# Wissensbasierte Systeme

INF 5T

Wintersemester 2004/2005

# *Inhaltsverzeichnis*

# 1 Einleitung

## 1.1 Motivation

**Entwicklung von intelligenten Computersystemen mit u.a. folgenden Eigenschaften:**

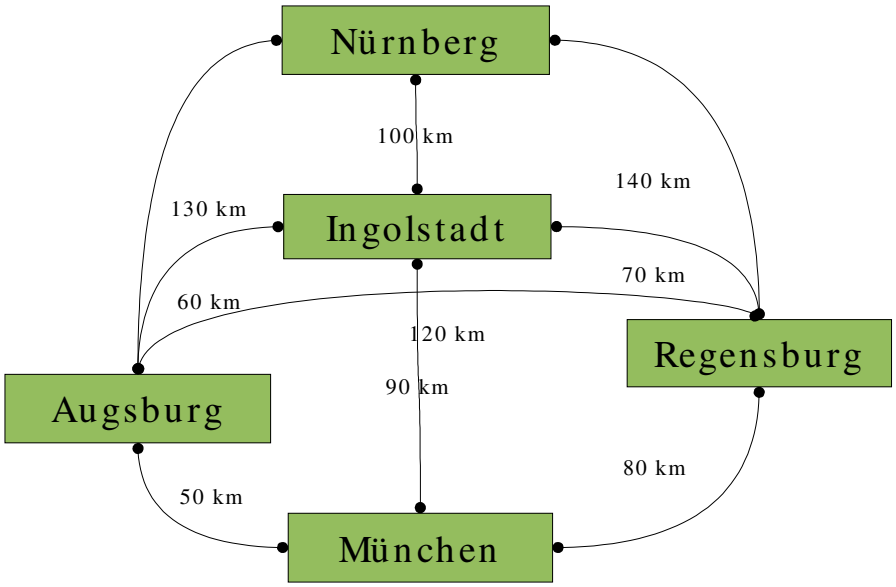
- Lernfähigkeit und Flexibilität
- Natürlichsprachige Schnittstelle
- Lösung komplexer Probleme
- Erklärungsfähigkeit
- Autonomie

**Erforschung von „intelligenem“ Problemlösungsverhalten bei Lebewesen, wie z.B.:**

- Simulation von Denkprozessen im Gehirn für Rückschlüsse auf die Arbeitsweise des Gehirns
- Formalisierung von Expertenwissen für wissensbasierte Systeme erweitert das Expertenwissen

**Beispiel**: TSP (Travelling Salesman Problem)

**Geg.**: Verbindungskarte mit Städten




**Aufgabe**: Mache, ausgehend von Ingolstadt, eine Rundreise durch alle Städte auf dem kürzesten Weg.

**Wege** (u.a.): I → A → M → R → N → I oder
I → N → A → R → M → I etc.

**Hinweise zum TSP:**

- Variante des TSP: Knoten können auch mehrfach besucht werden.

- Prozeduraler Algorithmus: Berechnung aller Wege durch ineinander geschachtelte Schleifen und Auswahl des kürzesten Weges (vollständige Enumeration).
- Komplexität des Problems:
  5 Städte: $4! = 24$ Durchläufe
  21 Städte: $20! = 2.43 \cdot 10^{18}$ Durchläufe
  n Städte: $(n-1)!$ ⇒ kein Polynom
  ⇒ non polynomial vollständig (np-vollständig)

**Definitionen**:

Ein **wissensbasiertes** System ist ein Computersystem, das das fachliche Wissen von Experten über ein bestimmtes Anwendungsgebiet (**Domäne**) einsetzt, um Problemstellungen daraus zu lösen. Ein **Expertensystem** ist ein wissensbasiertes System, das vorwiegend interaktiv genutzt wird. Die Methoden zur Entwicklung wissensbasierter Systeme sind im Forschungsgebiet „**Künstliche Intelligenz**“ (KI, Artificial Intelligence) zusammengeschlossen.

## Zielsetzung:

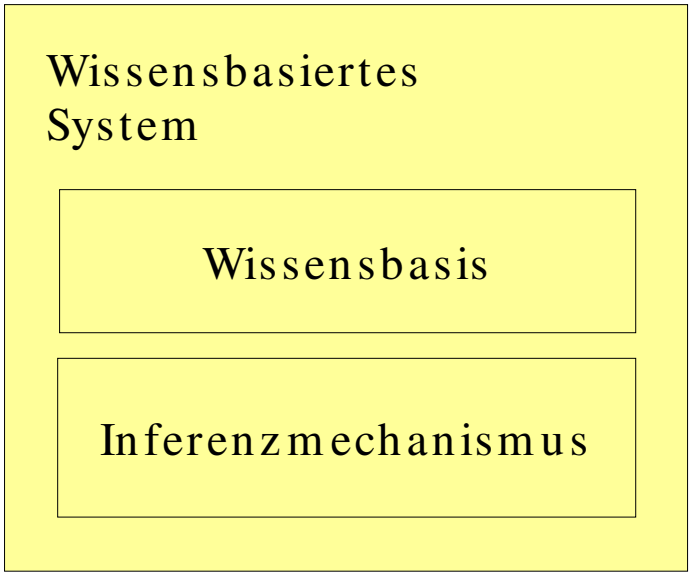
In dieser Vorlesung werden ausgewählte Grundlagen zu wissensbasierten Systemen vermittelt. Anhand praktischer Anwendungen werden einzelne Themen am Rechner vertieft.

## Überblick:

- Wissensrepräsentationen
- Problemlösungsstrategien

# 2 Wissensrepräsentation

### Architektur eines wissensbasierten Systems:




Die Wissensrepräsentation ist die Darstellung des Wissens in der Wissensbasis. Jede Wissensrepräsentation verwendet einen speziellen Inferenzmechanismus zur Auswertung des Wissens in der Wissensbasis. Der Inferenzmechanismus wird von der Entwicklungsumgebung bereitgestellt. Dies ermöglicht eine deklarative anstatt einer prozeduralen Beschreibung des Wissens (Was statt Wie). In einem wissensbasierten System sind in der Regel mehrere Wissensrepräsentationen problemspezifisch eingesetzt (hybride Wissensrepräsentation).

Die Erstellung einer Wissensrepräsentation wird von einem Knowledge Engineer im Rahmen einer Wissensaquisition durchgeführt. Hauptproblem dabei ist die Interpretation des informellen Wissens und dessen Formalisierung in einer Wissensrepräsentation (Methoden siehe Literatur).

### Wichtigste Formen der Wissensrepräsentation:

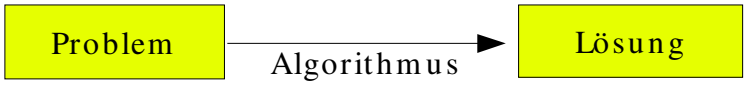
- objektorientierte Repräsentation
- Produktionsregeln (vorwärts verkettete Logik)
- Klauseln (rückwärts verkettete Logik)
- Constraints
- Neuronale Netze
- Fuzzy-Regeln (Fuzzy Logic)
- Semantische Netze


Semantische Netze sind Entity-Relationship-Modellen sehr ähnlich (siehe Vorlesung Datenbanksysteme). Die objektorientierte Repräsentation ist prozedural (→ Grundstudium)

## 2.1 Produktionsregeln

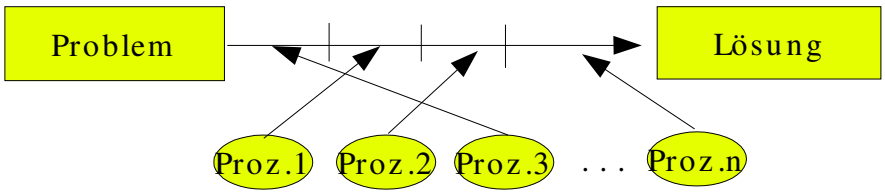
### Motivation:

Prozedurales Programm:




Nachteil: Begrenzte Flexibilität

Abhilfe:




Ein Produktionssystem besteht aus:

- Produktionsregeln
- Datenbasis
- Regelinterpreter

Die Datenbasis ist entweder als gewöhnliche Datenstrukturen (Records) oder als Objekte strukturiert. Die Instantiierungen der Datenstrukturen oder der Klassen werden Fakten genannt.

**Beispiel**: Deklaration eines Faktums der Klasse "Auto".

| | | |
|---|---|---|
| Attribute: | Zeichenkette | Autonummer |
| | Zeichenkette | Halter |
| | Integer | Kilometerstand |

Instantiierung des Faktums 1 der Klasse Auto:
Autonummer = EI-WZ 123
Halter = Schweiger Claudia
Kilometerstand = 12000 km

Instantiierung des Faktums 2 der Klasse Auto:
Autonummer = EI-EJ 24
Halter = Schweiger Johann
Kilometerstand = 95000 km

Eine Produktionsregel ist ein Sprachkonstrukt der Form:
$if\, B_1 \wedge ... \wedge B_n\, then\, A_1 ; .. ; A_m$

Die $B_i (i = 1, .., n)$ sind Bedingungen über den Daten der Datenbasis des Produktionssystems.
Die $A_i (i = 1, ..., m)$ sind Aktionen, die die Daten der Datenbasis bearbeiten.

Eine Bedingung kann sich wie in prozeduralen Sprachen üblich auf die Belegung eines Attributs eines Objektes beziehen. Darüber hinaus ist in jeder Bedingung implizit die Prüfung der Existenz der Fakten, die in der Bedingung vorkommen, enthalten. Zugriffsverletzungen sind damit ausgeschlossen.

**Beispiel**:

Bedingung $B_1 = \text{Auto.Kilometerstand} < 100000$
impliziert: $\exists$ Faktum $x$ zur Klasse Auto: Kilometerstand $< 100000$
Erfüllung der Bedingung für beide Fakten 1 und 2

Die Aktionen erzeugen, modifizieren und löschen die Fakten der Datenbasis.

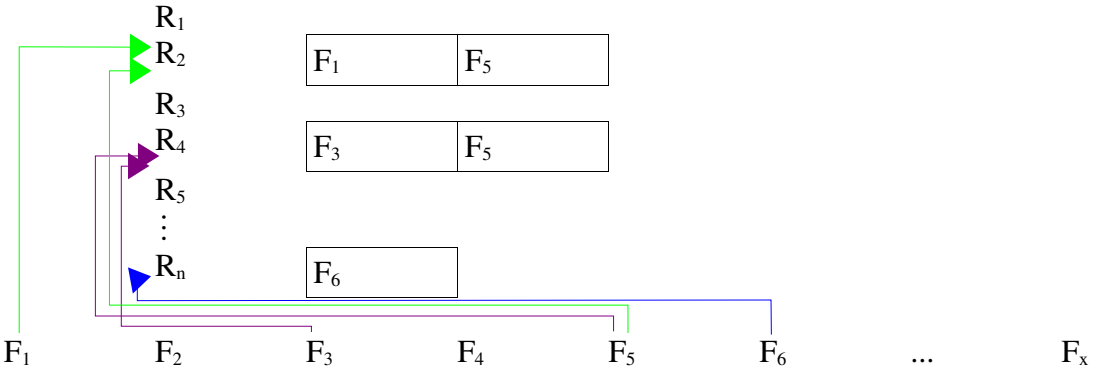
**Beispiel**:

Aktion $A1 =$ Erzeuge eine Instanz zur Klasse Auto mit Halter = Schweiger Andreas ,
Autonummer = EI – HH 10 ,
Kilometerstand = 0
als Faktum 3.

Die Datenbasis und die Produktionsregeln bilden die Wissensbasis des wissensbasierten Systems. Der Regelinterpreter bildet den Inferenzmechanismus, indem er die Produktionsregeln anhand der Fakten auswertet. Grundlegende Vorgehensweisen sind dabei die Vorwärts- und die Rückwärtsverkettung.

Bei der Vorwärtsverkettung werden die Produktionsregeln durch den Recognize-Act-Cycle interpretiert, der zyklisch eine Produktionsregel auswählt und deren Aktionen ausführt. Der Regelinterpreter kann eine Produktionsregel nur dann auswählen, wenn alle Bedingungen $B_1$, ..., $B_n$ erfüllt sind!

Im allgemeinen sind nicht alle Attribute eines Faktums in den Bedingungen einer Regel eingeschränkt. Die Attribute, die keiner Beschränkung unterliegen, nennt man freie Variablen. Die anderen Variablen bezeichnet man als gebunden. Durch Anwendung einer Regel werden die freien Variablen mit Werten belegt (gebunden). Diesen Vorgang nennt man Unifikation oder Matching. Formale Definition siehe Literatur, z.B. Lloyed.




**Beispiel**: Bedingung $B_1$ = Auto.Kilometerstand < 50000

Gebunden: Kilometerstand
frei: Halter, Autonummer

Unifikation mit Faktum1: Halter = Schweiger Claudia; Autonummer = EI-WZ 123
Unifikation mit Faktum2: nicht möglich
Unifikation mit Faktum3: Halter = Schweiger Andreas, Autonummer = EI-HH 10

Jedes Faktum kann für eine Regel mehrfach unifiziert werden. Jede Kombination von Fakten kann für eine Regel nur einmal unifiziert werden. Ansonsten würde der Regelinterpreter in einer Endlosschleife enden. Können mehrere Produktionsregeln unifiziert werden, so wird durch eine Konfliktlösungsstrategie entschieden, welche Regel ausgewählt wird.

**Beispiel**: Zusätzliche Fakten zu vorherigem Beispiel

Faktum 4: Klasse Stellplatz mit Name S1 und Eigentümer = Schweiger
Faktum 5: Klasse Stellplatz mit Name S2 und Eigentümer = Schweiger

**Bedingung**: $B_1 \wedge B_2$ mit:

$B_1 = \text{Auto.Kilometerstand} < 50000$

$B_2 = \text{Stellplatz.Eigentümer} = \text{Schweiger}$

**Unifikationen**:
Fakten 1 und 4
Fakten 1 und 5
Fakten 3 und 4
Fakten 3 und 5

Durch die Aktionen ändern sich die Fakten der Datenbasis. Das Ausführen der Aktionen einer Produktionsregel nennt man feuern. Der Recognize-Act-Cycle terminiert, wenn keine Produktionsregel mehr unifiziert werden kann.

**Beispiel**: Geschäftsszenario

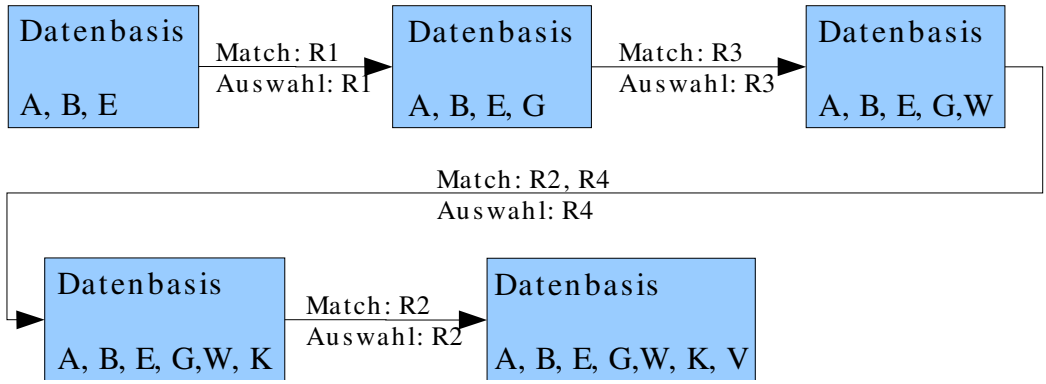
**Abkürzungen für die Datenbasis**:

$A \sim \exists$ Faktum der Klasse $\underline{A}$uftrag mit Auftragsvolumen $> 100.000\,EUR$
$B \sim \exists$ Kunde: $\underline{B}$onität des Kunden ist gut
$E \sim \exists$ Meldung: $\underline{E}$ndmontage ist ausgefallen
$G \sim \exists$ Kunde: Kunde ist $\underline{G}$roßkunde
$K \sim \exists$ Meldung: $\underline{K}$ritische Unternehmenslage ist eingetreten
$V \sim \exists$ Tätigkeit: $\underline{V}$ertriebsleiter muss verständigt werden
$W \sim \exists$ Kunde: Kunde ist ein $\underline{w}$ichtiger Kunde
$Z \sim \exists$ $\underline{Z}$ahlungsunfähigkeit: Das Unternehmen ist zahlungsunfähig

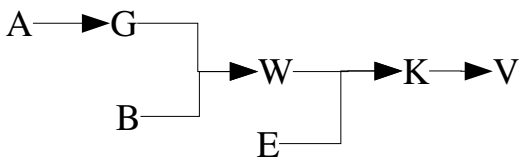
**Produktionsregeln**:

$R_1 \sim$ if $A$ then erzeugeG
$R_2 \sim$ if $W$ then erzeugeV
$R_3 \sim$ if $G \wedge B$ then erzeugeW
$R_4 \sim$ if $W \wedge E$ then erzeugeK

**Vorwärts verkettende Regelinterpretation**:

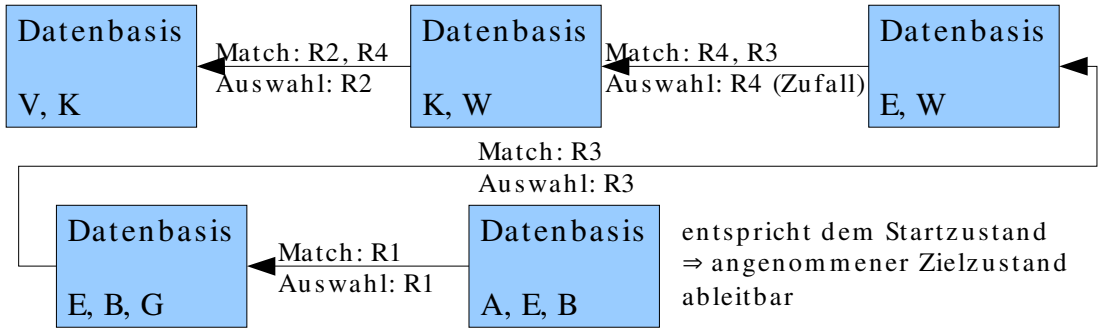



**Ableitungsbaum**:




Bei der Rückwärtsverkettung wird der angestrebte Zielzustand durch revertierte Anwendung der Produktionsregeln auf einen gegebenen Startzustand zurückgeführt.

**Beispiel**: obiges Geschäftsszenario mit rückwärtsverkettender Regelinterpretation

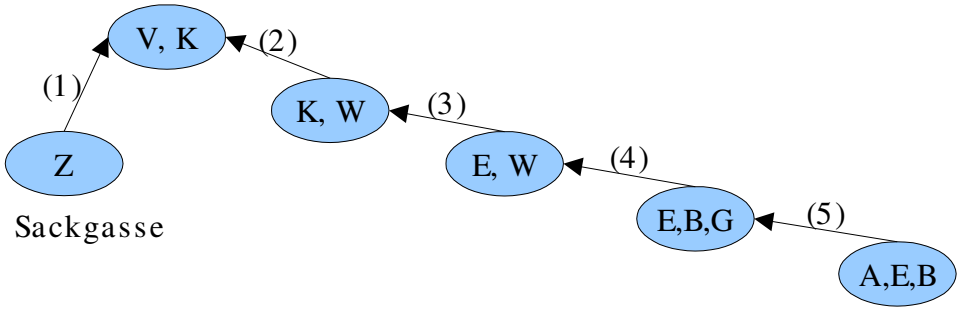



Kann für einen Zustand der Datenbasis keine Regel gefunden werden (Sackgasse), so wird falls möglich in einen früheren Zustand zurückgesetzt und dort eine andere Regel gewählt. Das Zurücksetzen in den zuletzt gefundenen Zustand nennt man (gewöhnliches) Backtracking. Kann gemäß einer Strategie auch in frühere Zustände zurückgesetzt werden, nennt man dies intelligentes Backtracking.

**Beispiel:** obiges Geschäftsszenario

Zusätzliche Produktionsregel:
$R5 \sim$ if $Z$ then erzeugeK, V

Ableitungsbaum mit gewöhnlichem Backtracking:




Ein Beispiel für ein Produktionssystem ist OPS5. Dieses wurde an der TU München zu MAGSY (Multiagentensystem) (K. Fischer, H. Windisch).

Als Konfliktlösungsstrategien werden in dieser Reihenfolge verwendet:

1. Einmaligkeit der Unifikation
2. Neuheit der Fakten
3. Spezifität der Regel (Anzahl der Bedingungen)
4. Zufällige Auswahl

<Folie Sprachbeschreibung MAGSY>

**Beispiel**: Steuerung eines Fahrzeugs zum Ausweichen eines Hindernisses

```
(literalize Fahrzeug
   Name
   Geschwindigkeit
   Winkel)
(Literalize Hindernis
   NameFahrzeug
   Richtung
   Abstand)
```

```
(literalize FahrzeugZustand
   NameFahrzeug
   Zeitpunkt
   Zustand)
(startup (make Fahrzeug ^Name PB2
            ^Geschwindigkeit -100
            ^Winkel 178)
         (make Fahrzeug ^Name DX1
            ^Geschwindigkeit 200
            ^Winkel -45)
         (make FahrzeugZustand ^NameFahrzeug PB2
                   ^Zeitpunkt aktuell
                   ^Zustand andocken)
         (make FahrzeugZustand ^NameFahrzeug DX1
                   ^Zeitpunkt aktuell
                   ^Zustand reisen)
         (make FahrzeugZustand ^NameFahrzeug DX1
                   ^Zeitpunkt Mittagspause
                   ^Zustand laden)
)
```

### a) Regel, die die Geschwindigkeit des Powerbots anzeigt

```
  (p AusgabePB2Geschwindigkeit
     (Fahrzeug ^Name PB2
               ^Geschwindigkeit <vel>)
     --> (write |Powerbot PB2 fährt derzeit mit|)
         (write <vel>)
         (write |mm/s|)
         (write (CRLF))
  )
```

### Übung: Geschwindigkeit aller Fahrzeuge ausgeben

```
p) Ausgabe Geschwindigkeiten
    (Fahrzeug ^Name <na>
              ^Geschwindigkeit <vel>)
    --> (write |Fahrzeug | <na> | fährt mit | <vel>)
        (write |mm/s| (CRLF))
)
```

### b) Regel, die die Geschwindigkeiten der Fahrzeuge im aktuellen Zustand reisen auflistet:

```
  (p Reisegeschwindigkeiten
     (Fahrzeugzustand ^NameFahrzeug <na>
                      ^Zustand reisen
                      ^Zeitpunkt aktuell)
     (Fahrzeug ^Name <na>
               ^Geschwindigkeit <vel>)
       --> (write |Fahrzeug | <na> | reist mit Geschwindigkeit |
             <vel> (CRLF))
  )
```

### c)

```
  (p AusgabeBewegteFahrzeuge
     (FahrzeugZustand ^Zeitpunkt aktuell ^NameFahrzeug <na>
                      ^Zustand <<andocken reisen>>)
     --> (write |Fahrzeug | <na> | ist in Bewegung| (CRLF)))
```

### d) Umfahrung eines Hindernisses

```
  (p R1
     (Hindernis ^Abstand < 1000
                ^Richtung vorne
                ^NameFahrzeug <na>)
     {<fzg> (Fahrzeug ^Name <na>)}
     {<zust> (FahrzeugZustand ^NameFahrzeug <na>
                              ^Zeitpunkt aktuell
                              ^Zustand reisen)
```

```
    --> (modify <fzg> ^Geschwindigkeit 10)
        (modify <zust> ^Zustand Kurskorrektur)
)


(p R2
   {<fzg> (Fahrzeug ^Name <na> ^Winkel <wi>)}
   {<zust> (FahrzeugZustand ^NameFahrzeug <na> ^Zeitpunkt aktuell
                            ^Zustand Kurskorrektur)}
   - (Hindernis ^NameFahrzeug <na> ^Abstand < 2000 ^Richtung links)
   --> (modify <fzg> ^Winkel (compute <wi> + 90))
       (modify <zust> ^Zustand QuerfahrtLinks)
)


(p R3
   {<fzg> (Fahrzeug ^Name <na> ^Winkel <wi>)}
   {<zust> (FahrzeugZustand ^NameFahrzeug <na> ^Zeitpunkt aktuell
                            ^Zustand Kurskorrektur)}
   - (Hindernis ^NameFahrzeug <na> ^Abstand < 2000 ^Richtung rechts)
   --> (modify <fzg> ^Winkel (compute <wi> - 90))
       (modify <zust> ^Zustand QuerfahrtRechts)
)


(p R4
   {<fzg> (Fahrzeug ^Name <na> ^Winkel <wi>)}
   {<zust> (FahrzeugZustand ^NameFahrzeug <na> ^Zeitpunkt aktuell
                            ^Zustand QuerfahrtLinks)}
   - (Hindernis ^NameFahrzeug <na> ^Abstand < 2000 ^Richtung rechts)
   --> (modify <fzg> ^Winkel (compute <wi> - 90))
       (modify <zust> ^Zustand Reisen)
)


(p R5
   {<fzg> (Fahrzeug ^Name <na> ^Winkel <wi>)}
   {<zust> (FahrzeugZustand ^NameFahrzeug <na> ^Zeitpunkt aktuell
                            ^Zustand QuerfahrtRechts)}
   - (Hindernis ^NameFahrzeug <na> ^Abstand < 2000 ^Richtung links)
   --> (modify <fzg> ^Winkel (compute <wi> + 90))
       (modify <zust> ^Zustand Reisen)
)
```

e) Regel, die den Abstand und die Richtung des nächsten Hindernisses unter 3 Meter zum Fahrzeug DX1 ausgibt.

```
(p NaechstesHindernis
   (Hindernis ^NameFahrzeug DX1
              ^Abstand { < 3000 <abst> }
              ^Richtung <ri>)
   - (Hindernis ^NameFahrzeug DX1
                ^Abstand < <abst>)
   --> (write |Richtung nächstes Hindernis | <ri> (CRLF))
)
```

## 2.2 Klauseln

Eine Klausel ist eine prädikatenlogische Formel erster Ordnung der Form: $\forall x_1, \ldots, \forall x_S : B_1 \vee \ldots \vee B_k \vee \overline{C_1} \vee \ldots \vee \overline{C_n}$ mit $B_1, \ldots, B_K, C_1, \ldots, C_n$: atomare Formeln und $x_1, \ldots, x_s$ alle Variablen in $B_1, \ldots, B_k, C_1, \ldots, C_n$.

Wird in folgender Notation dargestellt, weil $x \vee \overline{y} \Leftrightarrow x \rightarrow y : B_1, \ldots, B_k \rightarrow C_1, \ldots, C_n$ .

Eine atomare Formel ist eine Funktion mit einem booleschen Ergebnis über einer Menge von Variablen $x_1, \ldots, x_3$.

Klauseln mit $k \leqslant 1$ werden als Hornklauseln[1] bezeichnet. Klauseln mit $k=1$ und $n=0$ werden Einheitsklauseln[2] genannt und beschreiben Faktenwissen. Eine Klausel mit $k=0$ und $n>0$ heißt Ziel[3]. Eine Klausel mit $k=n=0$ ist eine leere Klausel[4].

**Beispiel**:

$\left.\begin{matrix} GibtVorlesung(Pöppel, GET) \\ IstProf(Pöppel) \end{matrix}\right\}$ *Hornklausel, Einheitsklausel*

ETProf(x) ← IstProf(x), GibtVorlesung(x, GET) ⇒ Hornklausel
$\overline{ETProf}(y)$ ⇒ Ziel

Der übliche Inferenzmechanismus für Klauseln ist die Resolution. Sie basiert auf dem Modus Ponens:

### Modus Ponens:

Gegeben seien atomare Formeln F und H. Dann gilt:

$$\frac{H \qquad \leftarrow \qquad F}{H} \qquad\qquad \frac{H \qquad \vee \qquad \neg F}{H}$$

**Beispiel**:

F ~ „es regnet“
F→H ~ „wenn es regnet, dann ist die Straße naß“
H ~ „die Straße ist naß“

Frage: Ist die Straße naß? ⇒ H∨¬F

Annahme: Die Straße ist nicht naß ⇒ H = false ⇒ ¬F ⇒ es regnet nicht

⇒ Widerspruch, weil es regnet! (F)

Bei der Resolution wird das zu beweisende Theorem negiert und und als Ziel der Faktenmenge hinzugefügt. Durch einzelne Resolutionsschritte wird versucht, einen Widerspruch abzuleiten. Ein Widerspruch liegt vor, wenn die leere Klausel abgeleitet werden konnte. Kann ein Widerspruch abgeleitet werden, ist das Theorem bewiesen.

Die Resulution basiert auf der Technik des Widerspruchsbeweises. Dabei wird eine Annahme bewiesen, indem man die Negation der Annahme durch ein Gegenbeispiel widerlegt.

1 auf der linken Seite des Pfeils nur eine Bedingung
2 links nur eine Bedingung, rechts nichts
3 links nichts, rechts mind. eine Bedingung
4 Pfeil alleine ;-)

**Beispiel**:

Satz: die natürlichen Zahlen besitzen kein Supremum.

Widerpsruchsbeweis: Angenommen es gäbe ein Supremum $\sup \in \mathbb{N} \Leftrightarrow \forall n \in \mathbb{N} : n \leq \sup \Leftrightarrow n = \sup + 1 \in \mathbb{N} \wedge n \leq \sup \Rightarrow 2 \leq 1$ Widerspruch!!

sup = 1 ist hier ein Gegenbeispiel, das den Widerspruch hervorruft.

Die Belegung der freien Variablen, die beim Widerspruch gebunden wurden, bilden das Ergebnis des Programmlaufs.

**Beispiel**:

$\text{GibtVorlesung}(Schweiger, WBS)$
$\text{GibtVorlesung}(Gold, SE)$
$\text{GibtVorlesung}(Gold, MA)$
$\text{GibtVorlesung}(Hunsinger, DB)$
$\text{GibtVorlesung}(Glavina, SE)$
$\text{GibtVorlesung}(Brüdigam, GET)$
$\text{IstProf}(Gold)$
$\text{IstProf}(Hunsinger)$
$\text{IstProf}(Glavina)$
$\text{IstProf}(Brüdigam)$
$\text{IstProf}(Schweiger)$
$\text{IstLB}(Hammer)$

$\text{INFProf}(x) \leftarrow \text{IstProf}(x), \text{GibtVorlesung}(x, SE)$
$\text{INFProf}(x) \leftarrow \text{IstProf}(x), \text{GibtVorlesung}(x, DB)$
$\text{INFProf}(x) \leftarrow \text{IstProf}(x), \text{GibtVorlesung}(x, WBS)$

Anfrage an das System: Gibt es einen INF-Professor an der Hochschule?
Ziel: $\neg INFProf(y)$

$\leftarrow INFProf(y) \leftarrow IstProf(y), GibtVorlesung(y, SE) \leftarrow$
$IstProf(Gold), GibtVorlesung(Gold, SE) \leftarrow \{\ \}$

Unifikation: $y = Gold$

Welche zwei Klauseln für einen Resolutionsschritt ausgewählt werden, wird durch eine Suchstrategie festgelegt. Die Resolution ist vollständig, d.h. falls es eine Ableitung der leeren Klausel gibt, wird sie gefunden.

Eine Programmiersprache, in der mit Klauseln programmiert wird, ist Prolog. Die Resolution entspricht der Rückwärtsverkettung von Produktionsregeln. Sackgassen werden durch Backtracking behandelt.

**Beispiel** (vorheriges Beispiel fortgesetzt):

Anfrage: Ist Schweiger ein INF-Professor?
Ziel: $\neg INFProf(Schweiger)$

$INFProf(Schweiger) \leftarrow IstProf(Schweiger), GibtVorlesung(Schweiger, SE)$
$\leftarrow GibtVorlesung(Schweiger, SE)$ !! Sackgasse !!

$INFProf(Schweiger) \leftarrow IstProf(Schweiger), GibtVorlesung(Schweiger, DB)$
$\leftarrow GibtVorlesung(Schweiger, DB)$ !! Sackgasse !!

$INFProf(Schweiger) \leftarrow IstProf(Schweiger), GibtVorlesung(Schweiger, WBS)$
$\leftarrow \{\ \}$

Es liegt ein Widerspruch vor, weil die leere Klausel abgeleitet werden konnte. Unifikation: nein, gibt es nicht, weil es keine freie Variable gibt.

Anfrage: Ist Brüdigam ein INF-Professor?
Ziel: $\neg INFProf(Brüdigam)$

$\leftarrow INFProf(Brüdigam) \leftarrow IstProf(Brüdigam), GibtVorlesung(Brüdigam, SE)$
$\leftarrow GibtVorlesung(Brüdigam, SE)$ !! Sackgasse !!

$\leftarrow INFProf(Brüdigam) \leftarrow IstProf(Brüdigam), GibtVorlesung(Brüdigam, DB)$
$\leftarrow GibtVorlesung(Brüdigam, DB)$ !! Sackgasse !!

$\leftarrow INFProf(Brüdigam) \leftarrow IstProf(Brüdigam), GibtVorlesung(Brüdigam, WBS)$
$\leftarrow GibtVorlesung(Brüdigam, WBS)$ !! Sackgasse !!

Es liegt kein Widerspruch vor. Also ist Brüdigam kein INF-Professor.

Durch das Sprachkonstrukt cut werden in Prolog Teile des Suchbaumes abgeschnitten. Damit wird zwar an Effizienz gewonnen, aber die Vollständigkeit der Resolution geht verloren.

Eine andere Inferenzmethode für Klauseln stammt aus dem Bereich der deduktiven Datenbanken und basiert auf der Abbildung von Klauseln in Ausdrücke relationaler Algebra. Details siehe Literatur.

## 2.3 Constraints

### Motivation

Problemstellungen, in denen freie Variablen so mit Werten belegt werden müssen, daß eine Reihe von Bedingungen erfüllt ist. Praktische Beispiele: Terminplanung, Stundenplanung, Produktionsplanung, Integritätsbedingungen in Datenbanken, autonome, mobile Roboter.

**Akademische Beispiele**: Kryptoarithmetische Probleme

a)

```
  F H
+ I N
--------
T O P
```

$F, H, I, N, O, P, T \in \{0, 1, ..., 9\}$
$F \neq H \neq I \neq ... \neq P \neq T$

Lösung, z.B.:

```
  4 1
+ 5 2
--------
0 9 3
```

b)

```
  S E N D
+ M O R E
------------
M O N E Y
```

$D, E, M, N, O, R, S, Y \in \{0, 1, ..., 9\}$
$D \neq E \neq M \neq ... \neq S \neq Y$

Lösung: $(D, E, M, N, O, R, S, Y) = (7, 5, 1, 6, 0, 8, 9, 2)$

(c) N-Damen-Problem

| | | | | |
|---|---|---|---|---|
| | | x3 | | A |
| x1 | | | | B |
| | | | x4 | C |
| | x2 | | | D |
| 1 | 2 | 3 | 4 | |

$x_i \neq x_j \wedge x_i - x_j \neq i - j \wedge x_j - x_i \neq j - i$ )

Erforderlich zur Berechnung der Lösung in einem Programmsystem ist die Formalisierung der Aufgabenstellung.

Gegeben seien:

- ohne Beschränkung der Allgemeinheit (O.B.d.A.) eine geordnete, nicht-leere Menge von Variablen $var = \{v_1, ..., v_k\}$
- eine Grundmenge $dom(v_i)$ von möglichen Werten für jede Variable $v_i$. Die Menge $dom$ nennt man Domäne von $v_i (i = 1, ..., k)$ und entspricht dem Datentyp.

**Beispiel** (SEND + MORE = MONEY):

$Var=\{D,E,M,N,O,R,S,Y\}\cup\{C0,C1,C2\}$
$dom(D)=dom(E)=\ldots=dom(Y)=dom(C0)=\ldots=dom(C2)=\{0,2,\ldots,9\}$

Ein Constraint ist ein Tupel der Form

$$\text{c\_name}=(\text{c\_vars},\text{c\_def})$$

mit

- c_name ist der Name des Constraints
- $\text{c\_vars}=(v_r,\ldots,v_s)$ ist ein Tupel von paarweise verschiedenen Variablen mit $v_i\in Var$ für $i=r,\ldots,s$. Die Variablen sind entsprechend Var geordnet.
- c_def ist die Definition einer Relation $\text{c\_rel}\subseteq dom(v_r)\,\text{x}\ldots\text{x}\,dom(v_s)$

Ein Constraint-Netzwerk ist eine Menge von Constraints.

**Beispiel** „Send more Money“

$\text{smm\_net}=\{cons_0,\ldots,const_3,cons_{distinct}\}$
$cons_0=((D,E,Y,C_0),D+E=Y+C_0\cdot 10)$
$cons_1=((E,N,R,C_0,C_1),N+R+C_0=E+C_1\cdot 10)$
$cons_2=((E,N,O,C_1,C_2),E+O+C_1=N+C_2\cdot 10)$
$cons_3=((M,O,S,C_2),S+M+C_2=O+M\cdot 10)$
$cons_{distinct}=((D,E,M,N,O,R,S,Y),distinct(D,E,M,N,O,R,S,Y))$

Ein Constraint-Problem $CP=(C,init)$ besteht aus

- einem Constraint-Netzwerk C
- einer Anfangseinschränkung (initiale Zuweisung) init, die jede Variable $v_i$ mit einer Teilmenge ihrer Domäne $dom(v_i)$ belegt.

**Beispiel** „Send more Money“

Constraint-Problem $\text{SMM}=(\text{smm\_net},\text{smm\_init})$
$\text{smm\_init}(E)=\ldots=\text{smm\_init}(Y)=\{0,\ldots,9\}$
$\text{smm\_init}(C_0)=\ldots=\text{smm\_init}(C_2)=\{0,1,2\}$

Eine Einzellösung eines Constraint-Problems $CP=(C,init)$ ist ein Tupel $d=(w_1,\ldots,w_k)$ mit

- $w_i\in init(v_i)$ für $i=1,\ldots,k$ und
- für alle Constraints $\text{c\_name}=(\text{c\_vars},\text{c\_def})\in C$ gilt: d bezogen auf die Variablen in $\text{c\_vars}(Projektion)\in\text{c\_rel}$

Die (volle) Lösung $sol(CP)$ eines Constraint-Problems $CP=(C,init)$ ist definiert als: $sol(CP)=\{d:\text{d ist Einzellösung von CP}\}$

Ist $sol(CP)$ die leere Menge, so nennt man CP unerfüllbar, da es keine Einzellösung gibt. $sol(CP)$ ist eine Menge von Tupeln.

**Beispiel**: ANYCP = ( ANYC, init ANYC )

$ANYC = (\text{constr\_1}, \text{constr\_2})$
$\text{constr\_1} = ((x, y), x + y \leqslant 6)$
$\text{constr\_2} = ((x), x \geqslant 4)$
$initANYC(x) = initANYC(y) = \{1, \ldots, 100\}$

dann ist $sol(CP) = \{(4,1), (4,2), (5,1)\}$

Die Bestimmung der Lösung eines Constraint-Problems ist i.a. np-vollständig. Daher verwendet man zur Lösungsfindung eine zweistufige Vorgehensweise:

1. Einschränkung des Suchraumes durch lokale Propagierungsmethoden
2. Suche nach Einzellösungen im Tupel von Mengen

Die Grundidee der lokalen Propagierung ist, jedes Constraint für sich einzeln zu betrachten und die Ergebnisse bzgl. der möglichen Werte der Variablen eines Constraints an andere Constraints, die eine oder mehrere dieser Variablen enthalten, weiter zu geben (zu propagieren). Die Weitergabe wird so lange durchgeführt, bis ein stabiler Zustand erreicht ist.

Das sukzessive Ausschalten von unmöglichen Wertebelegungen liefert für jede Variable eine Menge von übriggebliebenen Werten. Dieses Tupel von Mengen wird nachfolgend als sol-val(CP) bezeichnet. Sol-val(CP) ist nicht vergleichbar mit der Menge von Tupeln sol (CP), weil sol-val(CP) im Gegensatz zu sol(CP) keinerlei Abhängigkeiten zwischen einzelnen Variablen darstellt.

**Beispiel:** obiges ANYCP

1. lokale Propagierung:
   Betrachtung von constr_2: $x = \{4, 5, \ldots, 100\}$
   Betrachtung von constr_1: $x = \{4, 5\}$ , $y = \{1, 2\}$
   $\text{sol-val}(\text{ANYCP}) = (\{4, 5\}, \{1, 2\})$
2. Suche:
   $\text{sol}(\text{ANYCP}) = \{(4,1), (4,2), (5,1)\}$

## Darstellungsformen für Constraints

- Extensional: Aufzählung aller Tupel
  Beispiel: $FixeFeiertage = ((Tag, Monat), \{(1, Mai), (25, Dezember), (3, Oktober), \ldots\})$
- Prädikativ: Angabe eines Prädikats
  Beispiel: $cons0 = ((D, E, Y, C0), D + E = Y + 10 \cdot C0)$
- Konstruktiv: Konstruktion der Tupelwerte durch Abbildungen
  Beispiel (unidirektional):

$$UST = (\{Brutto, Netto, MWST\}, (Brutto = Netto \cdot \left(1 + \frac{MWST}{100}\right),$$

$$Netto = \frac{Brutto}{(1 + \frac{MWST}{100})},$$

$$MWST = (\frac{Brutto}{Netto} - 1) \cdot 100))$$

### Beispiele für Constraint Systeme:

- YACSS TUM
- ILOGSolver ILOG
- CPL(R) IBM (frei)
- CHIP ECRC, COSYTEC

Fordert man, dass die Lösung eines Constraint-Problems alle Constraints erfüllt, so spricht man von harten Constraints. Man kann aber auch Lösungen zulassen, die nicht alle Constraints erfüllen. Dies kann zum Beispiel dann notwendig sein, wenn es keine Einzellösung gibt, die alle Constraints erfüllt. Diejenigen Constraints, auf deren Erfüllung verzichtet wird, nennt man weiche Constraints. Die Aufweichung von harten Constraints zur Bestimmung von Einzellösungen bezeichnet man als Relaxierung von Constraints.

## 2.4 Fuzzy-Logic

Die Relaxierung von harten zu weichen Constraints führt zu einem unscharfen Begriff der „Erfüllung“ eines Constraints (Constraint kann ganz, halb oder auch fast nicht erfüllt sein). Die Fuzzy Logic ermöglicht eine Behandlung unscharfer Begriffe durch Zurückführung auf unscharfe Mengen.

**Eine (scharfe) Menge lässt sich wie folgt darstellen:**

Gegeben: G Grundmenge von Werten
Dann lässt sich die Menge $M \subseteq G$ durch folgende Zugehörigkeitsfunktion charakterisieren

$$\forall x \in G : m_M(x) = \begin{cases} 1 \text{ falls } x \in M \\ 0 \text{ sonst} \end{cases}$$

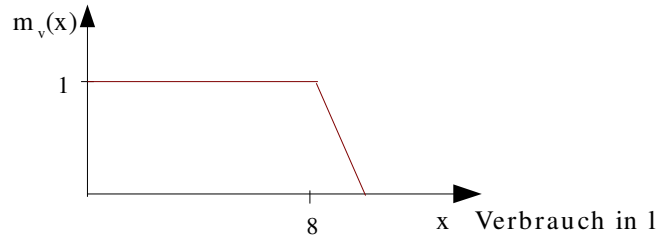
Der Wert 1 der Zugehörigkeitsfunktion zeichnet also einige Elemente aus G aus und markiert damit ihre Zugehörigkeit zu M. Eine unscharfe Menge (Fuzzy Set) lässt sich durch folgende Zugehörigkeitsfunktion herstellen:

$$\forall x \in G : x \rightarrow m_M(x) \in [0 \,..\, 1]$$

Damit kann man gewöhnliche Mengen als spezielle unscharfe Mengen betrachten, bei denen der Bildbereich der Zugehörigkeitsfunktion eingeschränkt ist.

**Beispiel**: Mercedes-Benz-Händler

Ein Kunde möchte ein Auto kaufen, das weniger als 8 l Kraftstoff auf 100 km verbraucht. Natürlich würde der Kunde den Wagen auch dann kaufen, wenn er 8,1 l auf 100 km verbraucht. Daher ist die Menge der akzeptierten Verbrauchswerte ein Fuzzy-Set wie folgt:



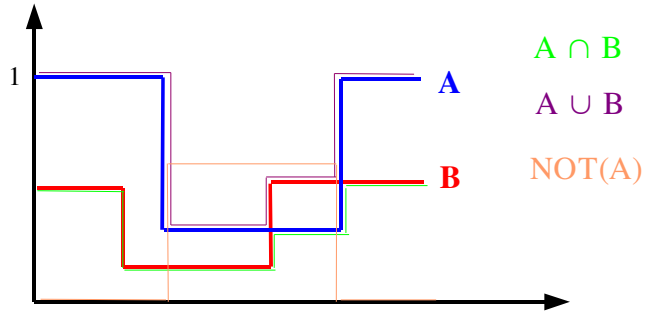

### 2.4.1 Operationen auf Fuzzy-Sets

Nachfolgend werden Operationen auf Fuzzy Sets o.B.d.A. Für zwei Fuzzy Sets dargestellt.

#### Mengenoperationen

Gegeben: Fuzzy-Sets $A, B \subseteq G$

- Durchschnitt: $C = A \cap B : m_C(x) = \min\left[m_A(x), m_B(x)\right]; \forall x \in G$
- Vereinigung: $C = A \cup B : m_C(x) = \max\left[m_A(x), m_B(x)\right]; \forall x \in G$
- Komplement: $C = \overline{A} : m_C(x) = 1 - m_A(x); \forall x \in G$



### Logikoperationen

Gegeben: Fuzzy-Sets $A, B \subseteq G, \gamma \in [0 \ldots 1]$

- Fuzzy AND: $A . \wedge . B : m_C(x) = \gamma \cdot \min\left[m_A(x), m_B(x)\right] + \frac{1}{2}(1-\gamma)(m_A(x)+m_B(x)), \forall x \in G$
- Fuzzy OR: $C = A \circ \vee \circ B : m_c(x) = \gamma \cdot \max\left[m_A(x), m_B(x)\right] + \frac{1}{2}(1-\gamma)(m_A(x)+m_B(x)), \forall x \in G$

$\gamma$ ist ein Parameter zur Justierung des Ergebnisses. In beiden Logikoperationen ergibt sich für $\gamma = 0$ das arithmetische Mittel. Bei $\gamma = 1$ wird das Fuzzy-AND zur Durchschnittsmenge und das Fuzzy-OR zur Vereinigungsmenge.

## 2.4.2 Praktische Anwendung:

### Optimierung

Mit diesen Logikoperationen läßt sich eine Bewertungsfunktion realisieren, die die Fuzzy-Resultate weicher Constraints zu einer Gesamterfüllung aller Constraints zusammenfaßt. Eine Bewertungsfunktion ist eine Funktion, die die Erfüllung einer Menge von in der Regel konkurrierenden Randbedingungen berechnet.

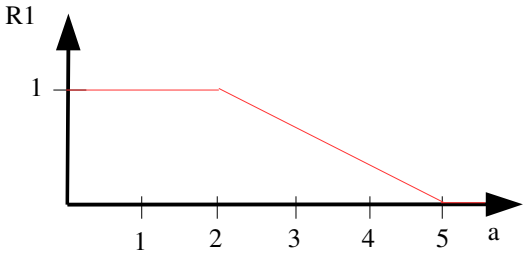
Bei Optimierungsaufgaben bildet jede Randbedingung ein Fuzzy Set. Die Zugehörigkeitsfunktion des Fuzzy-Sets zu einem weichen Constraint gib an, wie die Randbedingung für die betrachtete Alternative (Einzellösung) erfüllt ist. Bei vollständiger Erfüllung liefert die Zugehörigkeitsfunktion des Fuzzy Sets die 1, ansonsten 0. Die Bewertungsfunktion ist in der Regel ein Fuzzy-Logikoperator mit einer bestimmten Justierung.

**Beispiel**: Routenplanung

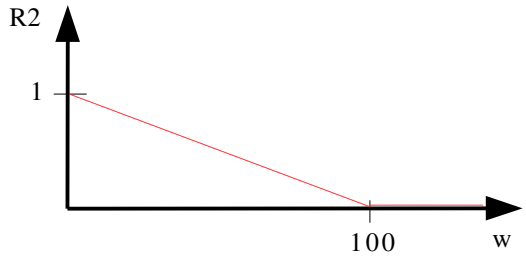
Randbedingungen:

R1: Anzahl der Ampeln bis 2 optimal, mehr als 4 Ampeln sind inakzeptabel
R2: Die Wegstrecke ist möglichst kurz und nicht über 100 km.
R3: Die mögliche Durchschnittsgeschwindigkeit liegt über 50 km/h, ist bei 80 km/h optimal und darüber unmöglich (alte Schepperkiste vom Schweiger (Polo))

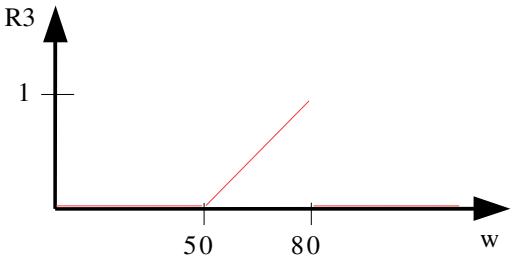
$$m_{R1}(a)=\begin{cases}1 & \text{falls } a<2\\ -\frac{1}{3}(a-2)+1 & \text{falls } a\le 2\wedge a<5\\ 0 & \text{sonst}\end{cases}$$




$$m_{R2}(w)=\begin{cases}-\frac{1}{100}w+1 & \text{falls } w<100\\ 0 & \text{sonst}\end{cases}$$




$$m_{R3}(g)=\begin{cases}0 & \text{falls } g<50\\ \frac{1}{30}(g-50) & \text{falls } 50\le g\le 80\\ 0 & \text{falls } g>80\end{cases}$$




**Gegebene Alternativen:**

$A_1: a=3, w=20, g=70$

$A_2: a=1, w=50, g=60$

**Berechnung:**

$A_1: m_{R1}(3)=\frac{2}{3},\quad m_{R2}(20)=\frac{4}{5},\quad m_{R3}(70)=\frac{2}{3}$

$A_2: m_{R1}(1)=1,\quad m_{R2}(50)=\frac{1}{2},\quad m_{R3}(60)=\frac{1}{3}$

**Bewertung:**

- Gleichmäßige Berücksichtigung aller Randbedingungen: Arithmetisches Mittel $(\gamma=0)$.

$$m(a,w,g)=0\cdot min\left(m_{R1}(a),m_{R2}(w),m_{R3}(g)\right)+\frac{1}{3}\cdot\left(m_{R1}(a)+m_{R2}(w)+m_{R3}(g)\right)$$

$$\begin{cases}m(3,20,70)=0{,}71\\ m(1,50,60)=0{,}61\end{cases}$$ => Entscheidung für Alternative $A_1$

- Berücksichtigung der besten Randbedingung: Fuzzy-OR

$$max(\frac{2}{3},\frac{4}{5},\frac{2}{3})=\frac{4}{5}$$
$$max(1,\frac{1}{2},\frac{1}{3})=1$$
$$\Rightarrow A2$$

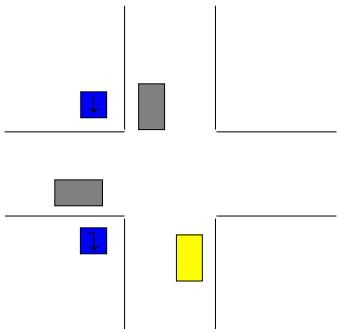
- Berücksichtigung der schlechtesten Randbedingung: Fuzzy-AND

$$min(\frac{2}{3},\frac{4}{5},\frac{2}{3})=\frac{4}{5}$$
$$min(1,\frac{1}{2},\frac{1}{3})=1$$
$$\Rightarrow A1$$

- Kombination aus der 1. und 3. Möglichkeit mit $\gamma=0{,}75$

$$m(3{,}20{,}70)=\frac{3}{4}\cdot min(\frac{2}{3},\frac{4}{5},\frac{2}{3})+\frac{1}{4}\cdot 0{,}71=0{,}7$$
$$m(1{,}50{,}60)=\frac{3}{4}\cdot min(1,\frac{1}{2},\frac{1}{3})+\frac{1}{4}\cdot 0{,}61=0{,}4$$
$$\Rightarrow A1$$

**Beispiel**: Kreuzungsassistent

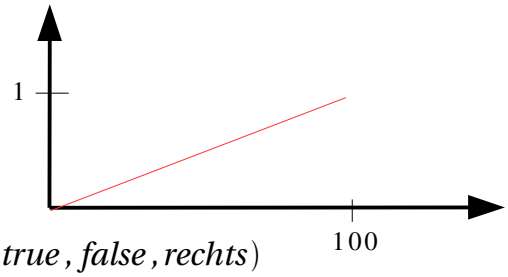


$Var=\{vel, bremse, blinker, frechts, fgegen, ziel\}$
$dom(vel)=\{0,10,20,\ldots,100\}$
$dom(bremse)=\{ein, aus\}$
$dom(frechts)=dom(flinks)=boolean$
$dom(ziel)=\{links, rechts, gerade\}$
$C_1=\{(vel), vel\leq 50\}$
$C_2=((vel, bremse, blinker, ziel),$
$(ziel=rechts\wedge vel<25\wedge blinker=rechts\wedge bremse=aus)\vee ziel\neq rechts)$
$C_3=((bremse, blinker, frechts, ziel),$
$(ziel=gerade\wedge bremse=ein\wedge \overline{frechts}\wedge blinker=aus)\vee$
$(ziel=gerade\wedge bremse=aus\wedge \overline{frechts}\wedge blinker=aus)\vee ziel\neq gerade)$
$C_4=((vel, bremse, blinker, frechts, fgegen, ziel),$
$(ziel=links\wedge blinker=links\wedge vel<25\wedge \overline{frechts}\wedge \overline{fgegen}\wedge bremse=aus)\vee$
$(ziel=links\wedge blinker=links\wedge(frechts\vee fgegen)\wedge bremse=ein)\vee ziel\neq links)$
$C_5=((vel, bremse),(vel>0\wedge bremse=aus)\vee(vel=0\wedge bremse=ein))$

$init(vel)=dom(vel), init(blinker)=dom(blinker),$
$init(bremse)=dom(bremse),$
$init(frechts)=\{true\},$
$init(fgegen)=\{false\},$
$init(ziel)=\{rechts\}$

$\text{sol\_val}(CP)=(\{10,20\},\{aus\},\{rechts\},\{true\},\{false\},\{rechts\})$
$sol(CP)=\{(10, aus, rechts, true, false, rechts),$
$(20, aus, rechts, true, false, rechts)\}$

$C_6=((vel), \frac{1}{100}vel)$



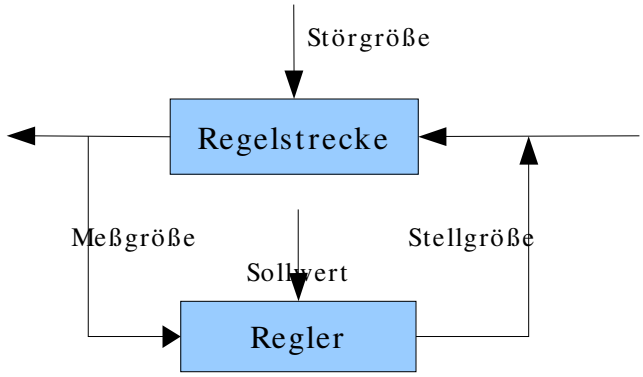

$C_6(10)=\frac{1}{10}$

$C_6(20)=\frac{1}{5}$

$\Rightarrow Verwendung\ der\ Lösung(20, aus, rechts, true, false, rechts)$

## Fuzzy-Regler

Prinzip der Regelung




Lösung komplexer Steuerungs- und Regelungsprobleme, für die kein hinreichendes mathematisches Modell existiert.

Eine unscharfe Zahl ist eine unscharfe Menge A auf der Grundmenge $\mathbb{R}$, für die gilt:

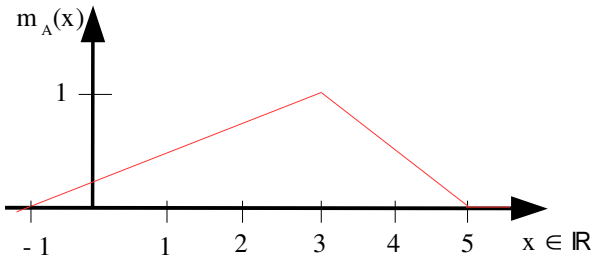
- Es gibt genau ein $x \in \mathbb{R} : m_A(x) = 1$
- $m_A(x)$ ist stückweise stetig

**Beispiel**: unscharfe Zahlen

**Gegeben**: Fuzzy-Set A mit Zugehörigkeitsfunktion $m_A(x)$:

$$m_A(x) = \begin{cases} (1+x)/4 & \text{für } x \in [-1 \ldots 3] \\ (5-x)/2 & \text{für } x \in [3 \ldots 5] \\ 0 & \text{sonst} \end{cases}$$

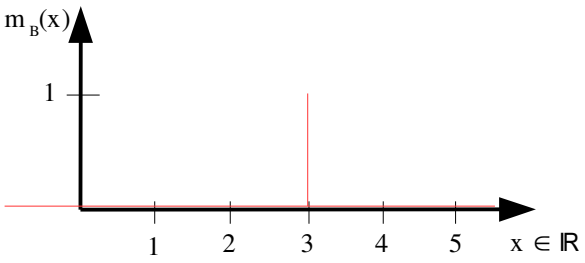
$\Rightarrow$ $m_A(x)$ beschreibt die unscharfe Zahl 3.




**Gegeben**: Fuzzy-Set mit $m_B(x)$

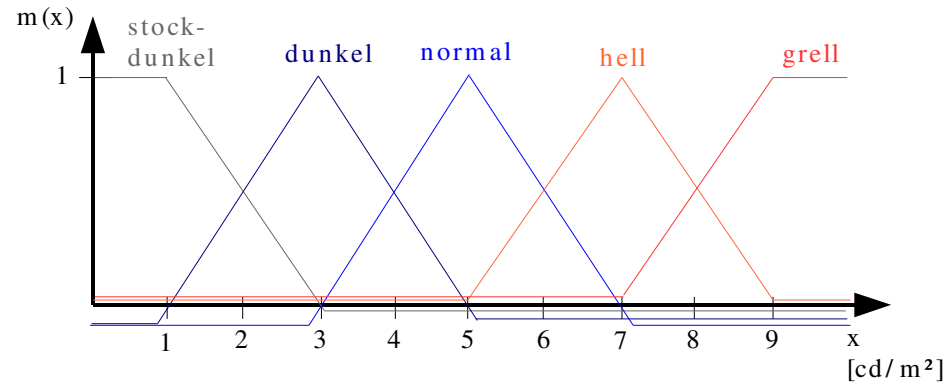
$$m_B(x) = \begin{cases} 1 & \text{für } x = 3 \\ 0 & \text{sonst} \end{cases}$$

$\Rightarrow$ $m_B(x)$ beschreibt die (scharfe) Zahl 3.




Auch Menschen benutzen gerne unscharfe Formulierungen. Diese werden mit linguistischen Variablen und linguistischen Werten modelliert.

**Beispiel**: Linguistische Variable „Helligkeit“




$$m_{hell}(x)=\begin{cases} 0 \text{ für } x\le 5 \vee x\ge 9 \\ \frac{1}{2}(x-5) \text{ für } 5<x<7 \\ -\frac{1}{2}(x-9) \text{ für } 7<x<9 \end{cases}$$

**Reglerbeschreibung:**

- Programmierung des Reglers durch Produktionsregeln, in denen unter Umständen auch Disjunktionen zugelassen sind.
- Die Bedingungen beziehen sich auf die Sensordaten.
- Die Aktionen sind Vorgaben an die Aktoren.
- Verwendung linguistischer Variablen und Werte auf beiden Seiten der Regel.

**Beispiel**:

```
if (Geschwindigkeit = hoch AND Abstand = gering) then Geschwindigkeit = niedrig
```

## Inferenzmechanismus

Um ein Schwingen des Regelkreises zu vermeiden, werden die Aktionen aller Produktionsregeln, die in einem Zyklus unifiziert wurden, zusammengefaßt. Der Bedingungsteil einer Fuzzy-Regel ist im Allgemeinen nicht vollständig erfüllt. Daher ist auch die vollständige Ausführung der Regel generell nicht sinnvoll. Stattdessen ist die Ausführung entsprechend dem Erfüllungsgrad der Bedingungen angebracht. Der Erfüllungsgrad kann z.B. mit der Min-Max-Inferenz-Methode bestimmt werden. <siehe Folie Beispiel 8.3>

- Fuzzifizierung der Meßwerte der Sensoren
- Bestimmung der Überdeckung der Fuzzy-Sets der Meßwerte mit dem Fuzzy-Set der linguistischen Variablen
- logische Verknüpfung der Einzelbedingungen und „Abschneiden“ der Fuzzy-Sets der Stellgröße
- Vereinigung der „abgeschnittenen“ Fuzzy-Sets
- Transformation des resultierenden Fuzzy-Sets in einen scharfen Wert (Defuzzifikation)

## Wichtige Methoden zur Defuzzifikation:

- Bildung des Mittelwerts aller Maxima

- Auswahl eines Maximums, z.B. von links nach rechts
- Schwerpunktbildung: $x_{res} = \frac{M}{F}$, $M = \int_{x_{min}}^{x_{max}} x f(x)\,dx$, $F = \int_{x_{min}}^{x_{max}} f(x)\,dx$

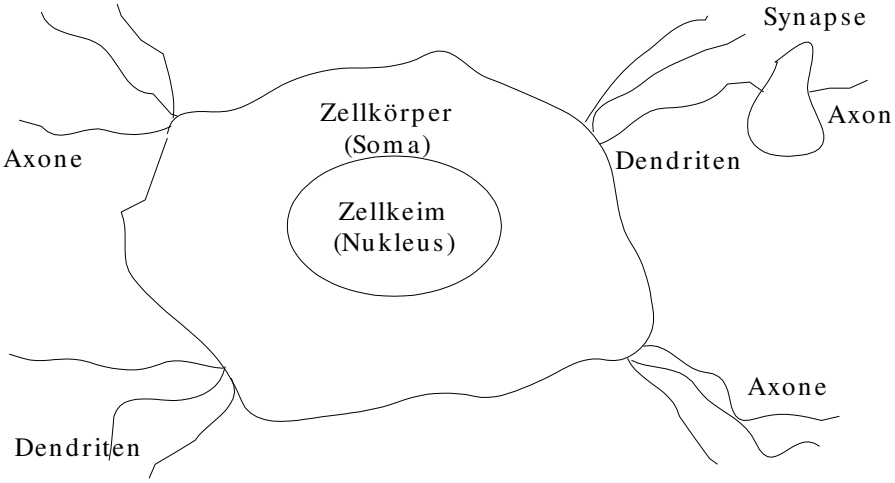
<siehe Handout>

## 2.5 Neuronale Netze

Die neuronalen Netze sind eine Wissensrepräsentation, die nach dem Vorbild des menschlichen Gehirns arbeitet. Sie versucht in Struktur und Funktionsweise Gehirnkomplexe nachzubilden, um dadurch eine Simulation menschlicher Denkvorgänge zu erreichen.

Aufbau des menschlichen Gehirns:

Es besteht aus mehr als 100 Milliarden Nervenzellen, sogenannter Neuronen.

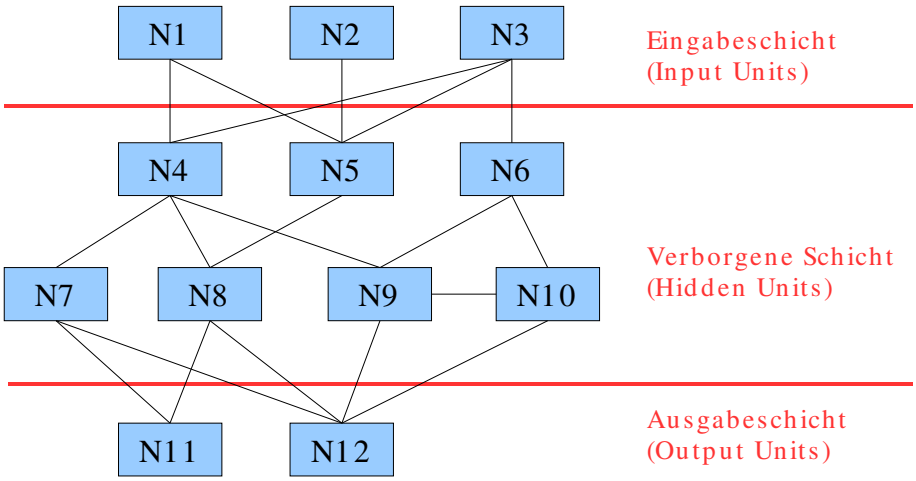



Der Zellkörper ist der Informationsträger. Die Information wird als Grad der Erregung der Zelle gespeichert. Im einfachsten Fall sind dies die beiden Zustände erregt und nicht erregt. Das Neuron gibt seine Erregung über Axone an andere Nervenzellen und Körperteile (Muskeln etc.) weiter. Ein Axon kann bis zu 1,5 m lang sein. Die Reize gelangen über die Dendriten in das Neuron. Am Ende einer Dendrite sitzt eine Synapse, ein Anschluss zu einem Axon einer anderen Zelle. Über die Synapsen können Erregungszustände verstärkt oder gedämpft an andere Neuronen weitergegeben werden. Überschreitet das weitergegebene Erregungssignal die Reizschwelle der verbundenen Nervenzelle (ca. 70 mV), so verändert diese Nervenzelle die eigene Erregung.

Die Eingabe von Information in das Gehirn erfolgt durch Reize, die durch Rezeptoren hervorgerufen werden. Zur Ausgabe dienen die Effektoren.

### Aufbau neuronaler Netze

Ein neuronales Netz besteht aus einer Menge von Prozessoren (Neuronen), die in Form einer Hierarchie miteinander verschaltet sind. Die Anzahl und die Verschaltung der Neuronen ist anwendungsspezifisch, z.B. Perzeptron, Kohonen-Netz. Das hierarchische Netz ist in Eingabeschicht, verborgene Schicht und Ausgabeschicht unterteilt.




Der Informationsfluss erfolgt von der Eingabeschicht über die verborgene Schicht zur Ausgabeschicht oder umgekehrt. Jedes Neuron besitzt eine Menge von Ein- und Ausgängen. Jedes Eingangssignal wird mit einem Faktor

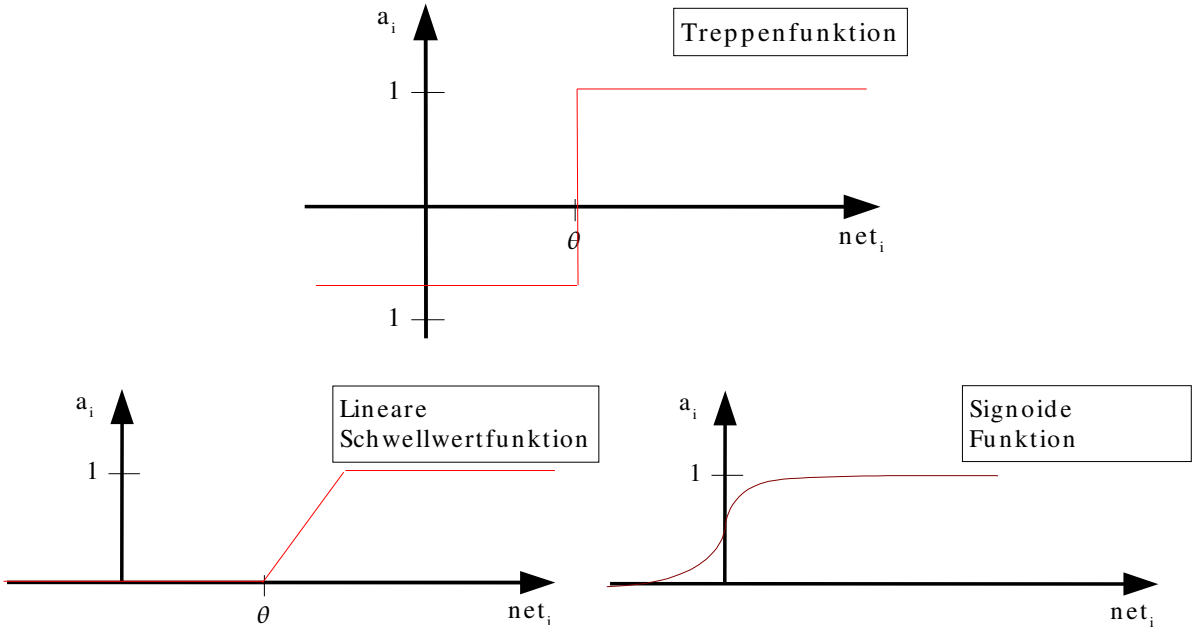
gewichtet. Die Verknüpfung der gewichteten Eingangssignale ruft im Neuron eine Aktivierung hervor, falls das Ergebnis der Verknüpfung einen Schwellwert überschreitet. Diese Aktivierung wird über die Ausgänge an andere Neuronen weitergegeben.

## Funktionen eines Neurons

Seien $o_j$ die Ausgangsaktivierung des Neurons $N_j$, $w_{ij}$ die Gewichtung der Verbindung zwischen den Neuronen $N_j$ und $N_i$.

- Propagierungsfunktion: $net_i = \sum_j w_{ij} \cdot o_j$
- Aktivierungsfunktion: $a_i(t+1) = F_i(net_i(t+1), \text{Eingabereize}_i(t+1), a_i(t))$
- Ausgangsfunktion: $o_i(t) = f_i(a_i(t))$

Gängige Aktivierungsfunktionen beschränken sich auf $net_i(t+1)$ als Parameter. Die wichtigsten Aktivierungsfunktionen sind:




Die Ausgabefunktion schaltet in der Regel den Aktivierungszustand ohne Veränderung weiter.

Damit ist die Hauptanwendung von neuronalen Netzen die Erkennung von Mustern. Die Aktivierung von Neuronen der Eingabeschicht von außen ruft die Aktivierung von Neuronen der Ausgabeschicht hervor. Ein derartiger Betrieb des neuronalen Netzes nennt man Recall-Phase.

Damit das neuronale Netz aus dem Eingabemuster das richtige Ausgabemuster erzeugt, muss es in einer Trainingsphase die Mustererkennung erlernen. Das Lernen erfolgt durch Veränderung der Gewichte der Verbindungen.

Das Lernen kann nach verschiedenen Lernregeln erfolgen:

**Delta-Regel:** Lernregel für zweischichtige Netze

$$w_{ij}^{neu} = w_{ij}^{alt} + \Delta w_{ij}$$

$$\Delta w_{ij} = \epsilon \cdot \delta_i \cdot o_j$$

$$\delta_i = z_i - o_i$$

mit:

$z_i$ Ziel (gewünschter Output) des Neurons $i$

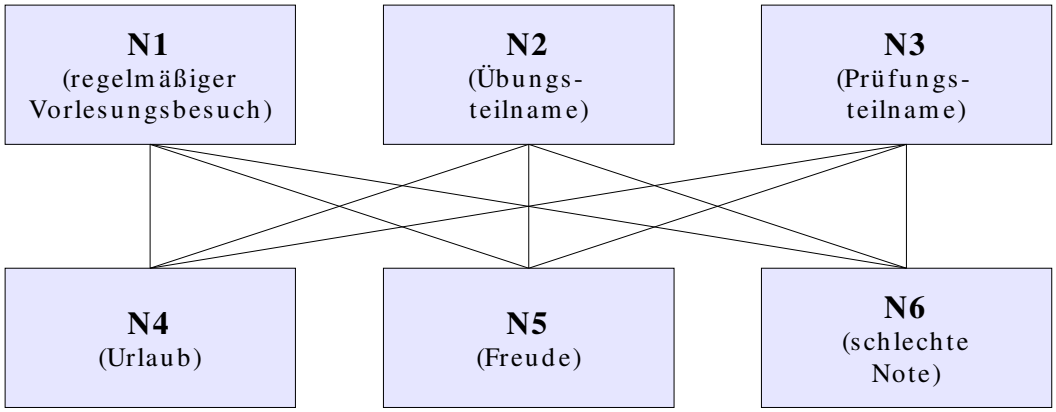
$\delta_i$ Fehlersignal des Neurons $i$

$\epsilon$ Lernrate $\epsilon \in [0 \ldots 1]$

Die Lernrate bestimmt die Größe der Lernschritts. Für $\epsilon = 0$ wird nicht gelernt. Ist $\epsilon = 1$, dann werden die Gewichte, die schon vorher erlernte Muster erkennen konnten, wahrscheinlich verdrängt. Das Netzwerk merkt sich das Muster so stark, dass die alten Muster unter Umständen nicht mehr erkannt werden.

Wird für ein Eingabemuster kein passendes Ausgabemuster erzeugt, so wird die Differenz zwischen Soll- und Istausgang bestimmt. Danach speist man die Differenz über die Ausgabe in das Netz ein und korrigiert die Gewichte. Dadurch ergibt sich bei der nächsten Erkennung ein Ausgabemuster, das näher am Sollwert liegt. Recall- und Trainingsphase wechseln sich ab.

**Beispiel**: Training eines neuronalen Netzes




- Aktivierungsfunktion: $a_i(t) = \begin{cases} 1 & \text{falls } net_i > 0 \\ -1 & \text{sonst} \end{cases}$
- Initiale Gewichte: $w_{ij} = 1 \quad \forall i, j$
- Lernrate: $\epsilon = \frac{1}{3}$

**Recall-Phase**

Eingabemuster: $N1 = N2 = N3 = 1$
Sollausgabemuster: $N4 = N5 = 1$, $N6 = -1$

$net_4 = 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 = 3 \quad \Rightarrow a(net_4) = 1 \quad \Rightarrow o_4 = 1$
$net_5 = \quad \Rightarrow o_5 = 1$
$net_6 = \quad \Rightarrow o_6 = 1$

Istausgabemuster: $N4 = N5 = N6 = 1 \approx >$ N6 nicht in Ordnung

**1. Trainings-Phase**

$\Delta w_{4,1} = \frac{1}{3} \cdot (1-1) \cdot 1 = 0 \Rightarrow \quad w_{4,1}^{neu} = 1 + 0 = 1$

$\Delta w_{5,1} = \frac{1}{3} (1-1) \cdot 1 = 0 \Rightarrow w_{5,1}^{neu} = 1 + 0 = 1$

$$\Delta w_{6,1} = \frac{1}{3}(-1-1)\cdot 1 = \frac{1}{3}(-2) = -\frac{2}{3} \Rightarrow w_{6,1}^{neu} = 1 - \frac{2}{3} = \frac{1}{3}$$

$$w_{4,2}^{neu} = w_{5,2}^{neu} = w_{5,3}^{neu} = 1$$

$$w_{6,2}^{neu} = w_{6,3}^{neu} = \frac{1}{3}$$

**2. Recall-Phase**

$$net_4 = 1\cdot 1 + 1\cdot 1 + 1\cdot 1 = 3 \Rightarrow o_4 = 1$$

$$net_5 = \qquad\qquad = 3 \Rightarrow o_5 = 1$$

$$net_6 = \frac{1}{3}\cdot 1 + \frac{1}{3}\cdot 1 + \frac{1}{3}\cdot 1 = 1 \Rightarrow o_6 = 1 \text{ n.i.O}$$

**2. Trainingsphase**

$$\Delta w_{4,1} = \frac{1}{3}(1-1)\cdot 1 = 1 \quad \Rightarrow w_{4,1}^{neu} = 1$$

$$\Delta w_{5,1} = w_{4,2}^{neu} = w_{5,2}^{neu} = w_{4,3}^{neu} = w_{5,3}^{neu} = 1$$

$$\Delta w_{6,1} = \frac{1}{3}\cdot(-1-1)\cdot 1 = -\frac{2}{3} \Rightarrow w_{6,1}^{neu} = \frac{1}{3} - \frac{2}{3} = -\frac{1}{3}$$

$$w_{6,2}^{neu} = w_{6,3}^{neu} = -\frac{1}{3}$$

**3. Recall-Phase**

$$net_4 = 1\cdot 1 + 1\cdot 1 + 1\cdot 1 = 3 \Rightarrow o_4 = 1$$

$$net_5 = \qquad\qquad = 3 \Rightarrow o_5 = 1$$

$$net_6 = -\frac{1}{3}\cdot 1 + \left(-\frac{1}{3}\right)\cdot 1 + \left(-\frac{1}{3}\right)\cdot 1 = -1 \Rightarrow a_6 = -1$$

**4. Recall-Phase**

$$N1 = N2 = -1, N3 = 1$$

$$net_4 = 1\cdot(-1) + 1\cdot(-1) + 1\cdot 1 = -1 \Rightarrow o_4 = -1$$

$$net_5 = \qquad\qquad = -1 \Rightarrow o_5 = -1$$

$$net_6 = -\frac{1}{3}\cdot(-1) + \left(-\frac{1}{3}\right)\cdot(-1) + \left(-\frac{1}{3}\right)\cdot 1 = \frac{1}{3} \Rightarrow o_6 = 1$$

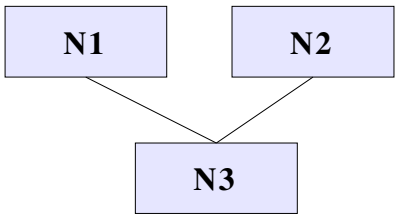
⇒ Sollausgabemuster

Die Delta-Regel ist nur für zweischichtige Netze anwendbar. Durch Prüfung der linearen Trennbarkeit kann festgestellt werden, ob ein Problem durch ein zweistufiges Netz gelöst werden kann. Dabei werden die Sollausgaben in ein Koordinatensystem eingetragen. Die Sollausgaben sind linear trennbar, wenn jede mögliche Lösung nur in einer zusammenhängenden Fläche vorkommt.

Ein weiteres Lernverfahren für zweischichtige Netze ist die Hepp'sche Regel. Diese besagt: Falls zwei Neuronen gleichzeitig aktiv sind, dann wächst die Stärke ihrer Verbindung. Sie läßt sich besonders gut bei Problemen mit orthogonalen Eingabevektoren anwenden.

$\Delta w_{ij} = \epsilon \cdot o_i o_j$

**Beispiel**: Logisches UND



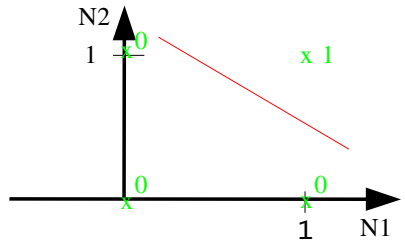

| N1 | N2 | N3 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Lernrate: $\epsilon = 1$

Initiale Gewichte: $w_{ij} = 0 \quad \forall i, j$

Aktivierungsfunktion: Treppenfunktion mit $\theta = 1{,}5$

Prüfungsfunktion lineare Trennbarkeit:




Bestimmung Lernregel: Bis auf Eingabevektor $N1 = N2 = 1$ orthogonal → Versuch Hebb'-sche Regel

Lernen von Muster 4:

$net_3 = 1 \cdot 0 + 1 \cdot 0 + 1 \cdot 0 \quad \Rightarrow o_3 = 0$ n.i.O:

$\Delta w_{3,1} = 1 \cdot 1 \cdot 1 = 1 \quad \Rightarrow w_{3,1}^{neu} = 0 + 1 = 1$

$\Delta w_{3,2} = 1 \cdot 1 \cdot 1 = 1 \quad \Rightarrow w_{3,2}^{neu} = 0 + 1 = 1$

$net_3 = 1 \cdot 1 + 1 \cdot 1 = 2 \quad \Rightarrow o_3 = 1$

Ein Lernverfahren, das die verborgene Schicht einschließt, ist das Backpropagation-Lernverfahren.

## Backpropagation-Algorithmus

Gegeben: Neuronales Netz mit verborgener Schicht

- $x_1, \ldots, x_n$ Eingabemuster
- $y_1, \ldots, y_m$ Sollausgabemuster
- $y_1^*, \ldots, y_m^*$ Istausgabemuster

Dann berechnet sich der quadratische Fehler $E$ wie folgt:

$$E = \frac{1}{2} \sum_{k=1}^{m} \left( y_k - y_k^* \right)^2$$

Der Faktor $\frac{1}{2}$ ist zur Vereinfachung der Formeln zur Berechnung der Gewichte. Die Änderung der Gewichte berechnet sich wie folgt: $\Delta w_{ij} = -\epsilon \frac{\partial E}{\partial w_{ij}}$

Der Differentialquotient wird über die Kettenregel gebildet.

$$\Delta w_{ij} = -\epsilon \frac{\partial E}{\partial net_i} \cdot \frac{\partial net_i}{\partial w_{ij}}$$

Fehlersignal für das Neuron $i$:

$$\delta_i = \frac{-\partial E}{\partial net_i}$$

$$\frac{\partial net_i}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_j o_i w_{ij} = o_j$$

Daraus folgt:

$$-\epsilon \frac{\partial E}{\partial net_i} \cdot \frac{\partial net_i}{\partial w_{ij}} = \epsilon \delta_i \cdot o_j$$ (Delta-Regel für zweistufige Netze)

$$\delta_i = (z_i - o_i) \cdot F'(net_i)$$

Für den Backpropagation-Algorithmus wird eine signoide Aktivierungsfunktion gewählt:

$$a_i(t) = \frac{1}{1 + e^{-net_i(t)}}$$

$$F'(net_i) = a_i \cdot (1 - a_i)$$

mit Identität als Ausgabefunktion:

$\delta_i = (z_i - o_i) \cdot o_i (1 - o_i)$ für die Neuronen der Ausgabeschicht

$\delta_j = \sum_k \delta_k \cdot w_{jk} \cdot F'(net_j)$ für die Neuronen der inneren Schichten

$= o_j (1 - o_j) \sum_k \delta_k w_{kj}$

**Beispiel**: Signoide Funktion

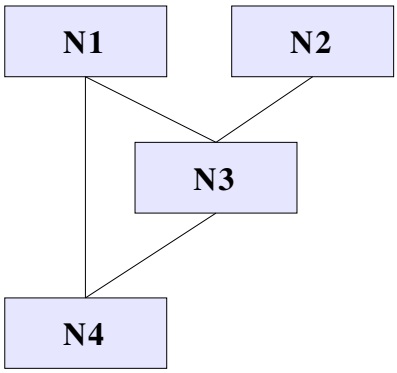
Eingabemuster: $N1 = 1, N2 = 2$

Ausgabemuster: $N4 = \frac{1}{2}$

Lernrate: $\epsilon = \frac{3}{4}$

Initiale Gewichte: $w_{ij} = 1 \quad \forall\, ij$

Aktivierungsfunktion: $a(net_i) = \frac{1}{1 + e^{-net_i}}$




**1. Recall-Phase:**

$net_3 = 1 \cdot 1 + 1 \cdot 2 = 3 \Rightarrow o_3 = 0{,}95$

$net_4 = 1 \cdot 1 + 1 \cdot 0{,}95 = 1{,}95 \Rightarrow o_4 = 0{,}87$

**1. Trainings-Phase:**

$$\delta_4 = \left(\frac{1}{2} - 0{,}87\right) \cdot 0{,}875 \cdot (1 - 0{,}87) = -0{,}04$$

$$\Delta w_{4,1} = \frac{3}{4}(-0{,}04) \cdot 1 = -0{,}03$$

$$w_{4,1}^{neu} = 0{,}97$$

$$w_{4,3}^{neu} = 0{,}97$$

$$\delta_3 = 0{,}95 \cdot (1 - 0{,}95) \cdot (-0{,}04) = -0{,}0019$$

$$\Delta w_{3,1} = \frac{3}{4} \cdot (-0{,}0019) \cdot 1 = -0{,}0014$$

$$w_{3,1}^{neu} = 0{,}9986$$

$$w_{3,2}^{neu} = 0{,}9973$$

**2. Recall-Phase**

$$o_4 = 0{,}869$$

**52. Trainings-Phase**

$$w_{3,1}^{neu} = 0{,}9756,\ w_{3,2}^{neu} = 0{,}9512,\ w_{4,1}^{neu} = -0{,}0058,\ w_{4,3}^{neu} = 0{,}0455$$

**52. Recall-Phase**

$$o_4 = 0{,}5093$$

**90. Trainings-Phase**

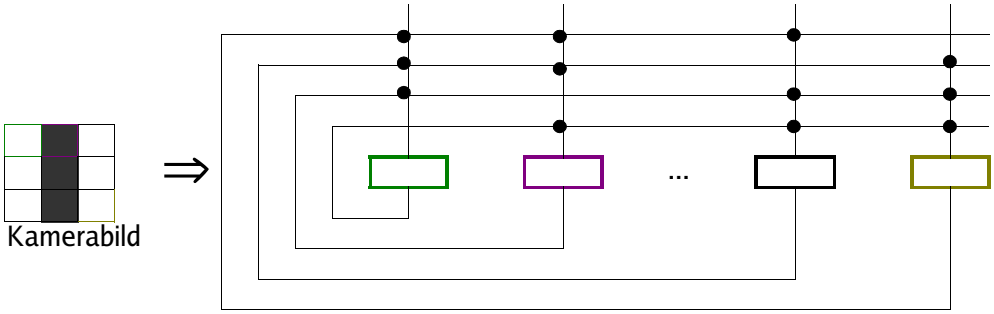
$$w_{3,1}^{neu} = 0{,}9755,\ w_{3,2}^{neu} = 0{,}9511,\ w_{4,1}^{neu} = -0{,}0249,\ w_{4,3}^{neu} = 0{,}0274$$

**91. Recall-Phase**

$$o_4 = 0{,}500271$$

Erhöhung der Lernrate auf 1 bringt Verkürzung auf 44 Trainingsphasen.

Die bisher behandelten Netze sind Feedforward-Netze, weil bei der Recall-Phase keine Zyklen auftreten. Feedback-Netze beinhalten Rückkoppelungen. Die Recall-Phase terminiert dann, sobald ein stabiler Erregungszustand bei allen Neuronen erreicht ist. Ein bekannter Vertreter der Feedback-Netze sind die Hopfield-Netze. Sie werden zur Mustererkennung bei SW-Bildern eingesetzt. Der Physiker Hopfield beschäftigte sich mit dem Verhalten von Spingläsern. Daraus hat er die Theorie der Hopfield abgeleitet.




Topologie des Hopfield-Netzes für das nebenstehende Kamerabil

Hinweis: die Verdrahtung der einzelnen Neuronen erfolgt auf alle anderen Knoten bis auf sich selbst!

Aktivierungsfunktion: $a_i(t+1)=\begin{cases}1 \text{ falls } net_i(t)\geq\theta\\ -1 \text{ sonst}\end{cases}$

$\theta$ ist in der Regel gleich $0$.

Energiefunktion: $E=-\frac{1}{2}\sum_i\sum_j w_{ij}\cdot o_j\cdot o_i$

Abbildung des Hopfield-Netzes in eine Matrix, so daß gilt:

$\vec{a}$ ein erkanntes Msuter, $W$ Matrix des Netzes $\Rightarrow \vec{a}=W\cdot\vec{a}$.

Wenn für Vektoren $\vec{a}_1,\ldots,\vec{a}_n$ Matrizen $w_1,\ldots,w_n$ für die Hopfield-Netze berechnet wurden, dann gilt:

$$w=w_1+\ldots+w_n$$

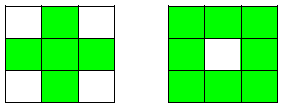
$w$ erkennt die Muster $\vec{a}_1,\ldots,\vec{a}_n$.

Für Netze mit einigen hundert Neuronen gilt:

$$M_{max}=0{,}15\cdot N$$

mit $M$ Anzahl speicherbarer Muster, $N$ Anzahl der Neuronen.

**Beispiel**: Hopfield-Netz für die Muster



$\vec{a}_1$ $\quad$ $\vec{a}_2$

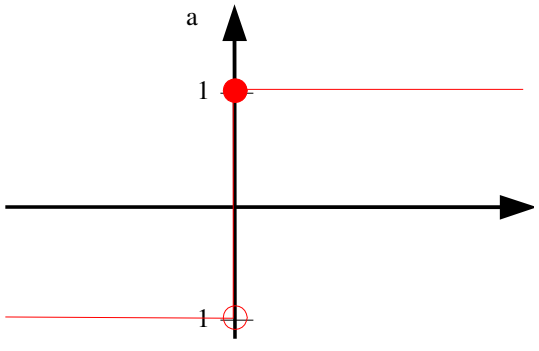
$w_1$ $\quad$ $w_2$

$\vec{a}_1=W_1\vec{a}_1\Leftrightarrow$

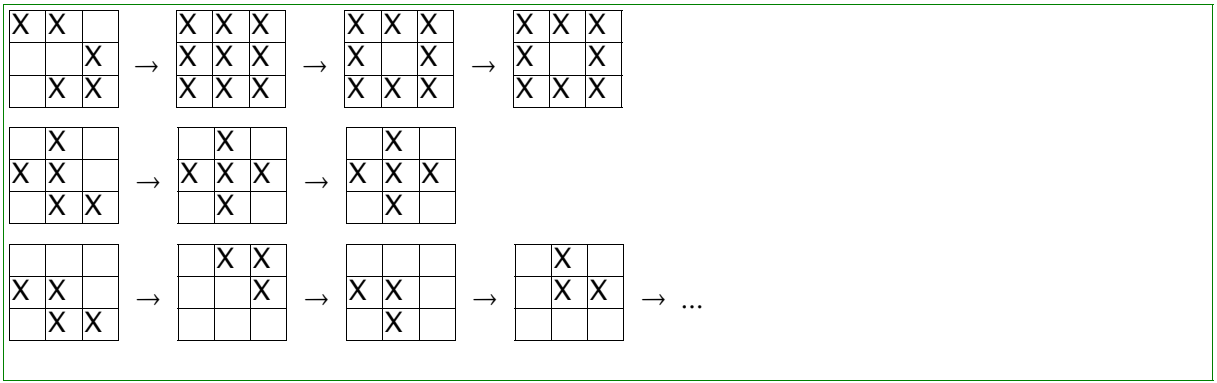
$\vec{a}_2=W_2\vec{a}_2\Rightarrow W=W_1+W_2$

$$\begin{pmatrix}-1\\1\\-1\\1\\1\\1\\-1\\1\\-1\end{pmatrix}=\begin{bmatrix}0&-1&1&-1&-1&-1&1&-1\\-1&0&-1&1&1&1&-1&1\\1&-1&0&-1&-1&-1&1&-1\\-1&1&-1&0&1&1&-1&1\\-1&1&-1&1&0&1&-1&1\\-1&1&-1&1&1&0&-1&1\\1&-1&1&-1&-1&-1&0&-1\\-1&1&-1&1&1&1&-1&0\\1&-1&1&-1&-1&-1&1&-1\end{bmatrix}$$

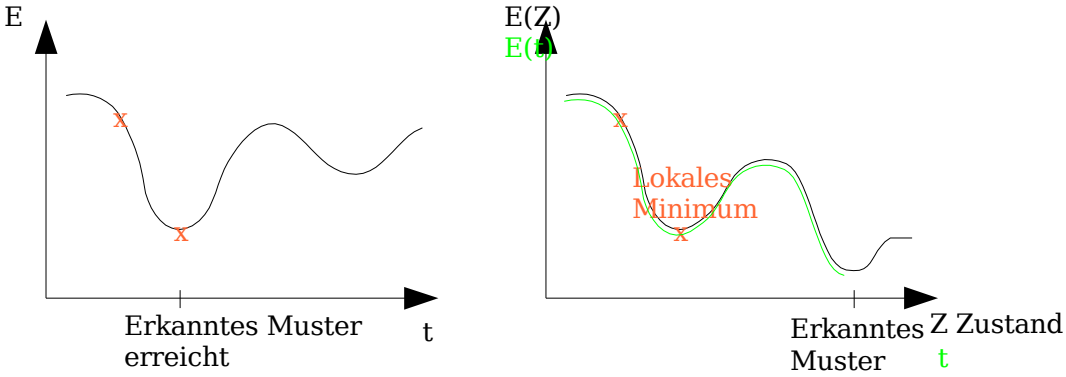
$$W=\begin{bmatrix}0&0&2&0&-2&0&2&0&2\\0&0&0&2&0&2&0&2&0\\2&0&0&0&-2&0&2&0&2\\0&2&0&0&0&2&0&2&0\\-2&0&-2&0&0&0&-2&0&-2\\0&2&0&2&0&0&0&2&0\\2&0&2&0&-2&0&0&0&2\\0&2&0&2&0&2&0&0&0\\2&0&2&0&-2&0&2&0&0\end{bmatrix}$$




**Beispiel**: Erkennung von gestörten Mustern



Jedem gelernten Muster entspricht einem Energieminimum.




Umgehen lokaler Minima durch Simulated Annealing.

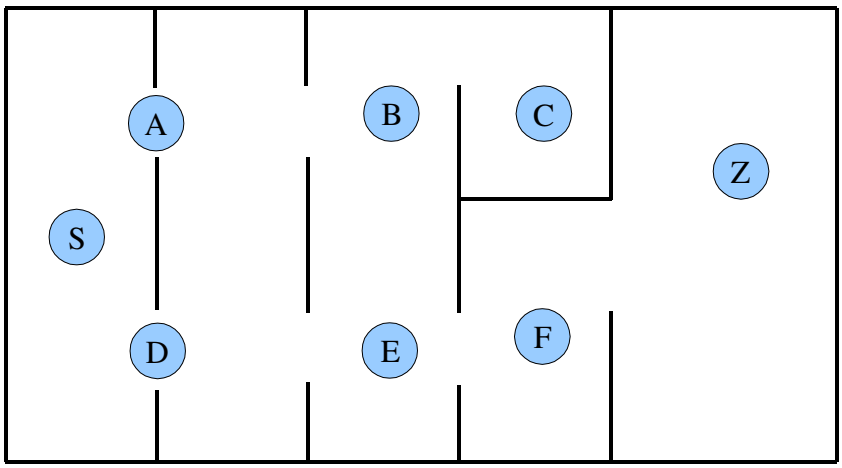
# 3 Problemlösungsstrategien

Man unterscheidet zwei Klassen von Problemlösungsstrategien:

- allgemeine Strategien
  einheitliche Vorgehensweisen, die für verschiedene Probleme anwendbar sind
  Vorteile: Standardisierung und Übertragbarkeit
  Betrachteter Vertreter: Suchverfahren
- problemspezifische Strategien:
  Vorgehensweisen, die in der Regel aus einer allgemeinen Strategie abgeleitet und auf spezielle Problemklassen zugeschnitten wurden.
  Vorteil: Höhere Leistungsfähigkeit innerhalb der Problemklasse.
  Betrachteter Vertreter: Planungsverfahren.

## 3.1 Suchverfahren

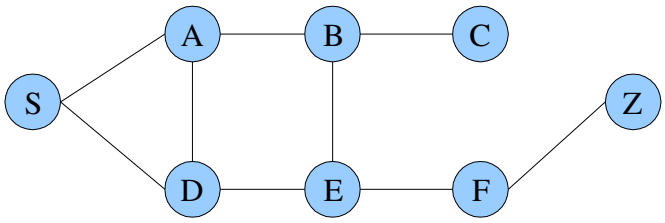
Szenario: Wegesuche




S Startpunkt
Z Zielpunkt

Die Wegsuche läßt sich als Suche im zugehörigen Graphen beschreiben.

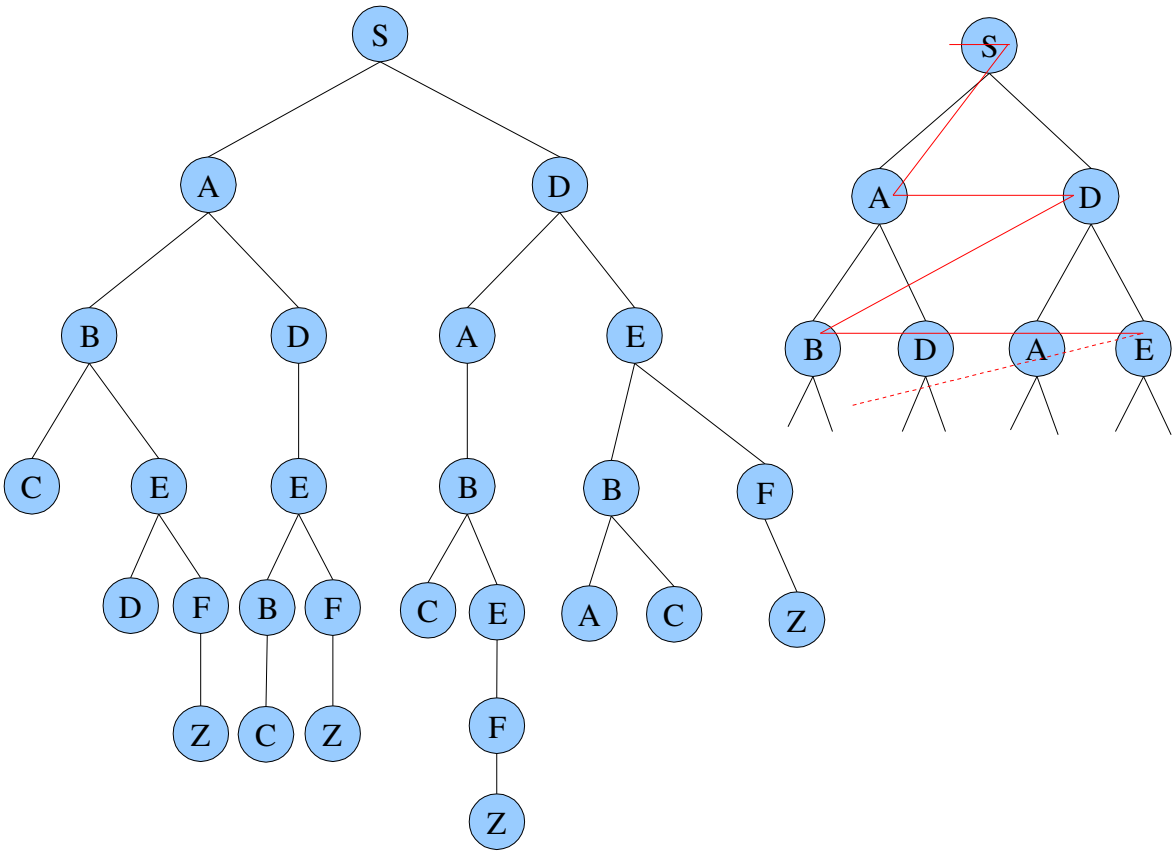
Sichtbarkeitsgraph:




$$G=\{(S,A),(A,S),(S,D),(D,S),(A,D),(D,A),(A,B),(B,A),(D,E),(E,D),$$
$$(B,E),(E,B),(B,C),(C,B),(E,F),(F,E),(F,Z),(Z,F)\}$$

Für die Lösung kommen in dieser Vorlesung nur Pfade ohne Zyklen in Frage. Dies ist im Allgemeinen nicht so.

Suchbaum:




Der Suchbaum steht für ein gegebenes Problem vorab im Allgemeinen nicht zur Verfügung, sondern wird im Zuge der Wegsuche aufgebaut. Der Aufbau des Suchbaumes kann nach verschiedenen Verfahren erfolgen. Die wichtigsten Verfahren werden nachfolgend betrachtet.

## Breitensuche

Der Suchbaum wird schichtenweise aufgebaut, d.h. es werden stets alle Nachfolger zu Knoten einer Schicht erzeugt, bevor in die nächste Schicht gewechselt wird. Die erste Schicht besteht nur aus dem Startknoten.

Prozedur Breitensuche
Eingabe: Graph G, Startknoten S, Zielknoten Z

(a) Initialisiere Liste L mit S

(b) Solange Ende noch nicht erreicht, wiederhole

b.1 Setze P als erstes Element von L

b.2 Prüfe, ob P gleich Z ist
Falls ja, Ende.

b.3 Lösche erstes Element in L

b.4 Strategische Aktionen

(a) Füge alle Nachbarn von P in G ans Ende von L

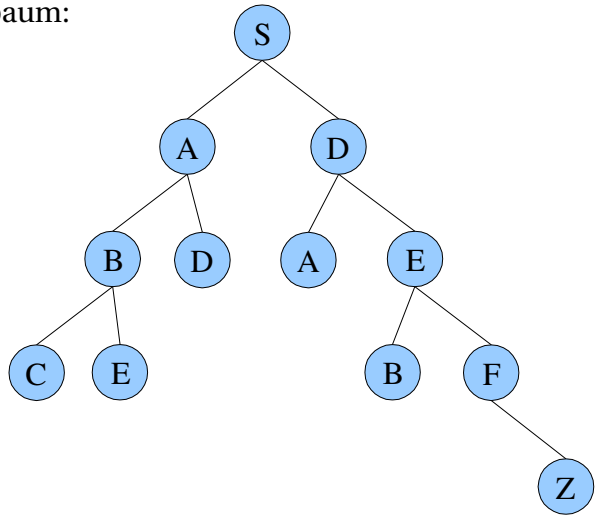
(b) lösche diejenigen neuen Knoten, die Pfade mit Zyklen ergeben

**Beispiel**: Listenentwicklung für Breitensuche

Listenentwicklung:

S
AD
DBD
BAE
ECE
CBF
F
Z

Suchbaum:




Die Breitensuche ist vollständig.

## Tiefensuche:

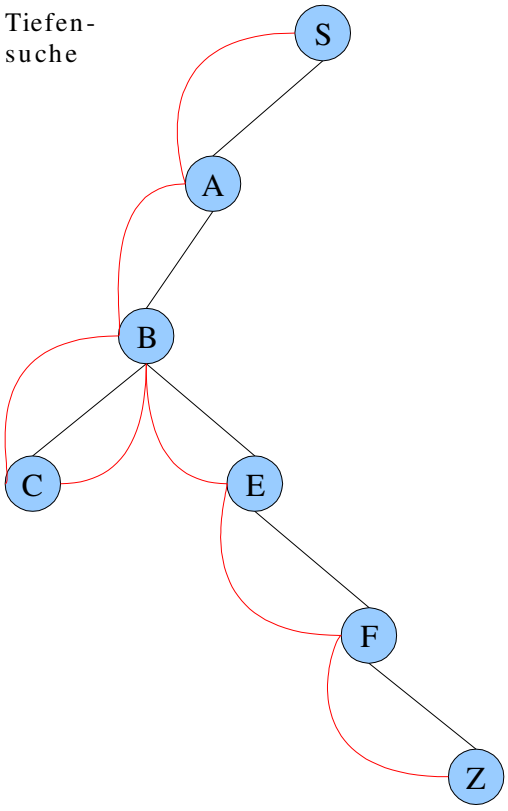
- Bei der Tiefensuche wird ein expantierter Knoten sofort weiter expandiert. Bei Sackgassen wird durch Backtracking in frühere expandierte Knoten zurückgesetzt.

  Prozedur Breitensuche
  Eingabe: Graph G, Startknoten S, Zielknoten Z

  (a) füge alle Nachbarn von P in G an den Anfang von L

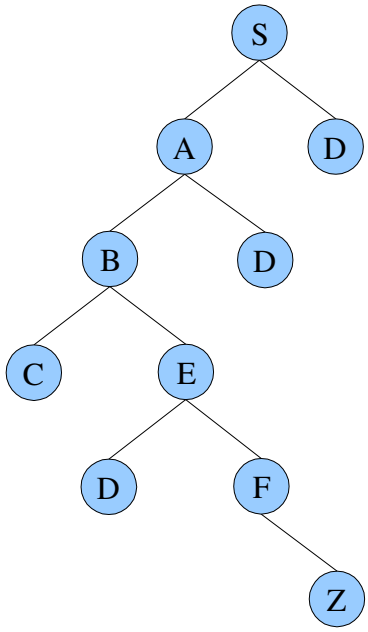
  (b) lösche alle neuen Knoten, die Pfade mit Zyklen darstellen

Tiefensuche



**Beispiel**: Listenentwicklung für Tiefensuche + <Grafik 3>

S
AD
BDD
CED
ED
BFD
ZD




Mit chronologischem Backtracking ist die Suche vollständig.

Die beiden Verfahren Breiten- und Tiefensuche unterscheiden sich in der Reihenfolge, in der die Knoten des Suchbaums expandiert werden. Diese lässt sich durch Hinzunahme von Wissen über die Wegpunkte (Heuristiken) verbessern.

## Gradientenmethode (Hill-Climbing)

Diese Methode befolgt die einfache Heuristik:

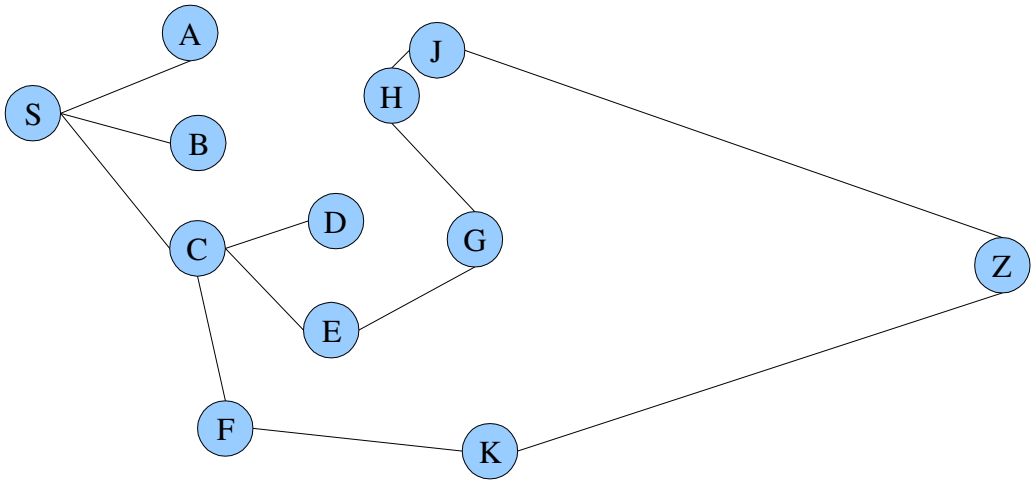
„Nimm den Knoten, der dich dem Ziel am nächsten bringt".

Falls eine Abstandsfunktion für Stellungen definiert ist, kann diese Heuristik wie folgt formalisiert werden:

„Gehe von $P$ zu $P'$, falls $\text{Abstand}(P', Z)$ minimal unter den Nachfolgern $P'$ von $P$ ist."

**Strategische Aktionen:**

1. Füge alle Nachbarn von P an den Anfang von L. Sortiere die neuen Knoten nach dem genschätzen Abstand zum Ziel.
2. Lösche alle neuen Knoten, die Pfade mit Schleifen darstellen.




Problem: Es kann vorkommen, dass nach der Abstandsfunktion lokal bestimmte Nachfolger weiter vom Ziel entfernt sind als Knoten, die bereits früher expandiert wurden. Eine Verbesserung stellt die „Best-First-Suche“ dar.

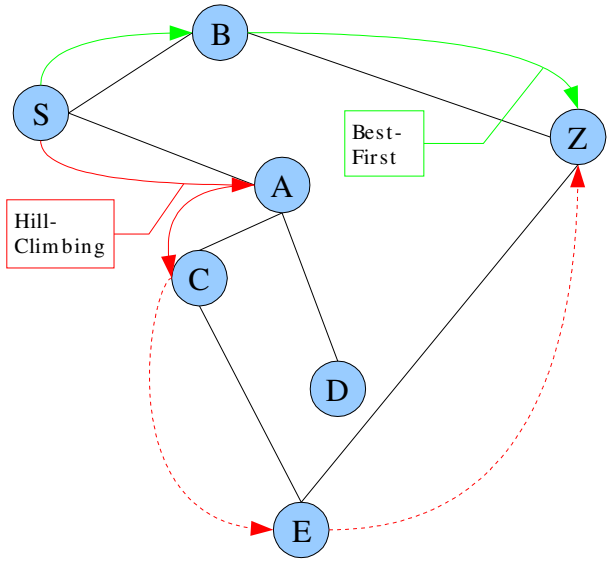
## Best-First-Suche

Dieses Verfahren arbeitet analog zum Hill-Climbing. Die Sortierung erfolgt jedoch über die gesamte Liste.

**Strategische Aktionen:**

a. Füge alle Nachbarn von P in L ein. Sortiere alle Knoten in L nach dem geschätzten Abstand zum Ziel.

b. Lösche die (neuen) Knoten, die Pfade mit Schleifen bilden.

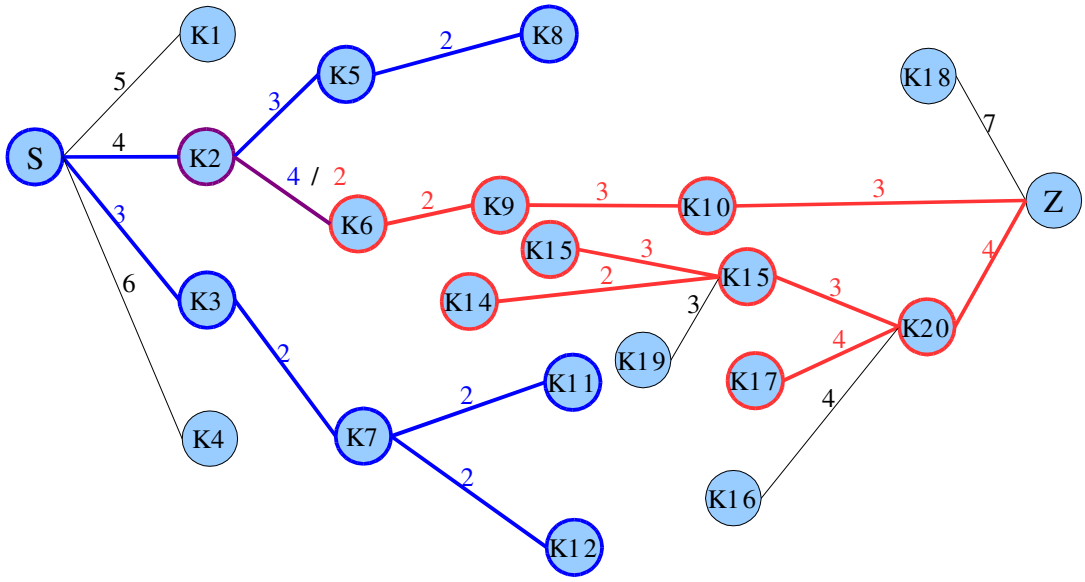



## Bidirektionale Suche

Die bisherigen Suchverfahren gehen von einem Startknoten zu einem Zielknoten. Lässt man dynamisch mehrere Start- und Zielknoten zu, so kann parallel ausgehend von den jeweiligen Start- und Zielknoten mit Breitensuche gesucht werden. Die Suche terminiert, sobald ein expandierter Knoten einem Startpunkt des entgegen gerichteten Suchlaufs ent-

spricht.
Das Verfahren wird sehr häufig in Kombination mit der Strahlensuche verwendet. Die Strahlensuche expandiert die Knoten einer Ebene entsprechend der Breitensuche. Danach wird die Menge der expandierten Knoten über eine Heuristik auf eine sinnvolle Anzahl von Knoten reduziert. Diese werden erneut expandiert.

**Beispiel**: Bidirektionale Strahlensuche mit Abstand als Kantengewicht und der Breite 2




## Suche über Nadelöhrmengen

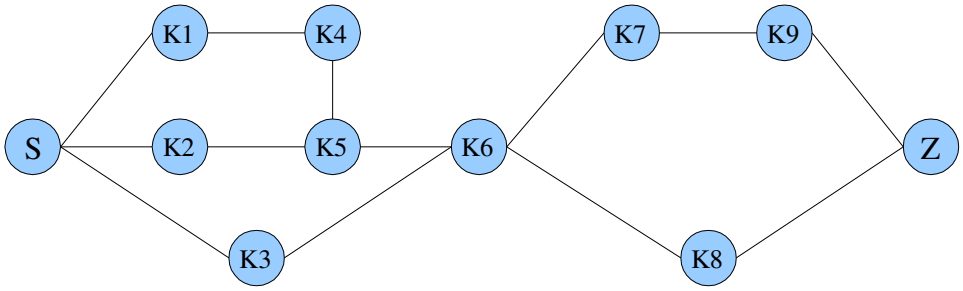
Je nach Problemstellung kann es im Graphen Knotenmengen geben, die in vielen Lösungen enthalten sind. Diese Knoten sind daher neuralgische Punkte oder Engstellen, durch die der gesuchte Weg mit großer Wahrscheinlichkeit führt. Diese Knoten nennt man Nadelöhrmenge.

**Beispiel**:

Donaubrücken sind Nadelöhre in Ingolstadt

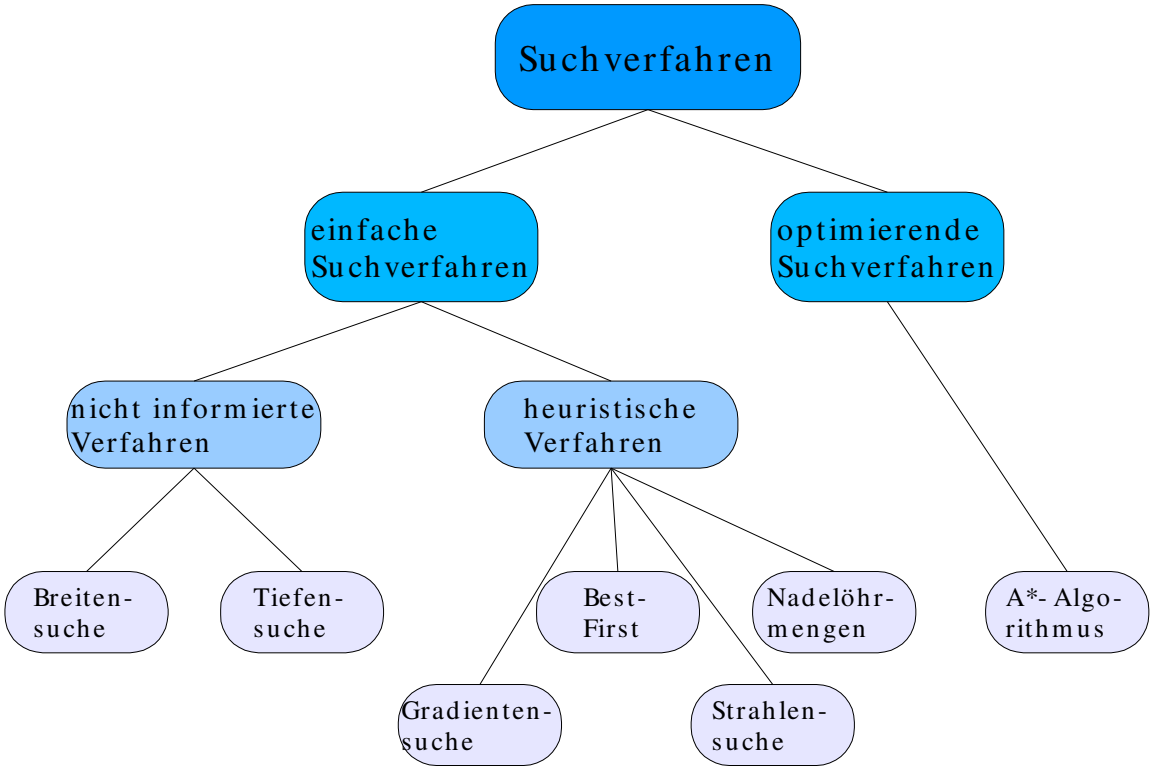
Lässt sich bei einem Problem eine Nadelöhrmenge finden, so liegt es nahe, die Suche zu parallelisieren. Es wird z.B. bidirektional vom Startknoten zur Nadelöhrmenge und vom Ziel zur Nadelöhrmenge gesucht.

**Beispiel**:




Die Menge $N=\{K6\}$ ist eine Nadelöhrmenge.

## Übersicht




## Optimierende Suchverfahren

Während bisher die grundsätzliche Erreichbarkeit des Zielknotens im Vordergrund stand, sollen nun beste Wege gesucht werden. Dazu betrachten wir Graphen, bei denen jeder Kante ein Gewicht zugeordnet ist. Das Gewicht einer Kante entsteht durch eine Kostenfunktion. Ein Weg von $S$ nach $Z$ ist optimal, wenn für ihn die niedrigsten Gesamtkosten entstehen.

## A*-Algorithmus

Für die Bewertung eines Wegpunktes Y als Nachfolger des Wegpunktes X werden stehen folgende Funktionen zur Verfügung:

$g(Y)$: tatsächliche Kosten vom Startknoten zum Knoten $Y$ über Knoten $X$

$h(Y)$: geschätzte Kosten des besten Weges von $Y$ zum Zielknoten

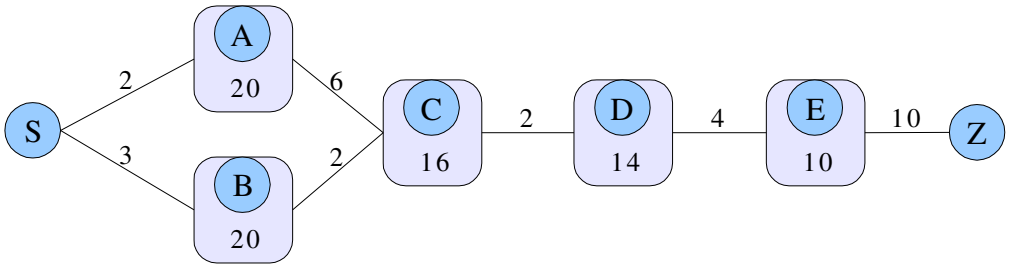
Liegt $Y$ auf dem besten Weg vom Startknoten zum Zielknoten und ist $h$ exakt, dann ist $K(Y)=g(Y)+h(Y)$ gleich der Kosten des besten Weges vom Start- zum Zielknoten. Für einen Nachfolger $W$ von $X$, der vom besten Weg abweicht, gilt: $K(W)<K(Y)$. Damit ist der Weg von $X$ über $W$ suboptimal.

**Strategische Aktionen:**

1. Expandierung von $P$ liefert $X_1,\ldots,X_n$
2. Berechne $K(x_i)=g(x_i)+h(x_i)$ für $i=1,\ldots,n$
   Füge $X_1,\ldots,X_n$ in $L$ ein
   Sortiere $L$ nach $K$
3. Lösche neue Knoten aus $L$, die im Suchbaum schon vorher aufgetreten sind, falls vorheriger Knoten einen niedrigeren $g$-Wert hat.
   Ansonsten lösche Nachfolger des vorherigen Knotens und lösche vorherigen Knoten.

Die Aktualisierung der Nachfolger eines gelöschten Knotens ist notwendig, falls dieser bereits expandiert wurde.

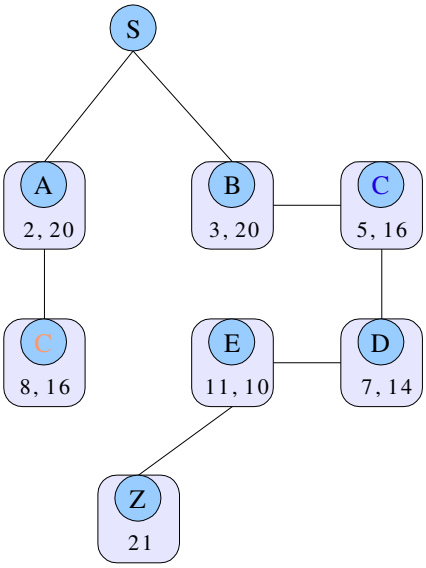
**Beispiel**: A*-Algorithmus




Listenentwicklung

1. S
2. A B
3. C B ~> B C
4. C C ~> C
5. D
6. E
7. Z

Suchbaum




Eine Funktion $h$ zur Bewertung eines Knotens heißt zulässig, wenn sie die Kosten, um zum Ziel zu kommen, nicht überschätzt. Für jeden Knoten $n$ und jeden Zielknoten $Z$, der von $n$ aus erreicbar ist, gilt: $h(n) \leq g(z) - g(n)$. Ist die Funktion $h$ zulässig, so liefert der A*-Algorithmus für das Problem eine optimale Lösung. Für praxisrelevante Problemstellungen ist es oft schwierig, eine zulässige Schätzfunktion $h$ zu finden. Daher läßt man diese Forderung im Hinblick auf eine Effizienzsteigerung bei der Suche gelegentlich fallen. Der A*-Algorithmus schließt Mehrfachwege zu einem Knoten aus und reduziert damit den Suchraum.

## 3.2 Planungsverfahren

Die Problemklasse der Planungsprobleme ist in der Technik sehr verbreitet. Beispiele für Planungsprobleme sind

- Konfigurieren
- Ressourcenplanung
- Aktionsplanung
- Trajektorienplanung bei Robotern

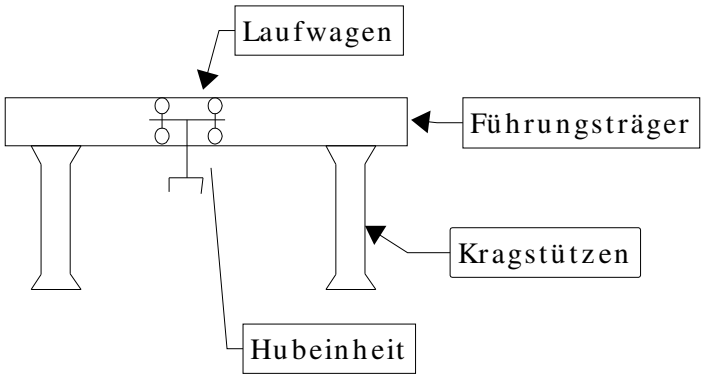
Ein Konfigurationsproblem KP ist ein Tupel der Form:

$$KP = (B, R, G, A, F)$$

mit
- B: Bauteilmenge
- R: Randbedingungsmenge für B
- G: Bewertungsfunktion für R
- A: Aggregation von B nach O mit O zusammengesetztes Objekt
- F: Anforderung für O mit $F = (RF, GF, HF, H)$
  - RF: Menge der Randbedingungen für O
  - GF: Bewertungsfunktion für RF
  - HF: Mindestbewertung für GF
  - H: Mindestbewertung für F

Die Aggregation A simuliert das Zusammensetzen der gewählten Bauteile aus B zu einem konfigurierten Objekt O. Die Aggregation projeziert beim Zusammensetzen die Eigenschaften der Bauteile aus B in Eigenschaften der Objekte O. Dies ist nötig, weil der Auftraggeber für die Konfigurierung in der Regel nur die Eigenschaften des zusammengesetzten Objekts kennt und nur darüber seine Randbedingungen formulieren kann. Die Eigenschaften der Bauteile will der Auftraggeber in der Regel auch nicht kennen.

**Beispiel**: Konfigurieren eines Ladeportals




| | |
|---|---|
| B | Menge aller Typen von Kragstützen, Führungsträgern, Laufwägen und Hubeinheiten |
| R | Menge aller technischen Restriktionen (Bremsweg des Laufwagens, Traglast der Hubeinheit |
| R | Das Ladeportal ist mit den vorhandenen Lagerbeständen produzierbar und etabliert einen neuen Laufwagen im Markt |
| A | Abbildung der Bauteileigenschaften in Eigenschaften des Portals (Gesamtpreis, Wartungsintervall) |
| GF | Das Portal ist hochwertig und kostengünstig |
| RF | Kundenrestriktion für Preis, Lieferzeit, Wartungsintervall |
| H, HF | Mindestqualitätsmaße, ab der der Kunde das Portal kauft |

Geeignete Wissenrepräsentationen für Konfigurationsprobleme sind Constraints in Verbindung mit Produktionsregeln.

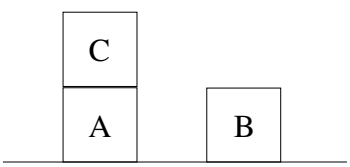
Planungsprobleme unterscheiden sich von Konfigurationsproblemen in der zeitlichen Anordnung der gewählten Objekte im Plan. Der Plan besteht aus einer Liste von Aktionen.

## STRIPS

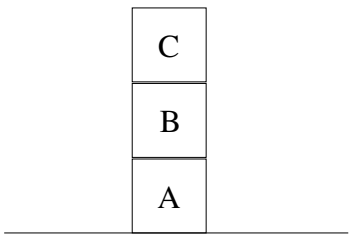
Anwendung der Suchverfahren zur Planung in STRIPS (Standford Research Institute Problem Solver). STRIPS wird nachfolgend anhand der Blocksworld erklärt.

Die Suche erfolgt im Zustandsraum. Jeder Zustand ist beschrieben durch eine Menge von Merkmalen.

Zustand S




Zustand Z



Jeder Zustand ist durch eine Menge von Merkmalen folgender Prädikate beschrieben:

| | |
|---|---|
| On (x, y) | Klotz x liegt auf Klotz y |
| Clear (x) | Auf Klotz x liegt kein anderer Klotz |
| Table (x) | Klotz x liegt auf dem Tisch |

Das Planungsproblem ist in STRIPS gegeben durch einen Ausgangs- und einen Zielzustand. Der Übergang von einem Zustand in einen anderen erfolgt durch Anwendung von Aktionen (Operatoren). In der Blocksworld sind folgende Operatoren verfügbar:

| | |
|---|---|
| move (x, y, z) | Bewege x von y auf z |
| unstack (x, y) | Nimm x von y und lege ihn auf den Tisch |
| stack (x, y) | Hebe x vom Tisch und lege ihn auf y |

Der Übergang von einem Zustand kann durch beliebige Anwendung von Operatoren erfolgen. In jedem Operator ist durch eine Vorbedingung festgelegt, in welchen Fällen er angewendet werden kann. Die Nachbedingung eines Operators legt fest, welche Merkmale des Zustands neu hinzukommen und welche nicht mehr gelten.

Beispiel: move (x, y, z)

Vorbedingungen: Clear (z), Clear (x), On (x, y)

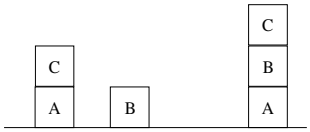
Nachbedingungen:
Hinzu: On (x, z), Clear (y)
Weg: Clear (z), On (x, y)


Die Problemlösung erfolgt nach den bekannten Suchverfahren.

Beispiel:

Übergang von obigem Zustand S zu Z:

unstack (C, A)
stack (B, C)
stack (A, B)




## Potentialfeldmethode

Anwendung bei Wegplanung für mobile Roboter: Die Lage eines zylindrischen Roboters wird durch eine Konfiguration p = (x, y) beschrieben. Die Kraft auf den Roboter ist der Gradient des Potentialfeldes u(p):

$$\vec{F}(p) = -\nabla u(p) = -\begin{pmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \end{pmatrix}$$

Der Roboter bewegt sich nach der Methode des steilsten Abstiegs auf das Ziel zu. Der Zielpunkt erhält ein anziehendes Potential. Durch abstoßendes Potential der Hindernisse wird ein größtmöglicher Abstand von den Hindernissen erreicht.

Sei:

$$\text{minabstand}(p) = \begin{cases} \text{minimaler Abstand des Punktes } p \text{ von den Hinternissen} & \text{für } p \text{ in Konfigurationsraum} \\ 0 & \text{für } p \text{ in einem Hindernis} \end{cases}$$

$$\text{zielabstand}(p) = \text{Abstand des Punktes } p \text{ vom Ziel } z$$

Möglische Definition des Potentials als

– anziehendes Potential: $U_{anz}(p) = A \cdot \text{zielabstand}(p)^2$

– Abstoßendes Potential: $U_{abst}(p)=\frac{1}{a_0+minabstand(p)}$

mit A Gewichtungsfaktor
$a_0$ Gewählte Potentialhöhe in den Hindernissen

Gesamtpotential: $U(p)=U_{anz}(p)+U_{abst}(p)$

Durch trichterförmige Verkleinerung des anzuziehenden Potentials wird die Planung effizienter.

## *Stichwortverzeichnis*