und Funktionssymbole von $\mathcal{L}$. Der Wertebereich von $I_{\mathcal{A}}$ besteht aus Prädikaten und Funktionen auf $U_{\mathcal{A}}$. Wir schreiben:

$$I_{\mathcal{A}}(P) = P^{\mathcal{A}},\ I_{\mathcal{A}}(f) = f^{\mathcal{A}}.$$

Sei φ eine $\mathcal{L}_1$-Formel und $\mathcal{A} :=_{def} (U_{\mathcal{A}}, I_{\mathcal{A}})$ eine $\mathcal{L}$-Struktur. $\mathcal{A}$ heißt zu φ passend, falls $I_{\mathcal{A}}$ für alle in φ vorkommenden Prädikatssymbole und Funktionssymbole definiert ist, falls also $\mathcal{L}_1 \subseteq \mathcal{L}$.

Definition 5.5.4 (Variablenbelegung ϱ)
Eine Variablenbelegung ϱ über einer $\mathcal{L}$-Struktur $\mathcal{A} = (U_{\mathcal{A}}, I_{\mathcal{A}})$ ist eine Funktion

$$\varrho : \mathcal{VAR} \rightarrow U_{\mathcal{A}};\ x \mapsto \varrho(x).$$

Definition 5.5.5 (Semantik der Prädikatenlogik)
Sei φ eine Formel, $\mathcal{A}$ eine zu φ passende Struktur und ϱ eine Variablenbelegung über $\mathcal{A}$. Für jeden Term t, den man aus den Bestandteilen von φ bilden kann, definieren wir nun induktiv den Wert von t in der Struktur $\mathcal{A}$, *den wir mit $\mathcal{A}(t)$ bezeichnen.*

1. *Für eine Variable x ist $\mathcal{A}(x) :=_{def} \varrho(x)$.*
2. *Falls t die Form hat $t = f^k(t_1, \ldots, t_k)$, wobei $t_1, \ldots, t_k$ Terme und f^k ein k-stelliges Funktionssymbol ist, so ist $\mathcal{A}(t) :=_{def} f^{\mathcal{A}}(\mathcal{A}(t_1), \ldots, \mathcal{A}(t_k))$.*

Nun definieren wir induktiv den Wahrheitswert einer Formel φ in $\mathcal{A}$:

1. *Falls $\varphi :=_{def} P^k(t_1, \ldots, t_k)$ mit den Termen $t_1, \ldots, t_k$ und k-stelligem Prädikatssymbol P^k, so ist*
$$\mathcal{A}(\varphi) :=_{def} \begin{cases} \mathbf{t}, & falls\, (\mathcal{A}(t_1), \ldots, \mathcal{A}(t_k)) \in P^{\mathcal{A}}, \\ \mathbf{f}, & sonst. \end{cases}$$
2. *Falls $\varphi :=_{def} \neg\psi$, so ist*
$$\mathcal{A}(\varphi) :=_{def} \begin{cases} \mathbf{t}, & falls\, \mathcal{A}(\psi) = \mathbf{f}, \\ \mathbf{f}, & sonst. \end{cases}$$
3. *Falls $\varphi :=_{def} (\psi \wedge \eta)$, so ist*
$$\mathcal{A}(\varphi) :=_{def} \begin{cases} \mathbf{t}, & falls\, \mathcal{A}(\psi) = \mathbf{t}\, und\, \mathcal{A}(\eta) = \mathbf{t}, \\ \mathbf{f}, & sonst. \end{cases}$$
4. *Falls $\varphi :=_{def} (\psi \vee \eta)$, so ist*
$$\mathcal{A}(\varphi) :=_{def} \begin{cases} \mathbf{t}, & falls\, \mathcal{A}(\psi) = \mathbf{t}\, oder\, \mathcal{A}(\eta) = \mathbf{t}, \\ \mathbf{f}, & sonst. \end{cases}$$
5. *Falls $\varphi :=_{def} \forall x\, \psi$, so ist*
$$\mathcal{A}(\varphi) = :=_{def} \begin{cases} \mathbf{t}, & falls\, für\, alle\, d \in U_{\mathcal{A}} : \mathcal{A}_{[x/d]}(\psi) = \mathbf{t}, \\ \mathbf{f}, & sonst. \end{cases}$$
6. *Falls $\varphi :=_{def} \exists x\, \psi$, so ist*
$$\mathcal{A}(\varphi) :=_{def} \begin{cases} \mathbf{t}, & falls\, es\, gibt\, ein\, d \in U_{\mathcal{A}} : \mathcal{A}_{[x/d]}(\psi) = \mathbf{t}, \\ \mathbf{f}, & sonst. \end{cases}$$

In den Fällen 5. und 6. wurde die Notation $[x/d]$ benutzt. Sie ist wie folgt definiert: Für $d \in U_{\mathcal{A}}$ sei $\mathcal{A}_{[x/d]}$ die Struktur $\mathcal{A}'$, die überall mit $\mathcal{A}$ identisch ist, bis auf die Definition von $x^{\mathcal{A}'}$: $x^{\mathcal{A}'} :=_{def} d$ (unabhängig davon, ob $I_{\mathcal{A}}$ für x definiert ist oder nicht). Wir schreiben:

- *$\mathcal{A} \models \varphi[\varrho]$ für $\mathcal{A}(\varphi) = \mathbf{t}$: $\mathcal{A}$ ist Modell für φ bezüglich ϱ.*
- *Enthält φ keine freien Variablen, so ist $\mathcal{A} \models \varphi[\varrho]$ unabhängig von ϱ. Wir lassen dann ϱ einfach weg.*
- *Falls es mindestens ein Modell für φ gibt, so heißt φ erfüllbar oder konsistent.*

Definition 5.5.6 (Tautologie)

1. *Eine Theorie ist eine Menge von Formeln ohne freie Variablen: $T \subseteq Fml_{\mathcal{L}}$. $\mathcal{A}$ erfüllt T falls $\mathcal{A} \models \varphi$ für alle $\varphi \in T$. Wir schreiben $\mathcal{A} \models T$.*
2. *Eine $\mathcal{L}$-Formel φ heißt $\mathcal{L}$-Tautologie, falls für alle möglichen $\mathcal{L}$-Strukturen $\mathcal{A}$ gilt: $\mathcal{A} \models \varphi$.*

Im folgenden lassen wir die zugrundeliegende Sprache $\mathcal{L}$ meistens weg, da sie aus dem Zusammenhang klar ersichtlich ist. Trotzdem muß sie natürlich immer festgelegt sein.

Definition 5.5.7 (Folgerungsmenge $Cn(T)$)
Eine Formel φ folgt aus T, falls für alle Strukturen $\mathcal{A}$ mit $\mathcal{A} \models T$ auch $\mathcal{A} \models \varphi$ gilt. Dafür schreiben wir: $T \models \varphi$.

Wir bezeichnen mit $Cn_{\mathcal{L}}(T) :=_{def} \{\varphi \in Fml_{\mathcal{L}} : T \models \varphi\}$, oder einfach $Cn(T)$, den semantischen Konsequenzoperator.

Lemma 5.5.8 (Eigenschaften von $Cn(T)$)
Der semantische Konsequenzoperator hat folgende Eigenschaften:

1. T-Erweiterung: $T \subseteq Cn(T)$,
2. Monotonie: $T \subseteq T' \Rightarrow Cn(T) \subseteq Cn(T')$,
3. Abgeschlossenheit: $Cn(Cn(T)) = Cn(T)$.

Lemma 5.5.9 ($\varphi \notin$ Cn(T))
$\varphi \notin Cn(T)$ gdw *es gibt eine Struktur $\mathcal{A}$ mit $\mathcal{A} \models T$ und $\mathcal{A} \models \neg\varphi$.*

Definition 5.5.10 (MOD(T), $Cn(\mathcal{U})$)
Ist $T \subseteq Fml_{\mathcal{L}}$, so bezeichnen wir mit MOD(T) die Menge aller $\mathcal{L}$-Strukturen $\mathcal{A}$ die Modelle von T sind:

$$MOD(T) :=_{def} \{\mathcal{A} : \mathcal{A} \models T\}.$$

Ist $\mathcal{U}$ eine Menge von Strukturen, so können wir alle Sätze betrachten, die in allen Strukturen gelten. Diese Menge bezeichnen wir ebenfalls mit $Cn(\mathcal{U})$:

$$Cn(\mathcal{U}) :=_{def} \{\varphi \in Fml_{\mathcal{L}} : \forall \mathcal{A} \in \mathcal{U} : \mathcal{A} \models \varphi\}.$$

Offenbar ist MOD der zu Cn duale Begriff:

$$Cn(\text{MOD}(T)) = Cn(T), \;\; \text{MOD}(Cn(T)) = \text{MOD}(T).$$

Definition 5.5.11 (Vollständigkeit einer Theorie T)
T heißt vollständig, falls für jede Formel $\varphi \in Fml_{\mathcal{L}}$: $T \models \varphi$ oder $T \models \neg\varphi$.

Diese letzte Bedingung ist nicht zu verwechseln mit der entsprechenden Eigenschaft in einer Struktur: Jede Struktur hat natürlich diese Eigenschaft.

Lemma 5.5.12 (Ex Falso Quodlibet)
T ist konsistent $\underline{\text{gdw}}$ $Cn(T) \neq Fml_{\mathcal{L}}$.

Welche Axiome erfüllen die folgenden Strukturen:

Beispiel 5.5.13 (Die natürlichen Zahlen in versch. Sprachen)

- $\mathcal{N}_{Pr} = (\mathbb{N}, 0^{\mathcal{N}}, +^{\mathcal{N}}, =^{\mathcal{N}})$ *(Presburger Arithmetik),*
- $\mathcal{N}_{PA} = (\mathbb{N}, 0^{\mathcal{N}}, +^{\mathcal{N}}, \cdot^{\mathcal{N}}, =^{\mathcal{N}})$ *(Peano Arithmetik),*
- $\mathcal{N}_{PA'} = (\mathbb{N}, 0^{\mathcal{N}}, 1^{\mathcal{N}}, +^{\mathcal{N}}, \cdot^{\mathcal{N}}, =^{\mathcal{N}})$ *(Variante der Peano Arithmetik).*

Es geht also jedes Mal um die natürlichen Zahlen, aber jeweils in verschiedenen Sprachen $\mathcal{L}_{Pr}$, $\mathcal{L}_{PA}$, $\mathcal{L}_{PA'}$.

Ist die Sprache grösser, kann man natürlich auch mehr ausdrücken. Ist aber $\mathcal{L}_{PA'}$ echt ausdrucksstärker als $\mathcal{L}_{PA}$?

Nein, denn man kann die $1^{\mathcal{N}}$ durch eine $\mathcal{L}_{PA}$-Formel definieren: es gibt eine $\mathcal{L}_{PA}$-Formel $\phi(x)$ sodass für jede Variablenbelegung ρ gilt

$$\mathcal{N}_{PA'} \models_{\rho} \phi(x) \;\; \text{gdw} \;\; \rho(x) = 1^{\mathbb{N}}$$

Man wähle etwa für $\phi(x)$ die Formel $\forall z(z \cdot x = z)$.

Ist vielleicht $\mathcal{L}_{PA}$ ausdrucksstärker als $\mathcal{L}_{Pr}$, oder kann man die Multiplikation vielleicht auch irgendwie definieren? Später werden wir sehen (in Kapitel 6) dass $\mathcal{L}_{PA}$ tatsächlich echt ausdrucksstärker ist: denn die Menge der in $\mathcal{N}_{Pr}$ gültigen Sätze ist entscheidbar, während die Menge der in $\mathcal{N}_{PA}$ gültigen Sätze noch nicht einmal rekursiv aufzählbar ist.

Wir haben schon einen zur Aussagenlogik $\mathcal{L}_{PL\,1}$ korrekten und vollständigen Kalkül angegeben. Solche Kalküle gibt es auch für die gerade definierte Prädikatenlogik $\mathcal{L}_{PL\,1}$.

Theorem 5.5.14 (Vollständigkeit der Prädikatenlogik)
Eine Formel folgt semantisch aus einer Theorie T genau dann, wenn sie abgeleitet werden kann:

$$T \vdash \varphi \;\; \underline{\text{gdw}} \;\; T \models \varphi$$

Theorem 5.5.15 (Kompaktheit der Prädikatenlogik)
Eine Formel folgt aus einer Theorie T genau dann, wenn sie schon aus einer endlichen Teilmenge von T folgt:

$$Cn(T) = \bigcup\{Cn(T') : T' \subseteq T, T' \textit{ endlich}\}.$$

5.6 Herbrand's Theorem

Die bisher eingeführte Relation $T \models \phi$ sagt aus, daß jedes Modell von T auch eines von ϕ ist. Da es aber sehr viele Modelle mit sehr grossen Universen gibt, stellt sich die Frage, ob man das nicht einschränken kann.

Theorem 5.6.1 (Löwenheim-Skolem)
$T \models \phi$ gilt genau dann, wenn ϕ in allen abzählbaren Modellen von T gilt.

Oftmals bestehen die Universen der Modelle die uns eigentlich interessieren genau aus den Grundtermen $Term_{\mathcal{L}}(\emptyset)$. Dies führt uns zu dem folgenden Begriff:

Definition 5.6.2 (Herbrand-Modell)
Ein Modell $\mathcal{A}$ heißt Herbrand-Modell bezüglich einer Sprache $\mathcal{L}$, falls das Universum von $\mathcal{A}$ genau aus $Term_{\mathcal{L}}(\emptyset)$ besteht und die Funktionssymbole f_i^k wie folgt interpretiert werden:

$$f_i^{k\,\mathcal{A}} : Term_{\mathcal{L}}(\emptyset) \times \ldots \times Term_{\mathcal{L}}(\emptyset) \rightarrow Term_{\mathcal{L}}(\emptyset);\ (t_1, \ldots, t_k) \mapsto f_i^k(t_1, \ldots, t_k)$$

Wir schreiben $T \models_{Herb} \phi$ falls jedes Herbrand Modell von T auch Modell von ϕ ist.

Theorem 5.6.3
Ist T universell und ϕ existentiell, so gilt

$$T \models_{Herb} \quad \textit{genau dann wenn} \quad T \models_{Herb} \phi$$

Ist $T \models_{Herb} \phi$ nicht viel einfacher, da man nur Herbrand-Modelle beachten muß? Ist es vielleicht sogar entscheidbar? Mehr darüber in Kapitel 6.

Der folgende Satz von Herbrand ist die Grundlage der Resolution. Er führt, sozusagen, PL1 auf AL zurück.

Theorem 5.6.4 (Herbrand)
Sei T universell und ϕ enthalte keine Quantoren. Dann gilt:

$$T \models \exists\phi \quad \textit{gdw} \quad \textit{es gibt } t_1, \ldots, t_n \in Term_{\mathcal{L}}(\emptyset) \textit{ mit: } T \models \phi(t_1) \vee \ldots \vee \phi(t_n)$$

Oder: Sei M eine Menge von Klauseln der PL1 (also Formeln der Form $P_1(t_1) \vee \neg P_2(t_2) \vee \ldots \vee P_n(t_n)$ wobei $t_i \in Term_{\mathcal{L}}(X)$). Dann ist M unerfüllbar genau dann, wenn es eine endliche unerfüllbare Menge M_{inst} von Grundinstanzen von M gibt.

Im automatischen Beweisen interessiert uns immer die Frage

$$M \models \exists x_1, \ldots x_n \bigwedge_i \phi_i$$

Dann ist $M \cup \{\neg\exists x_1, \ldots x_n \bigwedge_i \phi_i\}$ eine Menge von Klauseln, die genau dann unerfüllbar ist, wenn die obige Relation gilt.

5.7 Robinson's Resolution

Theorem 5.7.1 (Unifikation)
Gegeben sei eine endliche Menge von Gleichungen zwischen Termen oder Gleichungen zwischen Literalen. Dann gibt es einen Algorithmus der entweder eine allgemeinste Lösungssubstitution berechnet (d.h. eine Substitution für die beteiligten Variablen derart, daß linke und rechte Seiten aller Gleichungen syntaktisch identisch werden) oder fail *zurückgibt. Im letzten Fall gibt es gar keine Lösungssubstitution. Im ersten Fall ist die allgemeinste Lösungssubstitution bis auf Variablenumbenennungen eindeutig bestimmt: sie heißt*

mgU, allgemeinster Unifikator

- $p(x, a) = q(y, b)$,
- $p(g(a), f(x)) = p(g(y), z)$.
  Grundsubstitutionen sind: $[y/a, x/a, z/f(a)]$, $[y/a, x/f(a), z/f(f(a))], \ldots$
  Der mgU lautet: $[y/a, z/f(x)]$.

Wir wollen den auf der vorigen Folie erwähnten Algorithmus nun anhand eines Beispiels kurz skizzieren.

Gegeben: $f(x, g(h(y), y)) = f(x, g(z, a))$

Der Algorithmus berechnet nacheinander folgende Mengen von Gleichungen:

$$\begin{array}{l} \{\ x = x,\ g(h(y), y) = g(z, a)\ \} \\ \{\ g(h(y), y) = g(z, a)\ \} \\ \{\ h(y) = z,\ y = a\ \} \\ \{\ z = h(y),\ y = a\ \} \\ \{\ z = h(a),\ y = a\ \} \end{array}$$

Damit ergibt sich der mgU als: $[x/x, y/a, z/h(a)]$.

Definition 5.7.2 (Robinson's Resolution für PL1)
Die Resolution besteht aus zwei Regeln

$$\textit{Resolution: } \frac{C_1 \cup \{A_1\} \quad C_2 \cup \{\neg A_2\}}{(C_1 \cup C_2) mgU(A_1, A_2)}$$

wobei $C_1 \cup \{A_1\}$ *und* $C_2 \cup \{A_2\}$ *als variablendisjunkt vorausgesetzt werden.*

$$\textit{Faktorisierung: } \frac{C_1 \cup \{L_1, L_2\}}{(C_1 \cup \{L_1\}) mgU(L_1, L_2)}$$

Als Beispiel betrachte man $M = \{r(x) \vee \neg p(x), p(a), s(a)\}$ und die Frage $M \models \exists x(s(x) \wedge r(x))$?

Warum braucht man Faktorisierung? Man betrachte

$$M = \{s(x_1) \vee s(x_2), \neg s(y_1) \vee \neg s(y_2)\}$$

Resolviert man beide Klauseln miteinander, erhält man

$$\{s(x_1)\} \cup \{\neg s(y_1)\}$$

oder Varianten davon. Resolviert man diese neue Klausel mit eine der beiden in M, erhält man nur Varianten der entsprechenden Klausel in M. *Also ist $\Box$ niemals ableitbar (mit Resolution allein).* Faktorisierung löst das Problem sofort, denn man kann sowohl $s(x)$ als auch $\neg s(y)$ ableiten, und daraus die leere Klausel.

5.8 Der Situationskalkül

Wie axiomatisieren wir die Wumpus-Welt in PL1?

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

   TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
   action ← ASK(KB, MAKE-ACTION-QUERY(t))
   TELL(KB, MAKE-ACTION-SENTENCE(action, t))
   t ← t + 1
   return action
```

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

Wir benutzen ein Funktionssymbol

$$result(aktion, situation)$$

als Term für die Situation die ensteht, wenn in $situation$ die Aktion $aktion$ ausgeführt wird. Abbildung 5.1 veranschaulicht die Bedeutung der Terme $result(\ldots)$.

Aktionen: $Turn\_right$, $Turn\_left$, $Foreward$, $Shoot$, $Grab$, $Release$, $Climb$. Wir brauchen auch ein Gedächtnis, in Form eines 3-stelligen Prädikates

$$At(person, location, situation)$$

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

Wichtige Axiome sind die sogenannten "Successor-state Axioms", die beschreiben, welche Auswirkungen Aktionen auf Situationen haben. Die allgemeine Form dieser

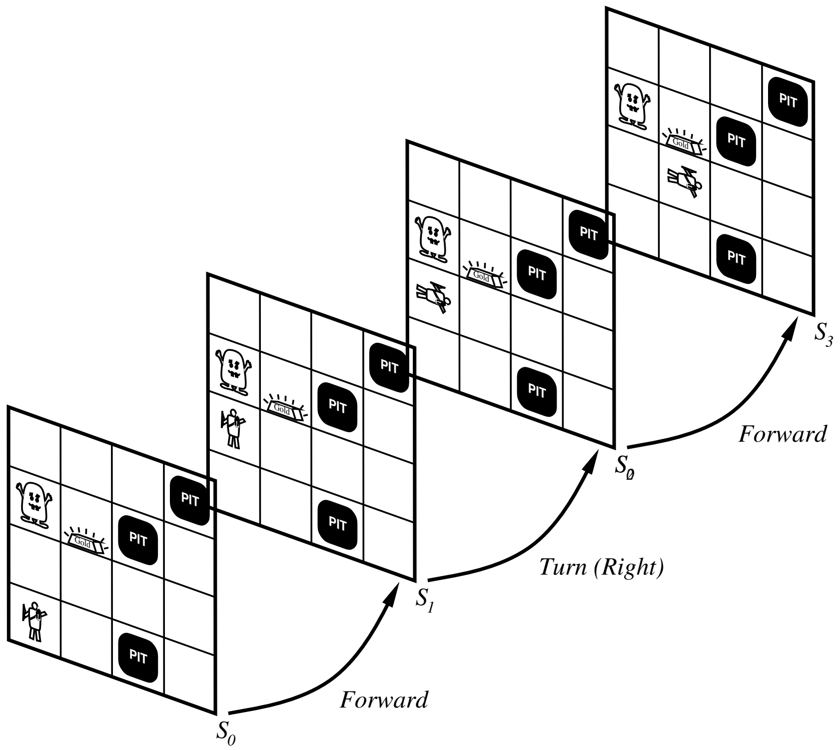


Abbildung 5.1: Der Situationskalkül.

Axiome ist

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

Axiome über $At(p, l, s)$**:**

$$\begin{aligned} At(p, l, result(a, s)) \leftrightarrow & ((l = location\_ahead(p, s) \wedge \neg Wall(l) \wedge a = Forward) \\ & \vee (At(p, l, s) \wedge \neg a = Forward)) \end{aligned}$$

$$\begin{aligned} At(p, l, s) \quad & \rightarrow Location\_ahead(p, s) = Location\_toward(l, Orient.(p, s)) \\ Wall([x, y]) \quad & \leftrightarrow (x = 0 \vee x = 5 \vee y = 0 \vee y = 5) \end{aligned}$$

$$\begin{aligned} & Location\_toward([x, y], 0) = [x + 1, y] \\ & Location\_toward([x, y], 90) = [x, y + 1] \\ & Location\_toward([x, y], 180) = [x - 1, y] \\ & Location\_toward([x, y], 270) = [x, y - 1] \end{aligned}$$

$$
\begin{aligned}
&Orient.(Agent, s_0) = 90\\
&Orient.(p, result(a,s)) = d \leftrightarrow ((a = turn\_right \wedge d = mod(Orient.(p,s) - 90, 360))\\
&\qquad \vee (a = turn\_left \wedge d = mod(Orient.(p,s) + 90, 360))\\
&\qquad \vee (Orient.(p,s) = d \wedge \neg(a = Turn\_right \vee a = Turn\_left))
\end{aligned}
$$

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

Axiome über Wahrnehmungen, sinnvolle neue Begriffe durch Definitionen:

$$
\begin{array}{lcl}
Percept([Stench, b, g, u, c], s) & \rightarrow & Stench(s)\\
Percept([a, Breeze, g, u, c], s) & \rightarrow & Breeze(s)\\
Percept([a, b, Glitter, u, c], s) & \rightarrow & At\_Gold(s)\\
Percept([a, b, g, Bump, c], s) & \rightarrow & At\_Wall(s)\\
Percept([a, b, g, u, Scream], s) & \rightarrow & Wumpus\_dead(s)\\
\\
At(Agent, l, s) \wedge Breeze(s) & \rightarrow & Breezy(l)\\
At(Agent, l, s) \wedge Stench(s) & \rightarrow & Smelly(l)
\end{array}
$$

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

$$
\begin{array}{lcl}
Percept([none, none, g, u, c], s) \wedge At(Agent, x, s) \wedge Adjacent(x, y) & \rightarrow & OK(y)\\
(\neg At(Wumpus, x, t) \wedge \neg Pit(x)) & \rightarrow & OK(y)\\
\\
At(Wumpus, l_1, s) \wedge Adjacent(l_1, l_2) & \rightarrow & Smelly(l_2)\\
At(Pit, l_1, s) \wedge Adjacent(l_1, l_2) & \rightarrow & Breezy(l_2)
\end{array}
$$

Axiome zur Beschreibung von Aktionen:

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

Alle Effekte müssen sorgsam beschrieben werden.

Axiome die Präferenzen zwischen Aktionen beschreiben:

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

$$\begin{array}{ll}
At(Agent,[1,1],s) \wedge Holding(Gold,s) & \to Great(Climb,s) \\
At_Gold(s) \wedge \neg Holding(Gold,s) & \to Great(Grab,s) \\
At(Agent,l,s) \wedge \neg Visited(Location_ahead(Agent,s)) \wedge & \\
\wedge OK(Location_ahead(Agent,s)) & \to Good(Forward,s)
\end{array}$$

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

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

5.9 Wichtige Probleme der Wissensrepräsentation

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

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

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

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

$$\begin{array}{lll}
\forall x & (Vogel(x) & \wedge\neg Pinguin(x) \wedge \neg Tot(x) \wedge \\
 & & \wedge\neg Strauss(x) \wedge \neg Flügel_gebr.(x) \wedge \\
 & & \wedge \ldots \\
 & \longrightarrow Fliegt(x) &
\end{array}$$

Ideal wäre es, nur "Normalerweise fliegen Vögel" aufzuschreiben.

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

Programmieren versus Knowledge Engineering

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

5.10 Aufgaben

Aufgabe 5.1 (Aussagenlogik)
Welche der folgenden aussagenlogischen Formeln F sind gültig, erfüllbar bzw. unerfüllbar? Nennen Sie alle Modelle zu F! Bestimmen Sie $Cn(\{F\})$!
(a) $\phi \wedge \psi$ *(b)* $\phi \vee \psi$ *(c)* $(\phi \wedge (\phi \vee \psi)) \rightarrow \phi$ *(d)* $(\phi \rightarrow \psi) \rightarrow (\neg\phi \rightarrow \neg\psi)$

Aufgabe 5.2 (Konnektive)
Welche logischen Konnektive außer $\wedge, \vee, \neg, \rightarrow$ halten Sie für angebracht? Wie viele binäre Konnektive kann es überhaupt geben? Warum sind nicht alle sinnvoll?

Aufgabe 5.3 (n-SAT)
Das n-SAT-Problem besteht darin herauszufinden, ob eine gegebene endliche Menge von Klauseln mit höchstens jeweils n Literalen erfüllbar ist. 3-SAT ist bekanntlich NP-vollständig. Was können Sie über die Komplexität von (a) 4– bzw. (b) 2-SAT sagen?

Aufgabe 5.4 (GSAT)
Betrachten Sie die folgende Prozedur GSAT – ein Greedy-Algorithmus zum Lösen des aussagenlogischen Erfüllbarkeitsproblems mit sogenanntem Random-Walk:

```
PROCEDURE GSAT
INPUT: S : Klauselmenge; m,n : Integers; p : Wahrscheinlichkeit
OUTPUT: T : Modell von S
FOR i := 1 TO m
  T := "Zufallsbelegung der Variablen in S";
  FOR j := 1 TO n
    IF "T erfüllt S" THEN RETURN(T);
    IF "Zufallszahl aus [0,1[" < p
      THEN x := "Variable aus unerfüllter Klausel in T"
      ELSE x := "Variable aus T, deren Belegungsänderung die Anzahl erfüllter Klauseln maxi-
    miert"
    T[x] := 1-T[x]   – "Wahrheitswert in T ändern"
```

1. *Ist der Algorithmus korrekt und vollständig, d.h. ist jede ausgegebene Belegung ein Modell bzw. wird in jedem Fall ein Modell gefunden, falls existent?*

2. *Implementieren Sie den Algorithmus in einer Programmiersprache Ihrer Wahl. Geben Sie neben dem Code bitte auch einige Testläufe mit ab!*

Aufgabe 5.5 (Prädikatenlogik)
Was ist falsch an folgenden Argumentationen?

1. *Falls eine Formel F nicht gültig ist, läßt sich folgendermaßen ein Beispiel zu $\neg F$ finden: Zähle solange alle möglichen Herbrand-Strukturen $\mathcal{A}$ passend zu F auf, bis ein Modell von $\neg F$ gefunden wird. Also ist das Erfüllbarkeitsproblem semi-entscheidbar.*

2. *Zur Herleitung der leeren Klausel wird Faktorisierung nicht benötigt, z.B. für die Menge $M = \{K_1, K_2\}$, wobei $K_1 = s(x_1) \vee s(x_2)$ und $K_2 = \neg s(y_1) \vee \neg s(y_2)$. Bilde dazu zunächst die Resolvente $R = s(x_2) \vee \neg s(y_2)$ aus K_1 und K_2. Resolviere dann R und K_1 zu $R' = s(x_2)$. Zweimalige Resolution von R' mit K_2 ergibt schließlich die leere Klausel.*

Aufgabe 5.6 (Resolution)
Widerlegen Sie $\forall x((p(x) \vee p(f(x))) \wedge (\neg p(x) \vee \neg p(f(f(x)))))$ *mittels prädikatenlogischer Resolution. Geben Sie jeweils eine Herleitung an, die (i) sowohl linear als auch eingabebeschränkt (d.h. Input-Ableitung), (ii) linear, aber nicht eingabebeschränkt, (iii) nicht linear, aber eingabebeschränkt bzw. (iv) weder linear noch eingabebeschränkt ist, oder überlegen Sie, warum dies nicht möglich ist!*

Aufgabe 5.7 (Situationskalkül)
Formalisieren Sie die Block-Welt im Situationskalkül. Objekte in dieser Domäne seien Blöcke, Tische und Situationen. Als Prädikate seien

on(x,y,s)	*"x ist auf y in Situation s",*
clear(x,s)	*"Block x ist frei in Situation s, d.h. liegt zuoberst" und*
block(x) *bzw.* table(x)	*"x ist ein Block bzw. Tisch".*

gegeben. Die einzig mögliche Aktion sei puton(x,y), *die als Vorbedingung* clear(x) *und* clear(y) *erfordert. – Im folgenden seien* a, b *und* c *Blöcke und* t *ein Tisch. Die Schreibweise* xyzt *bezeichne den Sachverhalt, daß* x *auf* y *auf* z *auf* t *liegt.*

1. *Geben Sie Axiome zur Beschreibung von* puton(x,y) *an! – Beschreiben Sie den Anfangszustand* s_0=abct. *– Welche Anfrage ist an einen Theorembeweiser zu stellen, damit dieser einen Plan findet, um* cbat *zu erzeugen?*

2. *Versuchen Sie, das Ganze in Prolog zu realisieren!*

6 Wissensrepräsentation II

6.1 Gleichheit

Es gibt eine einzige Relation, die in jeder Struktur eine kanonische Interpretation hat: die *Gleichheit* $\doteq$ (ein zweistelliges Prädikatssymbol), die man als *Identität* $=$ interpretieren kann. Zwei Elemente eines Individuenbereiches stehen genau dann in der Relation $\doteq$ wenn sie identisch sind.

Es gibt nun zwei mögliche Formulierungen für die Prädikatenlogik erster Stufe:

- **"PL 1 ohne $\doteq$":** dies ist die in Kapitel 5 dargestellte Version,
- **"PL 1 mit $\doteq$":** Dies bedeutet, daß es in der Sprache $\mathcal{L}$ ein zweistelliges Prädikatssymbol $\doteq$ gibt, daß in jeder Struktur als Identität interpretiert wird.

Ist "PL 1 mit $\doteq$" echt ausdrucksstärker als "PL 1 ohne $\doteq$"? Wir können Formeln Ψ_n angeben, so daß in allen Strukturen $\mathcal{A}$ mit Universum A gilt:

$$\mathcal{A} \models \Psi_n \ \text{ gdw } \ A \text{ enthält genau } n \text{ Elemente,}$$

nämlich Ψ_n: $\exists x_1 \dots x_n \, (\bigwedge_{i \neq j} \neg x_i \doteq x_j \ \wedge \forall x \bigvee_{i=1}^{n} x \doteq x_i)$
Kann man $\doteq$ vielleicht in "PL 1 ohne $\doteq$" simulieren? Man kann ja versuchen, $\doteq$ zu axiomatisieren (wie die Wumpus-Welt). Wir betrachten PL 1 ohne $\doteq$ mit einem zweistelligen Symbol eq und wir fordern folgende Axiome:

- $\forall x \; eq(x, x)$,
- für alle Funktionssymbole f geeigneter Stelligkeit:

  $$\forall x_1 \dots x_n, y_1 \dots y_n \, (eq(x_1, y_1) \dots eq(x_n, y_n)) \to eq(f(x_1, \dots, x_n), f(y_1, \dots, y_n)),$$

- für alle Prädikatssymbole P geeigneter Stelligkeit:

  $$\forall x_1 \dots x_n y_1 \dots y_n \, (eq(x_1, y_1) \dots eq(x_n, y_n)) \to (P(x_1, \dots, x_n) \to P(y_1, \dots, y_n)).$$

Diese Menge von Axiomen hängt von der zugrundeliegenden Sprache $\mathcal{L}$ ab, wir bezeichnen sie mit $\mathrm{EQ}_{\mathcal{L}}$.

Gilt folgende Äquivalenz, wobei ϕ eine Formel des PL 1 mit $\doteq$ ist[1]*:*

$$T \models_{mit \doteq} \phi \text{ gdw } T[\doteq /eq] \cup \mathrm{EQ}_{\mathcal{L}} \models_{ohne \doteq} \phi[\doteq /eq] \text{ ?}$$

Leider nein, denn

1. die Axiome $\mathrm{EQ}_{\mathcal{L}}$ lassen es immer zu, daß in einer Struktur $\mathcal{A}$ die Interpretation von eq aus ganz $A \times A$ besteht,
2. Modelle können auch Elemente enthalten, die nicht durch Terme der zugrundeliegenden Sprache bezeichnet werden – über diese Elemente können Axiome nur bedingt Aussagen machen.

Man kann zeigen, daß es auch mit weiteren Axiomen nicht gelingt, die Identität vollständig zu axiomatisieren.

6.2 (Un-)Entscheidbarkeit

Wir hatten früher schon die natürlichen Zahlen mit der Addition und Multiplikation betrachtet. Wir gehen nun etwas anders vor und betrachten die Sprache $\mathcal{L}_{PA}$ mit

- einer Konstanten 0 und einem einstelligen Funktionssymbol s,
- einem zweistelligen Relationssymbol eq und zwei dreistelligen Relationssymbolen $\oplus$ und $\otimes$.

Bevor wir einige Axiome über $\oplus$ und $\otimes$ angeben, wollen wir folgende Abkürzung definieren: die Formel $\exists! z\, \phi(z)$ steht für

$$\exists z\, (\, \phi(z) \;\wedge\; \forall y\, \phi(y) \rightarrow eq(y, z)).$$

Definition 6.2.1 (Peanoarithmetik PA$_{fin}$)
PA$_{fin}$ besteht aus EQ$_{\mathcal{L}_{PA}}$ und folgenden Axiomen:

$$\begin{array}{ll} \forall x \forall y \exists! z & \oplus(x, y, z) \\ \forall x & \oplus(x, 0, x) \\ \forall x \forall y \forall z & \oplus(x, y, z) \rightarrow \oplus(x, s(x), s(z)) \end{array}$$

$$\begin{array}{ll} \forall x \forall y \exists! z & \otimes(x, y, z) \\ \forall x & \otimes(x, 0, 0) \\ \forall x \forall y \forall z \forall z' & \otimes(x, y, z) \rightarrow (\otimes(x, s(y), z') \wedge \oplus(z, x, z')) \end{array}$$

Offenbar ist $\mathcal{N} := (\mathbb{N}, 0^{\mathcal{N}}, s^{\mathcal{N}}, \oplus^{\mathcal{N}}, \otimes^{\mathcal{N}}, eq^{\mathcal{N}})$ ein Modell all dieser Axiome. Dabei ist $0^{\mathcal{N}}$ die “richtige” 0, $s^{\mathcal{N}}$ ist die Nachfolgerfunktion, $\oplus^{\mathcal{N}}$ ist die Addition und $\otimes^{\mathcal{N}}$ die Multiplikation natürlicher Zahlen (jeweils gesehen als Relationen). $eq^{\mathcal{N}}$ ist die Identität. Es gilt:

[1] $\phi[\doteq /eq]$ entsteht aus ϕ indem jedes Vorkommen von $\doteq$ durch eq ersetzt wird.

- Die Menge $\{\phi : \ PA_{fin} \models \phi\}$ ist rekursiv aufzählbar aber nicht rekursiv (nicht entscheidbar).
- Die Menge $\{\phi : \ \mathcal{N} \models \phi\}$ ist noch nicht einmal rekursiv aufzählbar.

Es gilt sogar

Theorem 6.2.2 (Gödel)
Jede Formelmenge die PA_{fin} enthält und $\mathcal{N}$ als Modell hat, ist nicht rekursiv.

Jede rekursiv aufzählbare Formelmenge Φ die PA_{fin} enthält und $\mathcal{N}$ als Modell hat, ist unvollständig, d.h. es gibt ψ mit: $\mathcal{N} \models \psi$ aber $\Phi \not\models \psi$. $\mathcal{N}$ kann also nie vollständig axiomatisiert werden.

6.3 Varianten der Resolution

Unser allgemeines Ziel ist es aus einer Formelmenge eine existentiell quantifizierte Formel abzuleiten:

$$M \ \vdash \ \exists\varphi.$$

Um Resolution anzuwenden, müssen wir $M \cup \{\neg\exists\varphi\}$ bilden und dies in Klauselform bringen. Diese Menge heißt auch **Input**.

Anstatt beliebige Resolventen zu bilden und damit den Suchraum aufzublähen, versuchen wir dies etwas einzuschränken. Wir illustrieren dies an dem Beispiel $M := \{r(x) \vee \neg p(x),\ p(a),\ s(a)\}$ und der Query $\Box \leftarrow s(x) \wedge r(x)$. Eine unbeschränkte Resolution kann etwa so aussehen:

$$\dfrac{\dfrac{r(x) \vee \neg p(x) \quad p(a)}{r(a)} \qquad \dfrac{s(a) \quad \neg s(x) \vee \neg r(x)}{\neg r(a)}}{\Box}$$

Input-Resolution: In jedem Resolutions-Schritt muß eine der beiden Elter-Klauseln aus dem Input stammen. In unserem obigen Beispiel:

$$\dfrac{\dfrac{\dfrac{\neg s(x) \vee \neg r(x) \quad s(a)}{\neg r(a)} \qquad r(x) \vee \neg p(x)}{\neg p(a)} \qquad p(a)}{\Box}$$

Lineare Resolution: In jedem Resolutions-Schritt muß eine der beiden Elter-Klauseln entweder aus dem Input stammen, oder Vorgänger-Klausel der anderen Elter-Klausel sein.

Theorem 6.3.1
Lineare Resolution ist widerlegungsvollständig. Input-Resolution ist zwar korrekt, aber nicht widerlegungsvollständig.

6.4 SLD-Resolution

Definition 6.4.1 (Horn-Klauseln)
Eine Klausel heißt Horn-Klausel, *falls sie höchstens ein einziges positives Atom enthält.*

Eine Horn-Klausel hießt definit, *falls sie genau ein positives Atom enthält. Sie hat also die Form*

$$A(t) \leftarrow A_1(t_1), \ldots, A_n(t_n).$$

Eine Horn-Klausel ohne positives Atom heißt Query*:*

$$\Box \leftarrow A_1(t_1), \ldots, A_n(t_n).$$

Theorem 6.4.2 (Input-Resolution für Horn-Klauseln)
Input-Resolution ist widerlegungsvollständig für Horn-Klauseln.

Definition 6.4.3 (SLD-Resolution bez. P und Query Q)
SLD-Resolution bezüglich des Programms P und der Query Q ist Input-Resolution die mit der Query $\Box \leftarrow A_1, \ldots, A_n$ beginnt. Dann wird aus den A_i eines ausgewählt, und mit einer Klausel des Programms resolviert. Es entsteht eine neue Query, mit der genauso verfahren wird. Entsteht die leere Klausel $\Box \leftarrow$, so war die SLD-Resolution erfolgreich und die Instantiierungen der Variablen bezeichnet man als berechnete Antwort.

Theorem 6.4.4 (Korrektheit der SLD-Resolution)
Sei P ein definites Programm und Q eine Query. Dann ist jede berechnete Antwort für P bez. Q auch korrekt.

Welche Freiheitsgrade hat man bei der SLD-Resolution?

Definition 6.4.5 (Berechnungsregel)
Eine Berechnungsregel R ist eine Funktion, die jeder Query $\Box \leftarrow A_1, \ldots, A_n$ ein Atom $A_i \in \{A_1, \ldots, A_n\}$ zuordnet. Diese A_i ist das ausgewählte Atom, gegen das im nächsten Schritt resolviert wird.

PROLOG benutzt immer das linkeste Atom.

Beispiel 6.4.6 (Illustration der SLD-Resolution)

$$\begin{array}{lll} (1) & p(x,z) & \leftarrow \quad q(x,y), p(y,z) \\ (2) & p(x,x) & \\ (3) & q(a,b) & \end{array}$$

Ein SLD-Baum kann drei verschiedene Arten von Ästen haben:

1. unendliche,
2. Äste die mit der leeren Klausel enden (und zu einer berechneten Antwort führen), und

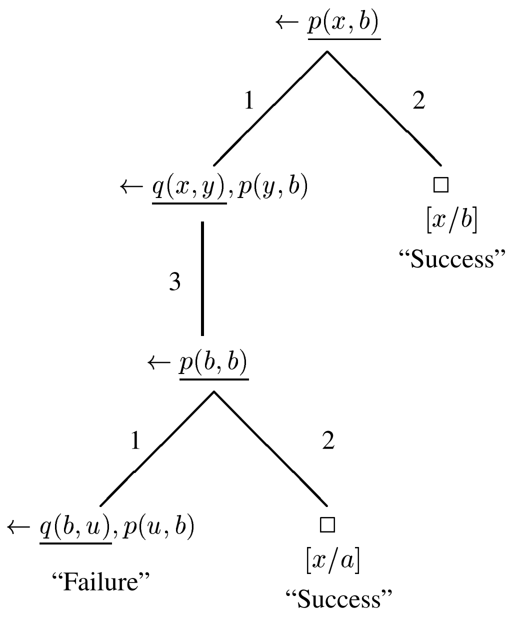


Abbildung 6.1: Ein endlicher SLD-Baum

3. Äste die scheitern (Sackgasse).

Theorem 6.4.7 (Unabhängigkeit von der Berechnungsregel)
Sei R eine Berechnungsregel und σ eine bezüglich R berechnete Antwort (d.h. es gibt eine erfolgreiche SLD-Resolution). Dann gibt es auch für jede andere Berechnungsregel R' eine erfolgreiche SLD-Resolution und die bezüglich R' berechnete Antwort σ' ist eine Variante von σ.

Theorem 6.4.8 (Vollständigkeit der SLD-Resolution)
Jede korrekte Antwort-Substitution wird durch eine berechnete Antwort-Substitution subsumiert. D .h.

$$P \models \forall Q\Theta$$
impliziert
SLD berechnet eine Antwort τ mit: $\exists \sigma : Q\tau\sigma = Q\Theta$

Wie findet man erfolgreiche Äste in einem SLD-Baum?

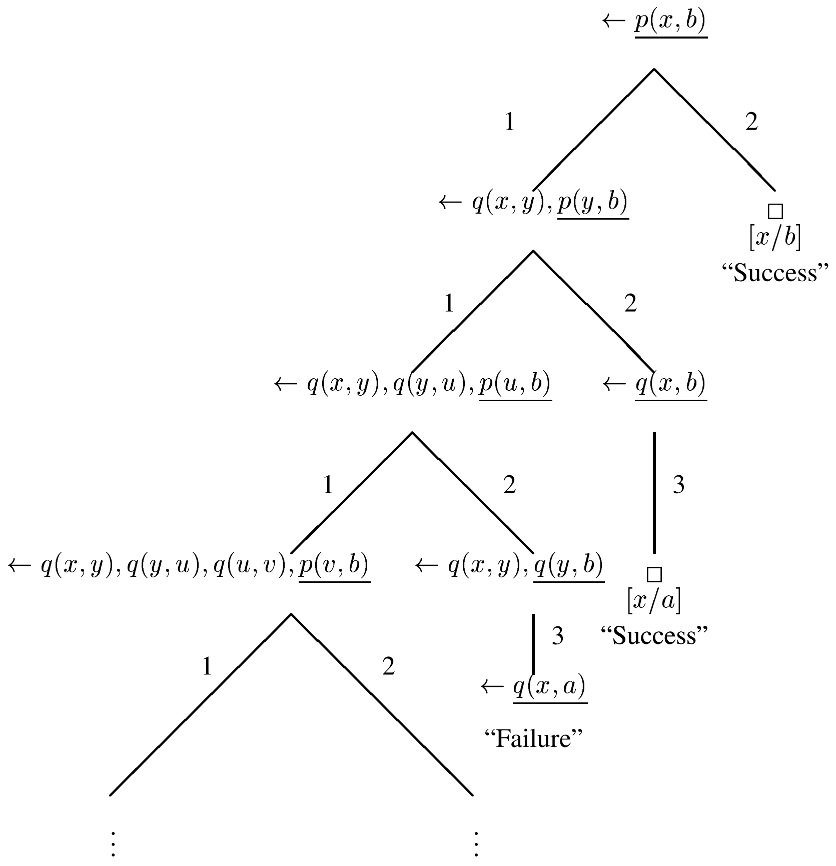


Abbildung 6.2: Ein unendlicher SLD-Baum

Definition 6.4.9 (Such-Regel)
Eine Such-Regel ist eine Strategie, SLD-Bäume nach erfolgreichen Ästen abzusuchen.

PROLOG benutzt Tiefen-Suche (depth-first). Eine SLD-Resolution ist durch eine Berechnungs- und eine Suchregel eindeutig bestimmt.

SLD-Bäume zu $P \cup \{Q\}$ sind durch die Berechnungsregel eindeutig bestimmt. PROLOG ist aus zwei Gründen unvollständig:

- Tiefen-Suche
- Inkorrekte Unifikation (kein Occur-Check).

Fragt man nicht nur nach erfolgreichen Ästen, sondern auch nach im Endlichen scheiternden SLD-Resolutionen, kommt noch eine weitere Ursache hinzu

- die Berechnungsregel muß **fair** sein, d.h. es muß garantiert werden, daß jedes Atom auf der Liste der Goals auch einmal ausgewählt wird.

6.5 Logik höherer Stufe

Definition 6.5.1 (Logik 2. Stufe $\mathcal{L}_{PL\,2}$)
Die Sprache $\mathcal{L}_{PL\,2}$ der Prädikatenlogik zweiter Stufe besteht aus der Sprache $\mathcal{L}_{PL\,1}$ und zusätzlich

- *für jedes $k \in \mathbb{N}$: $X_1^k, X_2^k, \ldots, X_n^k, \ldots$ einer abzählbaren Menge $\mathcal{RELVAR}^k$ von k-stelligen Prädikatsvariablen.*

Dadurch vergrößert sich die Menge der Terme,

- *ist X^k eine k-stellige Prädikatsvariable und sind $t_1, \ldots, t_k$ Terme, so ist auch $X^k(t_1, \ldots, t_k)$ ein Term,*

und auch die Menge der Formeln:

- *ist X eine Prädikatsvariable, φ eine Formel, so sind auch $(\exists X \varphi)$ und $(\forall X \varphi)$ Formeln.*

Man kann nun also nicht nur über Elemente des Individuenbereichs quantifizieren, sondern über beliebige Teilmengen, bzw. über beliebige k-stellige Relationen. In der Semantik der Logik zweiter Stufe ändert sich fast nichts — es kommen lediglich zwei neue Dinge dazu (nämlich die Interpretation von Formeln der Art $(\exists X \varphi)$, $(\forall X \varphi)$).

Außerdem verlangen wir von $I_\mathcal{A}$ daß die neuen k-stelligen Prädikatsvariablen auf k-wertige Relationen auf $U_\mathcal{A}$ abgebildet werden.

- Falls $\varphi :=_{def} \forall X^k \psi$, so ist
$$\mathcal{A}(\varphi) :=_{def} \begin{cases} \mathbf{t}, & \text{falls für alle } R^k \subseteq U_\mathcal{A} \times \cdots \times U_\mathcal{A} :\ \mathcal{A}_{[X^k/R^k]}(\psi) = \mathbf{t}, \\ \mathbf{f}, & sonst. \end{cases}$$

- Falls $\varphi :=_{def} \exists X^k \psi$, so ist
$$\mathcal{A}(\varphi) :=_{def} \begin{cases} \mathbf{t}, & \text{falls es ein } R^k \subseteq U_\mathcal{A} \times \cdots \times U_\mathcal{A} \text{ gibt mit } \mathcal{A}_{[X^k/R^k]}(\psi) = \mathbf{t}, \\ \mathbf{f}, & sonst. \end{cases}$$

Man kann nun also über beliebige n-stellige Relationen des Individuenbereichs quantifizieren, nicht mehr nur über einzelne Elemente.

Was bedeuten etwa die folgenden beiden Sätze aus PL 2?

$$\forall x \forall y\, (x \doteq y \iff \forall X\, (X(x) \iff X(y))),$$

$$\forall X\ \textbf{(}\ (\forall x \exists! y X(x,y) \land \forall x \forall y \forall z\, (\,(X(x,z) \land X(y,z)) \to x \doteq y)) \to \forall x \exists y X(y,x)\ \textbf{)}$$

Der erste Satz zeigt, daß die Gleichheit in PL 2 definierbar ist (im Unterschied zu PL 1). Der zweite Satz gilt in einer Struktur genau dann, wenn sie endlich ist. Man beachte, daß dies in PL 1 **nicht** ausdrückbar ist.

Während die Semantik von $\mathcal{L}_{PL\,2}$ eine kanonische Erweiterung von $\mathcal{L}_{PL\,1}$ ist, gilt dies nicht auf der Ebene der Kalküle. Man kann nämlich zeigen, daß die Menge der gültigen $\mathcal{L}_{PL\,2}$ Sätze noch nicht einmal rekursiv aufzählbar ist. Es kann also gar keinen korrekten und vollständigen Kalkül für die Prädikatenlogik zweiter Stufe geben.

6.6 Aufgaben

Aufgabe 6.1 (Theorie der ganzen Zahlen)
Sei die Theorie der ganzen Zahlen $(\mathbb{Z}, +, \cdot, =)$ *gegeben. Definieren Sie Formeln* $\chi(x)$*, die genau die folgenden Mengen ganzer Zahlen beschreiben:*
(a) $\{x \mid x \neq 0\}$ *(b)* $\{x \mid x$ *ist eine Primzahl* $\}$ *(c)* $\{x \mid x \in \mathbb{N}\}$

Aufgabe 6.2 (Herbrand-Theorie und Gleichheit)

1. *Nennen Sie alle Herbrand-Modelle der Axiome zur Peano-Arithmetik* $\mathbf{PA_{fin}}$ *(siehe Definition 6.1)!*
2. *Folgt aus den Gleichheitsaxiomen* $\mathbf{EQ}_{\mathcal{L}}$ *die Symmetrie von* $\mathbf{eq}$*? – Warum?*
3. *Die Gleichheit kann in der Prädikatenlogik erster Stufe nicht vollständig axiomatisiert werden. – Können Sie das vielleicht für Herbrand-Modelle tun?*

7 Planen

7.1 Planen vs. Problem-Lösen

Problem-lösender Agent: Die Effekte einer fixen Sequenz von Aktionen werden bestimmt.

Wissensbasierter Agent: Aktionen können ausgewählt werden.

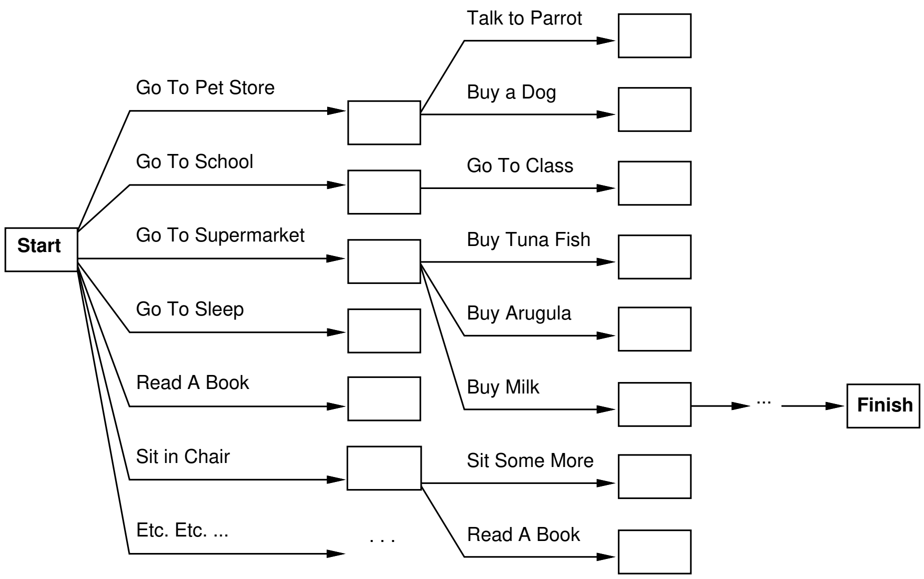
Beide zusammen versuchen wir zu einem **Planungsagenten** zu verschmelzen.

Beispiel 7.1.1
Wir wollen eine frische Bananenmilch trinken und dann ein paar Löcher zuhause in die Wand bohren. Dazu brauchen wir Bananen, Milch und eine Bohrmaschine.

Wie geht ein Problem-lösender Agent damit um?

```
function SIMPLE-PLANNING-AGENT(percept) returns an action
   static: KB, a knowledge base (includes action descriptions)
           p, a plan, initially NoPlan
           t, a counter, initially 0, indicating time
   local variables: G, a goal
                    current, a current state description

   TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
   current ← STATE-DESCRIPTION(KB, t)
   if p = NoPlan then
       G ← ASK(KB, MAKE-GOAL-QUERY(t))
       p ← IDEAL-PLANNER(current, G, KB)
   if p = NoPlan or p is empty then action ← NoOp
   else
       action ← FIRST(p)
       p ← REST(p)
   TELL(KB, MAKE-ACTION-SENTENCE(action, t))
   t ← t + 1
   return action
```

Tabelle 7.1: Einfacher Planungs-Agent



Planen unterscheidet sich vom Problem-Lösen in folgenden Punkten:

1. Benutze PL1 um Verbindungen zwischen Zuständen und Aktionen herzustellen. Wenn $have(Milk)$ das Ziel ist und $Buy(x) \rightarrow Have(x)$, so ist es sinnvoll, $Buy(Milk)$ zu betrachten.

   Dies vermeidet irrelevante Aktionen.

2. Beim Zusammenstellen eines Planes dürfen Aktionen beliebig addiert werden: es muß nicht in sturer Reihenfolge gehen. Achtung: Reihenfolge des Plans hat zunächst nichts mit der Reihenfolge der Ausführung zu tun.

   Dies setzt den Verzweigungsfaktor herab.

   Zustände sind Mengen von Sätzen, daher nicht notwendig vollständig.

3. Die meisten Teile der Welt sind unabhängig voneinander: $Have(Milk) \wedge Have(Bananas) \wedge Have(Drill)$.

   Dies ermöglicht Divide-et-Impera Strategien.

Planen im Situationskalkül

Anfangszustand: $At(Home, S_0)$, $\neg Have(Milk, S_0)$, $\neg Have(Bananas, S_0)$, $\neg Have(Drill, S_0)$.

Ziel: $\exists s\ (At(Home, s) \wedge Have(Milk, s) \wedge Have(Bananas, s) \wedge Have(Drill, s)$.

Axiome: z.B. über “Milch kaufen”:

$$\begin{aligned}&\forall a, s\; Have(Milk, result(a, s)) \leftrightarrow\\&(a = Buy(Milk) \wedge At(Supermarket, s)) \vee\\&(Have(Milk, s) \wedge a \neq Drop(Milk))\end{aligned}$$

Wir brauchen nun noch einen Term $Result'(l, s)$: die Situation die entsteht, wenn in s die Sequenz l von Aktionen ausgeführt wird.

$$\begin{array}{lll}\forall s & Result'([], s) & := \quad s\\ \forall a, p, s & Result'([a|p], s) & := \quad Result'(p, result(a, s))\end{array}$$

Die Aufgabe ist es nun, ein p zu finden, mit $At(Home, Result'(p, S_0)) \wedge Have(Milk, Result'(p, S_0)) \wedge Have(Bananas, Result'(p, S_0)) \wedge Have(Drill, Result'(p, S_0))$.
Es erbenen sich folgende Probleme:

1. $T \vdash \phi$ ist nur semi-entscheidbar.
2. Ist p ein Plan, so auch $[Leere\_Aktion|p]$ und $[A, A^{-1}|p]$.

Also sollte man möglichst keinen allgemeinen Beweiser, sondern einen auf den speziellen Fall zugeschnittenen benutzen. Außerdem sollte man die Sprache einschränken.

Man mache sich den Unterschied klar zwischen

- Goal an Planner und
- Query an Theorembeweiser.

Im ersten Fall sucht man einen Plan derart, daß *nach Ausführung* des Planes das Goal gilt. Im zweiten Fall fragt man ob die Query bezüglich der KB wahr gemacht werden kann: $KB \models \exists x \phi(x)$. Das dynamische steckt in den Termen. Die Logik selbst ist statisch.

7.2 STRIPS

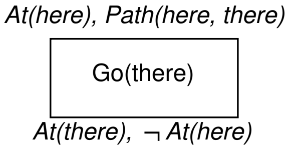
STRIPS steht für **ST**anford **R**esearch **I**nstitute **P**roblem **S**olver.

Zustände: Konjunktion von Funktions-freien Grund-Literalen

Ziel: Konjunktion von Funktions-freien Literalen

Aktionen: STRIPS-Operatoren bestehen aus 3 Komponenten

1. *Description*, Name der Aktion,
2. *Precondition*, Konjunktion von Atomen,
3. *Postcondition*, Konjunktion von Literalen.



Definition 7.2.1 (Anwendbarer Operator)
Ein Operator Op ist anwendbar in einem Zustand s, falls seine Variablen so instantiiert werden können, daß alle Preconditions in s wahr sind: $Precond(s) \subseteq s$.

Im resultierenden Zustand gelten alle positiven Literale aus $Effect(Op)$ und alle Literale die in s gelten, bis auf die negierten Literale aus $Effect(Op)$.

Bisher kann man alles auch als Problem-Lösen sehen. Wir benutzen STRIPS als Repräsentationsformalismus und suchen nach einem Lösungspfad:

- Knoten im Suchbaum $\approx$ Situationen,
- Lösungspfade $\approx$ Pläne.

Suche im Raum aller Pläne: Fange mit partiellem Plan an und erweitere sukzessive. Dazu brauchen wir Operatoren die auf Plänen operieren. Wit unterscheiden zwischen 2 Arten:

Refinement-Op: Einem Plan werden Constraints angehängt. Ein Plan steht dann für die Menge aller vollständigen Pläne (analog steht $Cn(T)$ für $MOD(T)$).

Modification-Op: Alle anderen.

Man mache sich eindringlich den Unterschied zum Problem-Lösen klar. *Suche im Raum aller Pläne*, also eine Abstraktionsebene höher.

Wie repräsentiert man Pläne? Es gilt zwei wichtige Punkte zu beachten:

- *Variablen-Instantiierungen:* Instantiiere nur, falls notwendig. Nimm also stets den mgU.
- *Partielle Ordnung:* Sieh' zunächst von der genauen Reihenfolge ab (dadurch wird der Suchraum kleiner).

Definition 7.2.2 (Plan)
Ein Plan ist ein Quadrupel bestehend aus

1. *Einer Multi-Menge von* Schritten. *Ein Schritt ist ein Operator des Problems.*
2. *Einer Menge von* Ordnungs-Bedingungen *(auch* Constraints *genannt) an die Schritte. Diese haben die Form $S_i \prec S_j$: S_i muß vor S_i ausgeführt werden.*
3. *Einer Menge von* Instantiierungs-Bedingungen. *Diese haben die Form $v = x$ wobei v eine Variable in einem Schritt und x eine Konstante oder Variable ist.*

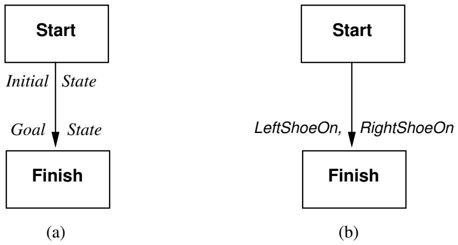


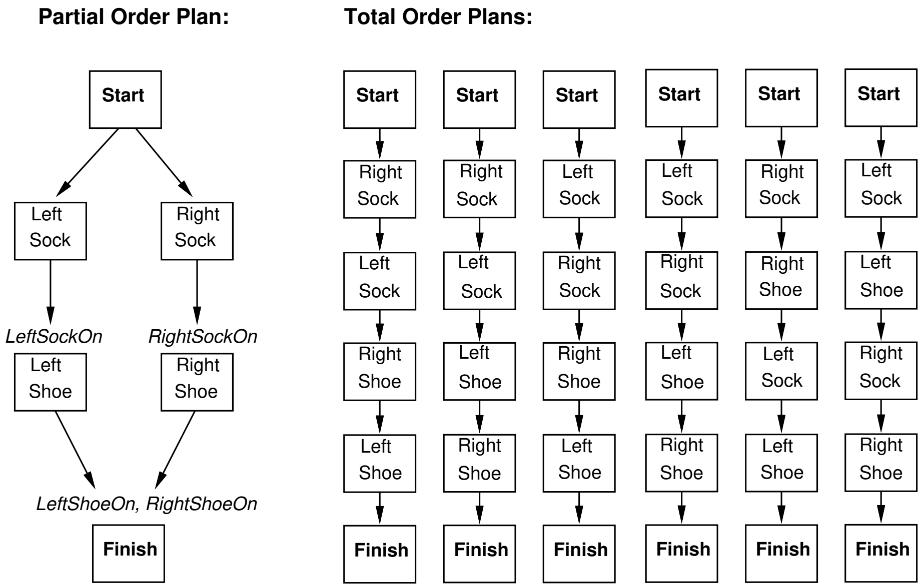
Abbildung 7.1: Der Anfangsplan

4. *Einer Menge von* kausalen Links. *Diese haben die Form* $S_i \xrightarrow{c} S_j$*:* S_i *stellt* c *sicher für* S_j*.*

Um den Anfangsplan eines Problems darzustellen definieren wir 2 Operatoren START und FINISH wie folgt
Als Anfangsplan eines Problems ergibt sich damit:

$$\textbf{Plan(} \begin{array}{ll} Steps: & \{S_1 : START, S_2 : FINISH\} \\ Orderings: & \{S_1 \prec S_2\} \\ Bindings: & \emptyset \\ Links: & \emptyset \end{array}$$
$$\textbf{)}$$

Wir wollen nun das Problem Schuhe und Socken anzuziehen betrachten.



Was ist eine Lösung? Das ist einfach, falls wir nur voll instantiierte, linear geordnete Pläne betrachten: dann können wir es nämlich einfach überprüfen. In unserem Fall ist es aber komplizierter:

Definition 7.2.3 (Lösung eines Planes)
Eine Lösung ist ein vollständiger *und* konsistenter *Plan. Dabei bedeutet vollständig, daß jede Precondition eines Schrittes von einem anderen Schritt sichergestellt wird und konsistent, daß es keine Widersprüche in den Instantiierungs oder Ordnungs-Bedingungen gibt.*

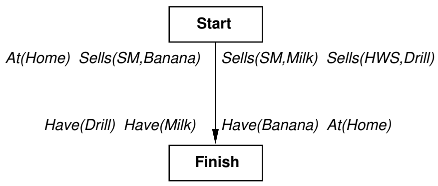
Noch präziser:

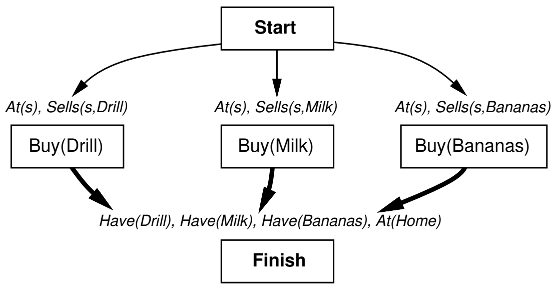
- "S_i stellt c sicher für S_j" bedeutet
  - $c \in Precondition(S_j)$,
  - $S_i \prec S_j$,
  - $\not\exists S_k : \neg c \in Effect(S_k)$ mit $S_i \prec S_k \prec S_j$ in irgendeiner Linearisierung des Plans.
- "Keine Widersprüche" bedeutet
  - Weder ($S_i \prec S_j$ und $S_j \prec S_i$) noch ($v = A$ und $v = B$ für verschiedene Konstanten A, B).

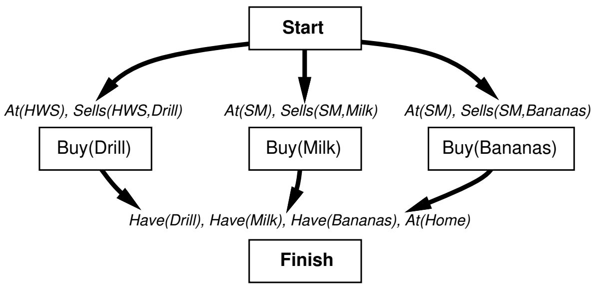
  Man beachte, daß dies nicht nur explizit gefordert wird. Die entsprechenden Aussagen können auch folgen, da $\prec$ und $=$ etwa transitiv sind.

7.3 Partial-Order Planning

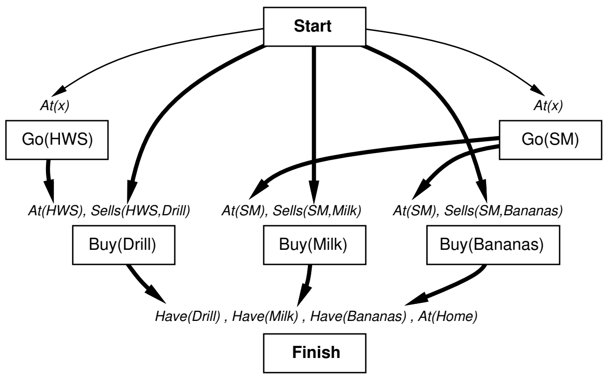
Wir betrachten unser Bananen-Milch-Beispiel. Als Operatoren verwenden wir Buy und Go.

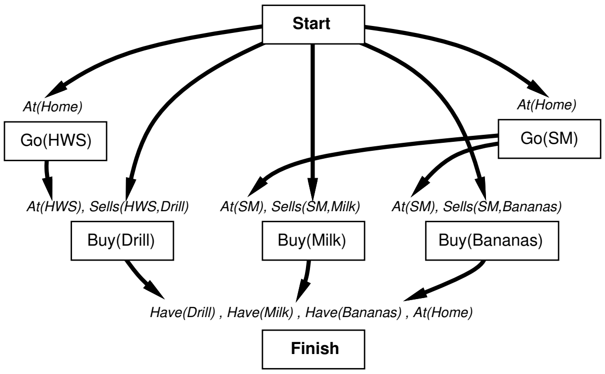






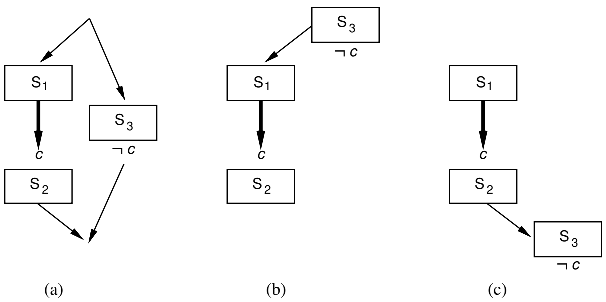
Kausale Links werden geschützt! Kausale Links sind fett eingezeichnet. Sie schließen den entsprechenden Ordnungs-Constraint ein!





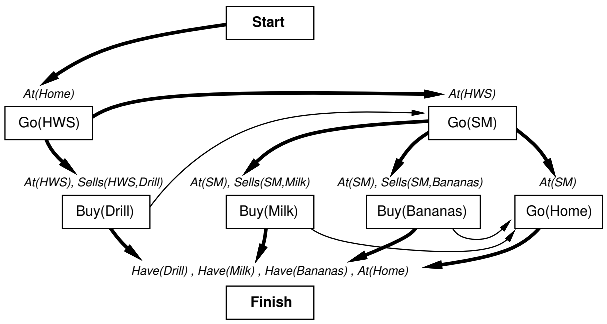
Der letzte partielle Plan kann nicht mehr erweitert werden. Wie stellt man das fest? Indem man feststellt, daß ein kausaler Link bedroht wird und die Drohung nicht aufgehoben werden kann!

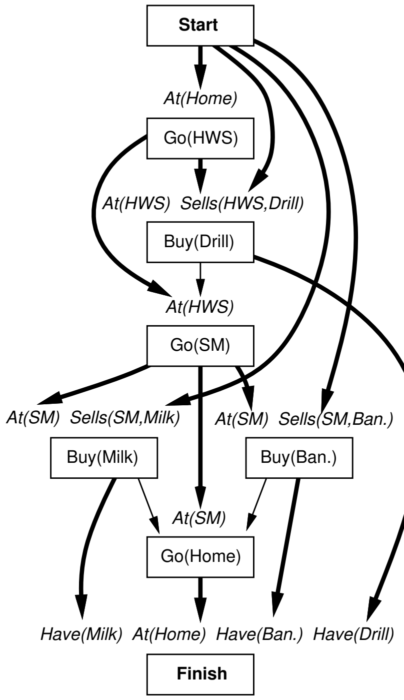
Definition 7.3.1 (Drohung)



Eine Drohung kann aufgehoben werden, indem man die Drohung vor- oder nachschaltet (das muss nicht immer gehen!).

Wir müssen einen Go-Schritt einführen, um die letzte Vorbedingung zu sichern. Aber wie können wir die Vorbedingung des Go-Schrittes sichern? Es gibt jetzt jede Menge Bedrohungen, von denen sich die meisten nicht auflösen lassen. Schließlich kommen wir zu





Das führt also zu folgendem Algorithmus:

- In jeder Runde wird der Plan erweitert um die Vorbedingung eines Schrittes zu sichern. Dies geschieht durch Auswahl eines geeigneten Operators.
- Der entsprechende kausale Link wird eingeführt. Drohungen werden durch Ordnungs-Constraints abgewehrt (es gibt 2 Fälle: der neue Schritt bedroht bestehende oder bestehende bedrohen den neuen).
- Falls es keinen Operator gibt oder die Drohungen nicht abgewehrt werden können — backtracken.

Theorem 7.3.2 (POP)
POP ist vollständig und korrekt.

Ist unser Beispiel eigentlich fair? Wir hatten doch nur sehr wenige Operatoren (ganze 2) und brauchen uns daher nicht zu wundern, daß der Verzweigungsfaktor so klein ist!

Dieser Einwand ist nicht gerechtfertigt, da auch wenn wir noch 1000 andere irrelevante Operatoren zulassen, diese nicht betrachtet werden, da wir sie nicht benutzen können, um unsere Vorbedingungen zu sichern. POP ist eben sehr ziel-gerichtet und vermeidet absolut irrelevante Aktionen.

Bisher haben wir uns überhaupt nicht um Variablen-Substitutionen gekümmert. Angenommen, S_1 sichert die $At(home)$ Vorbedingung eines anderen Schrittes S_2 und

function POP(*initial, goal, operators*) **returns** *plan*

plan ← MAKE-MINIMAL-PLAN(*initial, goal*)
loop do
  if SOLUTION?(*plan*) **then return** *plan*
  S_{need}, c ← SELECT-SUBGOAL(*plan*)
  CHOOSE-OPERATOR(*plan, operators*, S_{need}, c)
  RESOLVE-THREATS(*plan*)
end

---

function SELECT-SUBGOAL(*plan*) **returns** S_{need}, c

pick a plan step S_{need} from STEPS(*plan*)
  with a precondition c that has not been achieved
return S_{need}, c

---

procedure CHOOSE-OPERATOR(*plan, operators*, S_{need}, c)

choose a step S_{add} from *operators* or STEPS(*plan*) that has c as an effect
if there is no such step **then fail**
add the causal link $S_{add} \xrightarrow{c} S_{need}$ to LINKS(*plan*)
add the ordering constraint $S_{add} \prec S_{need}$ to ORDERINGS(*plan*)
if S_{add} is a newly added step from *operators* **then**
  add S_{add} to STEPS(*plan*)
  add *Start* $\prec S_{add} \prec$ *Finish* to ORDERINGS(*plan*)

---

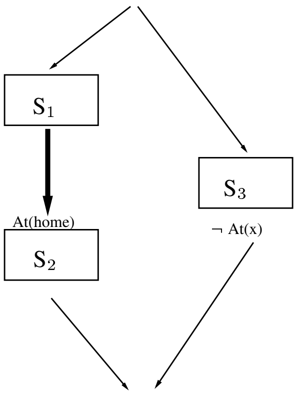
procedure RESOLVE-THREATS(*plan*)

for each S_{threat} that threatens a link $S_i \xrightarrow{c} S_j$ in LINKS(*plan*) **do**
  choose either
    Promotion: Add $S_{threat} \prec S_i$ to ORDERINGS(*plan*)
    Demotion: Add $S_j \prec S_{threat}$ to ORDERINGS(*plan*)
  if not CONSISTENT(*plan*) **then fail**
end

Tabelle 7.2: POP-Algorithmus.

es gibt einen paralellen Schritt S_3 mit der Nachbedingung $\neg At(x)$. Ist dies eine Bedrohung?

Wir nennen solch eine Drohung *möglich* und ignorieren sie zunächst, behalten sie aber im Hinterkopf. Falls aber irgendwann einmal x mit $home$ instantiiert wird, dann entsteht eine echte Drohung die aufgelöst werden muss.




```
procedure CHOOSE-OPERATOR(plan, operators, S_need, c)

  choose a step S_add from operators or STEPS(plan) that has c_add as an effect
          such that u = UNIFY(c, c_add, BINDINGS(plan))
  if there is no such step
     then fail
  add u to BINDINGS(plan)
  add S_add --c--> S_need to LINKS(plan)
  add S_add ≺ S_need to ORDERINGS(plan)
  if S_add is a newly added step from operators then
     add S_add to STEPS(plan)
     add Start ≺ S_add ≺ Finish to ORDERINGS(plan)
-----------------------------------------------------------------
procedure RESOLVE-THREATS(plan)

  for each S_i --c--> S_j in LINKS(plan) do
     for each S_threat in STEPS(plan) do
        for each c' in EFFECT(S_threat) do
           if SUBST(BINDINGS(plan), c) = SUBST(BINDINGS(plan), ¬ c') then
              choose either
                   Promotion: Add S_threat ≺ S_i to ORDERINGS(plan)
                   Demotion: Add S_j ≺ S_threat to ORDERINGS(plan)
           if not CONSISTENT(plan)
              then fail
        end
     end
  end
```

SHAKEY

STRIPS wurde ursprünglich für SHAKEY entworfen, einen kleinen mobilen Roboter. SHAKEY wurde duch 6 Operatoren beschrieben: $Go(x)$, $Push(b, x, y)$, $Climb(b)$, $Down(b)$, $Turn\_On(ls)$, $Turn\_Off(ls)$.

Damit SHAKEY den Lichtschalter an oder ausstellen kann, muss er auf einer Box stehen. Damit SHAKEY bei einer Go-Aktion nicht von einer Box fällt, ist $On(Shakey, floor)$ eine Vorbedingung der Go-Aktion.

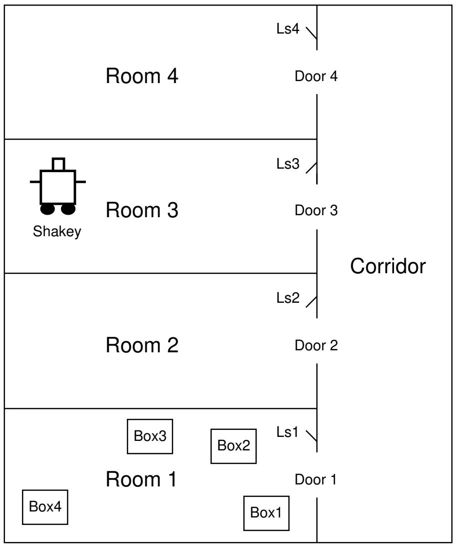


Abbildung 7.2: Shakey

7.4 Conditional Planning

Was tun, wenn die Welt nicht vollständig zugänglich ist? (Wo bekommt man Milch? Milch-Preis hat sich verdoppelt, man hat nicht genug Geld dabei!)

Führe neue Aktionen ein um gewisse Gegebenheiten abzufragen und entsprechend zu reagieren. (Wieviel kostet die Milch?)

Ein platter Reifen

$$
\begin{array}{lll}
\textbf{Op(} & Action: & Remove(x), \\
 & Prec: & On(x), \\
 & Effect: & Off(x) \land Clear\_Hub(x) \land \neg On(x) \\
\textbf{)} & & \\
\textbf{Op(} & Action: & Put\_on(x), \\
 & Prec: & Off(x) \land Clear\_Hub(x), \\
 & Effect: & On(x) \land \neg Clear\_Hub(x) \land \neg Off(x) \\
\textbf{)} & & \\
\textbf{Op(} & Action: & Inflate(x), \\
 & Prec: & Intact(x) \land Flat(x), \\
 & Effect: & Inflated(x) \land \neg Flat(x) \\
\textbf{)} & &
\end{array}
$$

Ziel: $On(x) \land Inflated(x)$,

```
function CONDITIONAL-PLANNING-AGENT(percept) returns an action
   static: KB, a knowledge base (includes action descriptions)
           p, a plan, initially NoPlan
           t, a counter, initially 0, indicating time
           G, a goal

   TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
   current ← STATE-DESCRIPTION(KB, t)
   if p = NoPlan then p ← CPOP(current, G, KB)
   if p = NoPlan or p is empty then action ← NoOp
   else
       action ← FIRST(p)
       while CONDITIONAL?(action) do
           if ASK(KB, CONDITION-PART[action]) then p ← APPEND(THEN-PART[action], REST(p))
           else p ← APPEND(ELSE-PART[action], REST(p))
           action ← FIRST(p)
       end
       p ← REST(p)
   TELL(KB, MAKE-ACTION-SENTENCE(action, t))
   t ← t + 1
   return action
```

Tabelle 7.3: Ein bedingter Planungs-Agent.

Anfangszustand: $Inflated(Spare) \wedge Intact(Spare) \wedge Off(Spare) \wedge On(Tire_1) \wedge Flat(Tire_1)$.

Was liefert unser POP? Da $Intact(Tire_1)$ nicht vorhanden, liefert POP als Plan

$$[Remove(Tire_1), Put\_on(Spare)].$$

Würden Sie das auch so machen? Besser wäre es doch, zunächst einmal nachzusehen, ob der Reifen wirklich kaputt ist, und nur in diesem Fall den ersatzreifen zu nehmen. Wir brauchen also einen *konditionalen Plan*:

Falls $Intact(Tire_1)$ dann: $Inflate(Tire_1)$.

Dazu müssen wir konditionale Schritte schon bei der Planerstellung erlauben:

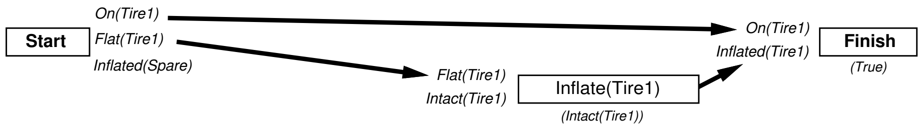
Definition 7.4.1 (Konditionale Schritte)
Ein konditionaler Schritt in einem Plan hat die Form

$$If(\{Condition\}, \{Then\_Part\}, \{Else\_Part\})$$

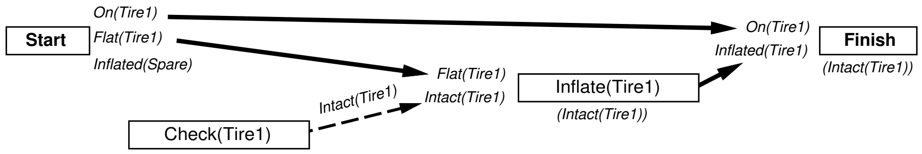
Der entsprechende Planungsagent sieht dann so aus: Der Agent muß beim Ausführen des Planes wissen (also erst zur Laufzeit) ob die entsprechende If-Bedingung gilt oder nicht. Daher führt man neue Aktionen zur Überprüfung ein:

$$\textbf{Op(}\quad \begin{array}{ll} Action: & Check\_Tire(x), \\ Prec: & Tire(x), \\ Effect: & \text{“Man weiss, ob } Intact(Tire) \text{ oder nicht.”} \end{array}$$
$$)$$

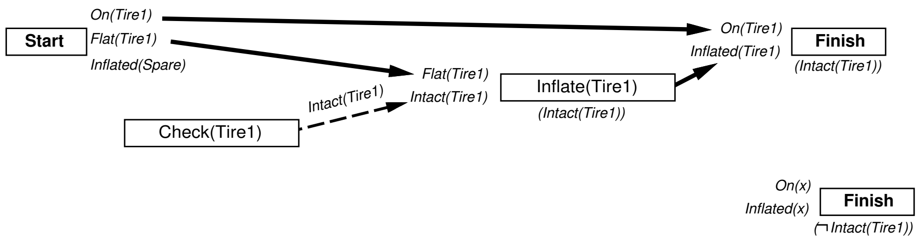
Definition 7.4.2 (Kontext)
Wir assoziieren mit jedem Schritt einen Kontext*: das ist die Menge der Bedingungen (Conditions), die gelten müssen, damit der Schritt ausgeführt werden kann.*



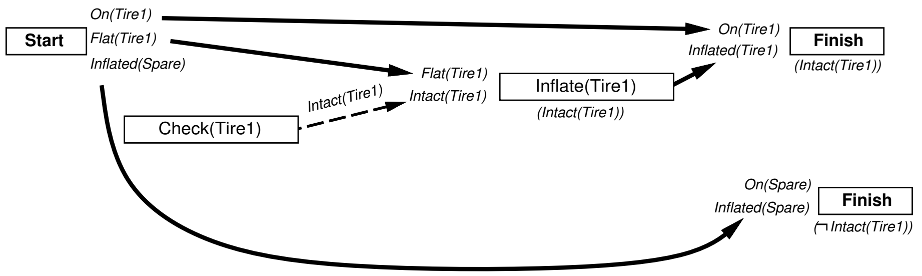
Hier würde POP schon backtracken, da $Intact(Tire_1)$ nicht gezeigt werden kann. Daher führt man eine neue Sorte von Links ein: *conditional links*.



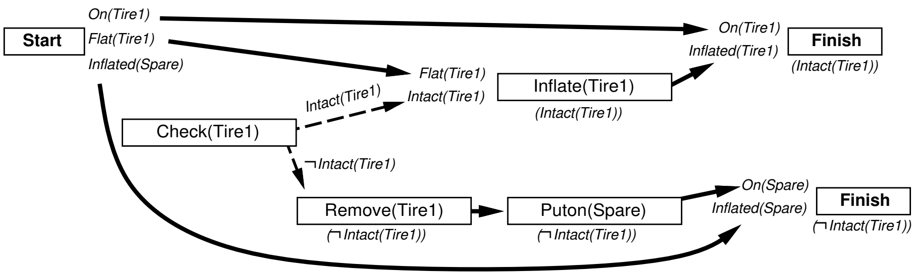
Man beachte, daß alle Fälle abgedeckt werden müssen. Also:



Wie können wir nun $Inflated(x)$ im FINISH-Schritt wahr machen? Das Hinzunehmen des Schrittes $Inflate(Tire_1)$ macht keinen Sinn, da ja die Vorbedingungen + Kontext inkonsistent miteinander sind.



Anstelle der letzten Modifikation könnte man auch einen $Inflate(Spare)$ Schritt einführen, aber das führt dann zu nichts. Es fehlt jetzt nur noch $On(Spare)$.



Achtung: zunächst steht "True" unter $Remove(Tire_1), Put\_on(Spare)$. Dann gibt es aber eine Drohung mit dem oberen Link. *Diese kann aufgelöst werden, indem die Kontexte inkompatibel gemacht werden. Die entsprechenden Kontexte vererben sich.* Genauer: Suche einen konditionalen Schritt, dessen Nachbedingung die Kontexte inkompatibel macht und damit Drohungen auflöst.

7.5 Erweiterungen

1. Ein Plan kann auch aus folgenden Gründen schiefgehen

- Aktionen können unerwartete Effekte haben, aber diese können aufgezählt werden (als Disjunktion).
- Die unerwarteten Effekte sind unbekannt. Dann muss man neu planen: *Replanning*.

```
function REPLANNING-AGENT(percept) returns an action
  static: KB, a knowledge base (includes action descriptions)
          p, an annotated plan, initially NoPlan
          q, an annotated plan, initially NoPlan
          G, a goal

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  current ← STATE-DESCRIPTION(KB, t)
  if p = NoPlan then
     p ← PLANNER(current, G, KB)
     q ← p
     if p = NoPlan or p is empty then return NoOp
  if PRECONDITIONS(p) not currently true in KB then
     p' ← CHOOSE-BEST-CONTINUATION(current, q)
     p ← APPEND(PLANNER(current, PRECONDITIONS(p'), KB), p')
     q ← p
  action ← FIRST(p)
  p ← REST(p)
  return action
```

D.h. man nimmt wahr und plant nur, wenn sich etwas verändert hat.

2. Man verbinde *Replanning* und *Conditional Planning*. Planen und Ausführung werden miteinander integriert.

```
function CPOP(initial, goals, operators) returns plan

  plan ← MAKE-PLAN(initial, goals)
  loop do
      Termination:
        if there are no unsatisfied preconditions
           and the contexts of the finish steps are exhaustive
            then return plan
      Alternative context generation:
        if the plans for existing finish steps are complete and have contexts C_1 ... C_n then
            add a new finish step with a context ¬ (C_1 ∨ ... ∨ C_n)
            this becomes the current context
      Subgoal selection and addition:
        find a plan step S_need with an open precondition c
      Action selection:
        choose a step S_add from operators or STEPS(plan) that adds c or
                knowledge of c and has a context compatible with the current context
        if there is no such step
            then fail
        add S_add --c--> S_need to LINKS(plan)
        add S_add < S_need to ORDERINGS(plan)
        if S_add is a newly added step then
            add S_add to STEPS(plan)
            add Start < S_add < Finish to ORDERINGS(plan)
      Threat resolution:
        for each step S_threat that potentially threatens any causal link S_i --c--> S_j
                  with a compatible context do
            choose one of
                Promotion: Add S_threat < S_i to ORDERINGS(plan)
                Demotion: Add S_j < S_threat to ORDERINGS(plan)
                Conditioning:
                    find a conditional step S_cond possibly before both S_threat and S_j, where
                        1. the context of S_cond is compatible with the contexts of S_threat and S_j;
                        2. the step has outcomes consistent with S_threat and S_j, respectively
                    add conditioning links for the outcomes from S_cond to S_threat and S_j
                    augment and propagate the contexts of S_threat and S_j
            if no choice is consistent
                    then fail
        end
  end
```

Tabelle 7.4: Der Conditional-Planning-Algorithmus

function SITUATED-PLANNING-AGENT(*percept*) **returns** an *action*
  static: *KB*, a knowledge base (includes action descriptions)
    p, a plan, initially *NoPlan*
    t, a counter, initially 0, indicating time
    G, a goal

  TELL(*KB*, MAKE-PERCEPT-SENTENCE(*percept, t*))
  current ← STATE-DESCRIPTION(*KB, t*)
  EFFECTS(START(*p*)) ← *current*
  if *p* = *NoPlan* **then**
    G ← ASK(*KB*, MAKE-GOAL-QUERY(*t*))
    p ← MAKE-PLAN(*current, G, KB*)
  action ← *NoOp* (the default)

  Termination:
    if there are no open preconditions and *p* has no steps other than START and FINISH **then**
      p ← *NoPlan* and skip remaining steps

  Resolving standard flaws:
    resolve any open condition by adding a causal link from any existing
      possibly prior step or a new step
    resolve potential threats by promotion or demotion

  Remove unsupported causal links:
    if there is a causal link START $\xrightarrow{c}$ S protecting a proposition c
      that no longer holds in START **then**
        remove the link and any associated bindings

  Extend causal links back to earliest possible step:
    if there is a causal link $S_j \xrightarrow{c} S_k$ such that
      another step S_i exists with $S_i < S_j$ and the link $S_i \xrightarrow{c} S_k$ is safe **then**
        replace $S_j \xrightarrow{c} S_k$ with $S_i \xrightarrow{c} S_k$

  Remove redundant actions:
    remove any step S that supplies no causal links

  Execute actions when ready for execution:
    if a step S in the plan other than FINISH satisfies the following:
      (a) all preconditions satisfied by START;
      (b) no other steps necessarily between START and S; and
      (c) S does not threaten any causal link in p **then**
        add ordering constraints to force all other steps after S
        remove S from p, and all causal links to and from S
        action ← the action in S

  TELL(*KB*, MAKE-ACTION-SENTENCE(*action*, *t*))
  $t \leftarrow t + 1$
  return *action*

7.6 Aufgaben

Aufgabe 7.1 (Partiell geordnete Pläne)
Definieren Sie ein Operatoren-Schema für das Problem, Schuhe, Socken, Hut und Mantel anzuziehen. Es sei angenommen, daß Hut und Mantel vorbedingungslos angezogen werden können. Geben Sie einen partiell geordneten Plan zur Lösung des Problems

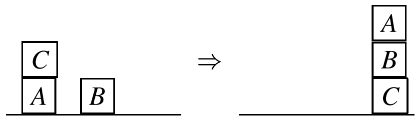
an! Wie viele Linearisierungen gibt es?

Aufgabe 7.2 (Makro-Operationen)

1. *Die STRIPS-Operatoren* pickup(x) *und* stack(x,y) *in einer Block-Welt kann man zu einem Makro-Operator* puton(x,y) *zusammenfassen. Wie sehen die drei Komponenten (Aktionsbeschreibung, Vorbedingung, Effekt) des Makro-Operators aus?*

2. *Versuchen Sie Bedingungen anzugeben, wie ganz allgemein Makro-Operatoren aus zwei gegebenen STRIPS-Operatoren automatisch abgeleitet werden können.*

Aufgabe 7.3 (Sussman-Anomalie)
Betrachten Sie das untenstehende Problem aus der Block-Welt. Formulieren Sie es mittels STRIPS-Operatoren und benutzen Sie die Funktion POP (siehe unten), um es zu lösen!



Aufgabe 7.4 (Algorithmus POP)
Eine wichtige Verfeinerung des POP-Algorithmus ist, daß der ausgewählte Schritt und die Vorbedingung in der Funktion SELECT-SUBGOAL nicht *Backtracking unterworfen werden. Wie wirkt sich das auf (a) die Korrektheit, (b) die Vollständigkeit und (c) die Komplexität des Verfahrens aus?*

Aufgabe 7.5 (Shakey's World I)
Beschreiben Sie die 6 Aktionen von Shakey *(a) im Situationskalkül und (b) als STRIPS-Operatoren. Betrachten Sie dazu auch Abbildung 7.3.*

Aufgabe 7.6 (Shakey's World II)

1. *Konstruieren Sie einen partiell geordneten Plan, um* Box2 *in Raum* Room2 *zu befördern, ausgehend von der Startsituation in Abbildung 7.3.*

2. *Angenommen,* Shakey *muß n Boxen von einem Raum in einen anderen befördern. Wie groß ist die Komplexität des Planungsprozesses in Abhängigkeit von n?*

Aufgabe 7.7 (Conditional Planning)
Diskutieren Sie die Anwendung von Conditional Planning *auf (a) die Staubsauger-Welt und (b) die Wumpus-Welt.*

8 Unsicheres Schließen

8.1 Motivation und Beispiele

Klassische Logiken lassen sich üblicherweise durch die Angabe zweier Dinge spezifizieren:

1. einer Menge von **Axiomen** und
2. einer Menge von **Schlußregeln**.

Durch Angabe verschiedener Axiomenmengen und Schlußregeln kann man verschiedene Logiken in uniformer Weise definieren. Formal macht man dies durch Definition einer Folgerungsrelation $\vdash \subseteq 2^{Fml} \times Fml$:

$\Phi \vdash \psi$ bedeutet *“aus der Menge Φ ist die Formel ψ ableitbar”*.

Dies haben wir in Definition 5.12 schon einmal formalisiert.

Warum Nichtmonotonie?

Datenbanken
Üblicherweise sind nur positive Fakten in einer Datenbank enthalten, z.B. eine Liste von verfügbaren Flügen. Klassisch gesehen kann keine negative Information aus den bekannten Fakten abgeleitet werden. Man hätte aber gerne, daß aus

$$\{flug(M, B, 9.00), flug(M, B, 12.00), flug(M, B, 15.00), flug(M, B, 18.00\}$$

auch

$$\neg flug(M, B, 11.00)$$

folgt. D.h., da $flug(M, B, 11.00)$ nicht explizit aufgelistet ist und wir annehmen können, daß die Tabelle vollständig ist, schliessen wir, daß ein solcher Flug nicht existiert. Die klassische Logik ist zu schwach, um solche Ableitungen zu erlauben: Wir können weder $flug(M, B, 11.00)$ noch $\neg flug(M, B, 11.00)$ ableiten.

Faustregeln
Oft ist ein Sachverhalt nicht genügend spezifiziert, da gewisse Hintergrundinformationen nötig sind. Im obigen Datenbank-Beispiel haben wir die Hintergrundinformation *"Die Liste ist vollständig"* benutzt. Anstatt streng logischer Schlüsse benutzen Menschen eher *Faustregeln* (engl. *Defaults*).

Beispiel 8.1.1 (Tweety und Fred)
Betrachten wir die folgenden beiden Faustregeln*:*

1. $vogel(x) \mid\!\sim fliegt(x)$*: Normalerweise können Vögel fliegen,*

2. $pinguin(x) \mid\!\sim \neg fliegt(x)$*: Normalerweise können Pinguine nicht fliegen.*

Angenommen, wir wissen auch noch, daß Pinguine Vögel sind und daß sowohl $tweety$ *als auch* $fred$ *zwei bestimmte Vögel sind. Natürlich ist es sinnvoll, aus diesen Informationen abzuleiten, daß sowohl* $tweety$ *als auch* $fred$ *fliegen können.*

In klassischer Logik:

$$\forall x(vogel(x) \rightarrow fliegt(x)),\ \forall x(pinguin(x) \rightarrow \neg fliegt(x)),$$

zusammen mit $vogel(tweety), vogel(fred)$ und $\forall x(pinguin(x) \rightarrow vogel(x))$.

Nun erfahren wir, daß $fred$ ein Pinguin ist. Addieren wir $pinguin(fred)$ zu unserer Formelmenge, können wir plötzlich alles mögliche ableiten: nicht nur $fliegt(fred)$, sondern auch $\neg fliegt(fred)$ — unsere Formelmenge ist also *inkonsistent*. Wir können jede beliebige Formel ableiten, also auch $\neg vogel(tweety), \neg fliegt(tweety), \forall x(fliegt(x) \leftrightarrow \neg fliegt(x))$.

Man könnte aber auch wie folgt vorgehen:

$$\forall x((vogel(x) \wedge \neg pinguin(x)) \rightarrow fliegt(x)),\ \forall x(pinguin(x) \rightarrow \neg fliegt(x)).$$

Dann hätte man zwar die Inkonsistenz vermieden, aber man kann nicht mehr

$$fliegt(tweety)$$

ableiten, da es keine Möglichkeit gibt, $\neg pinguin(tweety)$ abzuleiten.

Wir möchten gerne:

1. $\Phi \mid\!\sim fliegt(fred)$ und

2. $\Phi \cup \{pinguin(fred)\} \mid\!\sim \neg fliegt(fred)$.

In jeder monotonen Logik ist im zweiten Fall $fliegt(fred)$ immer noch ableitbar und daher ergibt sich zwangsläufig eine Inkonsistenz.

Erklärungen/Rechtfertigungen
Betrachten wir das folgende Beispiel. Wir wissen, daß meistens der Rasen dann naß wird, wenn es regnet. Er wird aber auch naß, wenn der Rasensprenger an ist (und dann bleibt die Straße trocken). Damit ergeben sich verschiedene Erklärungen, je nach Wissensstand:

Der Rasen ist naß.
1. Erklärung: Es hat geregnet.
Aber die Straße ist trocken.
Revidierte Erklärung: Der Rasensprenger war an.

Dieses Beispiel führt zum **Prinzip der Abduktion**:

Wenn ψ gilt und $\phi \rightarrow \psi$, dann schließen wir auf ϕ.

Wir listen im folgenden einige Beispiele auf, die typische nichtmonotone Phänomene beschreiben.

Beispiel 8.1.2 (Ausnahmen)
Folgende Fakten sind bekannt:

1. *Vögel fliegen normalerweise.*
2. *Ausnahmen sind Pinguine, sie fliegen nicht.*
3. *Tweety ist ein Vogel.*

Intuitiv möchten wir natürlich ableiten, daß Tweety fliegt. Nach einiger Zeit lernen wir aber: Tweety ist ein Pinguin. Frage: Fliegt Tweety immer noch? *Intuitiv erwarten wir: Nein.*

Beispiel 8.1.3 (Nixon-Diamant)
Folgende Fakten sind bekannt:

1. *Quäker sind normalerweise Pazifisten.*
2. *Republikaner sind normalerweise keine Pazifisten.*
3. *Nixon war sowohl Quäker als auch Republikaner.*

Frage: War Nixon ein Pazifist? *Wir erwarten hier, daß wir keine eindeutige Aussage treffen können.*

Beispiel 8.1.4 (Disjunktion)
Wir haben zwei Defaults und ein Faktum.

1. *Normalerweise trinken Italiener gerne Wein, Franzosen auch.*
2. *Georg ist Franzose oder Italiener.*

Frage: Trinkt Georg gerne Wein? *Hier erwarten wir die Antwort* Ja.

Beispiel 8.1.5 (Disjunktion mit Blockade)
Wir haben hier zwei Defaults und ein Faktum.

1. *Ein Arm ist normalerweise benutzbar.*
2. *Ein gebrochener Arm ist normalerweise eine Ausnahme zu 1.*
3. *Wir wissen, daß Peter den rechten oder den linken Arm gebrochen hat.*

Frage: Sind beide Arme benutzbar? *Hier erwarten wir die Antwort* Nein*, obwohl wir nicht wissen können, welcher Arm gebrochen ist.*

8.2 Zwei Ansätze mit klassischer Logik

Definition 8.2.1 (CWA)
Sei $\mathcal{A} \subseteq At$ eine Teilmenge der Menge der Atome der zugrundeliegenden Sprache $\mathcal{L}$ (also aussagenlogische Konstanten), W eine Menge von Sätzen, die Elemente aus $\mathcal{A}$ enthalten. Dann definieren wir die bezüglich $\mathcal{A}$ abgeschlossene Welt von W durch die folgende Formelmenge[1]

$$CWA^{\mathcal{A}}(W) :=_{def} Cn(W \cup \{\neg a : a \in \mathcal{A} \wedge W \not\vdash a\}).$$

Wir zeichnen also zunächst eine Menge $\mathcal{A}$ von Aussagen aus. Dann nehmen wir zu unserem sicheren Wissen W die Negate $\neg a$ all jener Aussagen $a \in \mathcal{A}$ hinzu, die nicht aus W ableitbar sind. Diese neue Menge schließen wir dann noch deduktiv ab. Denken wir an unser Fahrplan-Beispiel zurück, so bedeutet dies, daß wir $\neg flug(M, B, 11.00)$ explizit zu unserer Menge hinzunehmen.

Eine andere Möglichkeit besteht darin, maximal konsistente Teilmengen zu betrachten:

Definition 8.2.2 (Maximal konsistente Teilmengen)
Gegeben sei (D, W), wobei D (Defaults) und W (Fakten) beides Mengen von Sätzen sind. Eine Formelmenge $\mathcal{E}$ heißt Abschluß von (D, W), falls

$$\mathcal{E} = Cn(W \cup D_0)$$

wobei $D_0 \subseteq D$ maximal unter allen $D' \subseteq D$ so ist, daß $W \cup D'$ konsistent ist.

Hier betrachten wir also die verschiedenen Möglichkeiten, eine Theorie W maximal um Sätze aus D zu erweitern ohne inkonsistent zu werden.

Für Beispiel 8.1.2 erhalten wir folgende Formalisierung:

$$\begin{array}{lll}
T_1 :=_{def} & \{ & vogel(x) \wedge \neg ab_1(x) \rightarrow fliegt(x), \\
 & & pinguin(x) \rightarrow ab_1(x), \\
 & pinguin(x) \rightarrow vodel(x), & \\
 & & pinguin(x) \wedge \neg ab_2(x) \rightarrow \neg fliegt(x), \\
 & & vogel(tweety)\ \} \\
 & & \\
T_2 :=_{def} T_1 \cup & \{ & pinguin(tweety)\ \}
\end{array}$$

Mit $\mathcal{A} :=_{def} \{ab_1(tweety), ab_2(tweety)\}$ ergibt sich

$$CWA^{\mathcal{A}}(T_1) = Cn(T_1 \cup \{\neg ab_1(tweety), \neg ab_2(tweety)\},$$

also

$$CWA^{\mathcal{A}}(T_1) \models fliegt(tweety)$$

[1]Mit $Cn(T)$ bezeichnen wir den deduktiven Abschluß von $T \subseteq Fml_{\mathcal{L}}$, also die Menge aller Folgerungen von T in der jeweils betrachteten Logik.

und

$$CWA^{\mathcal{A}}(T_2) = Cn(T_2 \cup \{\neg ab_2(tweety)\} \models \neg fliegt(tweety),$$

also die gewünschten Ergebnisse. Der maximal konsistente Teilmengen-Ansatz liefert dasselbe. Für T_1 ergibt sich als $\mathcal{E}_1$ genau $CWA^{\mathcal{A}}(T_1)$ und für T_2 ergibt sich ebenfalls $CWA^{\mathcal{A}}(T_2)$ als einziger Abschluß.

Für Beispiel 8.1.3 ergibt sich als Formalisierung:

$$\begin{array}{ll} T :=_{def} \{ & r(x) \wedge \neg ab_1(x) \rightarrow \neg p(x), \\ & q(x) \wedge \neg ab_2(x) \rightarrow p(x), \\ & r(nixon) \wedge q(nixon) \ \} \end{array}$$

Wir erhalten mit $\mathcal{A} :=_{def} \{ab_1(nixon), ab_2(nixon)\}$

$$CWA^{\mathcal{A}}(T) = Cn(T \cup \{\neg ab_1(nixon), \neg ab_2(nixon)\} \models \Box,$$

es sind also alle Formeln ableitbar, da die gebildete Menge inkonsistent ist. Der CWA-Ansatz erweist sich also hier als zu stark. Wählen wir den maximal-konsistenten Teilmengen-Ansatz und setzen

$$D :=_{def} \{\neg ab_1(nixon),\ \neg ab_2(nixon)\},\ W :=_{def} T,$$

so sind für $D_1 :=_{def} \{\neg ab_1(nixon)\}$ und $D_2 :=_{def} \{\neg ab_2(nixon)\}$ jeweils

$$Cn(W \cup D_1),\ \text{bzw. } Cn(W \cup D_2)$$

maximal konsistente Mengen. In einer ist Nixon Pazifist, in der anderen nicht. Betrachten wir die Aussagen, die in beiden Mengen gelten, so erhalten wir unsere Ausgangsmenge zurück und können nichts darüber aussagen, ob Nixon Pazifist war oder nicht. Dieser Ansatz gibt uns also die gewünschten Ergebnisse.[2]

Die Formalisierung des Beispiels 8.1.4 ist ähnlich zu der von Beispiel 8.1.3:

$$\begin{array}{ll} T :=_{def} \{ & italiener(x) \wedge \neg ab_1(x) \rightarrow weintrinker(x), \\ & franzose(x) \wedge \neg ab_2(x) \rightarrow weintrinker(x), \\ & italiener(georg) \vee franzose(georg) \ \} \end{array}$$

Wir erhalten mit $\mathcal{A} :=_{def} \{ab_1(georg), ab_2(georg)\}$

$$CWA^{\mathcal{A}}(T) = Cn(T \cup \{\neg ab_1(georg), \neg ab_2(georg)\} \models weintrinker(georg).$$

In diesem Beispiel ergibt sich auch mit dem maximal-konsistenten Teilmengen-Ansatz nur genau ein Abschluß, der mit $CWA^{\mathcal{A}}(T)$ übereinstimmt.

In Beispiel 8.1.5 versagen allerdings beide Ansätze.

$$\begin{array}{ll} T :=_{def} \{ & \neg ab_1(x) \rightarrow useable(x), \\ & broken(x) \wedge \neg ab_2(x) \rightarrow ab_1(x), \\ & broken(left) \vee broken(right) \ \} \end{array}$$

[2]Man könnte hier auch auf die Formulierung mit ab_1, ab_2 verzichten und nur $r(nixon)$, $q(nixon)$ als Defaults betrachten.

Wir erhalten für $\mathcal{A} :=_{def} \{ab_1(left), ab_1(right), ab_2(left), ab_2(right)\}$

$$CWA^{\mathcal{A}}(T) = Cn(T \cup \{\neg ab_1(left), \neg ab_1(right), \neg ab_2(left), \neg ab_2(right)\}) \ \models \Box,$$

also wieder eine inkonsistente Menge!

Der konsistente Teilmengen-Ansatz mit

$$\mathcal{D} :=_{def} \{\neg ab_1(left), \neg ab_1(right), \neg ab_2(left), \neg ab_2(right)\}$$

liefert vier Abschlüsse:

1. Im ersten, $\mathcal{E}_1$ bezüglich $D_1 :=_{def} \{\neg ab_2(left),\ \neg ab_2(right),\ \neg ab_1(right)\}$ gilt: $ab_1(left)$, $broken(left)$, $useable(right)$.
2. Im zweiten, $\mathcal{E}_2$ bezüglich $D_2 :=_{def} \{\neg ab_2(left),\ \neg ab_2(right),\ \neg ab_1(left)\}$ gilt: $ab_1(right)$, $broken(right)$, $useable(left)$.
3. Im dritten, $\mathcal{E}_3$ bezüglich $D_3 :=_{def} \{\neg ab_2(left),\ \neg ab_1(right),\ \neg ab_1(left)\}$ gilt: $useable(left)$, $useable(right)$, $broken(left)$ und $ab_2(right)$.
4. Im vierten, $\mathcal{E}_4$ bezüglich $D_4 :=_{def} \{\neg ab_2(right),\ \neg ab_1(right),\ \neg ab_1(left)\}$ gilt: $useable(left)$, $useable(right)$, $broken(right)$ und $ab_2(left)$.

In den letzten beiden Abschlüssen sind also beide Arme benutzbar — was natürlich völlig unserer Intuition widerspricht. Dieser Ansatz liefert uns also zwei überflüssige Abschlüsse und ist daher zu schwach. Man beachte, daß der Schnitt der letzten beiden Formelmengen $\mathcal{E}_3$ und $\mathcal{E}_4$ genau der Menge

$$CWA^{\{\neg ab_1(left), \neg ab_1(right)\}}(T)$$

entspricht. Auch aus dieser Menge ist $useable(left) \wedge useable(right)$ ableitbar.

8.3 Default-Logik

Definition 8.3.1 (Regelsystem MP + D)
Sei D eine Menge von allgemeinen Inferenzregeln, d.h. Abbildungen, die einer endlichen Menge von Formeln $\varphi_1, \varphi_2, \ldots, \varphi_n$ eine Formel ψ zuordnen. Wir schreiben dies als

$$\frac{\varphi_1, \varphi_2, \ldots, \varphi_n}{\psi}.$$

Unter MP + D verstehen wir das Regelsystem, das entsteht, wenn die Regeln aus D zur Regel Modus Ponens hinzugenommen werden.

Für $W \subseteq Fml$ sei im folgenden

$$Cn^D(W)$$

die Menge aller Formeln φ, die man aus W und den Inferenzregeln aus MP + D ableiten kann.

Man beachte, daß $Cn(W)$ semantisch, $Cn^D(W)$ aber syntaktisch (über den Begriff des Beweises) definiert sind. Im Spezialfall $D = \emptyset$ stimmen beide Mengen nach dem Vollständigkeitssatz überein.

Lemma 8.3.2 (Eigenschaften von Cn^D)
Sei D eine Menge von allgemeinen Inferenzregeln und $W \subseteq Fml$. Dann gilt:

1. $Cn(W) \subseteq Cn^D(W)$.
2. $Cn^D(Cn^D(W)) = Cn^D(W)$.
3. $Cn^D(W)$ *ist die kleinste unter D abgeschlossene Menge die W enthält.*

Bemerkung 8.3.3 (Spezielle Form monotoner Inferenzregeln)
Wir können uns bei monotonen Inferenzregeln auf solche der Form

$$\frac{\alpha}{\beta}$$

beschränken, d.h. die Prämissenmenge besteht nur aus einer einzigen Formel.

Beweis: $\varphi_1 \wedge \ldots \wedge \varphi_k$ hat eine Ableitung in MP+D bezüglich einer Menge T gdw $\forall 1 \leq i \leq k : \varphi_i$ hat Ableitung in MP+D bezüglich T. Daher kann man

$$\frac{\varphi_1, \varphi_2, \ldots, \varphi_k}{\psi} \text{ durch } \frac{\varphi_1 \wedge \ldots \wedge \varphi_k}{\psi}$$

ersetzen. ■

Definition 8.3.4 (R^D Operator)
Für eine Menge von monotonen Inferenzregeln D und eine Theorie $W \subseteq Fml$ sei

$$R^D(W) :=_{def} Cn(W \cup \{\gamma : \frac{\alpha}{\gamma} \in D \textit{ und } \alpha \in W\})$$

Lemma 8.3.5 (Eigenschaften von R^D)
R^D ist monoton, kompakt und progressiv. Mit anderen Worten: Es gibt zu jedem $Y \subseteq Fml_{\mathcal{L}}$ eine kleinste Menge $Z \subseteq Fml_{\mathcal{L}}$ mit $Y \subseteq Z$ und $R^D(Z) = Z$. Außerdem ist $Z = R^D \uparrow^\omega (Y)$.

Der R-Operator leitet alles ab, was man in einem Schritt bekommt. Man betrachte etwa $W = \{\alpha, \gamma \to \beta\}$ und $D = \{\frac{\alpha}{\gamma}, \frac{\beta}{\eta}\}$. Dann ist

$$\begin{array}{lcl} R^D(W) & = & Cn(\{\alpha, \gamma, \beta\}) \\ R^D(R^D(W)) & = & Cn(\{\alpha, \gamma, \beta, \eta\}) \end{array}$$

Theorem 8.3.6 ($Cn^D = R^D \uparrow^\omega$)
Für alle $W \subseteq Fml$: $Cn^D(W) = R^D \uparrow^\omega (W)$.

Beweis:

$\supseteq$: Wir zeigen durch Induktion

$$R^D \uparrow^n (W) \subseteq Cn^D(W).$$

- $n = 0$: $R^D \uparrow^0 (W) = Cn(W)$.
- $n \mapsto n+1$: Sei $R^D \uparrow^n (W) \subseteq Cn^D(W)$. Dann gilt

$$R^D \uparrow^{n+1} (W) = Cn(\underbrace{\underbrace{R^D \uparrow^n (W)}_{\subseteq Cn^D(W)} \cup \underbrace{\{\gamma : \frac{\alpha}{\gamma} \in D \wedge \alpha \in R^D \uparrow^n (W)\}}_{\subseteq Cn^D(W)}}_{\subseteq Cn^D(W)}).$$

$\subseteq$: Sei $\varphi \in Cn^D(W)$, φ hat also einen Beweis in MP+D. Wir machen eine Induktion über die Länge dieses Beweises. Sei $\varphi_1, \ldots, \varphi_n$ ein Beweis von φ aus W in MP+D.
Wir haben zu zeigen, daß $\varphi \in R^D \uparrow^\omega (W)$.

- **Induktions-Anfang:** $n = 0$**.** Klar, aufgrund der Definition von R^D.
- **Induktions-Schluß:** $(n-1) \mapsto n$**.** Unsere Annahme sagt, daß die Aussage für Beweise der Länge höchstens $n-1$ gilt. Sei ein Beweis der Länge n vorgelegt. Dann gilt $\forall i < n : \ \varphi_i \in R^D \uparrow^\omega (W)$ also $\exists m \in \mathbb{N} : \forall i \leq n\, \varphi_i \in R^D \uparrow^m (W)$. Wir haben zu zeigen, daß $\varphi \in R^D \uparrow^\omega (W)$.

  Dies zeigen wir durch Fallunterscheidung: Wie entstand φ?

1. Fall: φ_n ist Axiom, dann $\varphi \in R^D \uparrow^1 (W)$.
2. Fall: $\varphi_n \in W$, dann erst recht $\varphi \in R^D \uparrow^1 (W)$.
3. Fall: φ_n entsteht durch Anwendung von Modus Ponens auf φ_l, φ_k mit $l, k < n$. Da $R^D \uparrow^m (W)$ abgeschlossen ist unter MP und nach Induktionsvoraussetzung $\varphi_l, \varphi_k \in R^D \uparrow^m (W)$ gilt, ist auch $\varphi \in R^D \uparrow^m (W)$.
4. Fall: φ_n entsteht durch Anwendung einer Regel aus D: $\frac{\varphi_i}{\varphi_n}$ mit $i < n$. Nach Definition des R^D Operators ist dann $\varphi_n \in R^D \uparrow^{m+1} (W)$.

■

Definition 8.3.7 (Default-Theorie $(W, \mathcal{D})$)
Eine Default Theorie ist ein Paar $(W, \mathcal{D})$ mit :

- $W \subseteq Fml_{\mathcal{L}}$ *ist eine aussagenlogische Menge von Formeln,*
- $\mathcal{D}$ *ist eine Menge von Defaults der Form :*

$$d = \frac{\alpha :\ \beta_1, \ldots, \beta_n}{\gamma}.$$

Dabei sind $\alpha,\ \beta_1, \ldots, \beta_n,\ \gamma\ \in Fml_{\mathcal{L}}$. Wir nennen

  - α *die Vorbedingung von d,*

- β_i *die Rechtfertigungen von* d *und*
- γ *die Konsequenz von* d.

Wir schreiben auch $\alpha = prereq(d)$, $\mathcal{J}(d) = \{\beta_1, \dots, \beta_n\}$, $\gamma = concl(d)$.

- *d heißt p-frei falls* $\alpha = \top$. *Wir schreiben auch* $\frac{\emptyset : \beta_1, \dots, \beta_n}{\gamma}$.
- *d heißt j-frei falls* $\mathcal{J}(d) = \emptyset$, *bzw.* $\mathcal{J}(d) = \{\top\}$.

Im folgenden definieren wir einen Beweiskalkül für Defaults. Die Grundidee dabei ist, daß die Anwendung eines Default von dem jeweiligen Kontext abhängt. Wir interpretieren

$$\frac{\alpha \ : \beta}{\gamma}$$

als

"Falls α gilt und es konsistent ist β anzunehmen, dann leite γ ab."

Um dies auszudrücken, definieren wir den Begriff des S-Beweises einer Formel φ aus W bez. einer Menge von Inferenzregeln MP+$\mathcal{D}$.

Definition 8.3.8 (S-Beweis bez. MP+$\mathcal{D}$)
Sei $(W, \mathcal{D})$ eine Defaulttheorie. Für $S \subseteq Fml_{\mathcal{L}}$ und $\varphi \in Fml_{\mathcal{L}}$ definieren wir den S-Beweis einer Formel φ aus W bez. MP+$\mathcal{D}$ als Folge $\varphi_1, \dots, \varphi_n$ so, daß gilt:

1. $\varphi_n = \varphi$,
2. $\forall\, 1 \leq i \leq n$ *gilt eine der folgenden Bedingungen:*
   (a) $\varphi_i \in W$,
   (b) φ_i *ist eine Substitutionsinstanz eines AL-Axioms,*
   (c) φ_i *entsteht durch Anwendung des Modus Ponens,*
   (d) φ_i *entsteht durch Anwendung eines Defaults, d.h. es gibt* $\frac{\varphi_j : \beta_1, \dots, \beta_m}{\varphi_i} \in \mathcal{D}$ *mit* $j < i$ *und* $\neg\beta_1 \notin S, \dots, \neg\beta_m \notin S$.

S heißt der Kontext. *Wir bezeichnen mit $Cn^{\mathcal{D},S}(W)$ die Menge aller Formeln, die einen S-Beweis aus W bez. MP+$\mathcal{D}$ besitzen.*

Lemma 8.3.9 ($Cn^{\mathcal{D},S}$ **ist eine Erweiterung von** Cn^{D})
Sei $\mathcal{D}$ eine Menge von j-freien Defaults. Sei $D_{mon} :=_{def} \{\frac{\alpha}{\gamma} : \frac{\alpha : \emptyset}{\gamma} \in \mathcal{D}\}$. Für jeden Kontext $S \subseteq Fml$ und für jede Theorie $W \subseteq Fml$ gilt:

$$Cn^{D_{mon}}(W) = Cn^{\mathcal{D},S}(W).$$

Das letzte Lemma sagt aus, daß für j-freie Defaults alle Konsequenzoperatoren, die man mit MP+$\mathcal{D}$ assoziieren kann, übereinstimmen. Zusätzlich fallen sie mit denen des monotonen Systems $MP + D_{mon}$ zusammen.

Theorem 8.3.10 (Eigenschaften von $Cn^{\mathcal{D},S}$)
$Cn^{\mathcal{D},S}(W)$ ist monoton in $\mathcal{D}$ und W, aber antimonoton in S:

1. *Monotonie in W: $W_1 \subseteq W_2 \Rightarrow Cn^{\mathcal{D},S}(W_1) \subseteq Cn^{\mathcal{D},S}(W_2)$.*
2. *Monotonie in $\mathcal{D}$: $\mathcal{D}_1 \subseteq \mathcal{D}_2 \Rightarrow Cn^{\mathcal{D}_1,S}(W) \subseteq Cn^{\mathcal{D}_2,S}(W)$.*
3. *Antimonotonie in S: $S_1 \subseteq S_2 \Rightarrow Cn^{\mathcal{D},S_2}(W) \subseteq Cn^{\mathcal{D},S_1}(W)$.*

Wir haben jetzt eine große Auswahl an Konsequenzoperatoren:

$$\{Cn^{\mathcal{D},S}(W) \ : \ S \subseteq Fml\}.$$

Welches sind die richtigen? Wie wählt man zu gegebener Default Theorie $(W, \mathcal{D})$ das "richtige" S?

Definition 8.3.11 (Extension)
Eine Theorie $E \subseteq Fml$ heißt Extension der Default Theorie $(W, \mathcal{D})$, falls

$$E = Cn^{\mathcal{D},E}(W).$$

E ist also Fixpunkt des $Cn^{\mathcal{D},\odot}(W)$-Operators im zweiten Argument.

Am besten stellt man sich eine Extension als nachträglich *gerechtfertigte*, in sich stabile Menge vor. Wir fangen mit einer potentiellen Menge S von Formeln an. Diese interpretieren wir als *Kontext*, relativ zu dem wir Folgerungen ziehen. Wenn nun die Menge aller aus $(W, \mathcal{D})$ relativ zu S folgerbaren Formeln genau mit S übereinstimmt, haben wir S im nachhinein als richtige Wahl rechtfertigt. S stabilisiert sich selbst — S ist somit ein möglicher Kandidat für den Abschluß von $(W, \mathcal{D})$.

Definition 8.3.12 (Abgeschlossen unter Defaults)
Ein Default $\frac{\alpha:\beta_1,\dots,\beta_n}{\gamma}$ heißt anwendbar in der Theorie $S \subseteq Fml$, falls $\alpha \in S$ und $\neg\beta_1 \notin S \dots, \neg\beta_n \notin S$.

Eine Theorie $S \subseteq Fml$ heißt abgeschlossen für eine Defaulttheorie $(W, \mathcal{D})$, falls

- *$W \subseteq S$ und $S = Cn(S)$,*
- *Die Konklusion jedes in S anwendbaren Defaults ist in S enthalten.*

Man macht sich leicht klar, daß jede Extension E einer Defaulttheorie $(W, \mathcal{D})$ auch wirklich abgeschlossen für $(W, \mathcal{D})$ ist.

Für Beispiel 8.1.2 (Ausnahmen), also $W :=_{def} \{vogel\}, \mathcal{D} :=_{def} \{\frac{vogel:fliegt}{fliegt}\}$, erhalten wir

S	$Cn(W)$	$Cn(W \cup \{fliegt\})$	$Cn(W \cup \{\neg fliegt\})$	Fml
$Cn^{\mathcal{D},S}(W)$	$Cn(W \cup \{fliegt\})$	$Cn(W \cup \{fliegt\})$	$Cn(W)$	$Cn(W)$

Es gibt also genau eine Extension, nämlich $Cn(W \cup \{fliegt\})$. Tweety fliegt also!

Für Beispiel 8.1.3 (Nixon Diamant), also $W :=_{def} \{q, r\}$, $\mathcal{D} :=_{def} \{\frac{q:p}{p}, \frac{r:\neg p}{\neg p}\}$:

S	$Cn(W)$	$Cn(W \cup \{p\})$	$Cn(W \cup \{\neg p\})$	Fml
$Cn^{\mathcal{D},S}(W)$	Fml	$Cn(W \cup \{p\})$	$Cn(W \cup \{\neg p\})$	$Cn(W)$

Hier gibt es also zwei Extensionen, nämlich $Cn(W \cup \{p\})$ und $Cn(W \cup \{\neg p\})$. Beide Extensionen entsprechen den möglichen Gegebenheiten. Auch hier stimmt also der Extensionsbegriff mit dem intuitiven Begriff überein.

Theorem 8.3.13 (Existenz von Extensionen)
Sei $\mathcal{D}$ eine Menge von j-freien Defaults, $W \subseteq Fml$, $D_{mon} :=_{def} \{\frac{\alpha}{\gamma} : \frac{\alpha:\emptyset}{\gamma} \in \mathcal{D}\}$. Dann ist $Cn^{D_{mon}}(W)$ die einzige Extension von $(W, \mathcal{D})$.

Definition 8.3.14 (Normaler Default)
Ein Default der Form $\frac{\alpha:\beta}{\beta}$ heißt normal.

Theorem 8.3.15 (Eigenschaften normaler DL-Theorien)
Sei $(W, \mathcal{D})$ normal. Es gilt:

1. **Existenz:** *$(W, \mathcal{D})$ normal besitzt mindestens eine Extension.*
2. **Orthogonalität:** *Sind E_1, E_2 verschiedene Extensionen von $(W, \mathcal{D})$ dann ist $E_1 \cup E_2$ inkonsistent.*
3. **Semi-Monotonie:** *Für jede Extension E_1 von $(W, \mathcal{D}_1)$ gibt es eine Extension E_2 von $(W, \mathcal{D}_1 \cup \mathcal{D}_2) : E_1 \subseteq E_2$.*

Definition 8.3.16 (CWA $\mapsto$ DL)
Jedem $\mathcal{A} \subseteq \mathcal{L}$ (also einer Teilmenge der aussagenlogischen Konstanten $\mathcal{AT}$) ordnen wir die folgende Menge von Defaults zu:

$$\mathcal{D}_{CWA^{\mathcal{A}}} :=_{def} \{\frac{\emptyset : \neg a}{\neg a} : a \in \mathcal{A}\}.$$

Theorem 8.3.17 (CWA ist eine spezielle normale DL)
Sei $\mathcal{A} \subseteq \mathcal{L}$, dann sind für jede Theorie $W \subseteq Fml_{\mathcal{L}}$ äquivalent

- *$CWA^{\mathcal{A}}(W)$ ist konsistent,*
- *$(W, \mathcal{D}_{CWA^{\mathcal{A}}})$ besitzt genau eine konsistente Extension.*

Es gilt ferner, daß diese eindeutig bestimmte Extension genau $CWA^{\mathcal{A}}(W)$ ist.

Definition 8.3.18 (Max. kons. Teilmengen $\mapsto$ DL)
Gegeben sei ein Paar (D, W) wobei $D, W \subseteq Fml$. Wir können diesem Paar (D, W) eine Defaulttheorie $(W, \mathcal{D})$ zuordnen indem wir die Defaults wie folgt definieren

$$\mathcal{D} :=_{def} \{\frac{\emptyset : d}{d} : d \in D\}.$$

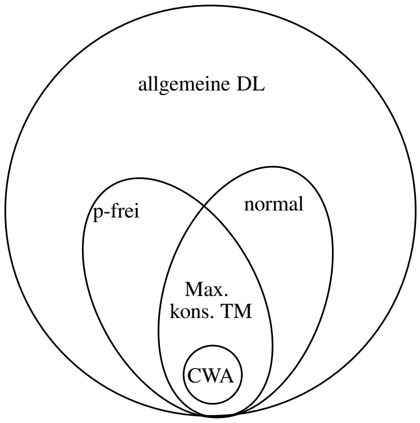


Abbildung 8.1: Teilsysteme der Default-Logik

Theorem 8.3.19 (Max. kons. TM = p-freier, normaler DL)
Sei $(W, \mathcal{D})$ die dem Paar (D, W) zugeordnete Defaulttheorie. Dann sind die Abschlüsse $\mathcal{E}_i$ von (D, W) genau die Extensionen von $(W, \mathcal{D})$.

Umgekehrt ergibt sich jede p-freie, normale Defaulttheorie $(W, \mathcal{D})$ als Bild einer Theorie (D, W) unter der in Def. 8.3.18 definierten Abbildung.

Die letzten beiden Ergebnisse haben gezeigt, daß die Defaulttheorie unsere beiden ad-hoc Ansätze CWA und max. konsistente Teilmengen enthält. Was leistet sie darüber hinaus?

Beispiel 8.3.20 (Spezifizität)
Folgende Fakten sind bekannt:

1. *Erwachsene sind normalerweise verheiratet.*
2. *Studenten sind normalerweise Erwachsene.*
3. *Studenten sind normalerweise nicht verheiratet.*
4. *Fred ist ein Erwachsener.*

Intuitiv möchten wir natürlich ableiten, daß Fred verheiratet ist. Nach einiger Zeit lernen wir aber: Fred ist ein Student. Frage: Ist Fred immer noch verheiratet? *Intuitiv erwarten wir: Nein, denn die spezifischere Information sollte die allgemeinere überschreiben.*

Die übliche Formalisierung lautet ($W := \{stud_{fred}\}$)

$$\mathcal{D} := \{\frac{erw_{fred} : verh_{fred}}{verh_{fred}}, \frac{stud_{fred} : erw_{fred}}{erw_{fred}}, \frac{stud_{fred} : \neg verh_{fred}}{\neg verh_{fred}}\}.$$

Daraus ist das gewünschte Ergebnis nicht herleitbar, da es eine Extension gibt, in der $verh_{fred}$ gilt.

Um eine Priorität zwischen den beiden widersprechenden Defaults auszudrücken, kann man den ersten Default $\frac{erw_{fred}:verh_{fred}}{verh_{fred}}$ durch

$$\frac{erw_{fred} : \neg stud_{fred}, verh_{fred}}{verh_{fred}}$$

ersetzen. Nun erhält man nur eine einzige Extension, in der $\neg verh_{fred}$ gilt.

8.4 Circumscription

Die wesentliche Idee der Circumscription ist es, Fakten über die Welt als gewöhnliche Formeln (der Prädikatenlogik) auszudrücken, aber dann nicht Gültigkeit in allen Modellen, sondern nur in bestimmten Modellen zu betrachten. Und zwar in solchen, die gewisse Prädikate möglichst klein machen.

Betrachten wir uns unsere Datenbank über Flugverbindungen und Beispiel 8.1.2. Im ersten Beispiel haben wir die Theorie T_0

$$\{flug(M, B, 9.00), flug(M, B, 12.00), flug(M, B, 15.00), flug(M, B, 18.00)\}$$

als gegebene Menge bezüglich einer zugrundeliegenden Sprache $\mathcal{L}$. Logisch kann aus dieser Menge nicht $\neg flug(M, B, 11.00)$ abgeleitet werden, da es ja ein Modell von $T_0 \cup \{flug(M, B, 11.00)\}$ gibt. Fragen wir aber nach den *minimalen* Modellen von T_0, also solchen, bei denen so wenig Instanzen von $flug(, ,)$ wie möglich gelten, kommen wir genau zu dem gewünschten Modell. Dieses stimmt mit $CWA^{\mathcal{A}}(T_0)$, wobei

$$\mathcal{A} :=_{def} \{flug(t_1, t_2, t_3) : \ t_1, t_2, t_3 \text{ sind } \mathcal{L}\text{-Terme}\}$$

überein.

Wenn wir unser Tweety-Beispiel wie folgt formalisieren:

$$\{vogel(tweety),\ \forall x\, (vogel(x) \land \neg pinguin(x) \longrightarrow fliegt(x))\},$$

und nach Modellen fragen, in denen die Extension von $pinguin$ minimal ist, kommen wir auch zu den gewünschten Modellen.

Der ab-Formalismus von McCarthy.

Unter Zuhilfenahme von neuen Prädikaten, sogenannten Abnormalitäts-Prädikaten (siehe die Beispiele 8.1.2, 8.1.3, 8.1.4, 8.1.5) wird ein gegebener Sachverhalt modelliert. Diese ab-Prädikate sollen im Normalfall nicht gelten, also nur Ausnahmefälle beschreiben:

- "$abnormal(aspekt, objekt)$": "$objekt$" ist abnormal bezüglich "$aspekt$".

Es geht also darum, abnormal-Prädikate zu minimieren. In unserem Tweety-Beispiel T_1 aus dem vorigen Abschnitt würden wir also anstatt

$$\forall x\,(vogel(x) \wedge \neg pinguin(x) \longrightarrow fliegt(x)$$

die beiden Sätze

$$\begin{array}{c}\forall x\,(vogel(x) \wedge \neg ab(x) \longrightarrow fliegt(x)) \\ \forall x\,(pinguin(x) \longrightarrow ab(x))\end{array}$$

benutzen und ab minimieren.

Angenommen, wir haben einen Sachverhalt in der Prädikatenlogik 1. Stufe formalisiert. Wir können uns also eine Theorie Σ vorstellen, die gewisse ausgezeichnete Prädikate $P_1, \ldots, P_n$ und $Z_1, \ldots, Z_m$ benutzt. Im folgenden kürzen wir solche endlichen Mengen einfach durch $\underline{P}$, $\underline{Z}$ ab: $\Sigma = \Sigma(\underline{P}; \underline{Z})$. Die $\underline{P}$ sind die zu minimierenden Prädikate (etwa die ab-Prädikate). In unserem Tweety Beispiel

$$T_0 :=_{def} \{vogel(tweety),\ \forall x\,(vogel(x) \wedge \neg pinguin(x) \longrightarrow fliegt(x))\}$$

wäre also $pinguin$ unter die P Prädikate aufzunehmen. In der ab-Formulierung:

$$\begin{array}{lll} T_1 :=_{def} & \{ & \forall x\,(vogel(x) \wedge \neg ab_1(x) \rightarrow fliegt(x)), \\ & & \forall x\,(pinguin(x) \rightarrow ab_1(x)), \\ & & \forall x\,(pinguin(x) \wedge \neg ab_2(x) \rightarrow \neg fliegt(x)), \\ & & vogel(tweety)\ \} \\ & & \\ T_2 :=_{def} T_1 \cup & \{ & pinguin(tweety)\ \} \end{array}$$

sind ab_1 und ab_2 die zu minimierenden Prädikate. Diese dürfen auf Kosten der $\underline{Z}$-Prädikate (hier $fliegt$) minimiert werden. D.h. $fliegt$ darf so variiert werden, daß ab_1, ab_2 möglichst klein werden.

Sehr wichtig ist, daß während der Minimierung andere (geeignet ausgesuchte) Prädikate variieren und wieder andere fix bleiben können. Dies führt zu folgender

Definition 8.4.1 (Semantik)
Wir definieren eine zweistellige Relation $<^{\underline{P};\underline{Z}}$ zwischen Modellen $\mathcal{W}_1$, $\mathcal{W}_2$ der zugrundeliegenden Sprache:

$$\mathcal{W}_1 <^{\underline{P};\underline{Z}} \mathcal{W}_2 \quad :\Longleftrightarrow_{def}$$

1. *$\mathcal{W}_1, \mathcal{W}_2$ besitzen dasselbe Universum.*
2. *Alle Symbole, die von $\underline{P}$, $\underline{Z}$ verschieden sind, werden gleich interpretiert.*
3. *Für jedes P_i: $P_i^{\mathcal{W}_1} \subseteq P_i^{\mathcal{W}_2}$.*

Wir schreiben dafür:

$$\Sigma \models_{circ(\underline{P};\underline{Z})} \phi \quad :\Longleftrightarrow_{def} \quad \phi \text{ gilt in allen } <^{\underline{P};\underline{Z}}\text{-minimalen Modellen von } \Sigma.$$

Die Relation $<^{\underline{P};\underline{Z}}$ ist reflexiv und transitiv, aber nicht antisymmetrisch und daher auch keine partielle Ordnung.

Was sind die minimalen Modelle von T_0? Wenn wir $pinguin$ minimieren, kommt es darauf an, ob wir $fliegt$ unter die variierenden Prädikate Z aufnehmen oder nicht. Denn sowohl

$$\mathcal{W}_1 :=_{def} \{vogel(tweety), \neg pinguin(tweety), fliegt(tweety)\}$$

als auch

$$\mathcal{W}_2 :=_{def} \{vogel(tweety), pinguin(tweety), \neg fliegt(tweety)\}$$

sind beides $<^{pinguin;\emptyset}$-minimale Modelle. Das liegt daran, daß sie im Sinne der $<^{pinguin;\emptyset}$ Relation nicht vergleichbar sind! Tatsächlich liefert die Circumscription nichts Neues, wenn keine Prädikate variieren.

Betrachten wir aber $<^{pinguin;fliegt}$-minimale Modelle, so gilt

$$\mathcal{W}_1 \ <^{pinguin;fliegt}\ \mathcal{W}_2,$$

also erhalten wir das gewünschte Resultat.

Für das zweite Beispiel betrachten wir analog T_1 bzw. T_2 wobei ab_1 und ab_2 minimiert werden, während $fliegt$ variiert. Auch hier erhalten wir die gewünschten Resultate — die $<^{ab_1,ab_2;fliegt}$-minimalen Modelle sind gerade die intendierten.

Definition 8.4.2 (2. Stufe Variable-Circumscription)
Sei $\Sigma :=_{def} \Sigma(P_1, \ldots , P_n; Z_1, \ldots , Z_m)$ *eine Formel der 2. Stufe Logik mit ausgezeichnetem (freiem) Vorkommen der Prädikate* $P_1, \ldots , P_n$ *(die* minimiert *werden sollen),* $Z_1, \ldots , Z_m$ *(die* variieren *sollen).*
Die 2. Stufe Variable-Circumscription **Circum**$(\boldsymbol{\Sigma(P_1, \ldots , P_n; Z_1, \ldots , Z_m)})$ *von*

- $P_1, \ldots , P_n$ in $\Sigma(\underline{P}; \underline{Z})$, wobei $Z_1, \ldots , Z_m$ variieren

ist der folgende zweite-Stufe-Satz:

- $\neg\exists \underline{\boldsymbol{X'}}\underline{\boldsymbol{Y'}}\ [\ \boldsymbol{\Sigma(\underline{X'}; \underline{Y'})}\ \wedge\ \forall \underline{\boldsymbol{x}}\ \underline{\boldsymbol{X'}}(\underline{\boldsymbol{x}}) < \underline{\boldsymbol{P}}(\underline{\boldsymbol{x}})\ ]$.

Die Relation "$<$" <u>*schließt*</u> *die Gleichheit aus; wir haben*

$$\underline{X'}(\underline{x}) < \underline{P}(\underline{x}) \ :\Longleftrightarrow_{def}\ [\underline{X'}(\underline{x}) \rightarrow \underline{P}(\underline{x}) \wedge \neg(\underline{P}(\underline{x}) \rightarrow \underline{X'}(\underline{x}))].$$

Da es keine "$\vdash$" Relation für die Logik zweiter Stufe gibt, können wir nur folgende Formel benutzen[3]*:*

$$\Sigma(\underline{P}; \underline{Z}) + Circum(\Sigma(P_1, \ldots , P_n; Z_1, \ldots , Z_m)) \models \psi.$$

Halten wir unser wichtigstes Ergebnis fest:

[3] $\underline{P}$ und $\underline{Z}$ werden als *Konstanten* und nicht als freie (Prädikats-) Variablen interpretiert.

> Wir haben einem Sachverhalt, der durch eine Theorie Σ beschrieben wird, eine erweiterte Theorie $Circum(\Sigma)$ zugeordnet. Die monotonen Folgerungen dieser Theorie betrachten wir als die nichtmonotonen Folgerungen aus Σ. Die Konstruktion von $Circum(\Sigma)$ aus Σ hängt von der Auswahl einzelner Prädikate in Σ ab: Einige werden minimiert (die $\underline{P}$), während andere (die $\underline{Z}$) variieren und wieder andere (die restlichen, nicht gesondert bezeichneten) fix bleiben.

Wie können wir dies implementieren? Es gibt ja keinen Kalkül für die 2. Stufe. Wir können versuchen Definition 8.4.2 in erster Stufe zu formulieren: dann haben wir einer Theorie Σ eine PL1-Theorie $Circum(\Sigma)$ zugeordnet und können automatisches Theorembeweisen benutzen.

- Solch eine Definition ist möglich, aber sie ist nur korrekt bezüglich der Semantik aus Definition 8.4.1 aber nicht vollständig.
- An Stelle der $\exists X'Y'$ in Definition 8.4.2 treten nun beliebige Formeln. Welche sind die richtigen? Dies macht das ganze sehr komplex.

Wie ist das mit der Konsistenz der Circumscription? Es kann tatsächlich vorkommen, daß es gar keine minimalen Modelle einer Theorie T gibt.

Beispiel 8.4.3 (Nichtexistenz minimaler Modelle)
Wir betrachten eine Sprache $\mathcal{L}$ mit genau einem einstelligen Prädikatssymbol P, einer Konstanten c (nullstelliges Funktionssymbol) und einem einstelligen Funktionssymbol f. Sei

$$\Sigma :=_{def} \{\exists x P(x),\, \forall x\, (P(x) \to P(f(x))\}.$$

Die Herbrand-Modelle von Σ sind offenbar genau die

$$\mathcal{M}_i :=_{def} \{P(f^j(c)) :\ i \leq j \in \mathbb{N}\},$$

mit $i \in \mathbb{N}$. Diese bilden eine unendlich absteigende Kette $\mathcal{M}_i \supset \mathcal{M}_{i+1}$ mit

$$\bigcap_{i=0}^{\omega} \mathcal{M}_i = \emptyset.$$

Die leere Struktur ist aber kein Modell von Σ!

Tabelle 8.1 zeigt alle hier betrachteten Ansätze noch einmal im Überblick.

8.5 Das Yale-Shooting-Problem

Dieses Problem stammt aus der Mitte der 80er Jahre. Nachdem man einige Formalismen (eben die Default Logik, die Circumscription und auch die Autoepistemische Logik) definiert hatte, stellte man fest, daß ein relativ einfaches Problem, das *Yale-Shooting Problem* in keinem der bekannten Formalismen in naheliegender Weise formuliert und richtig behandelt wird.

CWA	**Max. kons. TM**	**DL**	**CIRC**
$\mathcal{A} \subseteq At$	$D \subseteq Fml$	$\mathcal{D}$: $\frac{\alpha:\beta_1,\dots\beta_n}{\gamma}$	minim. Prädikate $P_1, \dots, P_n$
			varii. Prädikate $Z_1, \dots, Zm$
$W \subseteq Fml$	$W \subseteq Fml$	$W \subseteq Fml$	$W \subseteq Fml$
$(\mathcal{A}, W)$	(D, W)	$(W, \mathcal{D})$	$W = W(\underline{P}; \underline{Z})$
$CWA^{\mathcal{A}}(W)$	$\mathcal{E}_i$	Extensionen E_i	MinMod($W(\underline{P}; \underline{Z})$)

Tabelle 8.1: Nichtmonotone Ansätze im Überblick

Beispiel 8.5.1 (Yale-Shooting-Problem)
Die Prosa-Formulierung lautet:

> *Fred lebt zum Zeitpunkt T_0. Man lädt ein Gewehr bis T_1, man wartet bis T_2. Schließlich schießt man auf Fred und erreicht T_3. Die Axiome sagen aus, daß alle Fakten persistent in der Zeit sind mit der einzigen Ausnahme: Wenn man auf Fred mit einem geladenen Gewehr schießt, ist er nicht länger am Leben.*

Die folgende Formalisierung ist offensichtlich:

- "$ab(f, e, t)$" steht für "es ist abnormal, daß Zustand f zur Zeit $t + 1$ gilt, wenn die Aktion e zur Zeit t stattfindet ",
- "$holds(f, t)$" steht für "Zustand f gilt zur Zeit t"
- – Zustände sind: $LEBT$ und $GELADEN$.
  – Aktionen sind: $Laden$, $Warten$ und $Schießen$.

1. (a) $T_1 :=_{def}$ result($Laden$,T_0),
   (b) $T_2 :=_{def}$ result($Warten$,T_1),
   (c) $T_3 :=_{def}$ result($Schießen$,T_2),
2. holds($LEBT$,T_0),
3. ∀t holds($GELADEN$,result($Laden$,t)),
4. ∀t holds($GELADEN$,t) →
   ¬holds($LEBT$,result($Schießen$,t)) ∧ ab($LEBT$,$Schießen$,t),
5. ∀fet ((holds(f,t) ∧¬ab(f,e,t))→ holds(f,result(e,t))).

Wenn wir die Circumscription dieser Theorie betrachten und $ab(f, e, t)$ minimieren, so existiert immer ein minimales Modell in dem gilt:

1. $ab(GELADEN, Warten, T_1)$ (das Gewehr entlud sich),
2. $\neg holds(GELADEN, T_2)$ und
3. $holds(LEBT, T_3)$.

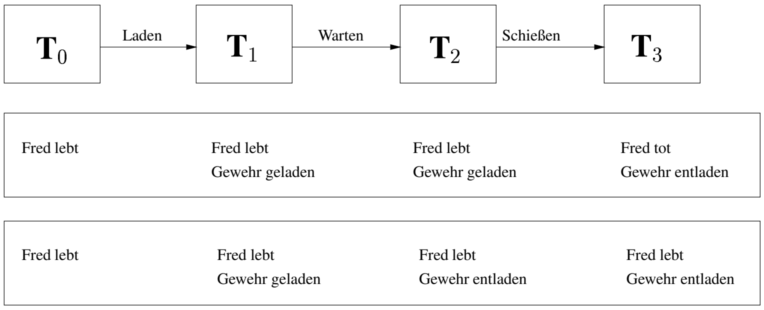


Abbildung 8.2: Das Yale-Shooting-Problem

Das eigentlich intendierte Modell, nämlich

1. $ab(LEBT, Schießen, T_2)$,
2. $holds(GELADEN, T_2)$ und
3. $\neg holds(LEBT, T_3)$.

ist mit dem ersten nicht vergleichbar, beide sind minimal!

Dasselbe Problem tritt auf, wenn man die Default Logik benutzt. Man kann dann die obige Formulierung beibehalten (natürlich muß man alles in Aussagenlogik formulieren, da wir DL ja nur für $\mathcal{L}_{AL}$ eingeführt haben) und muß nur noch die Defaults

- für alle f, e, t: $\frac{\emptyset : \neg ab(f,e,t)}{\neg ab(f,e,t)}$

hinzunehmen. Auch hier erhält man allerdings die Existenz von zwei Extensionen (den beiden gerade beschriebenen Modellen) anstatt einer einzigen.

Das Problem ist daher,

1. im voraus die gewünschten Extensionen zu charakterisieren, oder äquivalent dazu, das richtige *minimale*[4] Modell auszuwählen,
2. eine uniforme Methode anzugeben, wie man Defaults formulieren muß, damit die ungewollten Extensionen nicht auftreten.

8.6 Abstrakte Eigenschaften

Während einige nichtmonotone Logiken innerhalb der Prädikatenlogik erster Stufe formuliert sind (CIRC, CWA, Poole's Ansatz) gehen andere Ansätze (DL) darüber hinaus und führen neue Inferenzregeln auf Metaebene ein. Allen diesen Ansätzen und überhaupt jedem logischen System ist jedoch folgendes gemeinsam

[4] wobei ab zu minimieren ist

	endlicher Ansatz	**unendlicher Ansatz**
Reflexivität	$\alpha \in A \implies A \mid\!\sim \alpha$	$A \subseteq C(A)$
Transitivität	$A, \beta \mid\!\sim \alpha,\ A \mid\!\sim \beta \implies A \mid\!\sim \alpha$	$C(A) = C(C(A))$
Monotonie	$A \mid\!\sim \alpha \implies A, \beta \mid\!\sim \alpha$	$A \subseteq B \implies C(A) \subseteq C(B)$
	A **endlich!**	A, B **beliebig.**

Tabelle 8.2: Endlicher ($\mid\!\sim$) vs. unendlicher (C) Ansatz.

> *Jedes nichtmonotone System (DL, CIRC, CWA, ...) definiert in kanonischer Weise eine Relation*
>
> $$\mid\!\sim\ \subseteq Pot(Fml_{\mathcal{L}}) \times Pot(Fml_{\mathcal{L}}) : \ \Psi \mid\!\sim \Phi.$$

Wir sagen dann: Φ ist aus Ψ nichtmonoton herleitbar.

Axiomatische Ansätze: Abschlüsse von $\mathcal{K}$

Wir wollen zunächst die klassischen Tarski-Axiome betrachten. Wir haben zwei verschiedene Möglichkeiten, sie auszudrücken: Wir können eine *finite* Relation " $\mid\!\sim$ ", also eine Relation zwischen *endlichen* Mengen von Formeln, oder eine *unendliche* Relation "C", also eine beliebige Funktion von $Pot(Fml_{\mathcal{L}})$ nach $Pot(Fml_{\mathcal{L}})$, betrachten. Im klassischen, also monotonen Fall sind beide Vorgehensweisen äquivalent: Das ist genau die Aussage des Kompaktheitssatzes.

Der allgemeine Rahmen in diesem Kapitel ist die Aussagenlogik. Wir betrachten also im folgenden nur aussagenlogische Formeln α, β. Außerdem formulieren wir Defaults direkt unter Zuhilfenahme von $\mid\!\sim$:

> *Wir stellen uns eine endliche Default Basis $\mathcal{K}$ vor, die aus Paaren $\alpha \mid\!\sim \beta$ besteht. Wir lassen auch sicheres Wissen in Form einer Theorie Σ_s zu.*

Etwas genauer ausgedrückt, liegt dem Ansatz folgende Vorgehensweise zugrunde:

Default-Basis $\mathcal{K}$: Die vorgegebene Situation wird durch eine (endliche) Menge von *bedingten Behauptungen* (*conditional assertions*) der Form "$\alpha \mid\!\sim \beta$", wobei α, β aussagenlogische Ausdrücke sind, dargestellt. Diese Menge bildet die "Default-Basis" $\mathcal{K}$. Hierbei ist die Aussage "$\alpha \mid\!\sim \beta$" zu lesen als "*normalerweise (typischerweise, in der Mehrzahl aller Fälle, falls nichts dagegen spricht)* folgt aus α die Aussage β".

Abschluß von $\mathcal{K}$: Ausgehend von der Default-Basis $\mathcal{K}$ wird versucht, zusätzliche bedingte Behauptungen zu erschließen. Aus der Menge $\mathcal{K}$ von Aussagen der Form "$\alpha \mid\!\sim \beta$" werden also wieder Aussagen der Form "$\alpha \mid\!\sim \beta$" abgeleitet. Die Menge aller aus $\mathcal{K}$ abgeleiteten bedingten Aussagen heißt ein Abschluß von $\mathcal{K}$. Wir werden im folgenden verschiedene Abschlüsse von $\mathcal{K}$ betrachten.

Anfrage ?-$c \mid\!\sim d$: Schließlich können wir eine Anfrage an $\mathcal{K}$ stellen. Diese Anfrage muß auch wieder von der Form "$\gamma \mid\!\sim \delta$" sein, und sie wird mit "ja" oder "nein" beantwortet, je nachdem ob "$\gamma \mid\!\sim \delta$" im entsprechend gewählten Abschluß liegt.

Definition 8.6.1 (Bedingte Behauptungen: $\alpha \mid\!\sim \beta$)
Seien α und β Formeln. Das Paar $\alpha \mid\!\sim \beta$ heißt eine bedingte Behauptung.
$\alpha \mid\!\sim \beta$ steht für "aus α folgt nichtmonoton β". Es bedeutet, daß wenn α wahr ist, man normalerweise *bereit ist, β als wahr anzusehen.*

Für unser Tweety-Beispiel ergibt sich etwa:

$$\Sigma_s :=_{def} \{pinguin \to vogel\}$$
$$\mathcal{K} :=_{def} \{vogel \mid\!\sim fliegt, pinguin \mid\!\sim \neg fliegt\}$$

Die Anfragen die wir stellen können sind z.B.

$$pinguin \wedge vogel \mid\!\sim \neg fliegt \text{ bzw. } fliegt \mid\!\sim \neg pinguin.$$

Man beachte, daß jeder Vogel fliegt, bzw. jeder Pinguin nicht fliegt (denn die entsprechende Aussage ist ja explizit in der Basis enthalten). Interessant ist es nun zu fragen, ob auch unter der Annahme, daß ein Vogel und gleichzeitig ein Pinguin vorliegt, $\neg fliegt$ immer noch gilt. Im Unterschied zur Default Logik können wir nun auch fragen, ob *etwas das fliegt, normalerweise kein Pinguin ist*. In DL ist solch eine Aussage ein Default und die Ableitbarkeit von Defaults ist nicht definiert.

Der kumulative Abschluß $\mathcal{K}^{cum}$

Der kumulative und der präferentielle Abschluß einer Datenbasis ergeben sich unter Benutzung von Inferenzregeln einer bestimmten Form, Horn-Regeln: *Wenn A und B dann C*. Dabei sind A, B und C *positive* Aussagen, genauer *bedingte Behauptungen*.

Das kumulative System, $\mathcal{C}$ *-System* basiert auf dem Vorschlag Gabbays. Es ist das schwächste System, das Kraus, Lehmann und Magidor vorstellen. Allgemein gilt, daß Regeln, die in schwächeren Systemen abgeleitet werden können, auch in stärkeren Systemen ableitbar sind.

Definition 8.6.2 (System $\mathcal{C}$, $\mathcal{K}^{cum}$)
Man sagt eine Konsequenzrelation $\mid\!\sim$ ist kumulativ, wenn sie unter den Ableitungsregeln Reflexivität, Left Logical Equivalence, Right Weakening, Cut *und* Vorsichtige Monotonie *abgeschlossen ist.*

$\alpha \mid\!\sim \beta$ liegt in $\mathcal{K}^{cum}$ $:\Longleftrightarrow_{def}$ $\alpha \mid\!\sim \beta$ ist aus $\mathcal{K}$ mit den Regeln aus $\mathcal{C}$ ableitbar.

Die oben genannten Regeln werden im folgenden genauer betrachtet.

$$\frac{}{\alpha \mid\!\sim \alpha} \qquad (\textit{Reflexivität})$$

Diese Inferenzregel mit leerer Prämissenmenge kann man auch als Axiomenschema deuten. Axiome der Form $\alpha \mathrel{|\!\sim} \alpha$ sind immer wahr.

$$\frac{\Sigma_s \models \alpha \Leftrightarrow \beta,\ \ \alpha \mathrel{|\!\sim} \gamma}{\beta \mathrel{|\!\sim} \gamma} \qquad (\textit{Left Logical Equivalence})$$

Left Logical Equivalence drückt die Forderung aus, daß logisch äquivalente Formeln genau die gleichen Konsequenzen haben.

$$\frac{\Sigma_s \models \alpha \rightarrow \beta,\ \ \gamma \mathrel{|\!\sim} \alpha}{\gamma \mathrel{|\!\sim} \beta} \qquad (\textit{Right Weakening})$$

Right Weakening drückt aus, daß alles, was aus möglichen Konsequenzen logisch folgt, ebenfalls eine Konsequenz ist. Aus *Reflexivität* und *Right Weakening* folgt, daß $\alpha \mathrel{|\!\sim} \beta$ wenn $\alpha \models \beta$.

$$\frac{\alpha \wedge \beta \mathrel{|\!\sim} \gamma,\ \ \alpha \mathrel{|\!\sim} \beta}{\alpha \mathrel{|\!\sim} \gamma} \qquad (\textit{Cut})$$

Cut drückt aus, daß man bei der Suche nach einer Konklusion zuerst eine Hypothese zu den Fakten hinzufügen kann, um die Konklusion aus diesen so erweiterten Fakten zu erhalten. Dann muß man aber die hinzugefügte Hypothese aus den ursprünglichen Fakten herleiten.

$$\frac{\alpha \mathrel{|\!\sim} \beta,\ \ \alpha \mathrel{|\!\sim} \gamma}{\alpha \wedge \beta \mathrel{|\!\sim} \gamma} \qquad (\textit{Vorsichtige Monotonie})$$

Vorsichtige Monotonie drückt aus, daß das Hinzufügen eines neuen Faktums, das schon bewiesen wurde, die bisherigen Konklusionen bestehen läßt.

Lemma 8.6.3 (Kumulativität)
Die Regeln Cut *und* Vorsichtige Monotonie *können zusammen auch wie folgt ausgedrückt werden: Wenn $\alpha \mathrel{|\!\sim} \beta$ gilt, dann stimmen die Konsequenzen von α und $\alpha \wedge \beta$ überein:*

$$\textit{Für alle } \gamma \textit{ gilt: } \alpha \mathrel{|\!\sim} \gamma \ \ \underline{\text{gdw}}\ \ (\alpha \wedge \beta) \mathrel{|\!\sim} \gamma.$$

Ein stärkeres System ist das $\mathcal{P}$-*System*:

Definition 8.6.4 (System $\mathcal{P}$, $\mathcal{K}^{pref}$)
Das System $\mathcal{P}$ besteht aus allen Regeln von $\mathcal{C}$ und zusätzlich der folgenden Regel :

$$\frac{\alpha \mathrel{|\!\sim} \gamma,\ \ \beta \mathrel{|\!\sim} \gamma}{\alpha \vee \beta \mathrel{|\!\sim} \gamma} \qquad (\text{Oder})$$

Eine Konsequenzrelation, die alle Regeln von $\mathcal{P}$ erfüllt heißt präferentiell.

$$\alpha \mathrel{|\!\sim} \beta \textit{ liegt in } \mathcal{K}^{pref} \ \ :\Longleftrightarrow_{def}\ \ \alpha \mathrel{|\!\sim} \beta \textit{ ist aus } \mathcal{K} \textit{ mit den Regeln aus } \mathcal{P} \textit{ ableitbar.}$$

Oder besagt, daß eine Formel, die sowohl Konsequenz einer Formel α als auch Konsequenz einer Formel β ist, auch eine Konsequenz der Disjunktion $\alpha \vee \beta$ sein soll.

Einige abgeleitete Regeln im $\mathcal{C}$-System

Wir zeigen in diesem Abschnitt, daß und wie man einige nützliche Regeln aus dem kumulativen bzw. dem präferentiellen System herleiten kann.
Alle folgenden Regeln können in $\mathcal{C}$ hergeleitet werden

$$\frac{\alpha \mid\sim \beta, \;\; \alpha \mid\sim \gamma}{\alpha \mid\sim \beta \wedge \gamma} \qquad (\textit{And})$$

$$\frac{\alpha \mid\sim \beta \rightarrow \gamma, \;\; \alpha \mid\sim \beta}{\alpha \mid\sim \gamma} \qquad (\textit{MPC})$$

$$\frac{\alpha \mid\sim \beta, \; \beta \mid\sim \alpha, \; \alpha \mid\sim \gamma}{\beta \mid\sim \gamma} \qquad (\textit{Equivalence})$$

Die Regeln zur Definition des kumulativen und präferentiellen Abschlusses waren allesamt Horn-Regeln. Das folgende Beispiel zeigt, daß auch nicht-Horn Regeln interessant sind:

Beispiel 8.6.5 (Irrelevanz)
Seien p, q, r verschiedene aussagenlogische Variable und sei $\mathcal{K} :=_{def} \{p \mid\sim q\}$. Dann ist

$$(p \wedge r) \mid\sim q \;\; \notin \mathcal{K}^{pref}.$$

Dies widerspricht aber unserer Intuition, da r überhaupt nichts mit p oder q zu tun hat.

Solch ein Verhalten kann ausgeschlossen werden, wenn wir etwa die folgende Regel voraussetzen:

$$\frac{\alpha \not\mid\sim \neg\beta, \;\; \alpha \mid\sim \gamma}{\alpha \wedge \beta \mid\sim \gamma} \qquad (\textit{Rationalität})$$

Man beachte, daß diese Regel eine Verstärkung der vorsichtigen Monotonie ist: $\alpha \mid\sim \beta$ wurde gerade durch $\alpha \not\mid\sim \neg\beta$ ersetzt.

Weitere Varianten der Rationalität sind

$$\frac{\alpha \not\mid\sim \gamma, \;\; \beta \not\mid\sim \gamma}{\alpha \vee \beta \not\mid\sim \gamma} \qquad (\textit{Disjunktive Rationalität})$$

und

$$\frac{(\alpha \wedge \gamma) \not\mid\sim \beta, \;\; (\alpha \wedge \neg\gamma) \not\mid\sim \beta}{\alpha \not\mid\sim \beta} \qquad (\textit{Negations–Rationalität}).$$

Es gilt folgendes leicht zu verifizierendes Lemma

Lemma 8.6.6 (Varianten der Rationalität)
Bezüglich des Systems $\mathcal{C}$ gilt:

$$\textit{Rationalität} \implies \textit{Disjunktive Rationalität} \implies \textit{Negations–Rationalität}\,.$$

Offensichtlich ist es nicht möglich, analog zu $\mathcal{K}^{cum}$ oder $\mathcal{K}^{pref}$ in kanonischer Weise so etwas wie einen rationalen Abschluß zu definieren. Zwar können wir die Rationalität anwenden, aber es kann ja passieren, daß wir bedingte Behauptungen hinzunehmen müssen, die vorherige Anwendungen der Rationalitätsregel wieder zunichte machen.

8.7 Eigenschaften von DL und CIRC

Einige der im System $\mathcal{C}$ benutzten Regeln sind unkritisch, da sie von jedem nichtmonotonen System erfüllt werden. In der Tat fällt es schwer, sich Situationen vorzustellen, in denen die folgenden Eigenschaften nicht gelten:

- *Reflexivität*,
- *Left Logical Equivalence*,
- *Right Weakening*,
- *Cut*.

Interessant sind daher die *Kumulativität*, bzw. die *Oder*-Regel.

Theorem 8.7.1 (Eigenschaften von CWA)
a) *CWA erfüllt* Reflexivität, L. L. Equiv., Right Weak. *und* Cut.
b) *Ist $CWA(\Phi)$ konsistent, so gilt:*

$$\Phi \subseteq \Psi \subseteq CWA(\Phi) \implies CWA(\Phi) \subseteq CWA(\Psi).$$

Dies ist gerade die vorsichtige Monotonie*, also gelten alle Regeln aus $\mathcal{C}$, wenn CWA konsistent ist.*

Lemma 8.7.2 (Eigenschaften minimaler Modelle)
Gilt $\emptyset \neq MinMOD(\Sigma \cup \{\alpha\}) \models \beta$ so gilt auch

$$MinMOD(\Sigma \cup \{\alpha\}) = MinMOD(\Sigma \cup \{\alpha, \beta\})$$

Theorem 8.7.3 (CIRC erfüllt $\mathcal{P}$, falls konsistent)
a) *CIRC erfüllt* Refl., L. L. Equiv., R. Weak., Cut *und* Oder.
b) *Ist $CIRC(\Phi)$ konsistent, so gilt:*

$$\Phi \subseteq \Psi \subseteq CIRC(\Phi) \implies CIRC(\Phi) \subseteq CIRC(\Psi).$$

Dies ist gerade die vorsichtige Monotonie*, also gelten alle Regeln aus $\mathcal{P}$, wenn CIRC konsistent ist.*

8.8 Aufgaben

Aufgabe 8.1 (Closed World Assumption)

1. *Betrachten Sie einen Teil des Fahrplans der Deutschen Bahn (etwa Zugverbindungen Koblenz–Frankfurt/Main). Ist CWA*$^{At}(T)$ *konsistent? Warum?*

2. *Geben Sie eine möglichst einfache Theorie* T *und Atome* $A, B \in At$ *an, für die CWA*$^{\{A,B\}}(T)$ *inkonsistent ist.*

Aufgabe 8.2 (Maximal konsistente Teilmengen)
Betrachten Sie den Ansatz für maximal konsistente Teilmengen (Definition 8.7) einer Theorie (D, W).

1. *Kann ein Abschluß* $\mathcal{E}$ *inkonsistent sein? Geben Sie eine notwendige und hinreichende Bedingung an!*

2. *Angenommen,* D *enthält genau* n *Sätze. Wieviele Abschlüsse gibt es dann maximal?*

3. *Zeigen Sie, daß die Abschlüsse zu* (D, W) *orthogonal zueinander sind, d.h.* $\mathcal{E}_i \cup \mathcal{E}_j$ *ist inkonsistent für* $i \neq j$.

4. *Folgt aus* $(\bigcap \mathcal{E}) \cap D = \emptyset$, *daß* $\bigcap \mathcal{E} = Cn(W)$*?*

Aufgabe 8.3 (Disjunktion und Default-Theorie)
Formalisieren Sie Beispiel 8.4 („Weintrinker“) in Default-Logik und berechnen Sie die Extensionen.

Aufgabe 8.4 (Disjunktion mit Blockade)

1. *Formalisieren Sie Beispiel 8.5 („Armbenutzbarkeit“) in Default-Logik und berechnen Sie die Extensionen.*

2. *Welche Formalisierung ergibt sich, wenn Sie die ersten beiden Aussagen des Beispiels durch „Ein Arm ist normalerweise benutzbar, es sei denn, er ist gebrochen.“ ersetzen?*

3. *Diskutieren Sie den Unterschied beider Formalisierungen. Was passiert, falls Sie als Faktum in* W *„Der rechte Arm ist gebrochen.“ hinzunehmen? Wie unterscheidet sich die Extensionen in (a) und (b)?*

Aufgabe 8.5 (Konsistenz in der Default-Logik)

1. *Ist die Default-Theorie* $(\emptyset, \{\frac{\emptyset:\alpha}{\alpha}, \frac{\emptyset:\neg\alpha}{\neg\alpha}\})$ *konsistent?*

2. *Besitzt die Default-Theorie* $(\{\alpha\}, \{\frac{\alpha:\emptyset}{\neg\alpha}\})$ *eine Extension?*

3. *Default-Inferenzregeln haben die allgemeine Form* $\frac{\alpha:\beta_1,\ldots,\beta_n}{\gamma}$ *mit* $n \geq 1$. *Können wir uns auf den Fall* $n = 1$ *beschränken, ohne ausdrucksschwächer zu werden?*

Aufgabe 8.6 (Schwache Extensionen)
Die Definition einer Extension (Definition 8.18) ist aufwendig und kompliziert. In Definition 8.19 wird der viel einfachere Begriff „abgeschlossen unter $(W, \mathcal{D})$" definiert.

Wir sagen, daß $E \subseteq Fml$ eine schwache Extension von $(W, \mathcal{D})$ ist, falls E abgeschlossen ist unter $(W, \mathcal{D})$ und es keine kleinere Menge $E' \subsetneqq E$ mit dieser Eigenschaft gibt.

1. *Zeigen Sie, warum der Zusatz „und es gibt keine kleinere Menge … " nötig und intuitiv sinnvoll ist.*
2. *Zeigen Sie, daß jede Extension auch schwache Extension ist.*
3. *Betrachten Sie $W := \{\alpha \rightarrow \beta\}$ und $\mathcal{D} := \{\frac{\emptyset : \neg\alpha}{\alpha}, \frac{\beta : \emptyset}{\alpha}\}$ und berechnen Sie alle Extensionen und schwachen Extensionen. Gilt die Umkehrung von (b) ?*

Aufgabe 8.7 (Circumscription)

1. *Zeigen Sie, daß $<^{\underline{P};\underline{Z}}$ keine partielle Ordnung ist!*
2. *Was ist, falls $\underline{P} = \emptyset$ ist? Zu betrachten ist also $<^{\emptyset;\underline{Z}}$.*

Aufgabe 8.8 (Minimale Modelle)
Betrachten Sie die folgende Theorie!

$$\begin{array}{l} deutsch(x) \wedge \neg ab(x) \rightarrow Biertrinker(x) \\ deutsch(Peter) \\ \neg Biertrinker(Julius) \end{array}$$

Gilt in allen minimalen Modellen – minimal bezüglich $<^{\{ab\};\{deutsch, Biertrinker\}}$ – die Aussage $Biertrinker(Peter)$?

Aufgabe 8.9 (Weintrinker)

1. *Formalisieren Sie Beispiel 8.4 („Weintrinker")!*
2. *Bestimmen Sie alle minimalen Modelle!*

Aufgabe 8.10 (Circumscription und Rationalität)
Zeigen Sie, daß die Circumscription nicht die Rationalität erfüllt!

Hinweis zur Vereinfachung: *Sie können hierbei $\underline{Z} = \emptyset$ und $\underline{P}$ als die Menge aller auftretenden Prädikate ansetzen.*

9 Lernen aus Beobachtungen

Bisher: Die Intelligenz eines Agenten steckt in seinem Programm, sie ist fest verdrahtet.

Jetzt: Der Agent soll autonomer **werden**, durch eigene **Beobachtungen** (Erfahrungen) soll er seine Umgebung verstehen lernen.

Wichtig: Falls der Bereich in dem er arbeitet nur unvollständig beschrieben werden kann.

9.1 Grundlagen

Performance-Element: Beobachtet und wählt Aktionen aus. War bisher der gesamte Agent.

Kritik: Beobachtet das Resultat der Aktionen und bewertet sie. Bewertet anhand eines Standards **von aussen**!! (Warum? sonst wählt er seinen eigenen Standard so niedrig, dass er ihn immer erfüllt!)

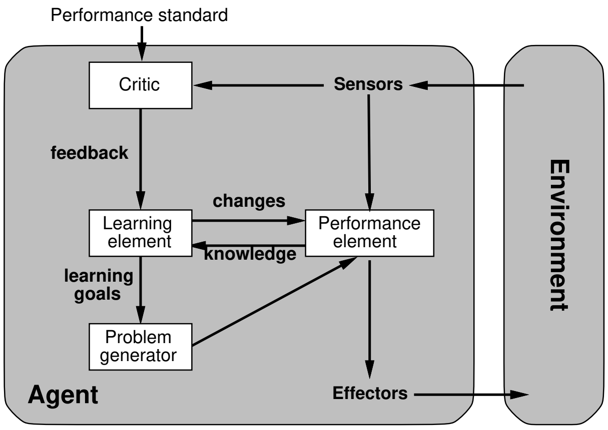


Abbildung 9.1: Ein Lern-Agent

Lern-Element: Modifiziert das Performance-Element indem es die Kritik berücksichtigt (und den inneren Aufbau des Perf.-Elem.). Außerdem stellt es neue Ziele auf (um die Effekte von Aktionen besser zu verstehen).

Problem Generator: Er schlägt vor Aktionen auszuführen, um die Goals des Learning-Elementes zu befriedigen. Dies müssen nicht immer die "besten" Aktionen sein (i. S. d. Perf.-Elem.): aber sie sollen **informativ** sein und neues Wissen über die Welt liefern.

Wovon hängt Lernen (also der Entwurf des Lern-Elementes) eigentlich ab?

1. Welche Komponenten des Performance-Elementes sollen verbessert werden?
2. Wie sind diese Komponenten repräsentiert?
   ($\rightsquigarrow$ Kapitel 9, Abschnitte 9.2 und 9.3
3. Welches feedback gibt es?
   - **Überwachtes L.:** Das Ausführen einer inkorrekten Aktion erzeugt als feedback die "richtige" Lösung (z.B. wie stark aufs Bremspedal treten). *Fahrlehrer*.
   - **Reinforcement L.:** Nur das Resultat wird wahrgenommen. Kritik sagt, ob es gut oder schlecht, aber nicht was richtig gewesen wäre.
   - **Nicht-Überwachtes L.:** Gar keine Hinweise über korrekte Aktionen.
4. Welche a-priori-Informationen gibt es?
   (Oft gibt es nützliches Hintergrundwissen.)

Komponenten des Performance Elementes

1. Abbildung von Bedingungen des aktuellen Zustandes in die Menge der Aktionen.
2. Ableiten relevanter Eigenschaften der Welt aus den Wahrnehmungen.
3. Informationen darüber, wie sich die Welt entwickelt.
4. Informationen über die Folgen von Aktionen.
5. Bewertungsfunktion über die Zustände der Welt.
6. Bewertungsfunktion über die Güte einzelner Aktionen in einem Zustand.
7. Beschreibung von Zustandsklassen, die die Nutzenfunktion des Agenten maximieren.

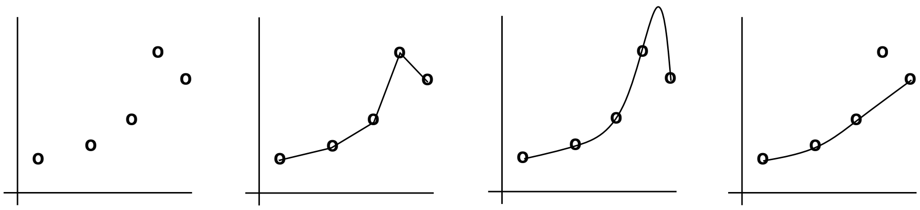
Alle diese Komponenten sind, mathematisch gesehen, *Funktionen*.

Lernen heißt diese Funktionen zu repräsentieren.

Beim Induktiven Lernen sind Paare (x, y) gegeben und die Aufgabe besteht darin, die Funktion f mit $f(x) = y$ zu bestimmen.

Beispiel 9.1.1 (Zahlenfolgen fortsetzen)
Wie lautet die nächste Zahl:

- *3, 5, 7, ?*
- *3, 5, 17, 257, ?*



(a) (b) (c) (d)

Einfacher Reflex-Lern Agent:

global *examples* ← {}

---

function REFLEX-PERFORMANCE-ELEMENT(*percept*) **returns** an action

  if (*percept*, *a*) **in** *examples* **then return** *a*
  else
    h ← INDUCE(*examples*)
    return *h*(*percept*)

---

procedure REFLEX-LEARNING-ELEMENT(*percept*, *action*)
  inputs: *percept*, feedback percept
      action, feedback action

  examples ← *examples* ∪ {(*percept*,*action*)}

Beispiel 9.1.2 (Verifizieren und falsifizieren)
Man betrachte eine Menge von Karten. Jede Karte enthält auf einer Seite einen Buchstaben und auf der anderen Seite eine Zahl. Nachdem man sich einige Karten angeschaut hat kommt man zu der Hypothese

Ist auf einer Seite einer Karte ein Vokal, so ist auf der anderen Seite eine gerade Zahl.

Es liegen nun folgende Karten auf dem Tisch:

A T 4 7.

Sie dürfen höchstens 2 Karten herumdrehen und sollen die Hypothese prüfen. Welche Karte drehen Sie herum?

Es müssen die **A** und die **7** herumgedreht werden. Die **A** weil damit die Hypothese direkt überprüft werden kann, die **7** weil das die einzige Möglichkeit ist, die Hypothese auch zu falsifizieren. Dies wird oft nicht gemacht: man sucht nur nach Bestätigung, aber man versucht nicht die Hypothese zu widerlegen.

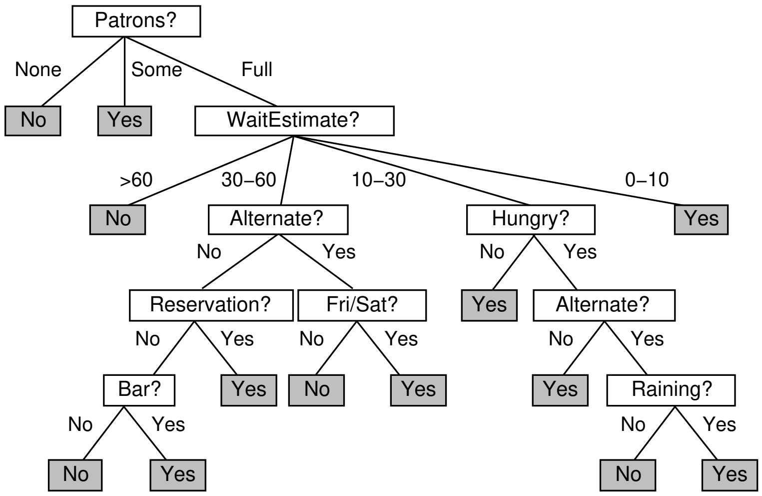


Abbildung 9.2: Ein Entscheidungsbaum

9.2 Entscheidungsbäume

Entscheidungsbäume sind Repräsentanten von booleschen Funktionen. Ein Beispiel: Man geht essen und erreicht ein Restaurant. Soll man auf einen freien Tisch warten oder weitergehen?

Entscheidungsbaum = Konjunktion von Implikationen[1]

$$\forall r\ (Patrons(r, Full) \wedge Wait\_estimate(r, 10-30) \wedge \neg Hungry(r)) \longrightarrow Will\_Wait(r)$$

Dies ist zwar in PL1 geschrieben, aber Entscheidungsbäume betreffen immer nur einzelne Objekte (das r oben), dessen Eigenschaften abgefragt werden. Also geht es um Aussagenlogik.

Kann jede boolesche Funktion als Entscheidungsbaum geschrieben werden? Ja: Zeile der Wahrheitstabelle entspricht Pfad im Baum.

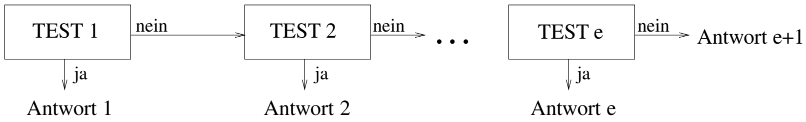
E-Bäume können sehr viel kleiner sein! Aber es gibt auch bool. Funktionen, die nur durch exponentiell große Bäume dargestellt werden können:

$$\text{Paritätsfunktion: } par(x_1, \ldots, x_n) := \begin{cases} 1, & \text{falls } \sum_{i=1}^{n} x_i \text{ ist gerade} \\ 0, & \text{sonst} \end{cases}$$

Eine Variante der E-Bäume sind

[1] Implikation = Jeder Pfad zu einem Blatt.

Beispiel 9.2.1 (Entscheidungslisten)
Alle Attribute seien booleschwertig. Eine Entscheidungsliste hat die Form



wobei $Antwort_i \in \{Ja, NEIN\}$ *und* $Test_i$ *eine Konjunktion von (eventuell negierten) Attributen ist. (*$\rightsquigarrow$ *Aufgaben 9.1 und 9.2).*

Sei $k - DL(n)$ *die Menge der booleschen Funktionen in* n*-Attributen, die durch E-Listen mit höchstens* k *Abfragen in jedem Test dargestellt werden können.*

Offensichtlich gilt:

- $n - DL(n)$ = Menge aller booleschen Funktionen,
- $card(n - DL(n)) = 2^{2^n}$.

Daß gilt $card(n - DL(n)) = 2^{2^n}$, sieht man anhand der Wahrheitstafel: für eine spezielle Funktion muss man 2^n Zeilen festlegen um sie zu definieren. In jeder Zeile gibt es 2 Möglichkeiten (0 oder 1).

Wie lernt man denn nun E-Bäume?

Example	Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	*Yes*
X_2	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	*No*
X_3	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	*Yes*
X_4	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*10–30*	*Yes*
X_5	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	*No*
X_6	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	*Yes*
X_7	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	*No*
X_8	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	*Yes*
X_9	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	*No*
X_{10}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	*No*
X_{11}	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	*No*
X_{12}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	*Yes*

Die Menge der Beispiele aus denen gelernt werden soll heißt *Trainingsmenge*. Beispiele können positiv (Attribut trifft zu) oder negativ (trifft nicht zu) sein.

Triviale Lösung des Lernens: Die Pfade im Baum sind genau die Beispiele. *Nachteil:* Neue Fälle können nicht behandelt werden.

Nimm den einfachsten Baum (bzw. den allgemeinsten) der mit den Beispielen verträglich ist.

Ockham's Messer: *Entia non sunt multiplicanda praeter necessitatem.*

Beispiel 9.2.2 (Raten Sie!)
Ein Computerprogramm kodiert Zahlentripel gemäß einer bestimmten Vorschrift. Sie sollen diese Vorschrift herausfinden.

Sie geben dem Programm Tripel (x_1, x_2, x_3) *Ihrer Wahl vor (*$X_i \in \mathbb{N}$*) und bekommen als Antwort "Ja" oder "Nein".*

Zur Vereinfachung gibt Ihnen das Programm schon zu Anfang bekannt, daß folgende Tripel in der Menge enthalten sind:

$$(4, 6, 8),\ (6, 8, 12),\ (20, 22, 40)$$

Stellen Sie weitere Anfragen (etwa 10) und versuchen Sie die Eigenschaft herauszufinden.

Das Problem ist, daß man nur Tripel der Form

$$(x, x + 2, 2x)$$

ausprobiert, also nur positive Beispiele versucht. Man ist zu speziell und versucht nicht, die einmal gefasste Hypothese kritisch zu hinterfragen, also sie zu falsifizieren. Man fragt nach einer speziellen Vorschrift, nicht nach der allgemeinsten, wie z.B. $\{(x, y, z) : x < y < z\}$.

Idee des Lern-Algorithmus

Ziel: Möglichst kleiner Baum. Teste zunächst die *wichtigsten* Attribute (um eine möglichst schnelle Klassifikation zu bekommen). *Dies formalisieren wir später, unter Rückgriff auf etwas Informationstheorie.*. Fahre dann rekursiv so fort: d.h. es gibt immer weniger Beispiele und Attribute. Wir machen folgende Fallunterscheidungen:

1. *Es gibt positive und negative Beispiele.*
   Wähle das beste Attribut.

2. *Es gibt nur positive oder nur negative Beispiele.*
   Fertig: Eindeutige Antwort ist möglich.

3. *Es gibt keine Beispiele mehr.*
   Dann muss man einen Default-Wert nehmen, etwa die Mehrheit der Beispiele des Elter-Knotens.

4. *Es gibt sowohl positive als auch negative Beispiele, aber es gibt keine Attribute mehr.*
   Dann reichen die zugrundeliegenden Attribute eben nicht aus, um zu einer Entscheidung zu kommen. Es ist einfach nicht genügend Information gegeben.

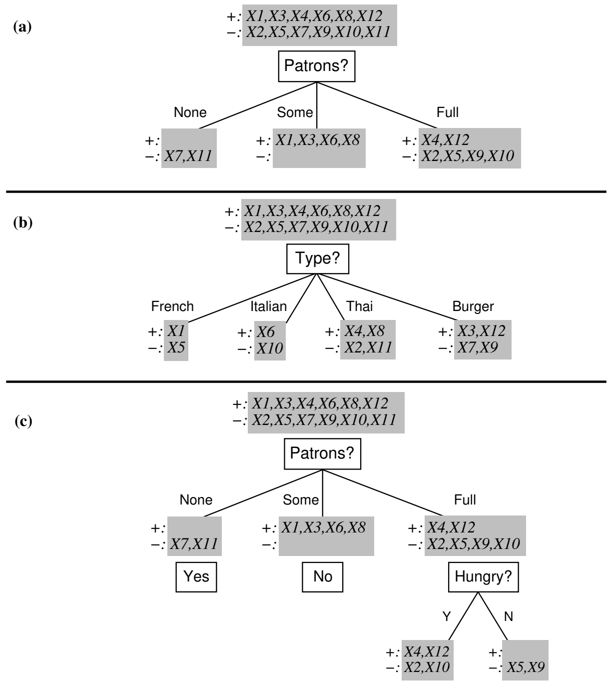
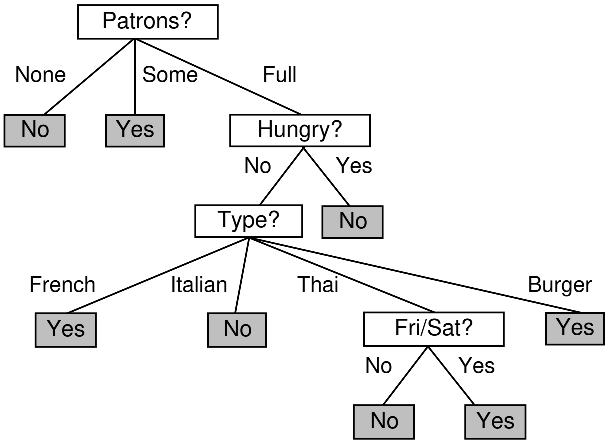


Abbildung 9.3: Auswahl der Attribute

function DECISION-TREE-LEARNING(*examples, attributes, default*) **returns** a decision tree
inputs: *examples*, set of examples
attributes, set of attributes
default, default value for the goal predicate

if *examples* is empty **then return** *default*
else if all *examples* have the same classification **then return** the classification
else if *attributes* is empty **then return** MAJORITY-VALUE(*examples*)
else
best ← CHOOSE-ATTRIBUTE(*attributes*, *examples*)
tree ← a new decision tree with root test *best*
for each value v_i of *best* **do**
$examples_i$ ← {elements of *examples* with $best\ =\ v_i$}
subtree ← DECISION-TREE-LEARNING($examples_i$, *attributes* − *best*,
MAJORITY-VALUE(*examples*))
add a branch to *tree* with label v_i and subtree *subtree*
end
return *tree*

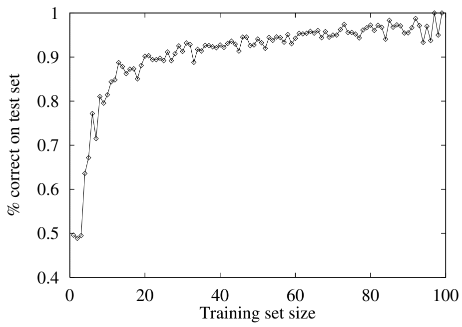
Tabelle 9.1: Lernen von Entscheidungsbäumen.



Der Algorithmus entwirft einen möglichst einfachen Baum, der mit den gegebenen Beispielen konsistent ist. *Wie gut ist der generierte Baum?* Wie weit ist er von dem "tatsächlichen" Baum entfernt? Kann man a-priori Abschätzungen geben? ($\rightsquigarrow$ Abschnitt 9.4).

Empirisches Vorgehen

1. Wähle eine Menge M_{Bsp} von Beispielen aus.
2. Teile diese in 2 Mengen auf: $M_{Bsp} = M_{Trai} \cup M_{Test}$.
3. Wende Lernalgorithmus auf M_{Trai} an und erhalte eine Hypothese H.
4. Bestimme den Anteil der Elemente aus M_{Test} die korrekt klassifiziert werden.
5. Wiederhole 1.-4. für viele $M_{Trai} \cup M_{Test}$ mit zufällig erzeugten M_{Trai}.



Informationstheorie

Wie wählt man das "beste" Attribut aus? Das beste Attribut ist dasjenige, das die größtmögliche Information liefert. Als Beispiel betrachten wir den fairen Münzwurf.

Definition 9.2.3 (1 Bit, Information)
1 Bit ist die Information die man braucht, um bei einem fairen Münzwurf das richtige Ergebnis anzusagen.

Allgemeiner: Angenommen, ein Experiment kann n-Ausgänge $v_1, \ldots, v_n$ haben. Jeder Ausgang v_i ergibt sich mit einer Wahrscheinlichkeit $P(v_i)$. Die Information die im Ergebnis (dem ausführen des Experimentes) steckt, ist definiert als

$$\mathbf{I}(P(v_1), \ldots, P(v_n)) := \sum_{i=1}^{n} -P(v_i) log_2 P(v_i)$$

Angenommen eine Münze ist präpariert. Zu 90 % kommt Kopf. Dann ist

$$\mathbf{I}(0.1, 0.9) = \ldots \approx 0.47$$

Für jedes Attribut A ist die folgende Frage wichtig: Wenn man diese Attribute auf der momentanen Trainigsmenge auswertet, wieviel Information gewinnt man dadurch? Das "beste" Attribut ist das mit der größtmöglichen Information!

Angenommen A hat ν verschiedene Ausgänge (Patron: 3, Hungry: 2, Type: 4) und teilt M_{Trai} in $E_1, \ldots, E_\nu$ auf. Jedes E_i enthalte p_i positive und n_i negative Beispiele. In jedem Zweig brauchen wir also noch

$$\mathbf{I}(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i})$$

Information um jedes Beispiel eindeutig zu klassifizieren.

Ein Beispiel ist in E_i mit der Wahrscheinlichkeit $\frac{p_i + n_i}{p+n}$ (wir nehmen an, daß M_{Trai} genau p positive und n negative Beispiele enthält).

Die Information um jedes Beispiel aus $M_{Trai} = \bigcup_i^\nu E_i$ zu klassifizieren ist

$$\text{Fehlende\_Inf}(A) = \sum_{i=1}^{\nu} \frac{p_i + n_i}{p+n} \mathbf{I}(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i})$$

Also gewinnen wir durch den Test des Attributes A folgende Information

$$\text{Gewinn}(A) = \mathbf{I}(\frac{p}{p+n}, \frac{n}{p+n}) - \text{Fehlende\_Inf}(A)$$

Choose_Attribute wählt das A aus mit maximalem Gewinn(A).

Aus Abb. 18.6 ergibt sich Gewinn(Patron)≈ 0.54. Berechnen Sie Gewinn(Type), Gewinn(Hungry) (Hungry als erstes Attribut), Gewinn(Hungry) (wenn vorher Patron ausgewählt wurde).

9.3 PL1 formalisierte Bereiche

Ziel: Allgemeinerer Rahmen.

Idee: Lernen ist Suchen im Hypothesen-Raum. ($\leadsto$ Kapitel 7)

Zielprädikat: $Q(x)$, einstellig (bisher: $Will\_Wait$)

Wir suchen eine Definition von $Q(x)$, d.h. eine Formel $C(x)$ mit

$$\forall x\ Q(x) \leftrightarrow C(x)$$

Jedes Beispiel X_i stellt eine Menge von Bedingungen dar, unter denen $Q(X_i)$ gilt oder nicht. Wir suchen eine Erklärung dafür: eine Formel $C(x)$, die die in den Beispielen auftretenden Prädikate benutzt. Z.B.:

$$\begin{array}{lll} \forall r \quad Will\_Wait(r) & \leftrightarrow & Patrons(r, Some) \\ & \vee & (Patrons(r, Full) \wedge \neg Hungry(r) \wedge Type(r, French) \\ & \vee & (Patrons(r, Full) \wedge \neg Hungry(r) \wedge Type(r, Thai) \wedge Fri\_Sat(r)) \\ & \vee & (Patrons(r, Full) \wedge \neg Hungry(r) \wedge Type(r, Burger) \end{array}$$

Definition 9.3.1 (Hypothese, Kandidatenfunktion)
Eine Formel $C_i(x)$ mit $\forall x C(x) \leftrightarrow C_i(x)$ heißt Kandidatenfunktion. Die gesamte Formel heißt Hypothese:

$$H_i: \ \forall x C(x) \leftrightarrow C_i(x)$$

Definition 9.3.2 (Hypothesenraum $\mathcal{H}$)
Der Raum aller Hypothesen $\mathcal{H}$ eines Lern-Alg. ist die Menge all jener Hypothesen, die der Algorithmus bilden kann.

Unter der Ausprägung einer Hypothese H bezüglich des Zielprädikates Q verstehen wir die Menge der Beispiele, Die H erfüllen (sie bestätigen H).

Hypothesen mit unterschiedlicher Ausprägung sind zusammen inkonsistent. Hypothesen mit gleicher Ausprägung sind logisch äquivalent.

Allgemeine Situation

- Wir haben eine Menge von Beispielen $\{X_1, \ldots, X_n\}$. Jedes Beispiel X beschreiben wir durch einen Satz und die Angabe $Q(x)$ (trifft zu) oder $\neg Q(x)$ (trifft nicht zu).

Example	Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	*Yes*
X_2	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	*No*
X_3	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	*Yes*
X_4	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*10–30*	*Yes*
X_5	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	*No*
X_6	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	*Yes*
X_7	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	*No*
X_8	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	*Yes*
X_9	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	*No*
X_{10}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	*No*
X_{11}	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	*No*
X_{12}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	*Yes*

X_1 ist beschrieben durch

$$Alt(X_1) \wedge \neg Bar(X_1) \wedge \neg Fri\_Sat(X_1) \wedge Hungry(X_1) \wedge \ldots$$

und $Will\_Wait(X_1)$. $\mathcal{H}$ ist die Menge der Entscheidungsbäume.

- Die Trainingsmenge ist die Menge all dieser Sätze.
- *Wir suchen eine Hypothese, die konsistent mit der Trainingsmenge ist.*

Unter welchen Bedingungen ist eine Hypothese inkonsistent mit einem Beispiel X?

Falsch negativ: Hypothese sagt nein ($\neg Q(X)$) aber in Wirklichkeit gilt '$Q(X)$.

Falsch positiv: Hypothese sagt ja ((X)) aber in Wirklichkeit gilt '$\neg Q(X)$.

Induktives Lernen in logik-basierten Bereichen heißt mit jedem Beispiel die Menge der möglichen Hypothesen weiter einschränken.

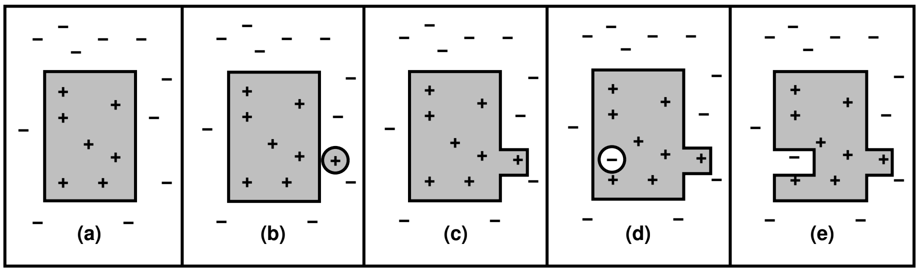
Für PL1 ist $\mathcal{H}$ im allgemeinen unendlich, daher ist automatisches Beweisen viel zu allgemein.

1. Ansatz: Wir behalten eine einzige Hypothese und modifizieren sie, falls die Beispiele mit ihr inkonsistent sind.

2. Ansatz: Wir behalten den ganzen Teilraum der noch mit den Beispielen konsistent ist (*Version-Space*. Dieser wird durch 2 Mengen effizient repräsentiert (so wie man mit [1,2] das entsprechende Intervall aller reellen Zahlen repräsentiert).

1. Ansatz: Current-Best-Hypothesis-Suche

Starte mit einfacher Hypothese H. Falls ein neues Beispiel damit konsistent: ok. Falls es "falsch negativ": Verallgemeinere H. Falls es "falsch positiv": Spezialisiere H.



Das ergibt einen Algorithmus:

function CURRENT-BEST-LEARNING(*examples*) **returns** a hypothesis

H ← any hypothesis consistent with the first example in *examples*
for each remaining example in *examples* **do**
  if *e* is false positive for *H* **then**
    H ← **choose** a specialization of *H* consistent with *examples*
  else if *e* is false negative for *H* **then**
    H ← **choose** a generalization of *H* consistent with *examples*
  if no consistent specialization/generalization can be found **then fail**
end
return *H*

Wie verallgemeinert, bzw. spezialisiert man?

$$H_1: \quad \forall x\ (Q(x) \leftrightarrow C_1(x))$$
$$H_2: \quad \forall x\ (Q(x) \leftrightarrow C_2(x))$$

H_1 verallgemeinert H_2, falls $C_2(x) \to C_1(x)$, H_1 spezialisiert H_2, falls $C_1(x) \to C_2(x)$. Verallgemeinern bedeutet: in einer Konjunktion lässt man $\wedge$-Glieder weg. Spezialisieren bedeutet: in einer Disjunktion nimmt man $\vee$-Glieder hinzu.

2. Ansatz: Version-Space

function VERSION-SPACE-LEARNING(*examples*) **returns** a version space
  local variables: *V*, the version space: the set of all hypotheses

V ← the set of all hypotheses
for each example *e* in *examples* **do**
  if *V* is not empty **then** *V* ← VERSION-SPACE-UPDATE(*V*, *e*)
end
return *V*

---

function VERSION-SPACE-UPDATE(V, e) **returns** an updated version space

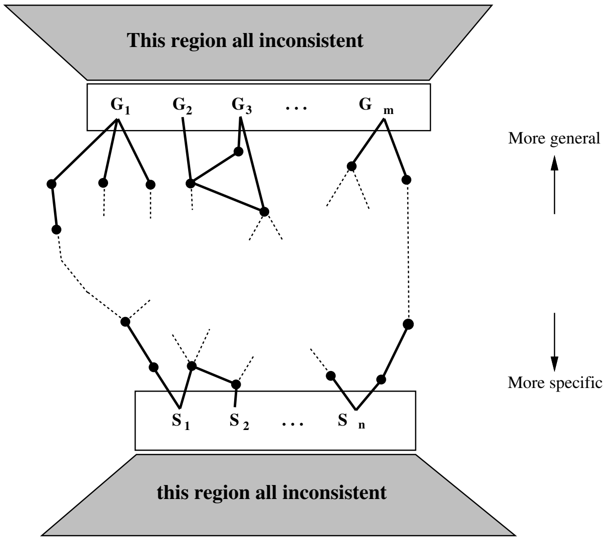
V ← $\{h \in V:$ *h* is consistent with $e\}$

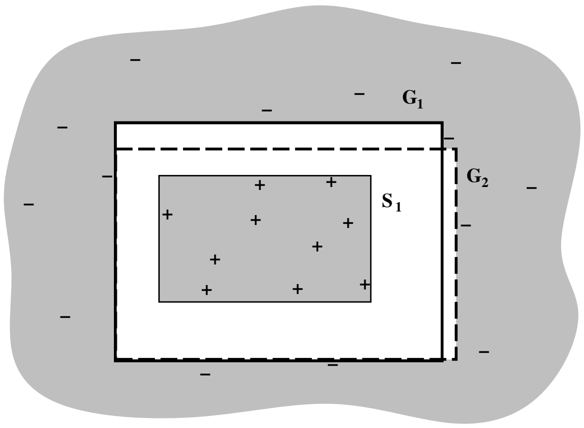
$\mathcal{H}$ ist eine große Disjunktion $H_1 \vee \ldots \vee H_n$. Wie soll man das repräsentieren?

Wie repräsentiert man die Menge aller reellen Zahlen zwischen 0 und 1? Als Intervall [0,1]. In unserem Fall gibt es eine partielle Ordnung auf $\mathcal{H}$ (Verallg./Spezial.). Die Grenzen sind gegeben durch

- *G-Menge*: ist konsistent mit allen bisherigen Beispielen und es gibt keine konsistente Hypothese die allgemeiner ist,
- *S-Menge*: ist konsistent mit allen bisherigen Beispielen und es gibt keine konsistente Hypothese die spezifischer ist.

Werden neue Beispiele betrachtet, müssen die G- und S-Mengen in VERSION-SPACE-UPDATE geeignet modifiziert werden.





Was kann passieren?

1. Es bleibt nur eine Hypothese übrig: Hurra!

2. Der Raum kollabiert: $G = \emptyset$ oder $S = \emptyset$.
3. Keine Beispiele mehr, aber mehrere Hypothesen. D.h. wir enden mit einer grossen Disjunktion.

9.4 Warum funktioniert das alles?

Wie nah ist eine aus dem Lern-Algorithmus berechnete Hypothese h von der richtigen Funktion f entfernt? Das führt zur *Computational Learning Theory*: **PAC**-Lernen – **P**robably **A**pproximately **C**orrect. Die Grundidee dabei ist: *Ist eine Hypothese konsistent mit einer großen Trainingsmenge, so kann sie nicht völlig falsch sein.*

Wie verhalten sich Trainings und Test-Menge zueinander? Wir machen folgende Annahme:

Trainings- und Testmenge sind gemäß derselben Wahrscheinlichkeitsverteilung aus der Menge X aller Beispiele entnommen.

Definition 9.4.1 (error(h), ϵ-approximativ korrekt)
Sei $h \in \mathcal{H}$ eine Hypothese und f die richtige (zu lernende) Funktion. Interessant ist die Menge

$$Diff(f,h) := \{x : h(x) \neq f(x)\}.$$

Wir bezeichnen mit $error(h)$ die Wahrscheinlichkeit, daß ein beliebiges Beispiel aus X in der Menge $Diff(f,h)$ liegt.

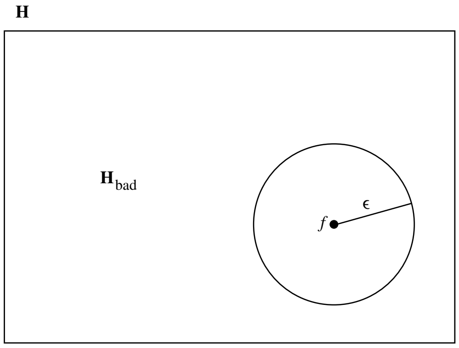
Für $\epsilon > 0$ heißt die Hypothese h ϵ-approximativ korrekt, falls $error(h) \leq \epsilon$.

Sei $\epsilon > 0$ gegeben. Wieviele Beispiele muß die Trainingsmenge enthalten, damit die durch einen Lern.-Alg. erzeugte Hypothese ϵ-approximativ korrekt ist?

Definition 9.4.2 (Beispielkomplexität)
Gegeben $\delta > 0$ und $\epsilon > 0$. Unter der Beispielkomplexität verstehen wir die Zahl m von Beispielen die ein beliebiger Lern-Algorithmus braucht, damit die von ihm erzeugte Hypothese h mit mindestens Wahrscheinlichkeit $1-\delta$ jedenfalls ϵ-approximativ korrekt ist.

Theorem 9.4.3
Die Beispielkomplexität m hängt in folgender Weise von ϵ, δ und dem Hypothesenraum $\mathcal{H}$ ab:

$$m \geq \frac{1}{\epsilon}(ln\frac{1}{\delta} + ln|\mathcal{H}|)$$



Was bedeutet das letzte Ergebnis? Komplexität hängt von $ln(|\mathcal{H}|)$ ab. Für boolesche Funktionen in n-Attributen ist

$$ln(|\mathcal{H}|) = 2^n$$

also braucht man exponentiell viele Beispiele selbst um nur mit einer Wahrscheinlichkeit ϵ-approximativ korrekt zu sein! Also:

- Verschiedene Mengen von Beispielen ergeben verschiedene Aussagen und damit Werte von ϵ. Es kommt also nicht nur darauf an, *wieviele* sondern auch *welche* Beispiele man auswählt.

  Wir relativieren darum unsere Frage und wollen nur noch fordern, daß die Hypothese mit einer Wahrscheinlichkeit von $1 - \delta$, wobei $\delta > 0$,ϵ-approximativ korrekt ist.

- Wir möchten auch von speziellen Lern-Algorithmen abstrahieren und eine Aussage über alle möglichen Lern-Algorithmen machen. Wir gehen daher davon aus, daß jeder Lern-Alg. jedenfalls eine Hypothese berechnet, die mit den bisherigen Beispielen konsistent ist.

Beweis: Sei $h_b \in \mathcal{H}$ mit $error(h_b) > \epsilon$.

- Wie groß ist die Wahrscheinlichkeit P_b, daß h_b konsistent mit m gewählten Beispielen ist? Für ein Beispiel ist es gerade $\leq (1 - \epsilon)$ wegen der Definition von $error$. Also:

$$P_b \leq (1 - \epsilon)^m$$

- Wie groß ist die Wahrscheinlichkeit P', daß es überhaupt eine Hypothese h_b mit $error(h_b) > \epsilon$ gibt, die mit m Beispielen konsistent ist?

$$\begin{array}{rcl} P' & \leq & \sum_{Hypothesen}(1-\epsilon)^m \\ & \leq & |\{h : error(h) > \epsilon\}|(1-\epsilon)^m \\ & \leq & |\mathcal{H}|(1-\epsilon)^m \end{array}$$

- Wir wollen, daß $P' \leq \delta$:

$$|\mathcal{H}|(1-\epsilon)^m \leq \delta$$

Einges umformen ergibt:

$$\begin{array}{lll} (1-\epsilon)^m & \leq & \frac{\delta}{|\mathcal{H}|} \\ mln(1-\epsilon) & \leq & ln(\delta) - ln(|\mathcal{H}|) \\ m & \geq & -\frac{1}{ln(1-\epsilon)}(ln(\frac{1}{\delta}) + ln(|\mathcal{H}|)) \\ m & \geq & \frac{1}{\epsilon}(ln(\frac{1}{\delta}) + ln(|\mathcal{H}|)) \end{array}$$

Die letzte Zeile gilt wegen $ln(1-\epsilon) < -\epsilon$.

■

das Theorem kann man auch so ausdrücken:

Lernen ist nie besser, als in einer Tabelle nachzuschlagen!

1. Ausweg: Wir fordern, daß nicht eine beliebige konsistente Hypothese generiert wird, sondern möglichst spezielle. (Das ist aber dann von der Komplexität noch schlechter.)

2. Ausweg: Wir geben es auf, beliebige boolesche Funktionen zu lernen und beschränken uns auf geeignete Teilklassen.

So können wir etwa **Entscheidungslisten** betrachten.

Theorem 9.4.4 (E-Listen werden in polynomialer Zeit gelernt)
Jede Funktion aus $k-DL(n)$ (E-Listen mit höchstens k-Tests) haben eine PAC-Komplexität von

$$m = \frac{1}{\epsilon}(ln(\frac{1}{\delta}) + O(n^k log_2(n^k)))$$

Mit anderen Worten: Jeder Algorithmus der für eine Menge von Beispielen eine konsistente Entscheidungsliste zurückgibt, kann in einen PAC-Lern-Algorithmus verwandelt werden, der eine $k - DL(n)$ Funktion nach höchstens m-Beispielen lernt.
Folgende Abschätzungen sind wichtig:

- $|Conj(n,k)| = \sum_{i=0}^{k} \binom{2n}{i} = \mathrm{O}(n^k)$,
- $|k - DL(n)| \leq 3^{|Conj(n,k)|}|Conj(n,k)\mathbf{!}$,
- $|k - DL(n)| \leq 2^{\mathrm{O}(n^k log_2(n^k))}$.

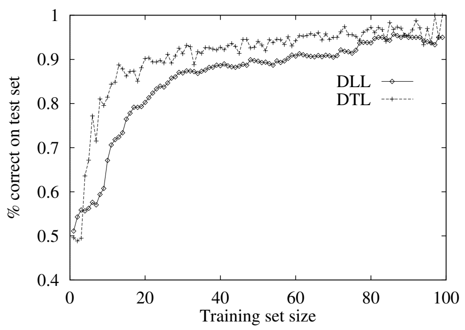


Abbildung 9.4: Vergleich E-Listen und E-Bäume

function DECISION-LIST-LEARNING(*examples*) **returns** a decision list, *No* or failure

if *examples* is empty **then return** the value *No*
t ← a test that matches a nonempty subset $examples_t$ of *examples*
such that the members of $examples_t$ are all positive or all negative
if there is no such *t* **then return** failure
if the examples in $examples_t$ are positive **then** $o \leftarrow Yes$
else $o \leftarrow No$
return a decision list with initial test *t* and outcome *o*
and remaining elements given by DECISION-LIST-LEARNING($examples - examples_t$)

9.5 Rauschen und Overfitting

Rauschen:

- Beispiele sind inkonsistent (einmal $Q(x)$, dann $\neg Q(x)$).
- Keine Attribute mehr um Beispiele zu klassifizieren.
- Macht Sinn, wenn die Umgebung nichtdeterministisch

Overfitting: Dual zum Rauschen: überbestimmt.

- Verbleibende Beispiele können mittels Attributen klassifiziert werden, die eine *Ordnung* ergeben, welche nicht vorhanden ist (irrelevante Attribute). *Beispiel:* Mehrere farbige Würfel werden geworfen. Jeder Wurf wird durch (Tag, Monat, Uhrzeit, Farbe) beschrieben. Solnge es keine Inkonsistenz gibt, ist jeder Wurf einzig und durch eine völlig überbestimmte Hypothese beschrieben.

Andere Beispiele:

- Pyramiden,
- Astrologie,
- “Mein magisches Fahrrad”.

9.6 Aufgaben

Aufgabe 9.1 (Entscheidungsbäume)
Wir haben es zugelassen, daß die Attribute in einem Entscheidungsbaum auch mehr als zwei Werte annehmen können.

1. *Zeigen Sie, daß man jeden Entscheidungsbaum in einen binären Entscheidungsbaum mit gleicher Funktionalität transformieren kann, der nur noch Boole'sche Attribute enthält.*
2. *Wie verändert sich die Anzahl der Attribute und die Tiefe des Baumes?*
3. *Gibt es Unterschiede zwischen binären Entscheidungsbäumen und Entscheidungslisten?*

Aufgabe 9.2 (Entscheidungslisten)
Zeigen Sie, daß Entscheidungslisten

1. *jede beliebige Boole'sche Funktion darstellen können, falls die Tests keiner Größenbeschränkung unterliegen, bzw.*
2. *Entscheidungsbäume der Tiefe k simulieren können, falls alle Tests höchstens k Literale enthalten.*

Aufgabe 9.3 (Fehlerabschätzungen)
Bei der rekursiven Konstruktion eines Entscheidungsbaumes kann es passieren, daß an einem Blatt eine gemischte Menge von Beispielen übrigbleibt, d.h. $p > 0$ positive und $n > 0$ negative Beispiele. Trotzdem muß der Algorithmus eine Entscheidung treffen, obwohl kein Attribut mehr zur Verfügung steht.

1. *Zeigen Sie, daß der* DECISION-TREE-LEARNING-*Algorithmus, der sich in diesem Fall nach der Mehrheit der Beispiele entscheidet, den absoluten Fehler minimiert.*
2. *In der Vorlesung wurde ein modifizierter Algorithmus besprochen, der nur eine Wahrscheinlichkeit zurückgibt, nämlich $p/(p+n)$. Zeigen Sie, daß diese Vorgehensweise die Summe der Fehler-Quadrate minimiert.*

Aufgabe 9.4 (Zahlen lernen)
Zu lernen ist die Eigenschaft natürlicher Zahlen, daß sie durch 6 teilbar sind. Als Lernbeispiele sind die natürlichen Zahlen von 1 bis 60 gegeben. Die Reste bei der Division durch 2, 3, 4 bzw. 5 stehen als Attribute zur Verfügung.

1. *Berechnen Sie den Informationsgewinn der vier Attribute!*
2. *Wie sieht ein Entscheidungsbaum mit möglichst wenig Knoten aus?*

Aufgabe 9.5 (Wahrscheinlichkeiten)
Ein Lern-Algorithmus suche eine mit den gegebenen Daten konsistente Hypothese. Die Zuordnung der Beispiele zu den Klassen „positiv“ und „negativ“ erfolge jedoch vollkommen zufällig. Die Beispiele haben n binäre Merkmale.

Sei nun eine Menge von x Beispielen gleichwahrscheinlich zufällig aus den 2^n möglichen ausgewählt. Schätzen Sie die Anzahl von Beispielen x ab, die genommen werden muß, so daß die Wahrscheinlichkeit, daß eine Inkonsistenz auftritt, gerade 0.5 ist.

Aufgabe 9.6 (Informationsgewinn)
Ein Attribut A unterteile eine Menge von Beispielen E in Teilmengen E_i, wobei jede Teilmenge p_i positive und n_i negative Beispiele enthalte. Zeigen Sie, daß $\textsc{Gewinn}(A) > 0$ *ist, außer wenn $p_i/(p_i + n_i)$ für alle i gleich ist.*

10 Lernen mit Netzen

- Methode, um komplizierte Funktionen mit sehr einfachen (aber vielen und vernetzten) Rechen-Einheiten darzustellen und solche Darstellungen aus Beispielen zu lernen.
- Die Funktionsweise unseres Gehirns besser kennenzulernen.

10.1 Das menschliche Gehirn

Neuronen bestehen aus

1. *Soma* (Zell-Körper),
2. *Nucleus* (Zell-Kern),
3. *Dendriten*,
4. *Axon* (1 cm - 1 m).

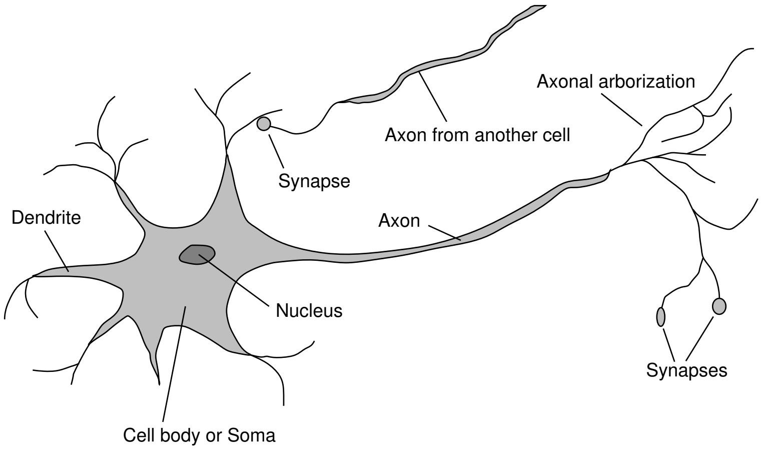


Abbildung 10.1: Struktur der Nervenzellen

	Computer	Human Brain
Computational units	1 CPU, 10^5 gates	10^{11} neurons
Storage units	10^9 bits RAM, 10^{10} bits disk	10^{11} neurons, 10^{14} synapses
Cycle time	10^{-8} sec	10^{-3} sec
Bandwidth	10^9 bits/sec	10^{14} bits/sec
Neuron updates/sec	10^5	10^{14}

Tabelle 10.1: Vergleich Gehirn/Computer

Axon spaltet sich auf und erreicht Dendriten anderer Neuronen: diese Stellen heißen *Synapsen*. Jedes Neuron hat Synapsen mit 10-100000 anderen Neuronen.

Transmitter übertragen Impulse von den Synapsen in die Dendriten und verändern das elektrische Potential in der Soma. Falls das über eine gewisse Reizschwelle geht, wird das Potential über das Axon gesendet: dies pflanzt sich dann fort.

Synapsen bilden sich durch Stimulation und äußere Reize.

Computer: seriell, sehr schnell, öfter mal Reboot,

Gehirn: hochgradig parallel, eher langsam, Fehler-tolerant (dauernd sterben Neuronen ab).

Untersuchungen haben gezeigt, daß in den 60 Jahren in Amerika aufgewachsene Kinder durch die Reizüberflutung (Fernsehen) nachweisbar mehr Nerven-Strukturen im Gehirn besitzen, als ihre direkten Vorfahren. Dies wirkt sich also schnell auf der körperlichen Ebene aus (schon in einer Generation).

10.2 Neuronale Netze

Definition 10.2.1 (Neuronales Netz)
Ein neuronales Netz besteht aus

1. Units *und*
2. Links *zwischen Units.*

Diese Links sind gewichtet. Es gibt 3 Arten von Units: Input-Units*,* Hidden-Units *und* Output-Units*.*

Eine Unit i bekommt einen Input über Links von anderen Units j. Die Input-Funktion

$$in_i := \sum_j W_{j,i}\, a_j$$

bestimmt die gewichtetet Summe davon.

Die *Aktivierungsfunktion* g bestimmt daraus den Output a_i, der über Output-Links an andere Units abgegeben wird:

$$a_i := g(in_i)$$

Notation	Meaning
a_i $\mathbf{a}_i$	Activation value of unit *i* (also the output of the unit) Vector of activation values for the inputs to unit *i*
g g'	Activation function Derivative of the activation function
Err_i Err^e	Error (difference between output and target) for unit *i* Error for example *e*
I_i $\mathbf{I}$ $\mathbf{I}^e$	Activation of a unit *i* in the input layer Vector of activations of all input units Vector of inputs for example *e*
in_i	Weighted sum of inputs to unit *i*
N	Total number of units in the network
O O_i $\mathbf{O}$	Activation of the single output unit of a perceptron Activation of a unit *i* in the output layer Vector of activations of all units in the output layer
t	Threshold for a step function
T $\mathbf{T}$ $\mathbf{T}^e$	Target (desired) output for a perceptron Target vector when there are several output units Target vector for example *e*
$W_{j,i}$ W_i $\mathbf{W}_i$ $\mathbf{W}$	Weight on the link from unit *j* to unit *i* Weight from unit *i* to the output in a perceptron Vector of weights leading into unit *i* Vector of all weights in the network

Tabelle 10.2: Einige Begriffe für neuronale Netze.

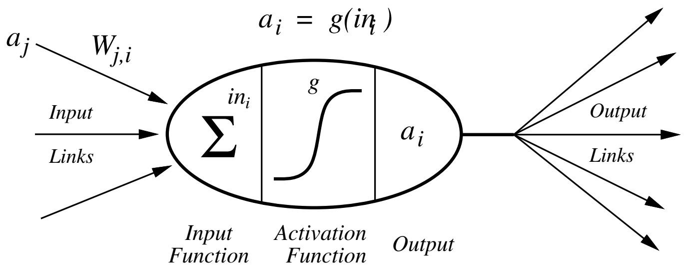


Abbildung 10.2: Ein idealisiertes Modell für ein Neuron

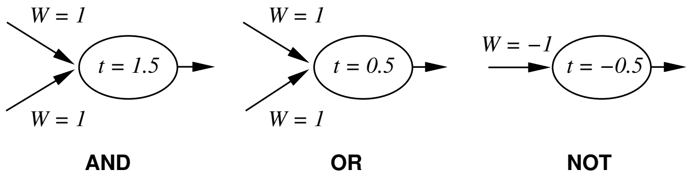
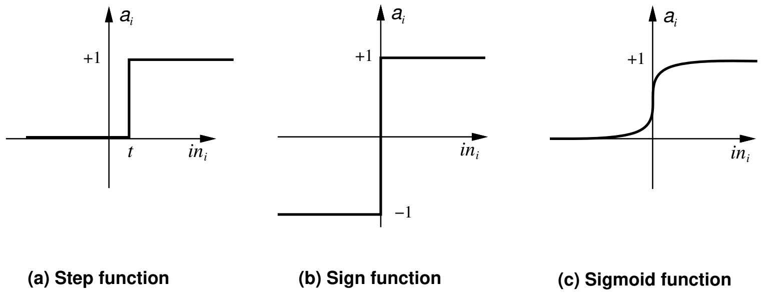


Abbildung 10.3: Einfache Units.

Beispiele für Aktivierungsfunktionen



Normierung: Anstatt $step_t$ betrachten wir $step_0$.
Benutzt ein Unit i die Aktivierungsfunktion $step_t(x)$, so führen wir einen zusätzlichen Input Link "0" ein, der einen konstanten Wert von $a_0 := -1$ beisteuert. Dieser Wert wird mit $W_{0,i} := t$ gewichtet. Wir können nun $step_0$ als Aktivierungsfunktion wählen:

$$step_t(\sum_{j=1}^{n} W_{j,i}a_j) \;=\; step_0(-t + \sum_{j=1}^{n} W_{j,i}a_j) \;=\; step_0(\sum_{j=0}^{n} W_{j,i}a_j)$$

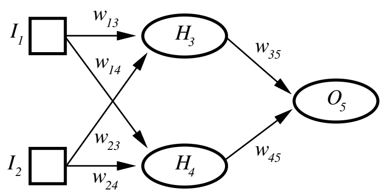
Netzwerke können

- **rekurrent**, also irgendwie miteinander verbunden,
- **feed-forward**, d.h. es gibt keine Zyklen durch die Links (azyklischer Graph),

sein. Üblicherweise sind Netzwerke in **Schichten** (Layers) aufgeteilt:

- Units einer Schicht haben nur Links zu Units in der nächsten Schicht.

Ein Beispiel sind **Multi-Layer Feed-Forward Netze**: diese haben keine internen Zustände, und damit kein Kurzzeit-Gedächtnis.



Der Output der Input-Units ist durch die Umgebung bestimmt.

Welche Funktion beschreibt die letzte Abbildung? Warum können überhaupt nicht-triviale Funktionen dargestellt (bzw. gelernt) werden?[1] *Wieviele Units brauchen wir?*

- **Wenig:** Zuwenige Funktionen können repräsentiert werden.
- **Viele:** Das Netz merkt sich alle Beispiele und schaut nur nach (*Overfitting*, $\leadsto$ Kapitel 9).

Hopfield-Netze:

1. Bidirektionale Links mit symmetrischen Gewichten.
2. Aktivierungsfunktion: $sign$.
3. Units sind sowohl Input- als auch Output-Units.

Lernen in neuronalen Netzen heißt alle Parameter so einstellen, daß sie mit den Trainingsdaten konsistent sind.

Wie findet man die optimale Netzwerk-Struktur? Man sucht im Raum aller Netzwerk-Strukturen (etwa mit genetischen Algorithmen, $\leadsto$ Abschnitt 10.6).

10.3 Das Perceptron

Definition 10.3.1 (Perceptron)
Ein Perceptron ist ein feed-forward *Netz mit nur einem Layer (also keinen hidden Units).*

Betrachten wir noch einmal die Darstellung von AND, OR und NOT in Abbildung 10.3. Kann man nicht einfach durch Komposition jede boolesche Funktion als Perceptron darstellen? Was ist mit

$$f(x_1, \dots, x_n) := \begin{cases} 1, & \text{falls } \sum_{i=1}^{n} x_i > \frac{n}{2} \\ 0, & \text{sonst.} \end{cases}$$

Lösung: $W_j := 1$ für alle $j = 1, \dots n$, und $step_{\frac{n}{2}}$. Sehr kompakte Darstellung (vergl. Entscheidungsbaum)!

- Man kann jede boolesche Funktion als Komposition von $\neg, \wedge, \vee$ darstellen, aber
- die Komposition der entsprechenden Perceptronen ist kein Perceptron mehr!

Was ist mit XOR?

[1] Weil die Aktivierungsfunktion **nicht-linear** ist. Falls die Aktivierungsfunktion *linear* ist, so sind nur lineare Funktionen darstellbar: $a_5 = g(W_{3,5}a_3 + W_{4,5}a_4) = g(W_{3,5}g(W_{1,3}a_1 + W_{2,3}a_2) + W_{4,5}g(W_{1,4}a_1 + W_{2,4}a_2))$.

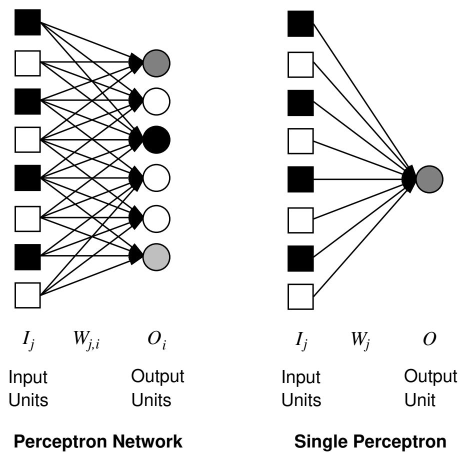
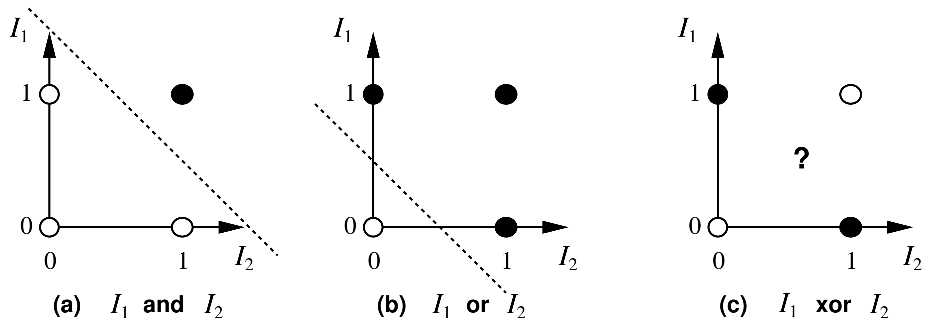


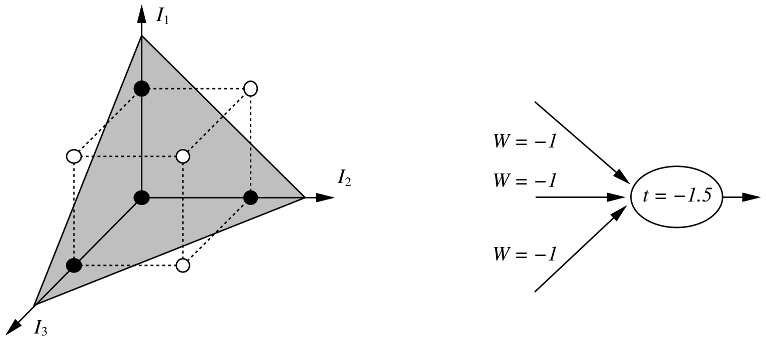
Abbildung 10.4: Modell des Perceptrons



$$\text{Output} \;=\; step_0(\sum_{j=1}^{n} W_j I_j)$$

$\sum_{j=1}^{n} W_j I_j = 0$ definiert eine **Hyperfläche** im n-dimensionalen Raum.

Definition 10.3.2 (Linear Separabel)
Eine boolesche Funktion in n-Attributen heißt linear separabel*, falls es eine Hyperfläche (also einen $n-1$-dimensionalen Teilraum gibt, der die positiven Funktionswerte (1, bzw. wahr) von den negativen (0, bzw. falsch) separiert.*



(a) Separating plane **(b) Weights and threshold**

Lern-Algorithmus: Ähnlich wie *Current-Best-Hypothesis* (⇝ Kapitel 9).

- **Hypothese:** Netzwerk mit momentanen Gewichtungen (zufälliger Anfangszustand).
- **UPDATE:** Mache es konsistent durch geringe Änderungen. *Wichtig:* Für jedes Beispiel wird UPDATE öfter durchlaufen. Diese Durchläufe heißen **Epochen**.

```
function NEURAL-NETWORK-LEARNING(examples) returns network

  network ← a network with randomly assigned weights
  repeat
     for each e in examples do
        O ← NEURAL-NETWORK-OUTPUT(network, e)
        T ← the observed output values from e
        update the weights in network based on e, O, and T
     end
  until all examples correctly predicted or stopping criterion is reached
  return network
```

Definition 10.3.3 (Perceptron-Lernen)
Beim Perceptron-Lernen werden die Gewichte W_j durch folgende Regel modifiziert:

$$W_j := W_j \; + \; \alpha \times I_j \times \mathit{Error}$$

wobei Error:= $T - O$ (also die Differenz des korrekten und des momentanen Output-Wertes ist). α heißt Lern-Rate.

Theorem 10.3.4 (Konvergenztheorem von Rosenblatt)
Jede durch ein Perceptron repräsentierbare Funktion wird durch den obigen Lern-Algorithmus gelernt. Genauer: Die W_j konvergieren zu einer Funktion die die Beispiele korrekt repräsentiert.

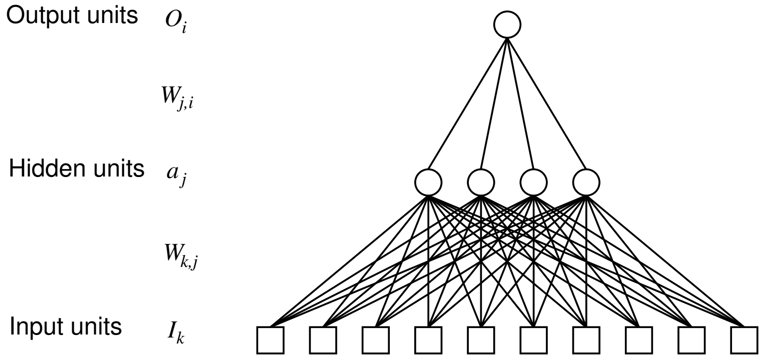
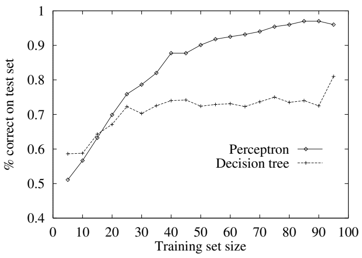
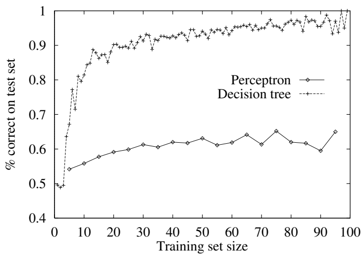


Abbildung 10.5: Multi-Layer Netze





Warum steigt Decision-Tree im ersten Bild nicht an? Je mehr Beispiele er bekommt, desto besser muss er doch sein.[2]

Das zweite Bild ist klar: Perceptron ist schlecht, weil die Funktion eben nicht linear separabel ist.

10.4 Multi-Layer Feed-Forward Netze

Problem: Wie sieht die error-Funktion für die hidden units aus?

Lernen in Multi-Layer-Netzen: **Backpropagation**.

In unserem Restaurant-Beispiel war das Perceptron nicht mächtig genug. Also versuchen wir es mit 2 Schichten. 10 Attrinute ergeben 10 Input-Units. *Wieviele hidden-units brauchen wir?* Für unser Beispiel reichen 4!

Der error beim Perceptron war leicht zu bestimmen, da es nur ein W_j zwischen In- und Out-put gab. Nun gibt es mehrere. *Wie soll der Fehler verteilt werden?*

Hier die Herleitung für die kommenden Gleichungen. Wir minimieren die Summe

[2] Es handelt sich aber um eine Majoritätsfunktion in 11 Input-Variablen. Also gibt es 2^{11} Beispiele: 100 ist noch weit davon entfernt!

der Fehler-Quadrate

$$E = \frac{1}{2}\sum_i (T_i - O_i)^2$$

Setzen wir den Output O_i ein, erhalten wir:

$$E = \frac{1}{2}\sum_i (T_i - g(\sum_j W_{j,i}a_j))^2 = \frac{1}{2}\sum_i (T_i - g(\sum_j W_{j,i}g(\sum_k W_{k,j}I_k)))^2$$

Man betrachte nun E als Funktion der $W_{j,i}$ und $W_{k,j}$ und versuche E zu minimieren. Wir machen also einen Gradientenabstieg.

$$\begin{aligned}\frac{\partial E}{\partial W_{j,i}} &= \tfrac{1}{2}\textstyle\sum_i (T_i - O_i)^2 \\ &= \tfrac{1}{2}2(T_i - g(\ldots))(-\tfrac{\partial g}{\partial x}(in_i))a_j \\ &= -a_j(T_i - O_i)\tfrac{\partial g}{\partial x}(in_i) \\ &= -a_j\Delta_i\end{aligned}$$

Und nun die $W_{k,j}$:

$$\begin{aligned}\frac{\partial E}{\partial W_{k,j}} &= \tfrac{1}{2}\textstyle\sum_i \big(2(T_i - g(\ldots))(-\tfrac{\partial g}{\partial x}(in_i))(W_{j,i}\tfrac{\partial g}{\partial x}(in_j)I_k)\big) \\ &= \textstyle\sum_i \big(\Delta_i W_{j,i}\tfrac{\partial g}{\partial x}(in_j)I_k\big) \\ &= \tfrac{\partial g}{\partial x}(in_j)I_k\textstyle\sum_i W_{j,i}\Delta_i\end{aligned}$$

Output-Units: ähnlich zum Perceptron,

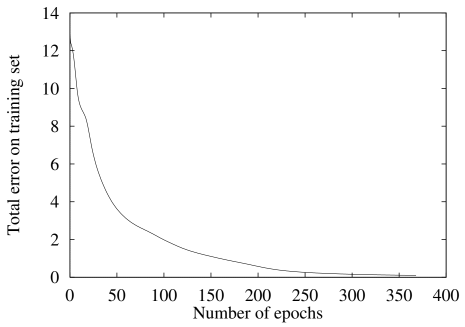
$$W_{j,i} := W_{j,i} \; + \; \alpha \times a_j \times \text{Error}_i \times g'(in_i)$$

Statt $\text{Error}_i \times g'(in_i)$ schreibt man kurz Δ_i.

Hidden-Units: *"Jedes Hidden-Unit j ist zu einem Teil für den Fehler Δ_i zuständig (wenn j mit Output-Unit i verbunden ist)."*

$$W_{k,j} := W_{k,j} \; + \; \alpha \times I_k \times \Delta_j$$

wobei $\Delta_j := g'(in_j)\sum_i W_{j,i}\Delta_i$.



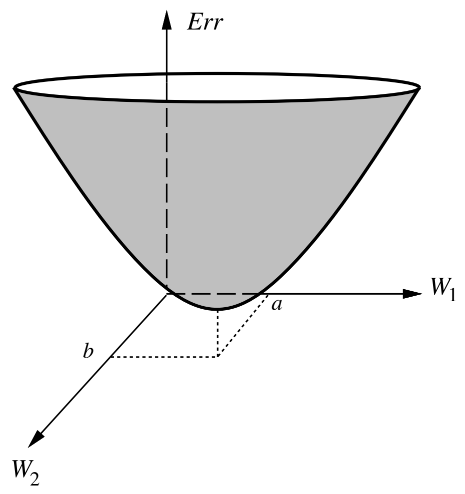
```
function BACK-PROP-UPDATE(network, examples, α) returns a network with modified weights
  inputs: network, a multilayer network
          examples, a set of input/output pairs
          α, the learning rate

  repeat
    for each e in examples do
      /* Compute the output for this example */
        O ← RUN-NETWORK(network, I^e)
      /* Compute the error and Δ for units in the output layer */
        Err^e ← T^e − O
      /* Update the weights leading to the output layer */
        W_{j,i} ← W_{j,i} + α × a_j × Err_i^e × g'(in_i)
      for each subsequent layer in network do
        /* Compute the error at each node */
          Δ_j ← g'(in_j) Σ_i W_{j,i} Δ_i
        /* Update the weights leading into the layer */
          W_{k,j} ← W_{k,j} + α × I_k × Δ_j
      end
    end
  until network has converged
  return network
```

Tabelle 10.3: Back-Propagation-Algorithmus.

Back-Propagation Algorithmus

1. Berechne Δ_i für die Output-Units aufgrund des beobachteten Fehlers $error_i$.
2. Rekursiv für jede Schicht (zuerst Output-Schicht):
   - Propagiere die Δ_i “zurück” (Schicht davor),
   - Modifiziere die Gewichte zwischen den momentanen Schichten.

Back-Propagation ist Gradientensuche! $error$ ist Funktion der Gewichte des Netzes. Diese Funktion gibt uns die Fehler-Oberfläche.



Allgemeine Bemerkungen

Ausdrückbarkeit: Neuronale Netze gut für stetige In- und Out-puts (Rauschen).

Um alle booleschen Funkt. in n-Attributen darzustellen, genügen $\frac{s^n}{n}$-hidden units. Oft geht es mit weniger: *die Kunst ist die Topologie des Netzes zu bestimmen.*

Effizienz: m Beispiele, $|W|$ Gewichte: Jede Epoche braucht $\mathbf{O}(m \times |W|)$-Zeit. Wir wissen ($\leadsto$ Kapitel 7, Abschnitt 9.4): Anzahl der Epochen ist exponentiell. In der *Praxis* ist die Konvergenzzeit sehr variabel.
Problem: Lokale Minima auf Fehler-Fläche.

Transparenz: Black-Box. Bäume oder Listen erklären wenigstens ihr Ergebnis.

10.5 Anwendungen

Alle folgenden Anwendungen konnten nach einer mehrere Monate dauernden Test-Phase zufriedenstellend gelöst werden.
Aussprache:

- (1987) *NETtalk* kann englischen Text aussprechen. Input besteht aus dem auszusprechenden Buchstaben und dem Vorgänger und 3 Nachfolgern.
- 80 Hidden Units.
- **Output-Schicht:** hoch, tief, betont, unbetont, . . .
- **Training:** 1024 Worte. Nach 50 Epochen: 95% korrekt auf Trainingsmenge.
- **Test-Menge:** Zu 78 % korrekt.

Schrifterkennung:

- (1989) Postleitzahlen auf Briefumschlägen erkennen.
- Input 6× 16 Pixel-array. 3 Hidden Layers, mit jeweils 768, 192, 30 Units. 10 Output Units (0 – 9). Das ergibt schon 200000 Gewichte!
  Trick: Units in der ersten Schicht untersuchen nur eine 5×5 Teilmatrix. $768 = 12 \times 64$. Jede der 12 Gruppen benutzt gleiche Gewichte. Jede Gruppe ist für ein Merkmal da. Insgesamt 9760 Gewichte.
- **Training:** 7300 Beispiele.
- **Test-Menge:** Zu 99 % korrekt.

Auto-Fahren:

- (1993) ALVINN. Aktionen: Lenken, beschleunigen, bremsen. Wir fahren nur geradeaus.
- **Input:** Stereo Farb-Kamera, Radar.
  Bild geht in ein 30×32 Pixel-array (Input-Units).
- **Output-Schicht:** 30 Units (entspricht dem Lenk-Winkel). Ausgewählt wird das Output-Unit mit höchster Aktivierung. Es wird also eine Funktion gesucht, die jedem Bild (der Strasse) einen Lenkwinkel zuordnet.
- **Hidden-Units:** Genau eine Schicht mit 5 Units.
- **Training:** Ein Mensch fährt 5 Minuten und das Netz “schaut zu”.
- **Back-Propagation:** 10 Minuten
- **Problem:**
  - Mensch fährt zu gut, keine Beispiele was getan werden muss wenn das Auto vom Weg abkommt.
  - Helligkeit! Hängt stark vom Belag und Wetter ab.

10.6 Genetische Algorithmen

Erinnern Sie sich an:

1. Darwin, Lamarck,
2. *Survival of the fittest*,
3. Mutation, Selektion, Reproduktion.

Algorithmus: arbeitet auf einer Population, reproduziert und selektiert gemäß einer Fitness-Funktion und gibt Individuen zurück.

```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
      parents ← SELECTION(population, FITNESS-FN)
      population ← REPRODUCTION(parents)
  until some individual is fit enough
  return the best individual in population, according to FITNESS-FN
```

Was hat das mit Lernen zu tun? Die Rolle des Individuums:

- Funktion eines Agenten
  (Fitness ist Performanz),
- Komponenten-Funktion eines Agenten
  (Fitness ist Kritik).

GENETIC-ALG. stellt parallele Suche mit Hill-Climbing dar. Wie wendet man diesen Algorithmus zum Lernen an?

1. **Fitness-Fkt.:** Menge aller Ind. $\longrightarrow \mathbb{R}$,
2. **Repr. eines Individuums:** Zeichenkette über endlichem Alphabet,
3. **Selektion von Individuen:** zufällig, Wahrscheinlichkeit proportional zur Fitness,
4. **Reproduktion von Individuen:** *Cross-Over* und *Mutation*. Selektierte Individuen werden zufällig gepaart. Für jedes Paar wird zufällig eine Cross-Over-Stelle ausgewählt ($n_0 \in \mathbb{N}$): d.h. Nachkomme hat die ersten n_0 Zeichen des einen Elters, ab n_{0+1} die des anderen Elters. Jedes Zeichen kann mit geringer Wahrscheinlichkeit mutieren.

Beispiel 10.6.1 (Entscheidungslisten im Restaurant-Beispiel)
Fitness-Funktion: *Anzahl der Beispiele mit der ein Individuum (eine Entscheidungsliste) konsistent ist.*

Repräsentation: (Zur Erinnerung: 10 Attribute, manche nicht-boolesch).
Um alle Paare (Attribut,Wert) darzustellen, brauchen wir 5 Bit. Für $k - DL(n)$ brauchen wir: $t \times (6 \times k + 1)$ Bit.

10.7 Aufgaben

Aufgabe 10.1 (Neuronale Netze)
Gegeben sei ein neuronales Netzwerk mit linearen Aktivierungsfunktionen. Das heißt, die Ausgabe jeder Einheit ist $c \cdot x + d$, wobei x die gewichtete Summe der Eingaben dieser Einheit ist.

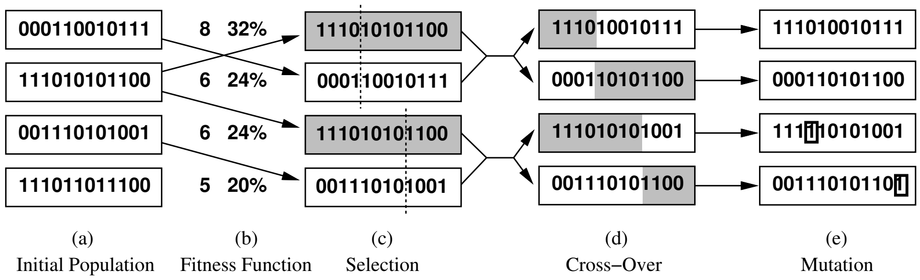


Abbildung 10.6: Der genetische Algorithmus

1. *Sei ein Netzwerk mit einem* Hidden Layer *gegeben. Entwickeln Sie Gleichungen für die Werte vom* Output Layer *als Funktion aller Gewichte* $\mathbf{W}$ *und den Eingaben* $\mathbf{I}$ *vom* Input Layer, *ohne die Ausgabe vom* Hidden Layer *explizit zu erwähnen.*

2. *Zeigen Sie, daß es ein Netzwerk ohne* Hidden Layer *gibt, das die gleiche Funktion berechnet!*

3. *Wiederholen Sie die Betrachtung aus Teil (a), aber diesmal mit mehr als einem* Hidden Layer. *Was können wir daraus schließen über Netzwerke mit linearen Aktivierungsfunktionen?*

Aufgabe 10.2 (Perceptron versus Entscheidungsbäume)
Betrachten Sie die folgende Menge von Beispielen. Jedes Beispiel hat 6 Eingaben und einen Ausgabewert.

I_1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
I_2	0	0	0	1	1	0	0	1	1	0	1	0	1	1
I_3	1	1	1	0	1	0	0	1	1	0	0	0	1	1
I_4	0	1	0	0	1	0	0	1	0	1	1	1	0	1
I_5	0	0	1	1	0	1	1	0	1	1	0	0	1	0
I_6	0	0	0	1	0	1	0	1	1	0	1	1	1	0
O	1	1	1	1	1	1	0	1	0	0	0	0	0	0

1. *Wenden Sie den Perceptron-Lernalgorithmus an. Berechnen Sie die resultierenden Kanten-Gewichte!*

2. *Wenden Sie die den* DECISION-TREE-LEARNING-*Algorithmus an. Wie sieht der resultierende Entscheidungsbaum aus?*

3. *Vergleichen und kommentieren Sie beide Resultate!*

11 Was gibt es noch in der KI?

11.1 Expertensysteme

Wie früher schon erwähnt, stellte sich bald heraus (60er Jahre), daß für viele Anwendungen *Problem-spezifisches* Wissen erforderlich ist (Siehe Kapitel 1, DENDRAL).

Dies führte zu wissensbasierten Systemen und schließlich zu Expertensystemen.

Zunächst betrachtete man vorwärtsverkettete *Produktionssysteme*: Wenn-Dann Regeln + Fakten + einfache Konflikt Resolution. Dies wurde kommerziell im XCON-System von DEC ausgewertet: XCON war ein System zur Konfigurierung von Komponenten für DEC-Computer.

Anspruchsvollere Systeme basieren auf *Semantischen Netzen*, oder *Frames*. Dies sind graphische Notationen zur Beschreibung von Wissen, insbesondere *taxonomischen* Wissens.

Viele solcher Systeme wurden definiert. Die jeweiligen Semantiken waren unterschiedlich und hingen stark vom verwendeten Formalismus ab. Die meisten Semantik waren operational: es wurde direkt ein Algorithmus angegeben.

Semantische Netze sind in gewissen Bereichen sehr effizient, weil ihre Ausdrucksstärke eingeschränkt ist (z.B. keine Disjuntion, Negation, nur eingeschränkter Gebrauch von Quantoren). Dafür sind sie

- modular,
- als Graphen einfach handhabbar.

Semantische Netze sind äquivalent zur PL1 bzw Teilmengen davon.

Schließlich wurden auch nichtmonotone Varianten solcher Netze betrachtet:

Verebung mit Ausnahmen: unser Student/Erwachsener Beispiel aus Kapitel 8 etwa. Erwachsen sollte möglichst nicht auf Studenten vererbt werden. Aber wie sieht das bei komplizierteren Graphen aus?

Multiple Vererbungsnetze: Wurden von Horty, Thomason und Touretzky betrachtet ([HTT90]).

Terminologische Logiken: Fing schon mit KL-ONE an. Dies sind Teillogiken von PL1, in denen der Begriff des Termes im Vordergrund steht. Man fragt sich, welche Konstrukte für die NP-Vollständigkeit zuständig sind (in AL) und lässt sie weg. Siehe etwa den Artikel von Franz Baader im KI-Sonderheft [Baa96].

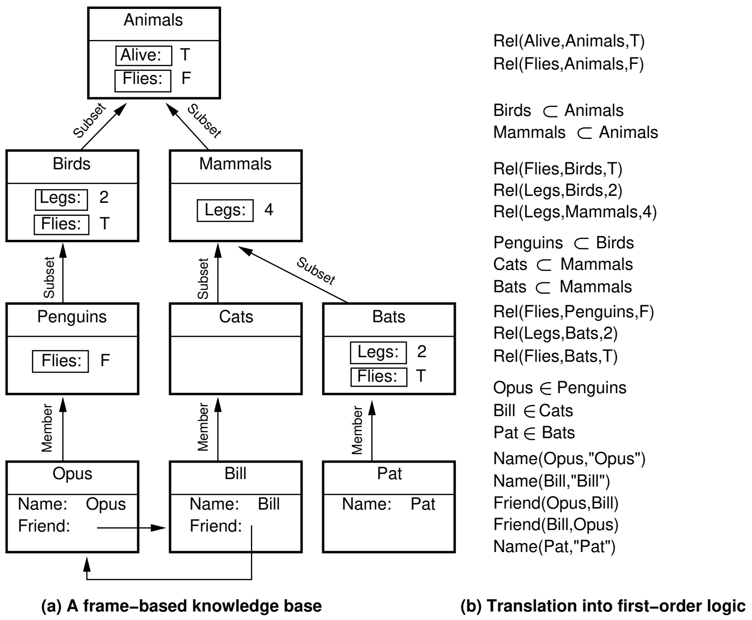


Abbildung 11.1: Frames versus PL1

Das klassische Expertensystem ist **MYCIN** von 1976 war für die medizinische Diagnostik entwickelt wurde. **MYCIN** benutzte einfache Regeln mit *Gewißheits-Faktoren*.

Jedes Expertensystem benutzt seinen eigenen ganz spezifischen Formalismus mit unterschiedlichen Konstrukten. Wir verweisen auf [Pup91].

11.2 Probabilistische Ansätze

Ein Satz ϕ ist hier, wie in der klassischen Logik, weiterhin entweder wahr oder falsch. Allerdings bekommt er ein Wahrscheinlichkeit

$$P(\phi) \in [0,1]$$

zugeordnet. Dies bedeutet, daß man zu einem gewissen Grad *glaubt*, daß der Satz wahr ist. Man benutzt folgende Axiome zur Berechnung dieser Grade:

$$\begin{array}{rcl} P(true) & = & 1 \\ P(false) & = & 0 \\ P(\phi \vee \psi) & = & P(\phi) + P(\psi) - P(\phi \wedge \psi) \end{array}$$

Außerdem gilt $\models \phi \leftrightarrow \psi$ genau dann wenn $P(\phi) = P(\psi)$.

Der Nachteil solcher Ansätze ist, dass sie hochgradig numerisch sind, und nicht klar ist, wo die Ausgangsdaten herkommen. menschliches Denken scheint eher qualitativ zu sein. Wir verweisen auf [Pea88].

Es wurde auch versucht, nichtmonotone Logiken als Spezialfall dieser Formalismen für Wahrscheinlichkeiten die gegen 0 und 1 gehen, zu deuten.

11.3 Fuzzy-Logik

Wurde 1965 von Lotfi Zadeh eingeführt. Hier bekommt jeder Satz ϕ einen *Grad an Wahrheit* Val zugeordnet:

$$Val(\phi) \in [0, 1].$$

Dies bedeutet, daß der Satz nur zu einem gewissen Grad wahr ist. Man benutzt folgende Axiome zur Berechnung dieses Grades:

$$\begin{array}{lcl} Val(true) & = & 1 \\ Val(false) & = & 0 \\ Val(\phi \wedge \psi) & = & min(\{Val(\phi), Val(\psi)\}) \\ Val(\phi \vee \psi) & = & max(\{Val(\phi), Val(\psi)\}) \end{array}$$

Gilt in dieser Logik $(\phi \vee \neg\phi) \leftrightarrow true$? Nein. In der Fuzzy-Logik geht es weniger um Unsicherheit, als vielmehr um die Behandlung von stetigen Variablen, also von Rausch-Phänomenen.

1978 wurde von Zadeh die *Possibility-Theorie* eingeführt: sie macht Fuzzy-Logik nichtmonoton (ähnlich wie die probabilistischen Ansätze).

11.4 Robotik

Der Name kommt aus dem tschechischen: Robota (die zu verrichtende Arbeit), Robotnik (Sklave) und wurde erstmals in einer Kurzgeschichte von J. Capek im Jahre 1917 verwendet.

Definition 11.4.1 (Roboter)
Ein Roboter ist ein künstlicher aktiver Agent, dessen Umgebung die reale Welt ist.

SHAKEY kann als erster, allerdings sehr eingeschränkter Roboter gesehen werden. In Bonn gibt es das WINO-Projekt unter der Leitung von Prof. Cremers.

11.5 Bilderkennen

Wir verweisen auf Prof. Priese, der sich z.B. mit *Verkehrszeichenerkennung* beschäftigt.

11.6 Spracherkennung

Wir verweisen hier auf Frau Prof. Harbusch (Institut für Computerlinguistik), die sich mit maschineller Übersetzung und dem Erkennen natürlicher Sprache beschäftigt.

Abbildung 11.2: Ein Roboter

In Saarbrücken gibt es das weltweit größte Projekt zu diesem Bereich: VERBMOBIL. Ein marktreifes System zum Übersetzen von deutschen und englischen Texten ist bei IBM entwickelt worden und wird von Langenscheidt vertrieben.

11.7 Kritisches zur KI

Hier sind vor allem die Bücher von H. Dreyfus zu nennen: *What Computers can't do* (1972) und *What Computers still can't do* (1992). Schließlich noch das von S. und H. Dreyfus: *Mind over Machine* (1986).

12 Ausgewählte Themen

12.1 Ordinalzahlen

Die wichtigste Eigenschaft natürlicher Zahlen ist deren Wohlfundiertheit: es gibt keine unendlich absteigenden Folgen von natürlichen Zahlen. Darauf beruht das Prinzip der vollständigen Induktion. Etwas allgemeiner:

Definition 12.1.1 (Wohlordnung)
$(X, \preceq)$ *heißt* Wohlordnung *(oder* $\preceq$ Wohlordnung *auf* X*), falls gilt :*

- $(X, \preceq)$ *ist eine bezüglich* $\preceq$ *partiell geordnete Menge,*
- *jede nichtleere Teilmenge* $Y \subseteq X$ *besitzt ein kleinstes Element.*

So bilden etwa die natürlichen Zahlen $(\mathbb{N}, \leq)$ mit ihrer natürlichen Ordnung eine Wohlordnung. Die rationalen Zahlen bilden keine Wohlordnung, man betrachte etwa das offene Intervall $]0,1[$. Es gibt in diesem Intervall kein kleinstes und kein größtes Element.

Wir wollen im folgenden die natürlichen Zahlen nur aus der leeren Menge $\emptyset$ zusammen mit einer sehr einfachen Mengenoperation definieren. Sei

$$0 :=_{def} \emptyset,\ 1 :=_{def} \{\emptyset\} = \{0\},\ 2 :=_{def} \{\emptyset, \{\emptyset\}\} = \{0, 1\}, \ldots$$

Jede natürliche Zahl ist also genau die Menge ihrer Vorgänger. Wir können diese Konstruktion fortsetzen und die folgenden Mengen definieren:

$$\begin{array}{l}
\omega :=_{def} \{0, 1, 2, \ldots, n, \ldots, \} \\
\omega + 1 :=_{def} \omega \cup \{\omega\}, \omega + 2, \ldots, \omega + n, \ldots, \omega + \omega \ {}_{def}{=:}\, \omega 2 \\
\omega 2 + 1, \ldots, \omega 2 + n, \ldots, \omega 3, \ldots, \omega 4, \ldots, \omega\omega \ {}_{def}{=:}\, \omega^2 \\
\omega^3, \ldots, \omega^n, \ldots, \omega^\omega, \ldots, \\
\omega^{\omega^\omega}, \ldots, \omega^{\omega^{\cdots^\omega}} \}\, n\text{-mal}, \ldots, \omega^{\omega^{\cdots^\omega}} \}\, \omega\text{-mal} \ \ {}_{def}{=:}\, \epsilon_0
\end{array}$$

Diese Mengen bezeichnet man auch als *Ordinalzahlen*. Man beachte, daß man sich die oben eingeführten Wohlordnungen auch als gewisse Ordnungen auf den natürlichen Zahlen $\mathbb{N}$ klar machen kann. Als Mengen gesehen sind ω und $\mathbb{N}$ natürlich identisch. Wenn wir aber ω (oder die anderen oben eingeführten Mengen) benutzen, dann meinen wir immer diese Mengen *zusammen mit ihrer natürlichen Ordnung* $\subseteq$, also eigentlich $(\omega, \subseteq)$, $(\omega + 1, \subseteq)$ etc. Analog bezeichnen wir mit $(\mathbb{N}, \leq)$ die natürlichen Zahlen mit ihrer natürlichen Ordnung. Man kann auf $\mathbb{N}$ aber auch andere Ordnungen definieren, z.B. $(\mathbb{N}, \leq_1)$ mit

$$i \leq_1 j \;:\Longleftrightarrow_{def} \begin{cases} \text{1. Fall: } i \text{ gerade und } j \text{ ungerade} \\ \text{2. Fall: } i \text{ gerade und } j \text{ gerade und } i \leq j \\ \text{3. Fall: } i \text{ ungerade, } j \text{ ungerade und } i \leq j \end{cases}$$

oder $(\mathbb{N}, \leq_2)$ mit

$$i \leq_2 j \;\;:\Longleftrightarrow_{def} \begin{cases} \text{1. Fall: } j = 17 \\ \text{2. Fall: } i \neq 17 \neq j \text{ und } i \leq j \end{cases}$$

Man sieht leicht, daß $(\mathbb{N}, \leq_1)$ ordnungsisomorph zu $\omega + \omega$ und $(\mathbb{N}, \leq_2)$ ordnungsisomorph zu $\omega + 1$ ist:

$$(\mathbb{N}, \leq_1) \cong (\omega + \omega, \subseteq), \;\; (\mathbb{N}, \leq_2) \cong (\omega + 1, \subseteq).$$

Dies ist kein Zufall; tatsächlich läßt sich jede Wohlordnung als eine Ordinalzahl darstellen. Wir schreiben auch $\alpha \leq \beta$ ($\alpha < \beta$) für $\alpha \subseteq \beta$ ($\alpha \subset \beta$).

Theorem 12.1.2 (Eigenschaften von Ordinalzahlen)
Es gelten folgende Aussagen:

1. *Für jede Ordinalzahl α ist $(\alpha, \subseteq)$ eine Wohlordnung.*

2. *Für jede Ordinalzahl α gilt: $\alpha = \{\beta : \beta \subset \alpha$ und β ist Ordinalzahl$\}$.*

3. *Zu jeder Wohlordnung $(X, \preceq)$ gibt es eine eindeutig bestimmte Ordinalzahl α so, daß $(X, \preceq) \cong (\alpha, \subseteq)$. α heißt der Wohlordnungstypus von $(X, \preceq)$.*

4. *Eine Ordinalzahl α ist entweder Nachfolger, d.h. α hat die Form $\alpha = \beta \cup \{\beta\}$ (man schreibt für $\beta \cup \{\beta\}$ auch $succ(\beta)$ oder $\beta + 1$) oder ist Limeszahl (hat keinen direkten Vorgänger, wie z.B. ω).*

5. *Eine Ordinalzahl repräsentiert die Klasse aller zu ihr ordnungsisomorphen Mengen.*

6. *Für je zwei Ordinalzahlen α, β gilt entweder $\alpha \geq \beta$ oder $\alpha \lneqq \beta$.*

Alle Ordinalzahlen die wir bisher betrachtet haben, kann man sich als Ordnungen auf $\mathbb{N}$ vorstellen. Insbesondere sind sie alle bijektiv (aber nicht ordnungserhaltend) auf $\mathbb{N}$ abbildbar. Abstrahiert man bei einer Menge vollständig von allen möglichen Ordnungen, so landet man bei den Kardinalzahlen:

Definition 12.1.3 (Kardinalität)
Kardinalzahlen sind Ordinalzahlen bei denen man von der Wohlordnung abstrahiert. Eine Kardinalzahl repräsentiert die Klasse aller zu ihr bijektiven Mengen. Sie ist also Repräsentant dieser Äquivalenzklasse.

Man sagt $card(M) \leq card(N)$ falls es eine injektive Abbildung von M nach N gibt.

Jede natürliche Zahl repräsentiert eine Kardinalzahl. Die kleinste unendliche Kardinalzahl ist ω, man bezeichnet sie auch als $\aleph_0$, um sie von der wohlgeordneten Menge $(\omega, \subseteq)$ zu unterscheiden. Mengen der Kardinalität $\aleph_0$ heißen auch *abzählbar*.

Das berühmte *Cantor'sche Diagonal-Argument* zeigt, daß die reellen Zahlen $\mathbb{R}$ eine echt größere Kardinalität als $\aleph_0$ haben.

Definition 12.1.4 (Hilbert's Hotel)
Gegeben sei ein Hotel mit einem sehr langen Gang, nämlich mit abzählbar vielen Zimmern der Nummern $1, 2, \ldots, n, n+1, \ldots$ *Alle Zimmer sind frei.*

Nun kommen Busse mit endlich vielen Fahrgästen an. Offensichtlich können diese alle untergebracht werden. Schließlich kommt ein Bus mit abzählbar unendlich vielen Fahrgästen an, und das Hotel ist voll. Was passiert, wenn nun noch ein Bus ankommt? Oder sogar einer mit nochmal abzählbar unendlich vielen Fahrgästen?

Alle Fahrgäste können untergebracht werden, denn man kann auf einen Schlag unendlich viele Zimmer frei machen: man telefoniert einfach alle Zimmer an, und bittet jeden Gast in Zimmer i auf das Zimmer $2i$ umzuziehen.

Theorem 12.1.5 (Transfinite Induktion)
Sei $\mathcal{P}$ eine Eigenschaft die auf Ordinalzahlen zutreffen kann oder nicht. Falls:

1. $\mathcal{P}(0)$ *und*
2. $\forall \beta + 1 : (\mathcal{P}(\beta)) \implies \mathcal{P}(\beta + 1)$ *und*
3. $\forall \lambda :$ $(\lambda$ *ist Limeszahl* $\wedge$ $(\forall \zeta < \lambda : \mathcal{P}(\zeta)) \implies \mathcal{P}(\lambda)$,

so gilt auch

$$\forall \alpha : \mathcal{P}(\alpha)$$

Die Addition und Multiplikation von Ordinalzahlen kann man sich konstruktiv vorstellen, wenn man sie auf bekannten Mengen definiert. Dies tun wir mit den nächsten beiden Definitionen.

Definition 12.1.6 (Addition von Wohlordnungen: $(X, \preceq) \oplus (X', \preceq')$**)**
Seien $(X, \preceq)$ und $(X', \preceq')$ Wohlordnungen. Wir definieren eine Wohlordnung $\preceq_{neu}$ auf der Menge $X \cup X'$ durch

$$y_1 \preceq_{neu} y_2 \;:\Longleftrightarrow_{def}\; \begin{cases} \text{1. Fall: } y_1, y_2 \in X \text{ und } y_1 \preceq y_2 \\ \text{2. Fall: } y_1, y_2 \in X' \text{ und } y_1 \preceq' y_2 \\ \text{3. Fall: } y_1 \in X,\, y_2 \in X' \end{cases}$$

Wir bezeichnen $(X \cup X', \preceq_{neu})$ als ordinale Summe von $(X, \preceq)$ und $(X', \preceq')$ und schreiben dafür $(X, \preceq) \oplus (X', \preceq')$.

Intuitiv setzt man also beide Wohlordnungen disjunkt hintereinander.

Definition 12.1.7 (Multiplikation von Wohlordnungen: $(X,\preceq)\odot(X',\preceq')$**)**
Seien $(X,\preceq)$ und $(X',\preceq')$ Wohlordnungen. Wir definieren eine Wohlordnung $\preceq_{neu}$ auf dem kartesischen Produkt $X \times X'$ durch

$$(x_1,y_1) \preceq_{neu} (x_2,y_2) \;:\Longleftrightarrow_{def}\; \begin{cases} \text{1. Fall: } y_1 \neq y_2 \text{ und } y_1 \preceq' y_2 \\ \text{2. Fall: } y_1 = y_2 \text{ und } x_1 \preceq x_2 \end{cases}$$

Wir bezeichnen $(X \times X', \preceq_{neu})$ als ordinales Produkt von $(X,\preceq)$ und $(X',\preceq')$ und schreiben dafür $(X,\preceq)\odot(X',\preceq')$.

Intuitiv bedeutet das ordinale Produkt gerade die antilexikographische Ordnung. Damit kann man sich in vielen Fällen einige nützliche Identitäten klarmachen.

Beispiel 12.1.8 ($+$ ist nicht kommutativ)
Sei $n \in \mathbb{N}$. Man betrachte etwa $n+\omega$. Dies ist dasselbe wie $(n,\leq)\oplus(\mathbb{N},\leq)$, was aber offensichtlich ordnungsisomorph zu $(\mathbb{N},\leq)$, also ω ist. Was ergibt sich nun für $\omega+n$? $(\mathbb{N},\leq)\oplus(n,\leq)$ ist offensichtlich nicht ordnungsisomorph zu ω, also

$$n+\omega < \omega+n.$$

Man zeigt leicht, daß die Multiplikation kommutativ und assoziativ ist. Außerdem ist $\omega+\omega=\omega\cdot 2$. Was ergibt aber $2\cdot\omega$? Wir erhalten durch Einsetzen in die Definition der Multiplikation $2\cdot\omega=\omega$. Gilt die Distributivität? Also etwa $\omega\cdot(\omega+\omega) \;=\; (\omega+\omega)\cdot\omega$? Der erste Teil der Gleichung vereinfacht sich zu $\omega\cdot(\omega\cdot 2)$ also zu $\omega^2\cdot 2$. Für den zweiten Teil ergibt sich $(\omega+\omega)\cdot\omega \;=\; (\omega\cdot 2)\cdot\omega \;=\; \omega\cdot(2\cdot\omega) \;=\; \omega\cdot\omega \;=\; \omega^2$.

Definition 12.1.9 (Der vergiftete Keks, endlicher Fall)
Gegeben sei eine $n \times m$ Matrix von Keksen. Keks $(1,1)$ ist vergiftet. 2 Spieler können abwechselnd eine Position (i,j) wählen und alle Kekse oberhalb i und rechts von j wegnehmen. Gewonnen hat, wer den vergifteten Keks **nicht** *nehmen muß. Man beachte Abbildung 12.1, dort wurde der Keks (i,j) ausgewählt.*

Theorem 12.1.10 (Gewinnstrategie für Spieler 1)
Es gibt eine Gewinnstrategie für Spieler 1.

Beweis: Angenommen Spieler 1 wählt als ersten Zug den Keks in der rechten oberen Ecke (also (n,m)). Falls sich so eine Gewinnposition ergibt, sind wir schon fertig.

Falls sich keine für Spieler 1 ergibt, kann Spieler 2 einen Gewinnzug wählen. Diesen kann aber Spieler 1 dann auch schon im ersten Zug wählen, und hat daher in jedem Fall einen Gewinnzug.

■

Was ist aber der Gewinnzug in einem konkreten Spiel? Das ist sehr schwierig und nur für einige Spezialfälle gelöst:

- $2\times n$: Hier wählt man $(2,n)$.
- $n\times n$: Hier wählt man $(2,2)$.

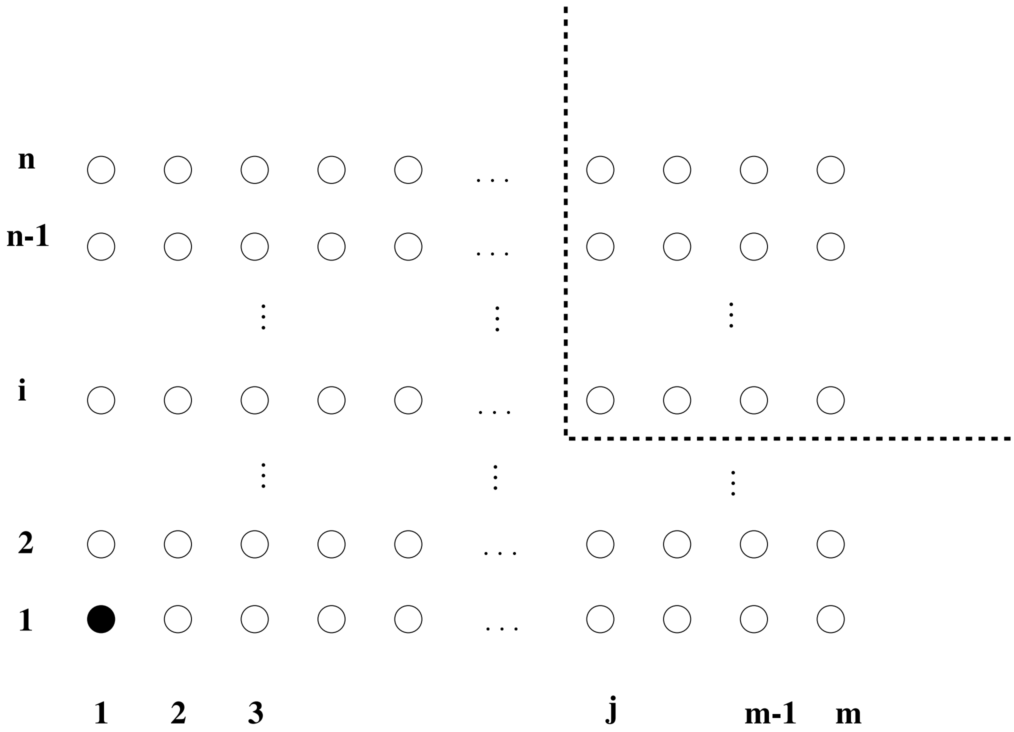


Abbildung 12.1: Der vergiftete Keks.

Für das 3×4-Spiel ist $(2, 3)$ der Gewinnzug. *Man kann dieses Spiel auch mit beliebigen Ordinalzahlen spielen.* Das einfachste Spiel ist das $2 \times \omega$-Spiel. Angenommen der erste Spieler wählt $(1, i)$. Dann kann der zweite Spieler $(2, i - 1)$ wählen. Wählt Spieler 1 aber $(2, i)$, so kann Spieler 2 die Position $(1, i + 1)$ wählen. So ergibt sich eine Gewinnstrategie für Spieler 2.

Wie ist es mit dem $\omega \times \omega$-Spiel? Hier kann Spieler 1 immer auf das $2 \times \omega$-Spiel reduzieren, also besitzt er eine Gewinnstrategie.

Enden solche unendlichen Spiele immer in endlicher Zeit?

12.2 Das Auswahlaxiom

Betrachten wir einmal die Definition der Kardinalzahlen. Man könnte doch auch versuchen, die Kardinalität einer Menge M als die kleinste Ordinalzahl α, so daß α isomorph zu $(M, \preceq)$ ist (wobei $\preceq$ irgendeine Wohlordnung auf M ist) zu definieren. Ist diese Definition zur ursprünglichen äquivalent? Nur, wenn man weiß, daß sich alle Mengen wohlordnen lassen.

- Jede Menge läßt sich wohlordnen.

- Für je zwei Mengen M, N gilt: $card(M) \leq card(N)$ oder $card(M) > card(N)$.
- Jede unendliche Menge besitzt eine abzählbare Teilmenge.

Alle diese Aussagen sind äquivalent zum **Auswahlaxiom**:

(AC) Sind $(X_i)_{i \in I}$ sämtlich nichtleere Mengen, so gilt: $\prod_{i \in I} X_i \neq \emptyset$

Man beachte, daß es hierbei auf die Index-Menge I ankommt. Ist I endlich, so ist die Aussage trivial (und beweisbar). Aber schon wenn I abzählbar ist, ist diese Variante (man nennt sie auch **(AC)**$_{abz}$) nichttrivial. **(AC)**$_{abz}$ ist äquivalent zu

Abzählbare Vereinigungen von abzählbaren Mengen sind selbst wieder abzählbar.

Kann man das Auswahlaxiom beweisen? Diese Frage ist ähnlich zu der, ob man das Paralellen-Postulat aus den anderen euklidischen Axiomen beweisen kann. Erst Gauß und Lobatschewskji konnten durch die Konstruktion nichteuklidischer Geometrien zeigen, daß dies nicht möglich ist.

Was aber entspricht in unserem Fall den euklidischen Axiomen? Die Mathematik gründet sich auf die Mengenlehre. Diese läßt sich in Prädikatenlogik erster Stufe mit Gleichheit formalisieren (siehe Kapitel 5). Als einziges nichtlogisches Symbol kommt man mit “$\in$” aus. Folgende Axiome haben sich im Verlauf der letzten 100 Jahre durchgesetzt:

Existenz: $\exists x \, x \doteq x$,

Extensionalität: $\forall x \forall y \, (\forall z \, (z \in x \leftrightarrow z \in y) \rightarrow x \doteq y)$,

Komprehension: $\forall z \exists y \forall x \, (x \in y \leftrightarrow (x \in z \wedge \phi))$, wobei y nicht frei in ϕ vorkommt,

Paarmenge: $\forall x \forall y \exists z \, (x \in z \wedge y \in z)$,

Vereinigung: $\forall x \exists y \forall z \forall z' \, ((z' \in z \wedge z \in x) \rightarrow z' \in y)$,

Ersetzung: $\forall w \, \big((\forall x \in w \exists! y \; \phi(x,y)) \rightarrow \exists z \forall x \in w \exists z' \in z \; \phi(x,z') \big)$, wobei z nicht frei in ϕ vorkommt,

Fundierung: $\forall x \, \big(\exists y \, (y \in x) \rightarrow \exists y (y \in x \wedge \neg \exists z (z \in x \wedge z \in y)) \big)$,

Das Komprehensionsaxiom gestattet die Konstruktion von Mengen mit einer Eigenschaft ϕ, üblicherweise schreibt man dafür $y = \{x : \; x \in z \wedge \phi(y)\}$. Ohne die zusätzliche Menge z wäre Russells Antinomie, also die Menge $\{x : \; x \notin x\}$ möglich, die sofort zu Widersprüchen führt.

Das Ersetzungsaxiom sagt aus, daß das Bild einer Funktion deren Definitionsbereich eine Menge ist, auch wieder eine Menge ist.

Die Fundierung schließlich schließt Mengen aus die sich selbst enthalten ($M \in M$).

Das obige Axiomensystem heißt **ZF**, nach Ernst Zermelo und Abraham Fraenkel.

Theorem 12.2.1 (Gödel 1938)
Die Negation des Auswahlaxioms ist nicht aus **ZF** *herleitbar:* $ZF \nvdash \neg AC$.

Theorem 12.2.2 (Cohen 1963)
Das Auswahlaxiom ist nicht aus **ZF** *herleitbar:* $ZF \nvdash AC$.

12.3 Nicht-Standard Kartenspiele

Definition 12.3.1 (Ein Nicht-Standard Kartenspiel)
Die Bank hält ein Kartenspiel und gibt dem Spieler Karten aus. In jeder Runde werden höchstens endlich viele Karten ausgegeben. Der Spieler wählt unter seinen Karten eine einzige aus und legt sie weg. Das Spiel endet nach ω*-vielen Schritten.*

Der Spieler hat gewonnen, wenn er am Ende keine Karten mehr hält, ansonsten hat er verloren.

Erstaunlicherweise gibt es eine einfache Gewinnstrategie für den Spieler: *Man sortiert die Karten so wie man sie bekommt und gibt sie in dieser Reihenfolge wieder her.* Dafür braucht man allerdings ein *unbegrenztes Gedächtnis*. Geht es auch mit begrenztem Gedächtnis?

Hier ist etwas Vorsicht geboten. Natürlich muß sich der Spieler seine momentane Hand anschauen. Auch dafür braucht er eigentlich unbegrenzten Speicher. Diesen unterscheiden wir aber von seinem Gedächtnis, daß ja etwas über vorausgegangene Spielzüge sagt.

Es zeigt sich, daß Gewinnstrategien in gewisser Weise von der Anzahl der Karten (abzählbar oder überabzählbar) abhängen.

Theorem 12.3.2 (Abzählbare Kartenmenge)
Für eine abzählbare Kartenmenge gibt es eine Gewinnstrategie für den Spieler, die ohne Gedächtnis und mit begrenztem Speicher (nämlich 2 Einheiten) auskommt.

Beweis: Wir numerieren die Karten bevor das Spiel losgeht, und werfen immer die Karte mit der kleinsten Nummer weg.

■

Wie ist das aber mit einer überabzählbaren Kartenmenge?

Theorem 12.3.3 (Überabzählbare Kartenmenge mit Gedächtnis)
Für eine überabzählbare Kartenmenge gibt es eine Gewinnstrategie für den Spieler, wenn man ein begrenztes Gedächtnis (1 Einheit) und begrenzten Speicher (nämlich 2 Einheiten) zulässt.

Beweis: Wir wenden das Auswahlaxiom an und wohlordnen unsere Kartenmenge bevor das Spiel losgeht. Im ersten Schritt werfen wir die Karte mit der höchsten Nummer α weg und merken uns diese im Gedächtnis. Dann gibt es in jeder Runde zwei Fälle:

> **1. Fall:** Es gibt auf der Hand eine Karte β mit $\beta < \alpha$. Dann werfen wir die Karte mit maximalem solchem β weg.

2. Fall: Falls der 1. Fall nicht auftritt, werfen wir die Karte mit der höchsten Nummer weg und schreiben diese in das Gedächtnis (α wird also überschrieben).

Man beachte, daß diese Strategie nur klappt, weil die Kartenmenge wohlgeordnet ist, es also keine unendlich langen, absteigenden Ketten gibt. ■

Was passiert, wenn man kein Gedächtnis zulässt?

Theorem 12.3.4 (Überabzählbare Kartenmenge ohne Gedächtnis)
Für eine überabzählbare Kartenmenge gibt es keine Gewinnstrategie für den Spieler, wenn man kein Gedächtnis zulässt. Auch nicht, wenn der Speicher unbegrenzt ist.

Beweis: Eine Gewinnstrategie ist eine Funktion F die aus einer Menge von Karten eine auswählt (hier ist schon wichtig, daß es kein Gedächtnis gibt). Angenommen es gibt eine solche Funktion. Dann definieren wir für eine Karte x

$$D_x := \{y \neq x : \; F(\{x, y\}) = y\}.$$

Es genügt nun zu zeigen, daß es ein x' mit card($D_{x'}$)=∞ gibt. Denn dann kann die Bank zuerst die Karte x' und eine weitere Karte y aus $D_{x'}$ dem Spieler ausgeben. Der Spieler muß dann y weglegen und die Bank wird in jeder Runde eine weitere Karte aus $D_{x'}$ ausgeben. Diese muss der Spieler gemäß seiner Strategie weglegen. Er behält so die Karte x' immer auf der Hand und verliert das Spiel.

Sei C eine abzählbare Teilmenge der Kartenmenge und sei

$$U := \bigcup_{x \in C} D_x \; \cup C$$

Ist eines der D_x mit $x \in C$ unendlich, so sind wir fertig. Seien also alle D_x endlich für $x \in C$. Dann ist U abzählbar. Dann gilt aber für jedes x' aus dem Komplement von U (dieses Komplement ist ja überabzählbar)

$$C \subseteq D_{x'}.$$

■

12.4 Kugel-Verdopplung

Wir betrachten Punkt-Mengen im $\mathbb{R}^n$ und wollen definieren, wann zwei solche Mengen *gleich groß* sind. Zur Erinnerung eine kurze Auffrischung der Kenntnisse aus Linearer Algebra.

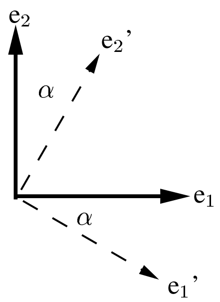
Definition 12.4.1 (GL$_n$, O$_n$)
Die invertierbaren (d.h. Determinante $\neq 0$) $n \times n$-Matrizen über $\mathbb{R}$ bilden eine Gruppe **GL**$_n$ *bezüglich der wohlbekannten Matrix-Multiplikation. Solch eine Matrix ist ein linearer Isomorphismus auf $\mathbb{R}^n$ vermöge*

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^n a_{1,i}x_i \\ \sum_{i=1}^n a_{2,i}x_i \\ \vdots \\ \sum_{i=1}^n a_{n,i}x_i \end{pmatrix}$$

Schließlich bilden die Abbildungen A aus $\mathbf{GL}_n$ mit $A^{-1} = A^t$ die orthogonale Gruppe $\mathbf{O}_n$.

Etwas anschaulicher, kann man sich $\mathbf{O}_n$ als diejenigen Isomrphismen vorstellen, die die kanonische Orthonormalbasis in eine andere Orthonormalbasis überführen.

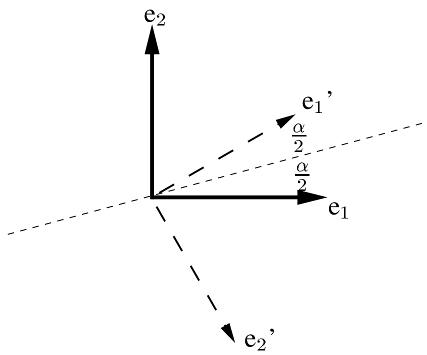
Betrachten wir etwa den $\mathbb{R}^2$. Man kann die Orthonormalbasis einfach um einen Winkel α drehen:



Solch eine Abbildung hat dann die Form

$$\begin{pmatrix} cos(\alpha) & -sin(\alpha) \\ sin(\alpha) & cos(\alpha) \end{pmatrix}$$

und die Determinante $+1$. Aber es gibt noch eine zweite Möglichkeit:



Dies ist eine Spiegelung an der eingezeichneten Achse! Solche Abbildungen haben die Form

$$\begin{pmatrix} cos(\alpha) & sin(\alpha) \\ sin(\alpha) & -cos(\alpha) \end{pmatrix}$$

und die Determinante -1. Man bezeichnet mit O_n^+ die Gruppe der orthogonalen Matrizen mit Determinante $+1$: dies sind genau die Rotationen.

Wie sehen die entsprechenden Abbildungen im $\mathbb{R}^3$ aus? Man lege einen Körper so, daß einer seiner Punkte mit dem Ursprung zusammenfällt. Dann bewege man ihn, so, daß der Ursprung fix bleibt. Jede solche Bewegung kann durch eine Rotation aus $\mathbf{O}_3^+$ realisiert werden. $\mathbf{O}_3$ ist etwas allgemeiner, es sind *rotierte Spiegelungen*. Sie haben die Form

$$\begin{pmatrix} \pm 1 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) \\ 0 & sin(\alpha) & cos(\alpha) \end{pmatrix}$$

Schließlich brauchen wir noch die Gruppe der Translationen $\mathcal{T}_n$:

$$\mathcal{T}_n = \{T_w : \ w \in \mathbb{R}^n\},$$

wobei $T_w : \mathbb{R}^n \to \mathbb{R}^n;\ v \mapsto v + w$.

Nun haben wir alles zusammen um zwei Punktmengen als *gleich groß* zu klassifizieren. Wir nehmen eine Menge, translatieren sie in den Ursprung und wenden eine Rotation an. Kann man zwei Mengen auf diese Weise identisch machen, so nennen wir sie gleich groß. Wir betrachten also

$$\mathcal{M}_n = \mathbf{O}_n\mathcal{T}_n = \{\sigma\tau : \ \sigma \in \mathbf{O}_n,\ \tau \in \mathcal{T}_n\},$$

die Gruppe der *distanz-invarianten* Transformationen.

Definition 12.4.2 (Äquivalenz)
Zwei Mengen $A, B \subseteq \mathbb{R}^n$ heißen äquivalent, falls sie so in endlich viele Mengen aufgeteilt werden können, die jeweils paarweise durch Abbildungen aus $\mathcal{M}_n$ ineinander übergeführt werden können. Wir schreiben dafür:

$$A \underset{\mathcal{M}_n}{\overset{fin}{\approx}} B$$

Überraschendwerweise folgt aus dem Auswahlaxiom das folgende paradoxe Resultat:

Theorem 12.4.3 (Kugelverdopplung)
*Sei **B** die Einheitskugel im $\mathbb{R}^3$ (**B**=$\{v : \| \ v \ \| \leq 1\}$). Dann gibt es $A_1, A_2 \subsetneqq B$ mit $A_1 \cap A_2 = \emptyset$ und*

$$A_1 \underset{\mathcal{M}_3}{\overset{fin}{\approx}} A_2 \underset{\mathcal{M}_3}{\overset{fin}{\approx}} B$$

Mit anderen Worten: Eine Kugel kann so in endlich viele Teile aufgespalten werden, daß durch rotieren und translatieren dieser Teile beim wiederzusammensetzen 2 Kugeln herauskommen!

Man kann sogar zeigen, daß je zwei beschränkte Teilmengen des $\mathbb{R}^3$ mit inneren Punkten äquivalent sind.

Warum ist das für uns so paradox? *Weil wir immer ein Maß im Kopf haben.* Wir würden gerne so argumentieren: Eine Kugel hat einen gewissen Rauminhalt. Wenn wir die Kugel aufspalten, ergibt die Summe der Rauminhalte der einzelnen Teile genau den der Kugel. Durch Translationen und Rotationen ändert sich dieser Rauminhalt sicher nicht. Also kann man diese Teile nicht zu zwei Kugeln, die ja doppelten Rauminhalt haben, zusammensetzen.

Definition 12.4.4 (Endliches additives Maß)
Ein Maß ist eine Funktion $\mu : \mathbb{R}^n \to \mathbb{R}_0^+ \cup \{\infty\}$ *mit:*

($\mathbf{A_{endl}}$) *Sei* $A, B \subseteq \mathbb{R}^n$ *und* $A \cap B = \emptyset$*. Dann ist* $\mu(A \cup B) = \mu(A) + \mu(B)$*.*

(Inv) *Sei* $A, B \subseteq \mathbb{R}^n$*. Falls es eine Transformation* $\rho \in \mathcal{M}_n$ *gibt, so daß* $A^\rho = B$*, so gilt* $\mu(A) = \mu(B)$*.*

(Norm) *Es ist* $\mu(\{x \in \mathbb{R}^n : \| x \| \leq 1\}) = 1$*.*

Es stellt sich nun die Frage, ob es solch ein Maß überhaupt gibt. Hier einige Antworten.

Theorem 12.4.5 (Hausdorff, 1914)
($A_{\mathbf{endl}}$), (Inv) und (Norm) sind unvereinbar im $\mathbb{R}^n$ *für* $n \geq 3$*.*

Theorem 12.4.6 (Banach, 1923)
($A_{\mathbf{endl}}$), (Inv) und (Norm) sind vereinbar im $\mathbb{R}^n$ *für* $n \leq 2$*.*

Der letzte Satz sagt insbesondere, daß 2 Segmente auf einer Geraden genau dann äquivalent sind, wenn sie die gleiche Länge haben. Analog sind 2 Kreise in der Ebene genau dann äquivalent sind, wenn sie den gleichen Radius haben.

Allerdings sind 2 beliebige Kugeln im $\mathbb{R}^n$ (für $n \geq 3$) immer äquivalent (unabhängig von ihrem Radius). Was ändert sich denn so radikal beim Übergang von $\mathbb{R}^2$ zu $\mathbb{R}^3$? Erstaunlicherweise ändern sich *nicht die topologischen Eigenschaften von* $\mathbb{R}^3$ *als normiertem Vektorraum, sondern die Gruppe* $\mathbf{O}_3^+$. Denn die $\mathbf{O}_3^+$ enthält eine freie Untergruppe vom Rang 2, und damit freie Untergruppen jedes beliebigen Ranges!

Sei etwa $\mathcal{F}$ die freie Gruppe in 2 Erzeugenden σ, τ, d.h.

$$\mathcal{F} := \{w : \ w \text{ ein Wort über } \sigma, \tau, \sigma^{-1}, \tau^{-1}\}$$

Als einzige Relationen gelten: $\sigma\sigma^{-1} = 1 = \sigma^{-1}\sigma$, $\tau\tau^{-1} = 1 = \tau^{-1}\tau$, $\sigma^n\sigma^m = \sigma^{m+n}$, $\tau^n\tau^m = \tau^{n+m}$. Offenbar gilt

$$\mathcal{F}_0 := \{\sigma^i\tau^i : \ i \in \mathbb{N}\} \subseteq \mathcal{F}$$

und diese Menge $\mathcal{F}_0$ erzeugt eine freie Untergruppe vom Rang ∞. Aber auch die Menge $\mathcal{F}(1,2)$ erzeugt durch

$$\{\sigma\tau, \sigma^2\tau^2\}$$

ist eine freie Untergruppe von $\mathcal{F}$ vom Rang 2. Zusammengenommen ergibt sich also:

$$\mathcal{F}_0 \subsetneqq \mathcal{F} \cong \mathcal{F}(1,2) \subsetneqq \mathcal{F}_0$$

was man auch das *Paradoxon der freien Gruppe vom Rang 2* nennt. Tatsächlich basieren alle paradoxen Zerlegungen auf entsprechenden Untergruppen der $\mathbf{O}_3^+$.

12.5 Gibt es unendlich kleine Zahlen?

Wir stellen uns die folgende Frage: Gibt es eine Funktion $\delta : \mathbb{R} \to \mathbb{R}$ mit

$$\int_a^b \delta(x)\partial x = \begin{cases} 0, & \text{falls } a < b < 0, \\ 1, & \text{falls } a < 0 < b, \\ 0, & \text{falls } 0 < a < b, \end{cases}$$

Intuitiv beschreibt δ eine Punktladung der Größe 1 (Dirac'sche Delta-Funktion). Eine schöne Eigenschaft solch einer Funktion ist die folgende. Ist f irgendeine andere Funktion, so gilt:

$$\int_{-\infty}^{\infty} \delta(x) f(x)\partial x \;=\; f(0).$$

Die Ableitung $\frac{\partial\delta}{\partial x}$ beschreibt genau ein *Dipolmoment*.

Man sieht leicht, daß es solch eine Funktion nicht geben kann, denn in jeder Umgebung des Ursprungs müsste sie identisch verschwinden, und im Ursprung selbst müsste sie unendlich groß sein.

Die Idee in der Nichtstandard-Analysis ist es, $\mathbb{R}$ zu einem Körper $\mathbb{R}^*$ zu erweitern, in dem man unendlich kleine Zahlen hinzunimmt. Dies hat natürlich noch weitere Auswirkungen. Ist z.B. η infinitesimal (unendlich klein), so ist dann natürlich $\frac{1}{\eta}$ unendlich groß. Man kann dies so tun, daß jede klassische Funktion $f : \mathbb{R}^n \to \mathbb{R}$ zu einer Funktion $f^* : \mathbb{R}^{n\,*} \to \mathbb{R}^*$ auf *eindeutige* Weise fortgesetzt werden. Darüberhinaus erfüllt $\mathbb{R}^*$ sogar genau die gleichen Sätze wie $\mathbb{R}$ (man spricht von einer elementaren Erweiterung).

Um also einen klassischen Satz in $\mathbb{R}$ zu beweisen, kann man es auch gleich in $\mathbb{R}^*$ versuchen, falls das einfacher ist. In $\mathbb{R}^*$ spielt sich viel mehr Leben ab. Z.B. kann man hier leicht die obige Funktion δ definieren:

$$\delta(x) := \begin{cases} 0, & \text{für } x \neq 0 \text{ und nicht infinitesimal,} \\ \infty, & \text{für } x = 0, \\ \text{geeignet infinitesimal} \neq 0, & \text{für infinitesimales } x. \end{cases}$$

Man beachte, daß ∞ ein *richtiges* Element von $\mathbb{R}^*$ ist.

Auch der Begriff der Stetigkeit wird in $\mathbb{R}^*$ sehr anschaulich: für unendlich nahe Argumente sind auch die Funktionswerte unendlich nahe beisammen.

Nicht-Standard-Analysis entstand in den 60'ern (Abraham Robinson). Viele etwas *hemdsärmelige* Argumente der Physiker konnten so nachträglich gerechtfertigt werden.

12.6 Zero-Knowledge-Spiele

Wir betrachten 2 Spieler von denen jeder eine natürliche Zahl (die 0 eingeschlossen) zugewiesen bekommt (jeder kennt nur seine eigene, nicht die des anderen). Beide bekommen gesagt, daß es sich um aufeinanderfolgende Zahlen handelt. Alle 10 Sekunden schlägt der Schiedsrichter einen Gong: wer die Zahl des anderen weiss, muss sie direkt nach dem ertönen des Gongs sagen und hat gewonnen.

Theorem 12.6.1 (Gewinnstrategie)
Der Spieler mit der Zahl n weiß nach $n + 1$-Gongs die Zahl seines Gegenspielers.

Beweis: Wir machen Induktion nach n. Für $n = 1$ ist die Sache klar. Sei also das Theorem für alle n_0 echt kleiner als n schon bewiesen. Ist die andere Nummer $n-1$ (die eigene ist ja n) so würde der Gegenspieler nach n Gongs das Spiel beenden. Geschieht dies nicht, so ist seine Nummer $n + 1$.

■

Das paradoxe an der Situation ist, daß uns jeder Gong scheinbar keine zusätzliche Information liefert. Hier ein anderes Beispiel.

Beispiel 12.6.2 (Studenten in der KI-Vorlesung)
30 Studenten kommen in die KI-Vorlesung. 20 von ihnen haben ein Asche-Kreuz auf der Stirn, können es aber nicht sehen. Der Dozent will die Intelligenz seiner Studenten messen und schlägt folgendes Spiel vor. Alle 30 Sekunden frage ich, ob jetzt jemand weiss, ob er selbst ein Kreuz auf der Stirn hat oder nicht. Wer es weiss, darf sich melden und bekommt einen Schein.

Nach 20 Minuten hat sich noch nichts getan. Da sagt der Dozent: "Also gut, es war offensichtlich zu schwer. Ich gebe Ihnen einen Tip: Mindestens einer von Ihnen hat ein Kreuz auf der Stirn."

Und tatsächlich, nach genau 10 Minuten melden sich alle Studenten und wissen die Lösung.

Das Paradoxon besteht darin, welche zusätzliche Information der Dozent gegeben hat. Offensichtlich wusste doch jeder, das es mindestens einen mit einen Kreuz auf der Stirn gibt (sogar daß es derer mindestens 19 gibt).

12.7 Aufgaben

Aufgabe 12.1 (Hilberts Hotel)
Gegeben sei ein Hotel mit abzählbar vielen Zimmern $1, 2, \ldots$. Alle Zimmer sind frei. Es kommen Busse an, deren Fahrgäste alle im Hotel untergebracht werden sollen. Wie geht das, wenn

1. *ein ω–Decker-Bus mit ω vielen Sitzplätzen auf jedem Deck bzw.*
2. *ω viele solcher ω–Decker-Busse*

kommen?

Aufgabe 12.2 ($\alpha \times \beta$–Spiel)
Eine $\alpha \times \beta$–Matrix ist mit Keksen belegt. Zwei Spieler wählen abwechselnd ein Feld (i, j) aus, auf dem noch ein Keks liegt. Sie dürfen dann alle Kekse, die sowohl rechts von Spalte i als auch oberhalb von Zeile j (jeweils einschließlich) liegen, aufessen. Aber Vorsicht, der Keks auf (1,1) ist vergiftet. Wer den nehmen muß, hat verloren.

1. *Zeigen Sie, daß $\alpha \times \beta$–Spiele immer nach endlich vielen Schritten zu Ende gehen.*

2. *Wer gewinnt das Spiel bei folgenden Matrix-Größen?*
   (1) $2 \times (\omega + 1)$
   (2) $\omega \times \omega$
   (3) $(\omega + 1) \times (\omega + 1)$

3. *Schreiben Sie ein Programm, das den Gewinnzug für den ersten Spieler berechnet, falls von einer endlichen* $m \times n$*–Matrix ausgegangen wird.*
   Was ist der Gewinnzug bei folgenden Matrix-Größen?
   (1) 3×5
   (2) 5×6
   (3) 7×7

Aufgabe 12.3 (Nicht-meßbare Mengen)

1. *Betrachten Sie die Relation* $\sim$ *auf dem Intervall* $[0, 1[\subseteq \mathbb{R}$*, definiert durch:* $x \sim y$ *gdw.* $x - y \in \mathbb{Q}$*. Zeigen Sie, daß* $\sim$ *eine Äquivalenzrelation ist!*

2. *Unter Anwendung des Auswahlaxioms (AC) bilden wir nun eine Menge* $M \subseteq [0, 1[$*, die von jeder Äquivalenzklasse bezüglich* $\sim$ *genau ein Element enthält. Läßt sich jedes* $x \in [0, 1[$ *eindeutig als Summe zweier Zahlen* $m \in M$ *und* $q \in \mathbb{Q}$ *darstellen? Warum?*

3. *Bilden Sie nun zu jedem* $q \in \mathbb{Q}$ *die Menge* $M_q = \{frac(m + q) \mid m \in M\}$*, wobei* $frac(x)$ *der Nachkommateil von* x *sei. Läßt sich* $[0, 1[$ *als Vereinigung abzählbar unendlich vieler disjunkter Mengen* M_q *darstellen? Wenn ja, wie?*

4. *Zeigen Sie, daß* M *nicht meßbar ist, d.h.* $\mu(M)$ *existiert nicht!*

Benutzen Sie dazu die folgende Definition eines Maßes. Ein Maß ist eine Funktion $\mu : \mathbb{R}^n \to \mathbb{R}_0^+ \cup \{\infty\}$ *mit:*

($\mathbf{A_{abz}}$) *Sei* I *eine abzählbare Indexmenge (nicht notwendigerweise unendlich) und* $A = \bigcup\limits_{i \in I} B_i$*, wobei die* $B_i \subseteq \mathbb{R}^n$ *paarweise disjunkt sind.*
Dann ist $\mu(A) = \sum\limits_{i \in I} \mu(B_i)$*.*

(Inv) *Sei* $A, B \subseteq \mathbb{R}^n$*. Falls es eine Transformation* $\rho \in \mathcal{M}_n$ *gibt, so daß* $A^\rho = B$*, so gilt* $\mu(A) = \mu(B)$*.*

(Norm) *Es ist* $\mu(\{x \in \mathbb{R}^n \mid \forall 1 \leq i \leq n : 0 \leq x_i < 1\}) = 1$*.*

Zeitplan

Abschnitte	Dauer	Gesamt
Kapitel 1	1.5 Doppel-Std.	0.75 Woche
Kapitel 2	1.5 Doppel-Std.	0.75 Woche
Kapitel 3	1.5 Doppel-Std.	0.75 Woche
Kapitel 4	2 Doppel-Std.	1 Woche
Kapitel 5	3 Doppel-Std.	1.5 Wochen
Kapitel 6	1.5 Doppel-Std.	0.75 Woche
Kapitel 7	3 Doppel-Std.	1.5 Wochen
Kapitel 8	4 Doppel-Std.	2 Wochen
Kapitel 9	3 Doppel-Std.	1.5 Wochen
Kapitel 10	2 Doppel-Std.	1 Woche
Kapitel 11	1 Doppel-Std.	0.5 Woche
Kapitel 12	2 Doppel-Std.	1 Woche

Literaturverzeichnis

[Baa96] Franz Baader. Logik-basierte Wissensrepräsentation. *KI*, 3, Special Issue on Logic Programming:8–16, 1996.

[BD96] Gerhard Brewka and Jürgen Dix. Knowledge representation with logic programs. Technical report, Tutorial Notes of the 12th European Conference on Artificial Intelligence (ECAI '96), 1996. Also appeared as Technical Report 15/96, Dept. of CS of the University of Koblenz-Landau. Will appear as Chapter 6 in *Handbook of Philosophical Logic*, 2nd edition (1998), Volume 6, Methodologies.

[BDK97] Gerd Brewka, Jürgen Dix, and Kurt Konolige. *Nonmonotonic Reasoning: An Overview*. CSLI Lecture Notes 73. CSLI Publications, Stanford, CA, 1997.

[DF96] J. Dix and U. Furbach. Logisches Programmieren mit Negation und Disjunktion. *KI*, 3, Special Issue on Logic Programming:24–31, 1996. Appeared also as Technical Report TR 21-95 , University of Koblenz-Landau.

[Dix95] Jürgen Dix. *Nichtmonotones Schliessen*. Universität Koblenz-Landau, Lecture Notes, 1995. Vorlesungsskriptum.

[HTT90] Jeff Horty, Richmond Thomason, and D. S. Touretzky. A skeptical Theory of Inheritance in Nonmonotonic Semantic Networks. *Artificial Intelligence*, 42:311–348, 1990.

[Pea88] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, San Mateo, 1988.

[Pup91] Frank Puppe. *Einführung in Expertensysteme*. Studienreihe Informatik. Springer, Berlin, 2nd edition, 1991.

[RN95] Stuart Russel and Peter Norvig. *Artificial Intelligence — A Modern Approach*. Prentice Hall, New Jersey 07458, 1995.

[TTH91] D. S. Touretzky, R. H. Thomason, and J. F. Horty. A skeptic's menagerie: Conflictors, preemptors, reinstaters, and zombies in nonmonotonic inheritance. In *Proc. 12th IJCAI, Sydney*, 1991.