Grundlagen von Decision Support und Expertensystemen

Peter Becker
Hochschule Bonn-Rhein-Sieg
Fachbereich Informatik
peter.becker@h-brs.de

Vorlesung Wintersemester 2012/13

Allgemeines zur Vorlesung

- Homepage zur Vorlesung:
  `http://www2.inf.h-brs.de/~pbecke2m/xps/`
- Die Vorlesung wird folienbasiert gehalten.
- Die Folien zur Vorlesung (Skript) stehen auf der Homepage vor der Vorlesung zur Verfügung.
- Format: PDF, einseitig

Übungen

- zweistündig nach der Vorlesung
- Besprechung und Analyse der Aufgaben
- In erster Linie theoretische und praktische Aufgaben zum Selbststudium, Einsatz von Softwarewerkzeugen (manchmal auch Programmierung)
- Bearbeitungszeit: abhängig von den Aufgaben, i.d.R. eine Woche
- Sie erhalten Zugriff auf das Labor Wissens- und Informationsmanagement.
  ☞ `ux-2e00.inf.fh-bonn-rhein-sieg.de`

Modulzuordnung und Studienleistung

BIS (3. Sem.):

- Modulgruppe: SPEZ-BI (Spezialisierung Business Intelligence)
- Prüfung im Prüfungszeitraum
- mündliche Prüfung über Vorlesung und Übung
- Credits: 6 (mit Bestehen der Prüfung)

Inhalt

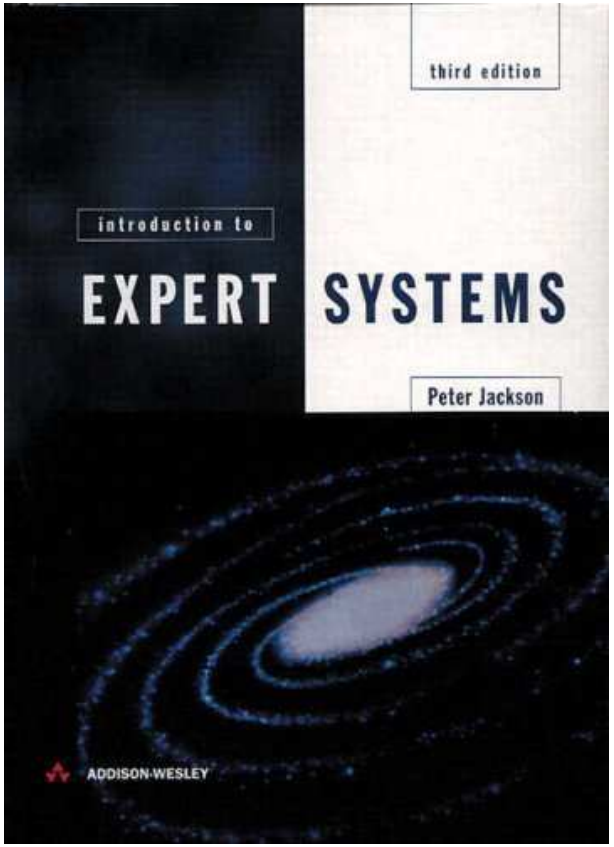
1. Business Intelligence: Einführung und Grundbegriffe
2. Problemlösung mittels Suche
3. Logikbasierte Wissensrepräsentation und Inferenz
4. Regelbasierte Systeme
5. Geschäftsregeln (Business Rules)
6. Betriebswirtschaftliche Fallbeispiele
7. Entscheidungen bei Unsicherheit

Lernziele/Kompetenzen:

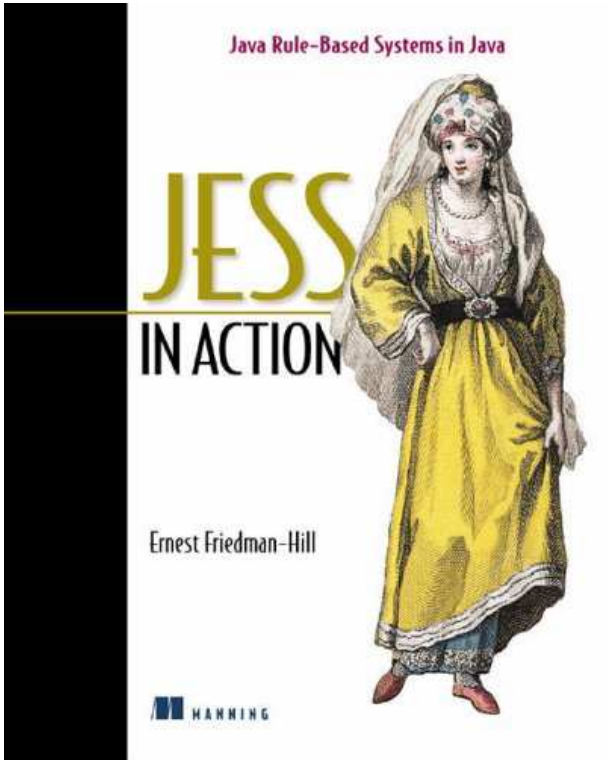
- Allgemein: Grundlegende Methoden für Decision Support und Expertensysteme kennen, verstehen und anwenden können;
- Lösungs methoden für schwierige Planungs- und Optimierungsprobleme kennen und anwenden können;
- Wissen mit Hilfe von Logik und Regeln repräsentieren können;
- Inferenzmethoden für Experten- und Regelsysteme beherrschen;
- Grundlegende Funktionen von Software-Werkzeugen für die Realisierung von Expertensystemen einsetzen können;
- Wichtige Konzepte für den Umgang mit unsicherem Wissen kennen;

Literatur

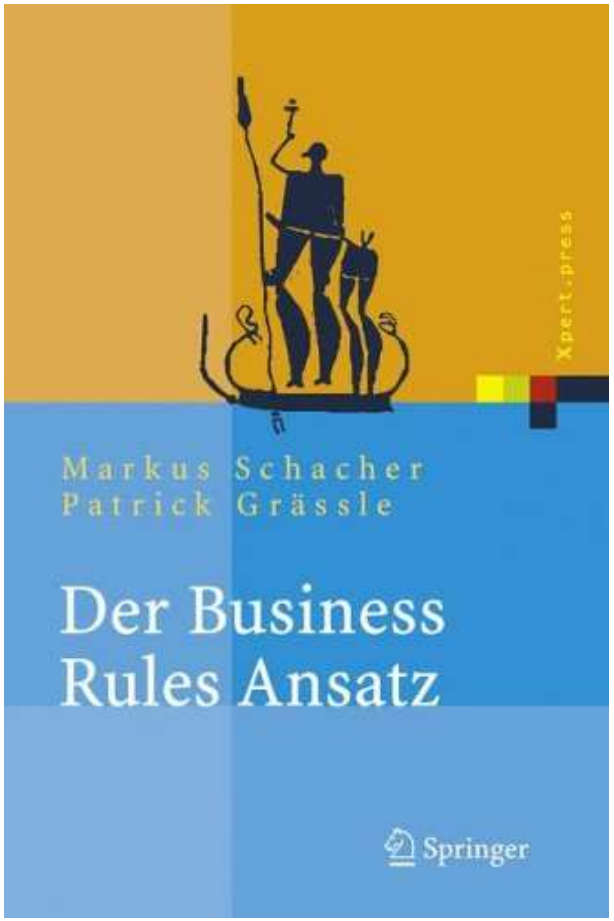
D. Karagiannis, R. Telesko
Wissensmanagement
Oldenbourg
2001



P. Jackson
Introduction to Expert Systems
Addison Wesley
1999



E. Friedman-Hill
Jess in Action: Rule-Based Systems in Java
Manning
2003



M. Schacher, P. Grässle
Agile Unternehmen durch Business Rules
Springer
2006

1. Einführung und Grundbegriffe

Lernziele:

- Wichtige Grundbegriffe verstehen, einordnen und erläutern können;
- Grundlegende Merkmale von Decision Support Systemen kennen;
- Arten von Wissen kennen und gegeneinander abgrenzen können;
- Grundbegriffe der Wissensverarbeitung kennen;

Business Intelligence

Business Intelligence ist ein verhältnismäßig junger und uneinheitlich verwendeter Begriff.

Anandarajan et al. 2004:

> Data analysis, reporting and query tools can help business users wade through a sea of data to synthesize valuable information from it — today these tools collectively fall into a category called “Business Intelligence”.

- Allgemein umfasst der Begriff *Business Intelligence (BI)* Methoden, Prozesse und Werkzeuge, um Unternehmensdaten in handlungsgerichtetes Wissen zu transformieren.
- handlungsgerichtetes Wissen: insbesondere zur Entscheidungsfindung
- Beispiele für Gebiete des BI: Data Mining, Data Warehouses, OLAP, Expertensysteme
- Berührungspunkte zu: Datenbanken, Künstliche Intelligenz, Wissensmanagement, Entscheidungstheorie, Statistik, ...

Definitionsvielfalt

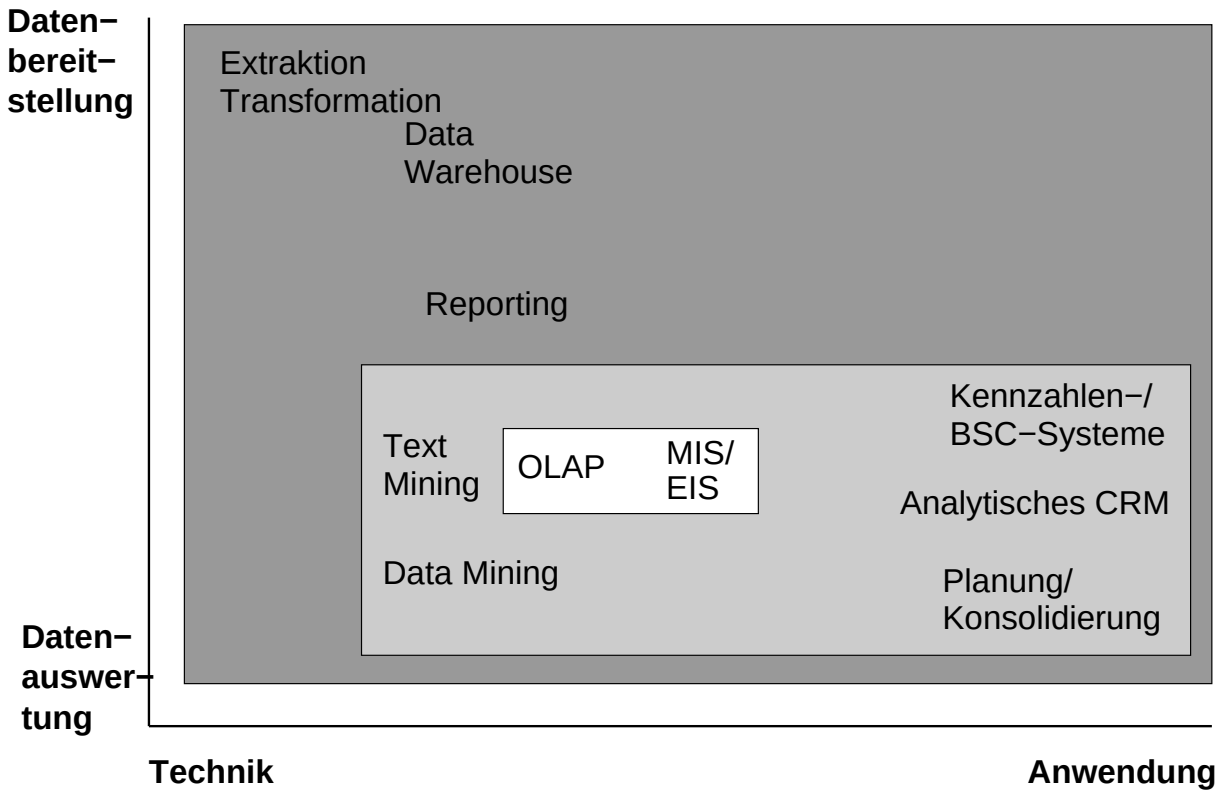
Mertens identifiziert sieben unterschiedliche Varianten der BI-Abgrenzung:

1. BI als Fortsetzung der Daten- und Informationsverarbeitung: Informationsverarbeitung für die Unternehmensleitung
2. BI als Filter in der Informationsflut: Informationslogistik
3. BI gleich Management Information Systems mit schnellen/flexiblen Auswertungen
4. BI als Frühwarnsystem
5. BI identisch zu Data Warehouse
6. BI als Informations- und Wissensspeicherung
7. BI als Prozess


1. Identify
your intelligence
needs
2. Gather
the information
3. Analyze
the information
4. Act
based on
your analysis

Gluchowski baut einen zweidimensionalen Ordnungsrahmen für BI auf:

horizontal: von der Technik- zur Anwendungsorientierung
vertikal: von der Datenauswertung zur Datenbereitstellung



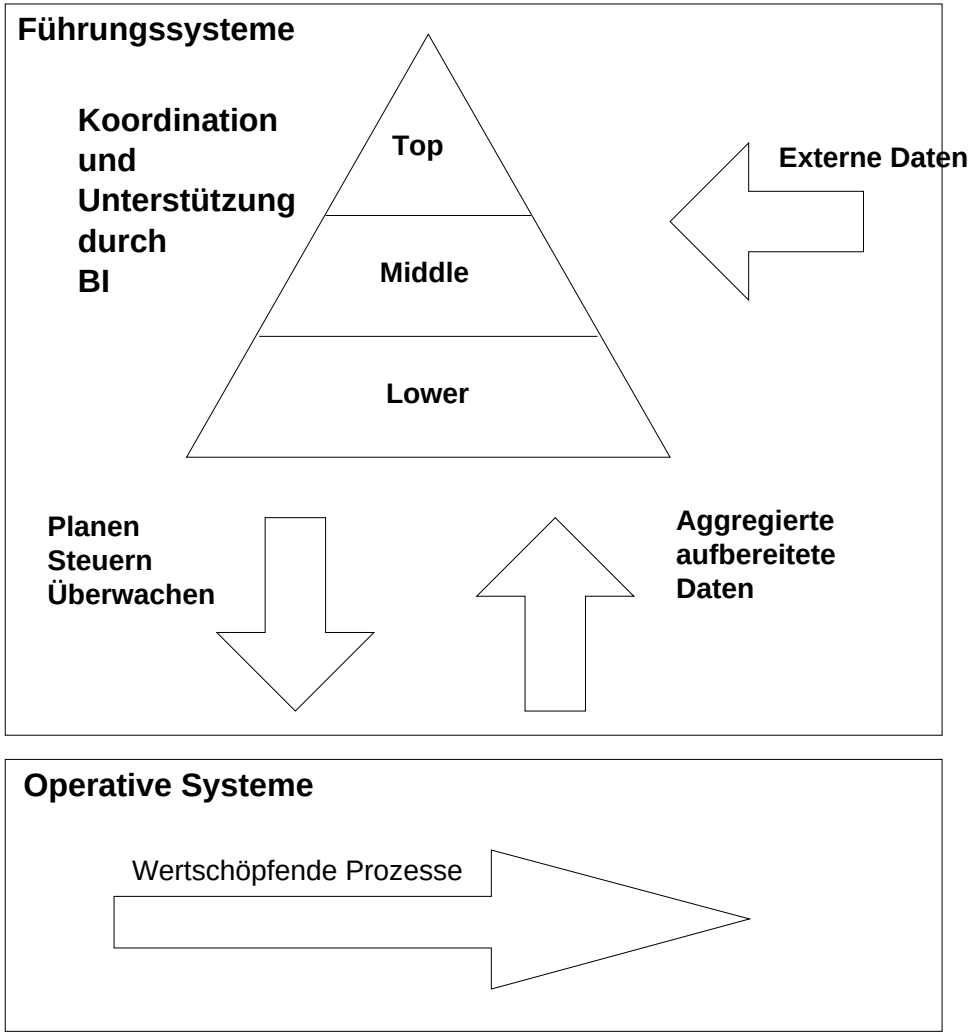
Hieraus leitet Gluchowski folgende Abgrenzung ab:

- **Business Intelligence im engeren Sinne**
  Ausschließlich Kernapplikationen, die die Entscheidungsfindung unterstützen
- **Analyseorientiertes Business Intelligence**
  Sämtliche Anwendungen, mit denen der Entscheider arbeitet
- **Business Intelligence im weiteren Sinne**
  Alle direkt und indirekt für die Entscheidungsunterstützung eingesetzten Anwendungen
  Auch Datenaufbereitung und Präsentationsfunktionalität

BI als integrierter Gesamtansatz

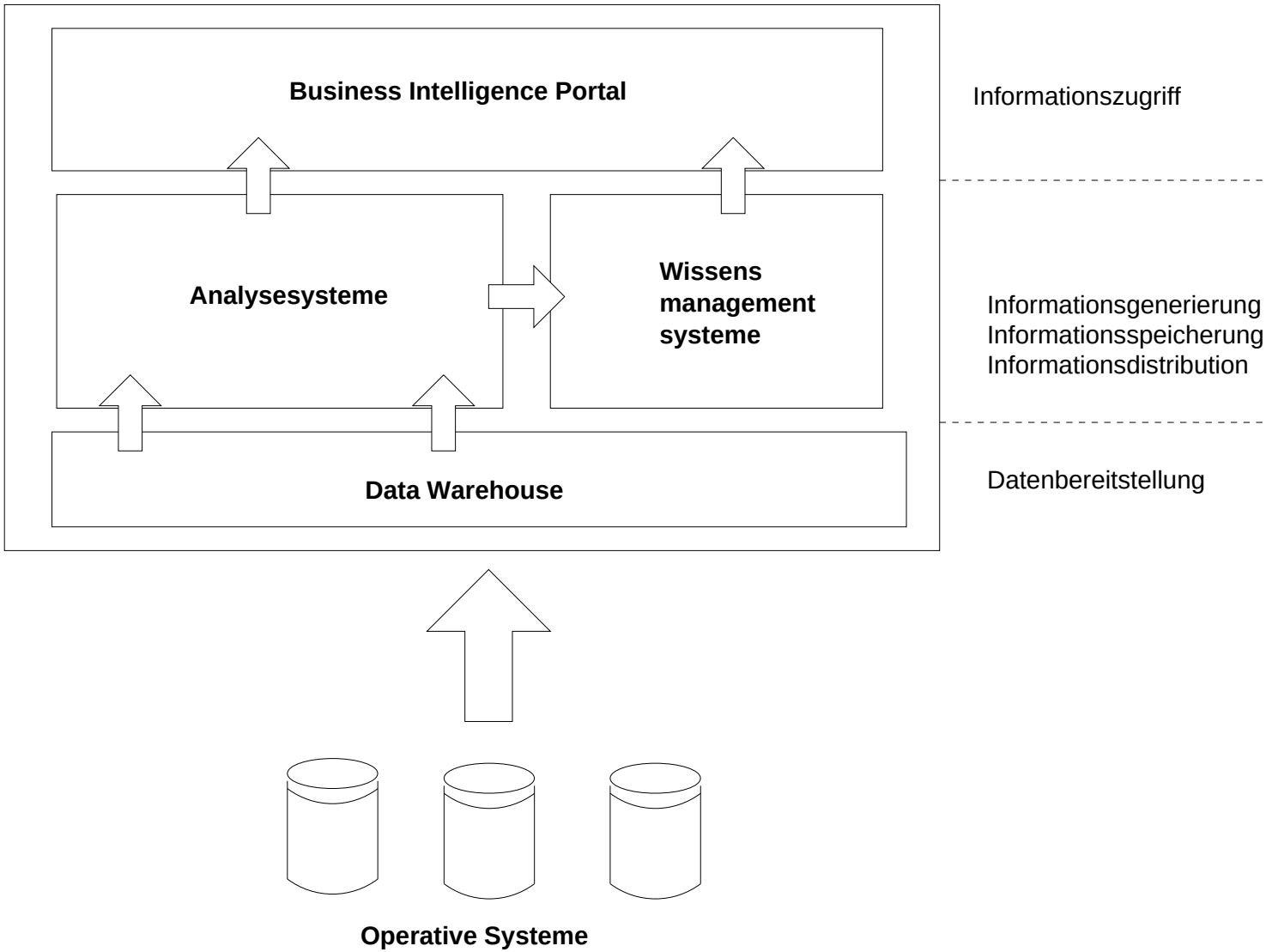
- Kemper, Mehanna, Unger sehen einen integrierten Gesamtansatz als definitorische Eigenschaft von Business Intelligence.
- Ausweitung der Datenbasen, massive Veränderung im Marktumfeld, höhere interne und externe Transparenz und Fundierung der Entscheidungen sind bei der Unternehmensführung zu berücksichtigen.
- Einzelsysteme zur Managementunterstützung können diesen Anforderungen nicht genügen.
- Daher ist ein integrierter Lösungsansatz erforderlich.

- Intelligence wird dabei als Information verstanden, die es zu generieren, speichern, recherchieren, analysieren, interpretieren und zu verteilen gilt.
- Erwerbbare BI-Werkzeuge werden ausschließlich als Entwicklungshilfen spezieller BI-Anwendungen gesehen.



Ordnungsrahmen für Business Intelligence

- BI als integrierter Gesamtansatz kann nur unternehmensspezifisch konkretisiert werden.
- BI-Ordnungsrahmen: generisches Konzept, das es auszufüllen gilt
- Data Warehouse: themenbezogene integrierte Datenhaltung
- Informationsgenerierung durch Analysesysteme, Speicherung und Verteilung als Teil des Wissensmanagements
- Portale sind eine zentrale Anlaufstelle für verschiedene Analysesysteme


Business Intelligence Portal
Analysesysteme
Wissens
management
systeme
Data Warehouse
Operative Systeme
Informationszugriff
Informationsgenerierung
Informationsspeicherung
Informationsdistribution
Datenbereitstellung

Entscheidungstheorie

- Eine *Entscheidung* ist eine rationale Wahl von Aktionen in einer gegebenen Umwelt.
- *Rational* ist ein Entscheidungsprozess, wenn er auf sinnvollen Kriterien beruht und diese berücksichtigt werden.
- Rational heißt nicht allwissend!
- Das Ergebnis einer Entscheidung ist eine *Aktion* oder *Strategie*.
- Deskriptive Entscheidungstheorien versuchen die empirischen Fragen der Wirtschafts- und Sozialwissenschaften zu beantworten.
- *Präskriptive (normative) Entscheidungstheorien* untersuchen, wie rationale Entscheidungen ausfallen müssen, damit Ziele unter Nebenbedingungen optimal erfüllt werden.
- Die präskriptiven Entscheidungstheorien der BWL, des Operations Research und der Informatik bilden die Grundlage entscheidungsunterstützender Systeme.

Decision Support Systeme

- *Entscheidungsunterstützende Systeme (Decision Support Systems, DSS)* sind rechnergestützte Informationssysteme, die Benutzer (hier typischerweise Entscheidungsträger) bei der Lösung komplexer Probleme unterstützen.
- Verwandte oder synonym gebrauchte Begriffe: Management Information System, Führungsinformationssystem, Execution Information System
- Entscheidungsprobleme sind *wohlstrukturiert*, wenn ihre wichtigsten Variablen und Beziehungen bekannt sind (Beispiel: mathematische Optimierungsprobleme)
- Fehlt Information über wichtige Variablen und Beziehungen ist das Entscheidungsproblem *schlecht strukturiert*.

Sicherheit, Risiko, Unsicherheit

DSS können *sichere*, *risikobehaftete* oder *unsichere* Entscheidungsprobleme unterstützen.

sicher: bekannte Umweltbedingungen, direkte Abhängigkeit zwischen Aktion und Nutzen

risikobehaftet: für die Bedingungen und Abhängigkeiten sind Wahrscheinlichkeitsverteilungen bekannt

unsicher: nur mögliche Zustände und Aktionen sind bekannt (Beispiel: Spiele)

Mögliche Lösungsansätze:

- Tabellenkalkulation
- Optimierungsverfahren
- Heuristiken
- regelbasiert
- Simulation

Verfahren der Entscheidungsunterstützung

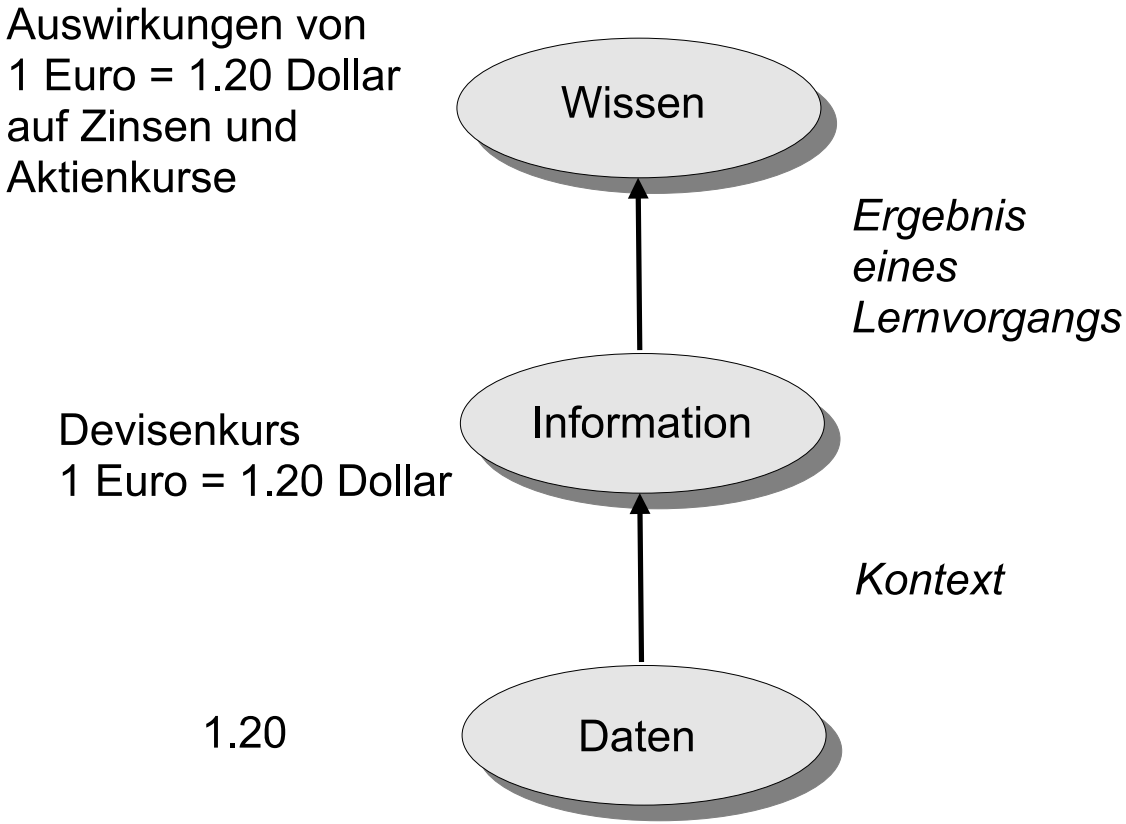
- Datenorientierte Verfahren zur Entscheidungsunterstützung leiten aus grossen Datenmengen Parameter ab, die sich auf die Analyse anderer Daten verallgemeinern lassen.

  Beispiel: OLAP, statistische Verfahren
- Modellorientierte Verfahren setzen die Gültigkeit eines Modells voraus und errechnen auf Basis dieses Modells eine optimale (oder gute) Entscheidung.

  Beispiel: Optimierungsverfahren
- Wissensbasierte Verfahren versuchen, das für eine gute Entscheidung notwendige Problemlösungswissen zu repräsentieren und anzuwenden.

  Hierzu gehören Expertensysteme, die sehr häufig regelbasiert arbeiten.

Wissen und Wissensverarbeitung



Wissen: Versuche einer Definition

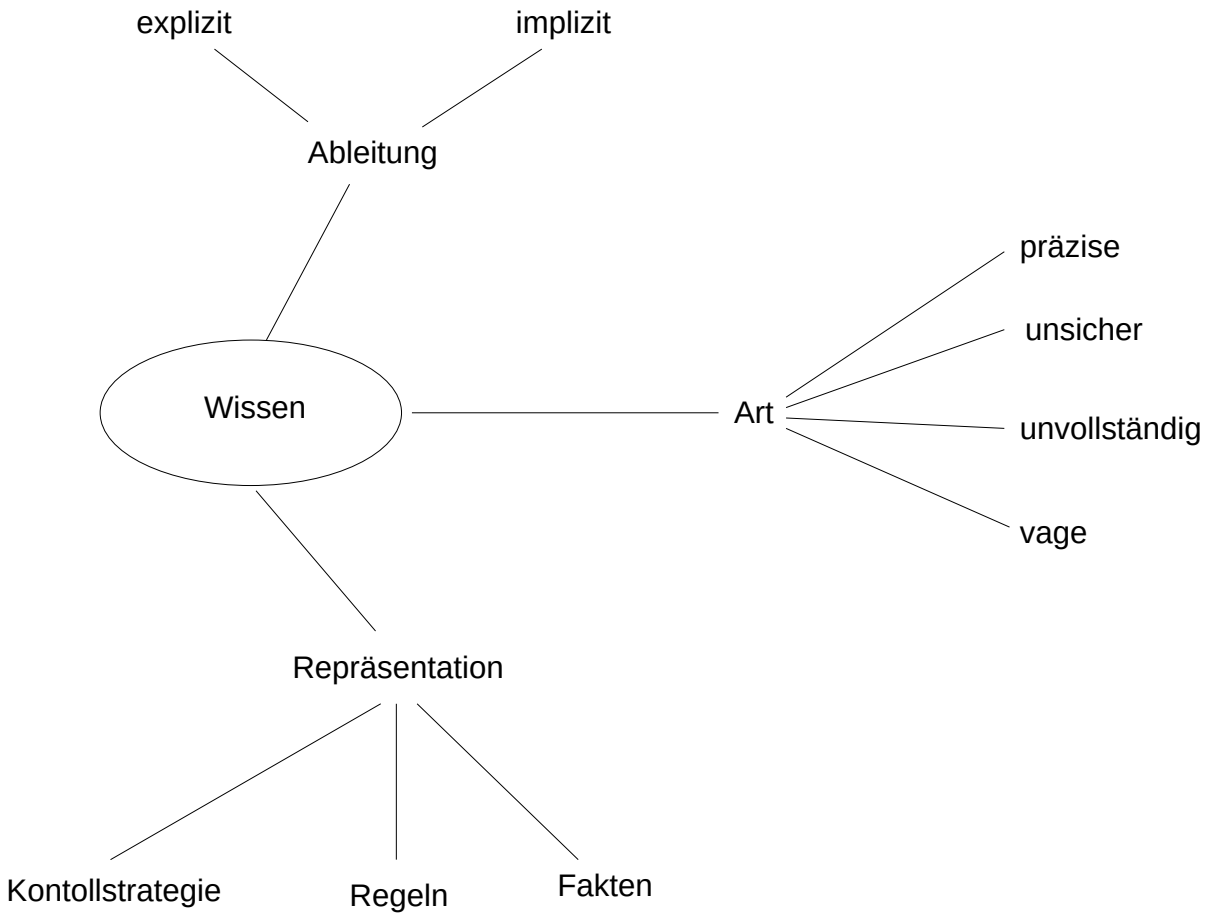
- Knowledge is organized information applicable to problem solving. (Woolf)
- Knowledge is information that has been organized and analyzed to make it understandable and applicable to problem solving or decision making. (Turban)

Wissen, Kennen, Können

Umgangssprachlich bezeichnet man das Ergebnis eines Lernvorgangs als

- *wissen*, wenn es sich um sprachlich-begriffliche Fähigkeiten handelt,
- *kennen*, wenn es sich um sinnliche Wahrnehmung handelt,
- *können*, wenn es sich um motorische Fähigkeiten handelt.

Arten von Wissen



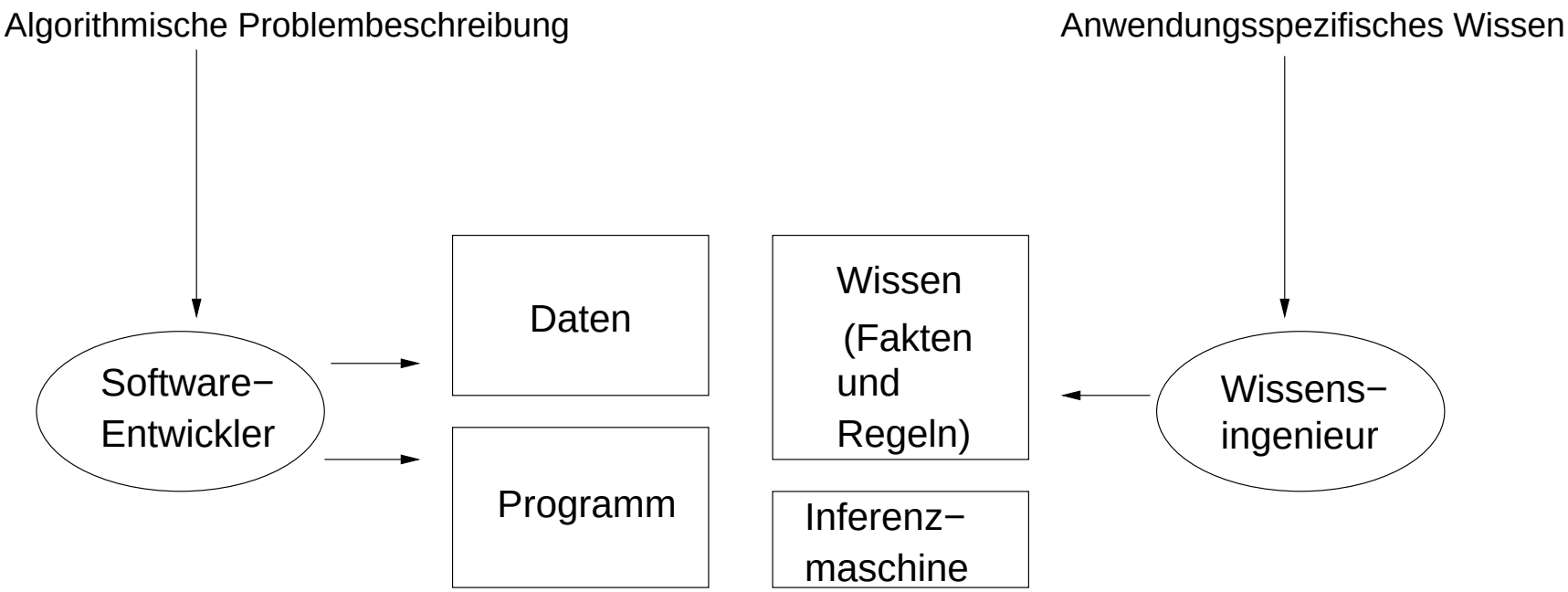
Wissensebenen

- kognitive Ebene (z.B. Erfahrung von Experten, Arbeitsanweisungen)
- Repräsentationsebene (z.B. Aussagenlogik, Prädikatenlogik)
- Implementierungsebene (z.B. Prolog-Statements)

☞ Bei der Wissensverarbeitung stehen die Repräsentationsebene und die Implementierungsebene im Vordergrund (Schließen der KI-Lücke).

☞ Beim Wissensmanagement stehen die kognitive Ebene und die Repräsentationsebene im Vordergrund.

Daten- vs. Wissensverarbeitung



Inferenz

- Nehmen wir an, es gibt eine Menge von Regeln, wie sich ein Autofahrer im Straßenverkehr zu verhalten hat.
- Die Regeln sind beispielsweise in “wenn...dann”-Form repräsentiert.
- Weiterhin gebe es Fakten, die Tatsachen widerspiegeln (Geschwindigkeit, Geschwindigkeitsbegrenzung, Ampel, etc.).
- Regeln und Fakten bilden die Wissensbasis.
- In solch einer Wissensbasis gibt es keine Kontrollstrukturen wie in einem herkömmlichen Programm, die festlegen, in welcher Reihenfolge die Regeln anzuwenden sind.
- Stattdessen muß ein Mechanismus vorhanden sein, der bestimmt, welche Regeln wie anzuwenden sind.
- Dieser Mechanismus heißt *Inferenzmechanismus*.
- *Inferenz* ist ein (Denk-)Prozeß, in dem aus vorhandenem Wissen (bzw. Annahmen oder Vermutungen) neues Wissen (Annahmen, Vermutungen) gewonnen werden.

- Neues Wissen heißt hier, daß nach Inferenz etwas verfügbar ist, was vorher nicht unmittelbar verfügbar war.
- Wissensbasis:
  - Wenn es regnet, dann ist die Straße nass. (Regel)
  - Es regnet. (Faktum)

  Inferenz (mit Modus Ponens): Die Straße ist nass. (abgeleitetes Faktum)

Logik und Inferenz

Gegenstand der Logik:

- *Repräsentation* von Wissen durch Formeln einer adäquaten Logiksprache
  - Syntax der Logiksprache
  - Bedeutung (Interpretation) von Formeln der Logiksprache
- *Herleitung (Inferenz)* von neuem Wissen auf Basis der Kalküls.
  - Definition des Folgerungsbegriffs
  - Übertragung der semantischen Folgerung auf äquivalente syntaktische Umformungen

Anwendungsgebiete der Logik in der Wissensverarbeitung:

- Inferenz in Expertensystemen
- Logikprogrammierung, deduktive Datenbanken
- automatisches Beweisen
- Programmverifikation

Zielrichtungen der Inferenz

- Prognosen, logische Ableitungen erstellen
  Es sind Fakten $\mathbb{F}$ und Regeln $\mathbb{R}$ gegeben. Was kann daraus gefolgert werden?

  Beispiel: Wenn es regnet, dann ist die Straße naß. Was kann aus der Tatsache, daß es regnet, gefolgert werden?
- Erklärungen finden
  Wie läßt sich ein Fakt $\mathbb{F}$ mit Hilfe der Regeln $\mathbb{R}$ erklären?
  Beispiel: Die Straße ist naß. Wie kann das sein?
- Hypothesen prüfen
  Können aus den Fakten $\mathbb{F}$ und den Regeln $\mathbb{R}$ die Hypothesen $\mathbb{H}$ hergeleitet werden?
  Beipiel: Wenn es regnet, dann ist die Straße naß. Es regnet. Ist die Straße dann naß?

Arten der Inferenz

- Deduktion

  Zum Starten eines Autos ist eine aufgeladene Batterie notwendig. Bei unserem Auto ist die Batterie leer. Wir schließen, daß wir unser Auto nicht starten können.

- Induktion

  Wir haben wiederholt beobachtet, daß ein Auto nicht startet und die Batterie leer ist. Wir haben noch nie beobachtet, daß ein Auto mit leerer Batterie gestartet werden konnte. Wir schließen daraus, daß ein Auto, das eine leere Batterie hat, nicht gestartet werden kann.

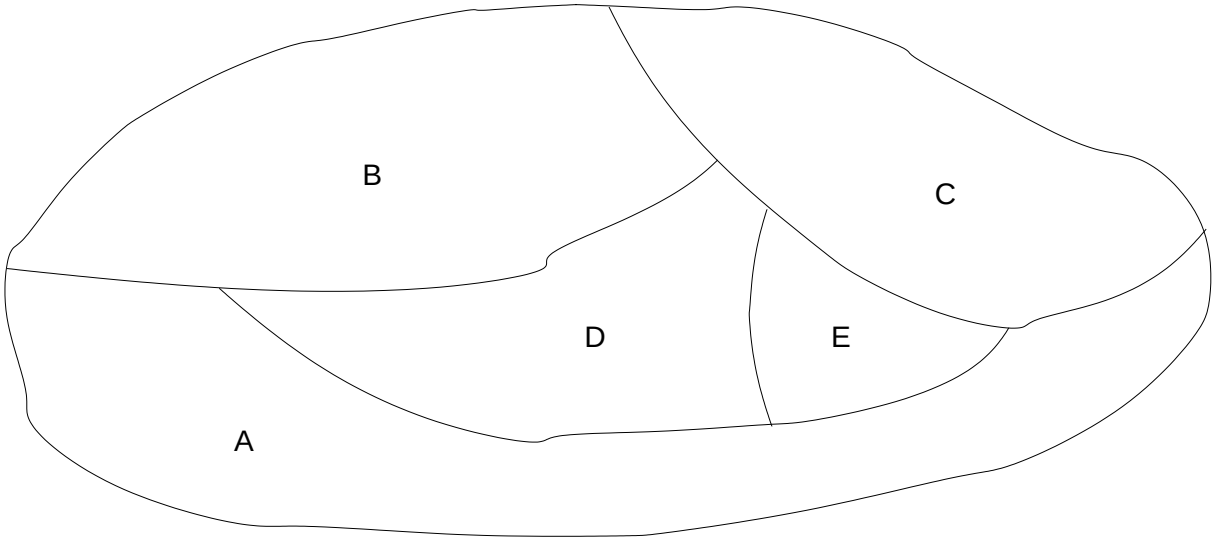
- Abduktion

  Zum Starten eines Autos ist eine aufgeladene Batterie notwendig. Unser Auto läßt sich nicht starten. Wir schließen, daß die Batterie leer ist.

2. Suchverfahren

- Viele Probleme der Entscheidungsunterstützung lassen sich auf ein Suchproblem zurückführen.
- Die Eigenschaften und Lösungsverfahren von Suchproblemen sind daher von grundlegender Bedeutung für dieses Gebiet.
- Suchverfahren sind ein klassisches Thema innerhalb der Wissensverarbeitung.

Färbeproblem



Beispiel 2.1. Die angegebene Landkarte mit den Ländern A, B, C, D und E ist so mit den Farben rot, blau, gelb und orange zu färben, daß keine zwei benachbarten Länder die gleiche Farbe haben.

Färbeproblem (2)

- Ein naives *generate-and-test* Verfahren würde 4^5 mögliche Farbkonstellationen prüfen.
- Allgemein sind $\mathfrak{m}^{\mathfrak{n}}$ Farbkonstellationen zu prüfen, mit $\mathfrak{m} :=$ Anzahl der Farben und $\mathfrak{n} :=$ Anzahl der Länder.

☞ Ineffizient!

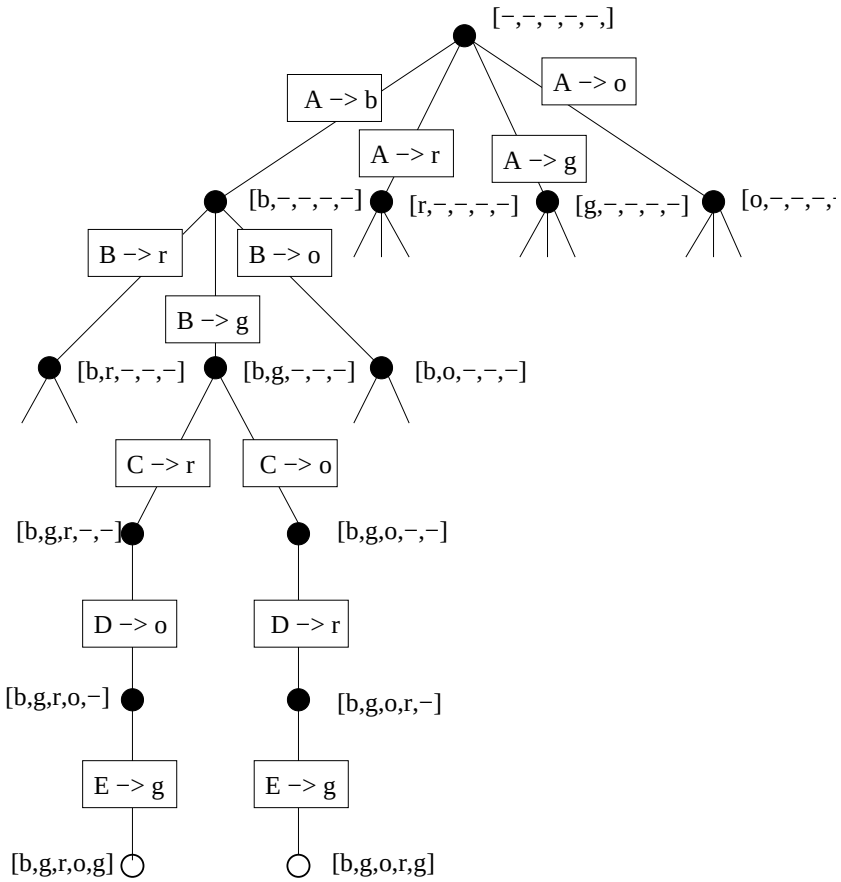
Färbeproblem (3)

- Es scheint sinnvoller zu sein, die Länder der Reihe nach zu färben.
- So kann man Zwischenzustände bei der Problemlösung durch Teilfärbungen beschreiben, etwa

$$(\mathrm{A} \leftarrow \mathrm{rot}, \mathrm{B} \leftarrow \mathrm{blau}, \mathrm{C} \leftarrow \mathrm{gelb})$$

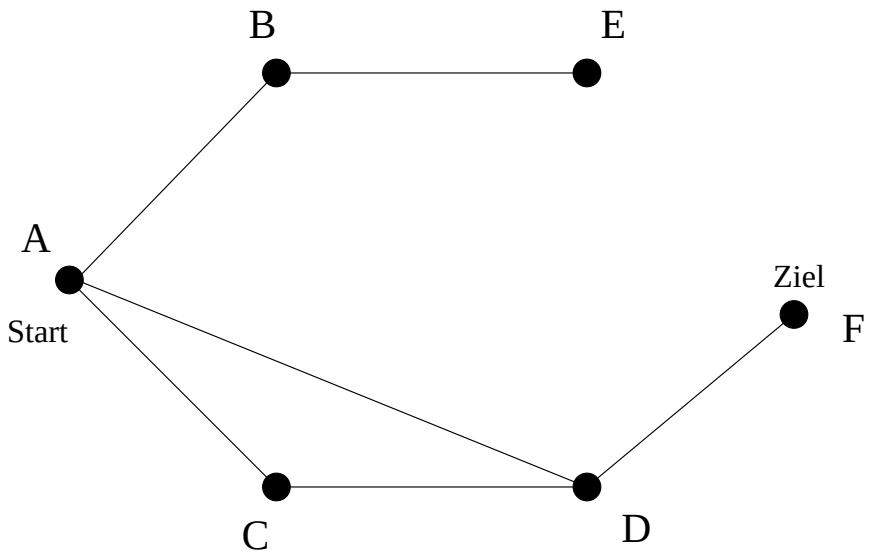
- Nach der Zuordnung $(\mathrm{A} \leftarrow \mathrm{blau}, \mathrm{B} \leftarrow \mathrm{blau})$ kann man direkt abbrechen.
- Die Problemlösung startet mit der leeren Färbung $()$.
- Ziel ist es, eine komplette zulässige Färbung zu erreichen.
- Die Schritte im Laufe der Problemlösung lassen sich durch Zustandsübergangsoperatoren beschreiben.

Suchbaum



- Die Lösung des Färbeproblems läßt sich als Suchbaum darstellen.
- Die Knoten des Suchbaums entsprechen den Zuständen (zulässige Teilfärbungen).
- Die Kanten entsprechen den Operatoren.

Routenproblem



Beispiel 2.2. Gegeben ist eine Karte mit Städten und Straßen, die die Städte miteinander verbinden.

Gesucht ist eine Route von einem Startort zu einem Zielort.

Suchbaum: Tafel ✎.

Zustandsraum

Ein *Problem* wird repräsentiert durch Wissen, das ein DSS nutzen kann, um zu entscheiden, welche Aktionen ausführt werden sollen.

Für Suchprobleme läßt sich das Wissen repräsentieren durch:

- Ein *Zustand* stellt das Wissen zu einem bestimmten Zeitpunkt der Lösungsfindung dar.
- Der *Zustandsraum* ist die Menge aller Zustände.
- *Zustandsübergangsoperatoren* beschreiben, wie ausgehend von einem Zustand andere Zustände des Zustandsraums erreicht werden können.
- Der *Startzustand* ist der Zustand, der zu Beginn der Lösungsfindung vorliegt. Er läßt sich explizit angeben.
- Die Menge der *Zielzustände* charakterisiert die Lösungen des Problems. Zielzustände lassen sich in der Regel nur implizit angeben, z.B. über ein Testprädikat.

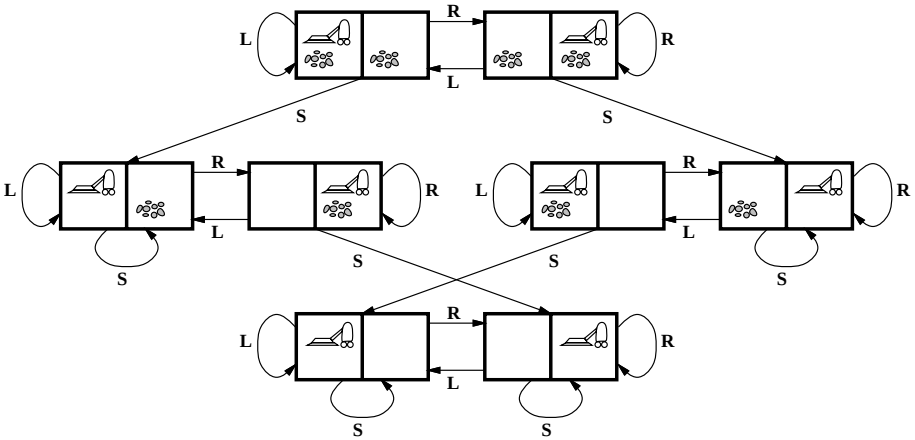
- Um verschiedene Lösungen vergleichen zu können, können Folgen von Aktionen *Kosten* zugewiesen werden.

  Typischerweise definiert man Kosten abhängig von einem Zustand und einer Aktion. Die Gesamtkosten für eine Aktionenfolge ergeben sich aus den Einzelkosten.

Beispiel 2.3.

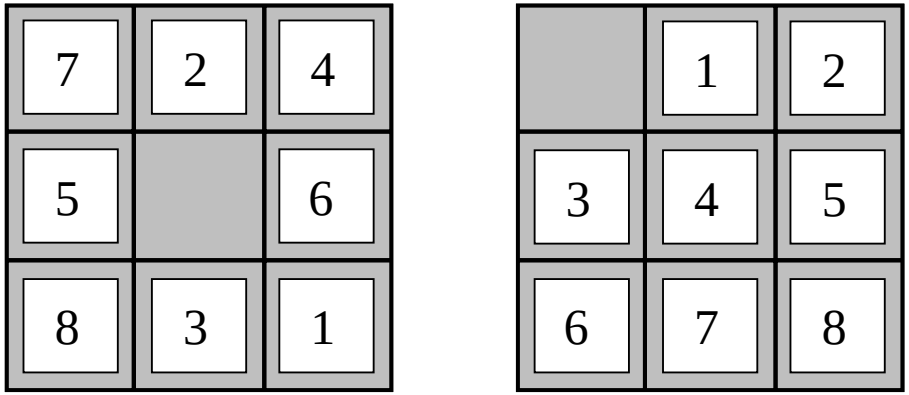
Staubsaugerwelt:

- Operationen: rechts, links, saugen
- Ziel: alle Räume müssen sauber sein



8-Puzzle:

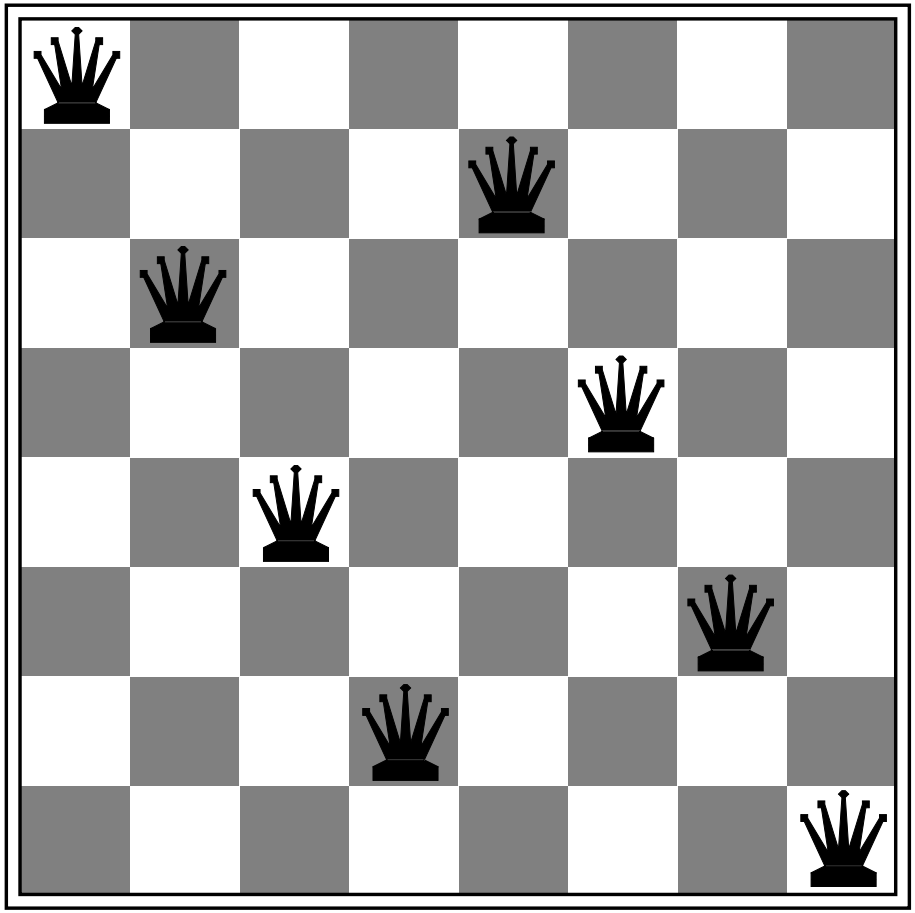
- Operationen: rechts, links, oben, unten
- Ziel: Endkonfiguration



Start State Goal State

8-Damen-Problem:

- Zustand: Plazierung zwischen 0 und 8 Damen
- Operationen: eine Dame plazieren
- Ziel: alle Damen so plaziert, daß sie sich nicht bedrohen



Beispiel 2.4. Suchprobleme in der realen Welt:

- Routenberechnung, Navigation
- Travelling Salesman Problem, Vehicle Routing: Tourenplanung, Fahrzeugeinsatzplanung, Gebietsplanung
- VLSI Layout: Cell Layout und Channel Routing
- Robot Navigation: virtuell und physisch, wie findet der Roboter/Avatar sein Ziel?
- Assembly Sequencing: Finde eine Reihenfolge zum Zusammenbau komplexer Werkstücke aus Einzelteilen

Zustandsraum und Suchbaum

- Die Knoten eines Suchbaums stellen die Zustände dar.
- Die Kanten entsprechen den Zustandsübergangsoperatoren.
- Die Wurzel entspricht dem Startzustand.
- Die Zielknoten sind die Zielzustände.
- Die Berechnung der Nachfolger eines Knotens s wird als *Expansion* des Knotens s bezeichnet.

Zustandsraum und Suchbaum (2)

- Der Zustandsraum beschreibt nur, wie man prinzipiell zu einer Lösung gelangen kann,
- aber nicht, wie man effizient zu dieser kommt.
- Ganz wesentlich für eine effiziente Problemsösung sind:
  - ☞ das Verfahren, das festlegt, in welcher Reihenfolge die Zustände untersucht bzw. expandiert werden sowie
  - ☞ die Bewertung der einzelnen Zustände.

Uninformierte Suchverfahren

- Suchverfahren, die über die Beschreibung des Zustandsraums hinaus keine Zusatzinformation benutzen, heißen *uninformierte Suchverfahren*.
- Insbesondere findet keine Bewertung der einzelnen Zustände statt.
- Dementsprechend unterscheiden sich die Verfahren im wesentlichen darin, in welcher Reihenfolge die Zustände expandiert werden.
- Die wichtigsten Vertreter der uninformierten Suchverfahren sind die *Breitensuche* und die *Tiefensuche*.

Uninformierte Suchverfahren (2)

- Ausgehend von der Wurzel des Suchbaums (Startzustand) werden die Knoten sukzessive expandiert.
- Später wird man von den Nachfolgern des expandierten Knotens weiterarbeiten, solange bis man einen Zielknoten gefunden hat.
- Die Liste der Knoten, die gerade in Bearbeitung sind, heißt *Agenda (open list)*.
- Knoten der Agenda sind generiert, aber noch nicht expandiert.
- Expandierte Knoten werden auch als *closed* bezeichnet.

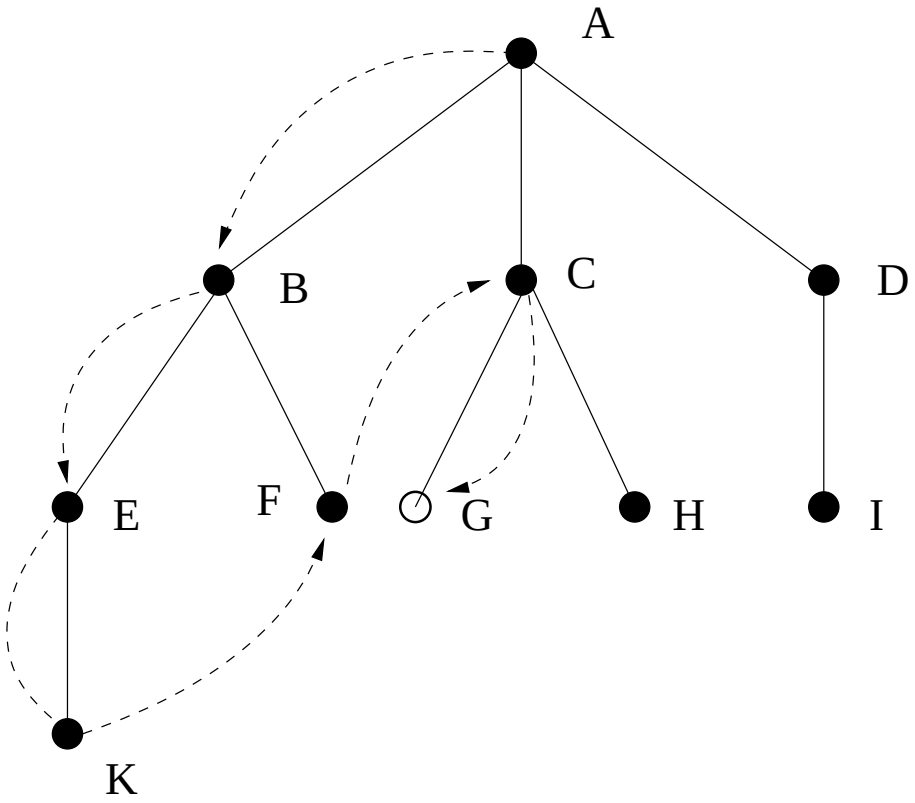
Uninformierte Suchverfahren (3)

- Breitensuche und Tiefensuche laufen nach dem gleichen Schema ab.
- Zu Beginn der Suche besteht die Agenda aus dem Startzustand.
- In einer beliebigen Iteration wird der erste Knoten s_{akt} aus der Agenda genommen.
- Wenn s_{akt} ein Zielzustand ist, hat man eine Lösung gefunden.
- Ist s_{akt} kein Zielzustand, so wird s_{akt} expandiert, d.h. alle Nachfolger von s_{akt} werden in die Agenda eingefügt.
- Breiten- und Tiefensuche unterscheiden sich darin, wo die Nachfolger in die Agenda eingefügt werden.

Tiefensuche

☞ Bei der Tiefensuche werden die Nachfolger eines expandierten Knotens s_{akt} an den Anfang der Agenda eingefügt.

- Die Agenda entspricht einem Kellerspeicher (Stack).
- Liefert ein Knoten, der kein Zielknoten ist, keine neuen Knoten, so wird die Suche fortgesetzt an dem nächstgelegenen Knoten, für den noch nicht alle Nachfolger expandiert wurden.
- Dies entspricht einem Backtracking.

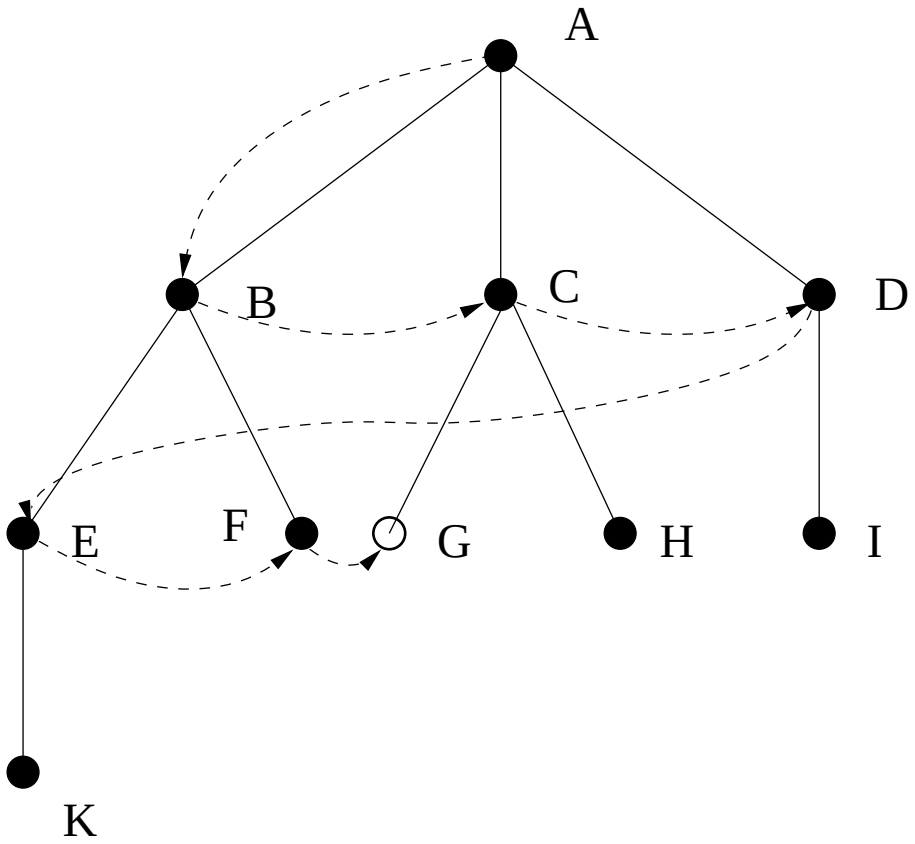


Schritt	Agenda	s_{akt}
1	(A)	A
2	(B,C,D)	B
3	(E,F,C,D)	E
4	(K,F,C,D)	K
5	(F,C,D)	F
6	(C,D)	C
7	(G,H,D)	G

Breitensuche

☞ Bei der Breitensuche werden die Nachfolger eines expandierten Knotens s_{akt} an das Ende der Agenda eingefügt.

- Die Agenda entspricht einer Warteschlange (Queue).



Schritt	Agenda	s_{akt}
1	(A)	A
2	(B,C,D)	B
3	(C,D,E,F)	C
4	(D,E,F,G,H)	D
5	(E,F,G,H,I)	E
6	(F,G,H,I,K)	F
7	(G,H,I,K)	G

Algorithmen

Algorithmus 2.1. [Tiefensuche]

$Agenda := (Startknoten)$;
`while` $Agenda \neq ()$ `do`
    $s_{akt} := first(Agenda)$;
    Entferne s_{akt} aus der Agenda;
    `if` s_{akt} ist Zielknoten `then` s_{akt} ist Lösung; STOP;
    $Agenda := Nachfolger(s_{akt}) + Agenda$;
`end`
Problem hat keine Lösung; STOP;

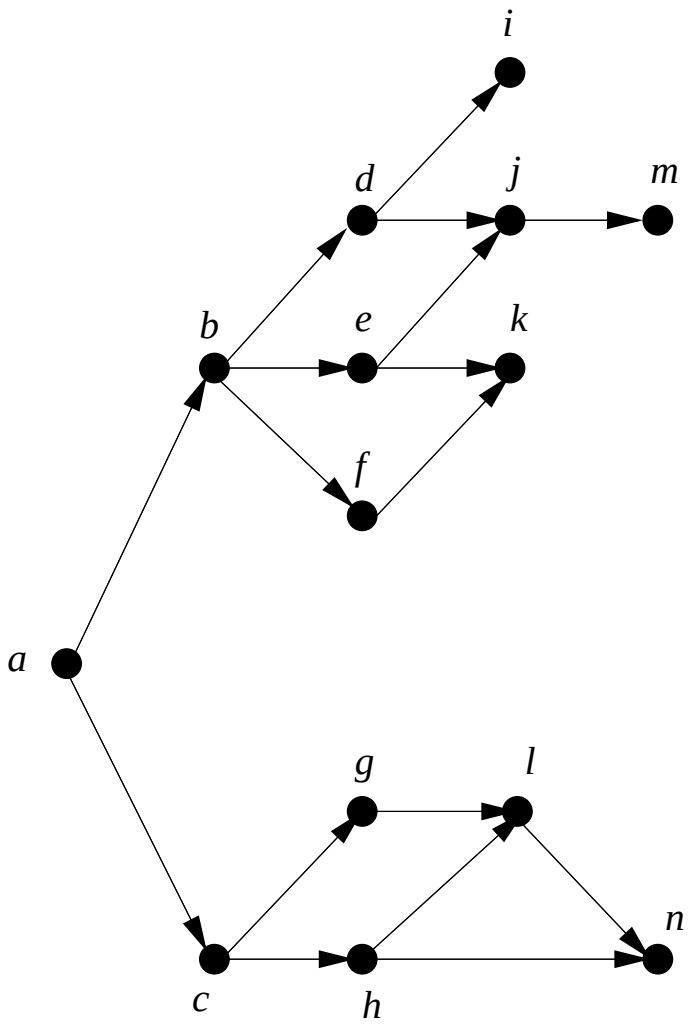
Algorithmen (2)

Algorithmus 2.2. [Breitensuche]

$Agenda := (Startknoten)$;
`while` $Agenda \neq ()$ `do`
    $s_{akt} := first(Agenda)$;
    Entferne s_{akt} aus der Agenda;
    `if` s_{akt} ist Zielknoten `then` s_{akt} ist Lösung; STOP;
    $Agenda := Agenda + Nachfolger(s_{akt})$;
`end`
Problem hat keine Lösung; STOP;

Beispiel 2.5. Suche einen Weg von a nach e mit Tiefensuche bzw. Breitensuche.

Tafel ✎.



Beispiel 2.6. Ein Weinhändler hat drei Krüge, einen von 9 Liter, einen von 7 Liter und einen von 4 Liter Inhalt.

Auf den Krügen sind keine Litermarkierungen angebracht.

Der 9-Liter-Krug ist gefüllt, die anderen sind leer.

Die Krüge sollen so umgefüllt werden, daß der 9-Liter-Krug sechs Liter und der 4-Liter-Krug drei Liter enthält.

Tafel ✎.

Eigenschaften von Suchverfahren

Definition 2.1. Ein Suchverfahren heißt *vollständig*, wenn für jeden Suchbaum jeder Knoten expandiert werden könnte, solange noch kein Zielknoten gefunden wurde.

- Ein vollständiges Suchverfahren ist fair in dem Sinne, daß jeder Knoten die Chance hat, expandiert zu werden.
- Ein vollständiges Sucherfahren findet auch bei unendlichen Suchbäumen stets eine Lösung, falls eine existiert.

☞ Breitensuche ist vollständig.

☞ Tiefensuche ist nur bei endlichen Suchbäumen vollständig.

Eigenschaften von Suchverfahren (2)

Definition 2.2. Für ein uninformiertes Suchverfahren heißt eine Lösung *optimal*, wenn sie unter allen Lösungen die geringste Tiefe im Suchbaum aufweist.

- ☞ Breitensuche findet eine optimale Lösung (falls existent).
- ☞ Tiefensuche findet i.a. keine optimale Lösung.

Eigenschaften von Suchverfahren (3)

Komplexitäten:

- Für Breiten- und Tiefensuche ist der ungünstigste Fall, daß die Lösung in der "äußersten rechten Ecke" des Suchbaums liegt.
- $\Longrightarrow$ Zeitkomplexität $\mathrm{O}(\mathrm{b}^{\mathrm{t}})$, mit $\mathrm{b} =$ Verzweigungsrate und $\mathrm{t} =$ Tiefe des Zielknotens.
- Bei der Tiefensuche enthält die Agenda die Knoten des aktuellen Suchpfades sowie deren Nachfolger $\Longrightarrow$ Platzkomplexität $\mathrm{O}(\mathrm{bt})$.
- Bei der Breitensuche kann die Agenda eine komplette Ebene des Suchbaums enthalten $\Longrightarrow$ Platzkomplexität $\mathrm{O}(\mathrm{b}^{\mathrm{t}})$.

Varianten von uninformierten Suchverfahren

Tiefensuche mit Begrenzung der Suchtiefe:

- Versucht die Nachteile der Tiefensuche auszuschalten.
- Eine *maximale Suchtiefe* wird vorgegeben.
- Zweige jenseits der maximalen Suchtiefe werden abgeschnitten.
- Hierdurch wird garantiert, daß eine Lösung gefunden wird, wenn eine innerhalb der Suchtiefe existiert.
- Nachteil: Wie groß soll man die maximale Suchtiefe wählen?

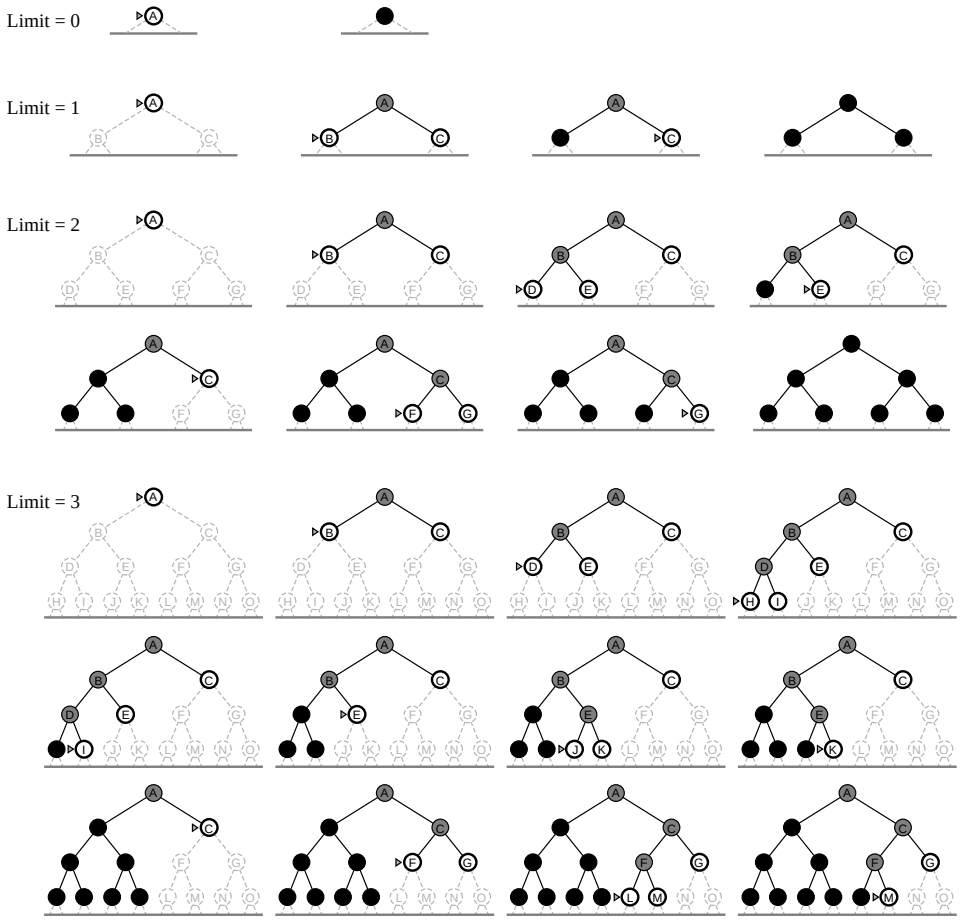
Iterative Deepening:

- Da man nicht weiß, wie groß man die maximale Suchtiefe wählen soll, probiert man dies einfach systematisch durch.
- D.h. man führt immer wieder Tiefensuche mit Begrenzung der Suchtiefe durch und erhöht die maximale Suchtiefe jeweils um 1.
- Bei einer maximalen Suchtiefe t und einem regulären Suchbaum mit Verzweigungsgrad b werden $O(b^t)$ Knoten erzeugt.

☞ Wegen

$$\sum_{d=0}^{t} b^d = \frac{b^{t+1} - 1}{b - 1} < \frac{b}{b - 1} b^t = O(b^t)$$

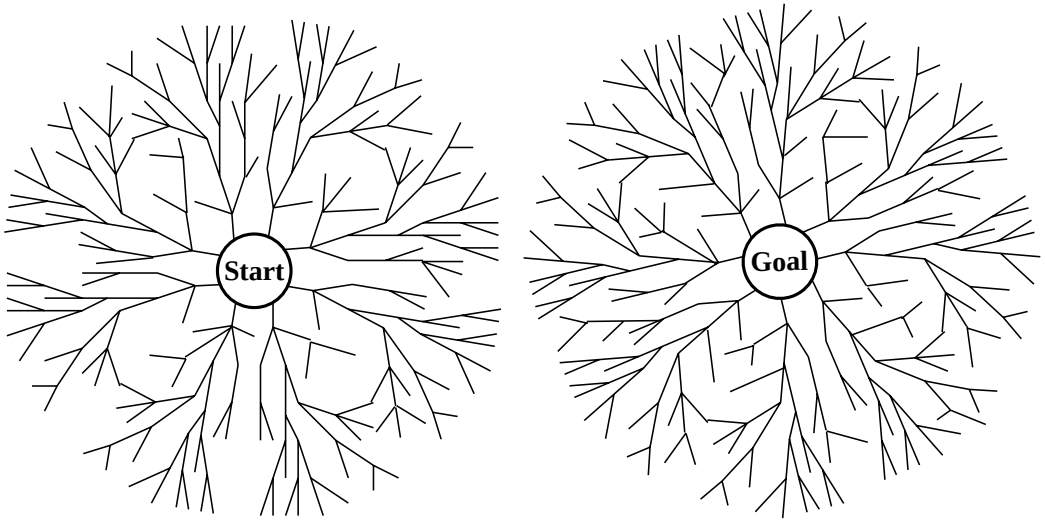
verliert man nur einen konstanten Faktor.



☞ Iterationen bei Iterative Deepening und einem binären Suchbaum

Bidirektionale Suche:

- Gleichzeitige Suche von Start zum Ziel und vom Ziel zum Start.
- Lösungen ergeben sich dort, wo sich die Suchpfade treffen.



- Durch Halbierung der Länge der Suchpfade ergibt sich ein Aufwand von $O(2b^{\frac{d}{2}}) = O(b^{\frac{d}{2}})$.
  ☞ drastische Reduzierung

- Probleme:
  - Zielzustände sind oft nicht explizit bekannt.
  - Zielzustände müssen nicht eindeutig sein.
  - Operatoren sind nicht unbedingt umkehrbar.

Informierte Suchverfahren

- Für größere Suchbäume sind Breiten- und Tiefensuche nicht effizient genug.
- Vielversprechender sind Ansätze, bei denen Problemwissen zur Steuerung des Suchprozesses eingesetzt wird.
- Dies können wir erreichen, indem wir die Zustände (Knoten) danach bewerten, wie erfolgversprechend sie sind.
- Wir schätzen beispielsweise für jeden Knoten, wie nahe er an einem Zielknoten liegt.
- Solch eine Bewertung heißt *heuristische Funktion*.

Heuristische Funktion

Definition 2.3. Eine Funktion, die jedem Zustand (Knoten) s eines Zustandsraums (Suchbaums) eine nichtnegative Zahl $h(s)$ zuordnet, heißt *heuristische Funktion*. Für einen Zielzustand s gilt dabei $h(s) = 0$.

Ein Suchverfahren, das eine heuristische Funktion zur Auswahl der zu expandierenden Zustände einsetzt, heißt *informiertes Suchverfahren* oder auch *heuristisches Suchverfahren*.

Beispiel 2.7. [Schiebepuzzle]

Startzustand

5	4	
6	1	8
7	3	2

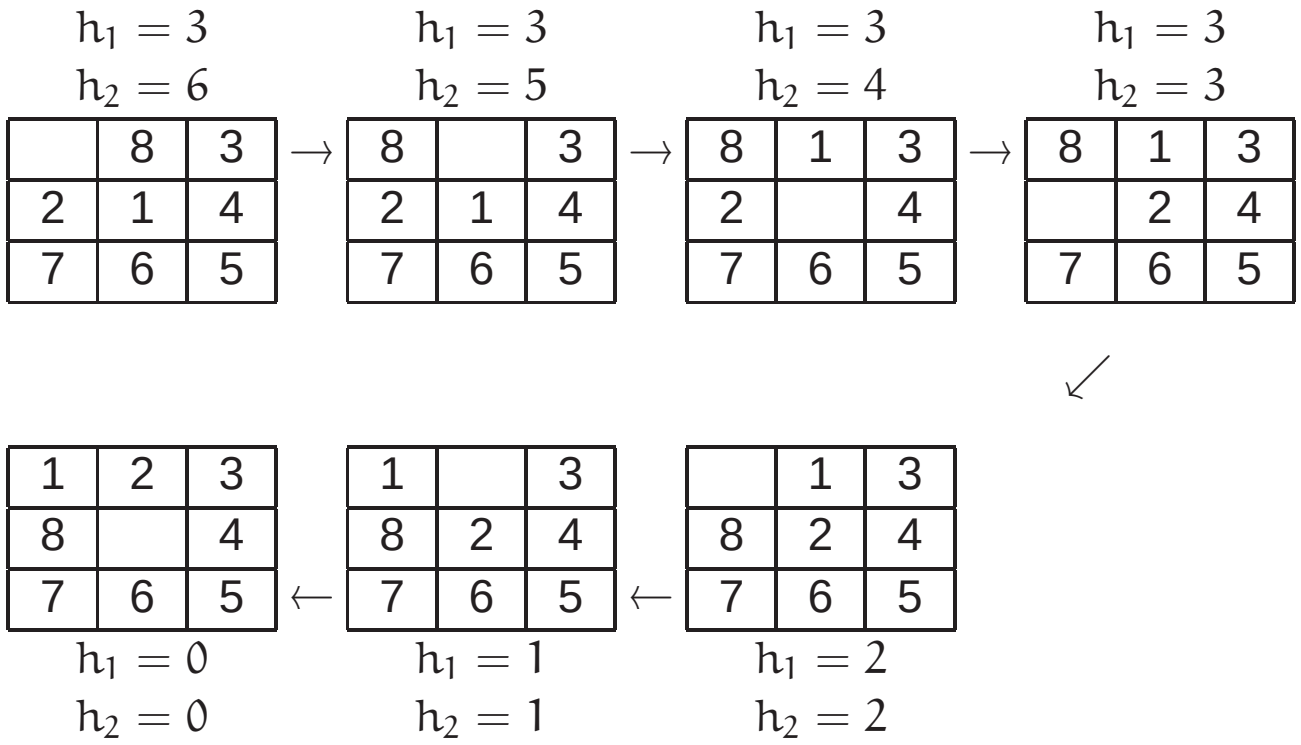
$\rightarrow$

1	2	3
8		4
7	6	5

Endzustand

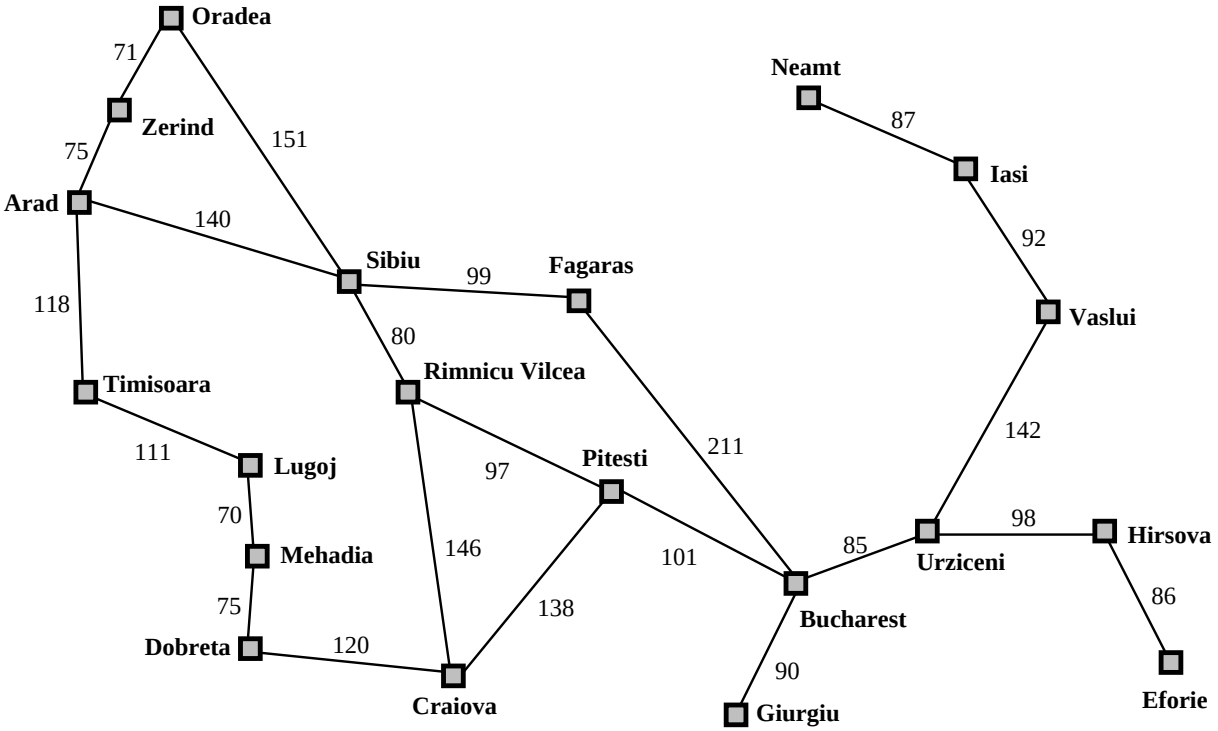
Mögliche heuristische Funktionen:

- $h_1(s) :=$ Anzahl der Plättchen, die nicht an der richtigen Stelle liegen.
  Hier: $h_1(s) = 7$.
- $h_2(s) :=$ Summe der Entfernungen aller Plättchen von der Zielposition.
  Hier: $h_2(s) = 2 + 3 + 3 + 2 + 4 + 2 + 0 + 2 = 18$.


$h_1 = 3$ $h_2 = 6$
$h_1 = 3$ $h_2 = 5$
$h_1 = 3$ $h_2 = 4$
$h_1 = 3$ $h_2 = 3$
$h_1 = 2$ $h_2 = 2$
$h_1 = 1$ $h_2 = 1$
$h_1 = 0$ $h_2 = 0$

- Die heuristische Funktion h_2 differenziert stärker als h_1, d.h.
- h_2 kann Zustände unterscheiden, die von h_1 gleich bewertet werden.
- Eine heuristische Funktion ist um so brauchbarer, je mehr Zustände sie unterschiedlich bewertet.
- Eine heuristische Funktion, die alle Zustände gleich bewertet, ist unbrauchbar.

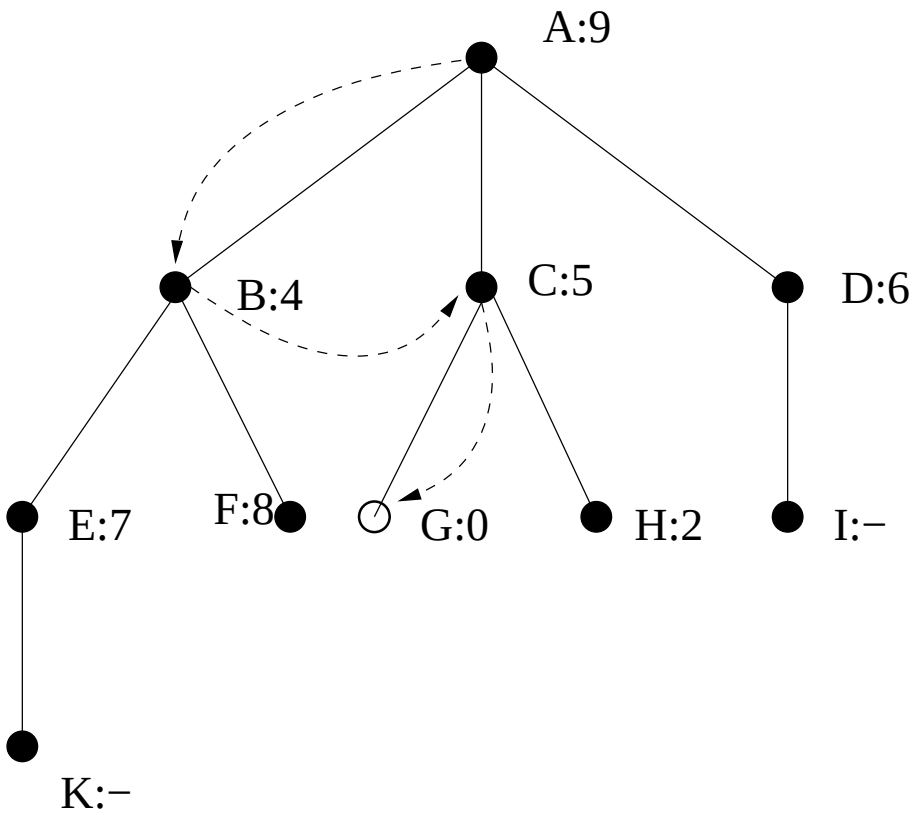
Beispiel 2.8. [Routenplanung] Zur Abschätzung der Straßenentfernung wird die Luftlinienentfernung benutzt.



Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Dobreta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Bestensuche

- Bei der *Bestensuche* erfolgt die Expansion eines Knotens auf Basis der heuristischen Funktion.
- Hierzu werden in der Agenda die Knoten zusammen mit ihrer Bewertung abgelegt.
- Es wird nun jeweils der Knoten der Agenda expandiert, der die geringste Bewertung aufweist.
- Die Agenda hat also die Form einer Prioritätswarteschlange (priority queue).
- Ansonsten ist die Bestensuche analog zur Tiefen- und Breitensuche.



Schritt	Agenda	s_{akt}
1	A:9	A
2	B:4, C:5, D:6	B
3	C:5, D:6, E:7, F:8	C
4	G:0, H:2, D:6, E:7, F:8	G

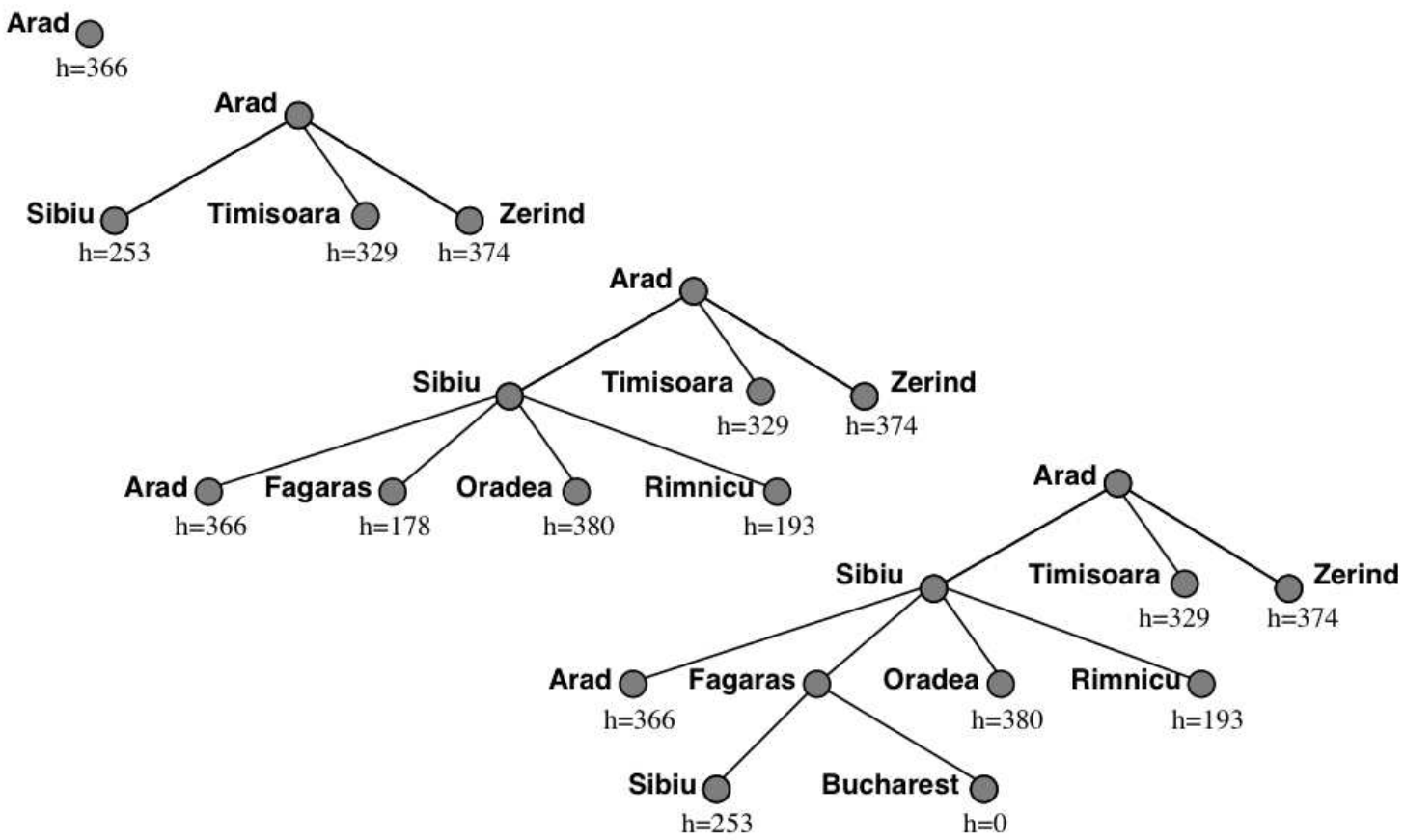
Algorithmus zur Bestensuche

Algorithmus 2.3. [Bestensuche]

$Agenda := (Startknoten)$;
`while` $Agenda \neq ()$ `do`
    $s_{akt} := first(Agenda)$;
    Entferne s_{akt} aus der Agenda;
    `if` s_{akt} ist Zielknoten `then` s_{akt} ist Lösung; STOP;
    $Agenda := einfuegen(Agenda, Nachfolger(s_{akt}))$;
`end`
Problem hat keine Lösung; STOP;

Beispiel 2.9. Suchbaum für Beispiel 2.7 mit Bestensuche. Tafel ✎

Beispiel 2.10. Bestensuche für die Suche einer Route von Arad nach Bucharest.


Arad
h=366
Arad
Sibiu
Timisoara
Zerind
h=253
h=329
h=374
Arad
Sibiu
Timisoara
Zerind
h=329
h=374
Arad
Fagaras
Oradea
Rimnicu
h=366
h=178
h=380
h=193
Arad
Sibiu
Timisoara
Zerind
h=329
h=374
Arad
Fagaras
Oradea
Rimnicu
h=366
h=380
h=193
Sibiu
Bucharest
h=253
h=0

Eigenschaften der Bestensuche

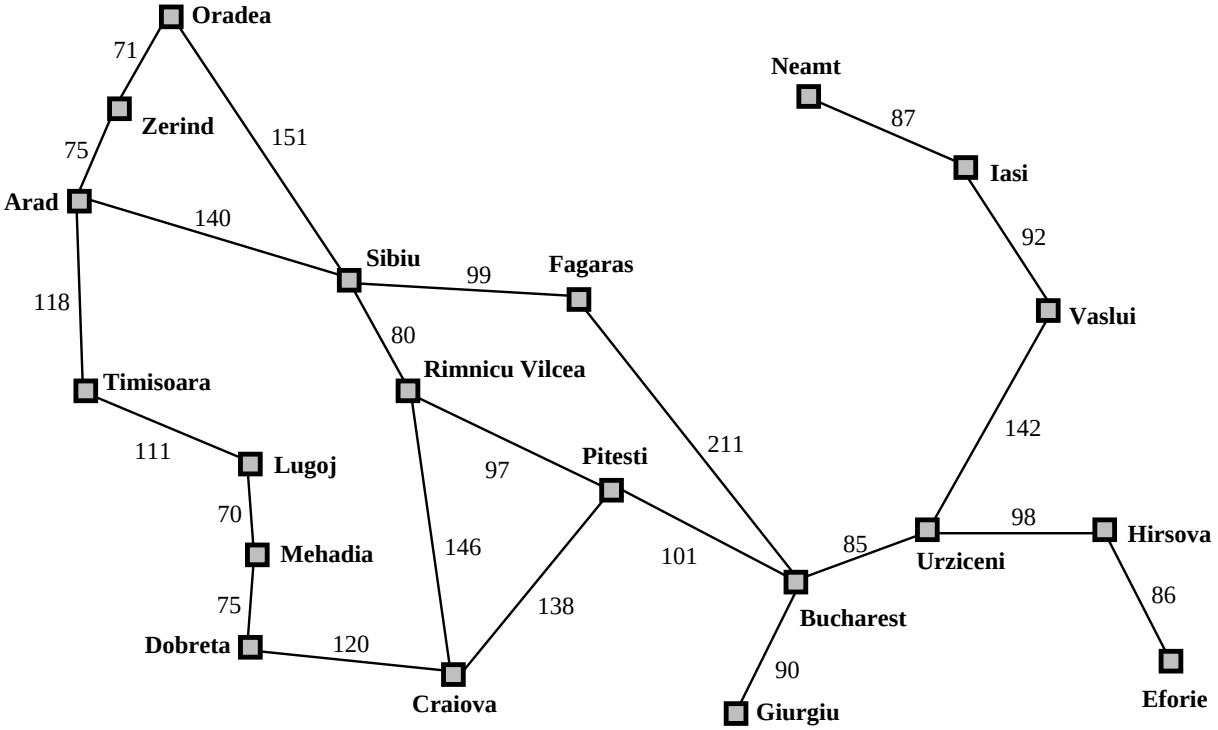
Definition 2.4. Eine heuristische Funktion h heißt *fair* gdw. es zu jedem $n \geq 0$ nur endlich viele Knoten s gibt mit $h(s) \leq n$.

- Fairness entspricht der Vollständigkeit bei uninformierten Suchverfahren.
- Ist eine heuristische Funktion fair, so wird ein Zielknoten gefunden, falls ein solcher existiert.

Bestensuche und Optimalität

- Die Bestensuche vernachlässigt die “Kosten” bei der Anwendnung der Operatoren.
- Wird die Güte einer Lösung charakterisiert durch diese Operatorkosten, so findet die Bestensuche allgemein keine optimale Lösung.

Beispiel 2.11. Suche einer Route von Arad nach Bucharest:



Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Dobreta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Bestensuche wählt Route über Sibiu und Fagaras, obwohl die Route über Rimnicu Vilcea und Pitesti kürzer ist.

Bewertung von Lösungen

Definition 2.5. Es sei $p = (s_0, s_1, \ldots, s_r)$ eine Folge von Zuständen und s_{i+1} sei durch Anwendung eines Zustandsübergangsoperators auf s_i erreichbar.

Beim Übergang von s_i nach s_{i+1} fallen Kosten in Höhe von $k(s_i, s_{i+1})$ an.

Die *Kosten $k(p)$ der Zustandsfolge* seien definiert durch:

$$k(p) := \sum_{i=0}^{r-1} k(s_i, s_{i+1})$$

Für einen Zustand s sei:

$g^*(s)$:= minimale Kosten für einen Weg vom Startzustand s_0 nach s

$h^*(s)$:= minimale Kosten für einen Weg von s zu einem Zielzustand

Problem: Finde (falls möglich) eine Zustandsfolge p^* vom Startzustand s_0 in einen Zielzustand z, die minimale Kosten aufweist, d.h.

$k(p^*) = h^*(s_0)$ bzw.

$k(p^*) = \min\{g^*(z) | z \text{ ist Zielzustand}\}$.

Zulässiger Schätzer

Definition 2.6. Eine heuristische Funktion h heißt *zulässiger Schätzer* bzw. *zulässig* gdw. $\mathrm{h}(s) \leq \mathrm{h}^*(s)$ für alle Zustände s des Zustandsraums.

Bemerkung: Eine zulässiger Schätzer überschätzt nie die anfallenden Restkosten!

Beispiel 2.12. Zulässige Schätzer sind:

- die heuristischen Funktionen aus Beispiel 2.3 für das Schiebepuzzle und
- die Luftlinienentfernung beim Routenproblem.
- Bei kombinatorischen Optimierungsproblemen werden als zulässige Schätzer häufig effizient lösbare Relaxationen des Problems verwendet. Beispiel: minimaler Spannbaum als Relaxation für die Berechnung eines minimalen Hamiltonschen Weges.
- Allgemein: Man vernachlässigt Nebenbedingungen, so daß man zu einem einfach lösbaren Problem kommt.

Der A*-Algorithmus

Der A*-Algorithmus basiert auf:

1. einer Bewertung $g(s)$ für die Zustände, wobei $g(s)$ die bisher geringsten Kosten zur Erreichung des Zustands s angibt,
2. einer (üblicherweise zulässigen) heuristischen Funktion $h(s)$ zur Schätzung der Restkosten und
3. einer Bewertungsfunktion $\Phi(s) = g(s) + h(s)$, die zur Auswahl des zu expandierenden Zustandes dient.

Steuerung der Suche bei A*:

☞ Es wird der Knoten der Agenda expandiert, der die geringste Bewertung $\Phi(s)$ aufweist.

Folgende Punkte sind beim A*-Algorithmus zu berücksichtigen:

- Durch eine Verringerung von $g(s)$ für einen Zustand s kann auch eine Verringerung von $\Phi(s)$ auftreten.
- Dies kann im allgemeinen auch für schon expandierte Knoten der Fall sein!
- Deshalb werden schon expandierte Knoten in einer speziellen Liste *Closed* verwaltet.
- Bewertungen sind dementsprechend anzupassen.

Algorithmus 2.4. [A*]

$Agenda := (Startknoten)$;
$g(Startknoten) := 0$;
$p(Startknoten) = \texttt{nil}$;
`while` $Agenda \neq ()$ `do`
    $s_{akt} := first(Agenda)$;
    Entferne s_{akt} aus der Agenda;
    Füge s_{akt} in *Closed* ein;
    `if` s_{akt} ist Zielknoten `then` s_{akt} ist Lösung; STOP;
    `forall` $s \in Nachfolger(s_{akt})$ `do`
        `if` $s \notin Agenda \wedge s \notin Closed$ `then`
            $g(s) := g(s_{akt}) + k(s_{akt}, s)$;
            $p(s) := s_{akt}$;
            Füge s in die Agenda mit Bewertung $\Phi(s)$ ein;
        `else`
            `if` $g(s_{akt}) + k(s_{akt}, s) < g(s)$ `then`

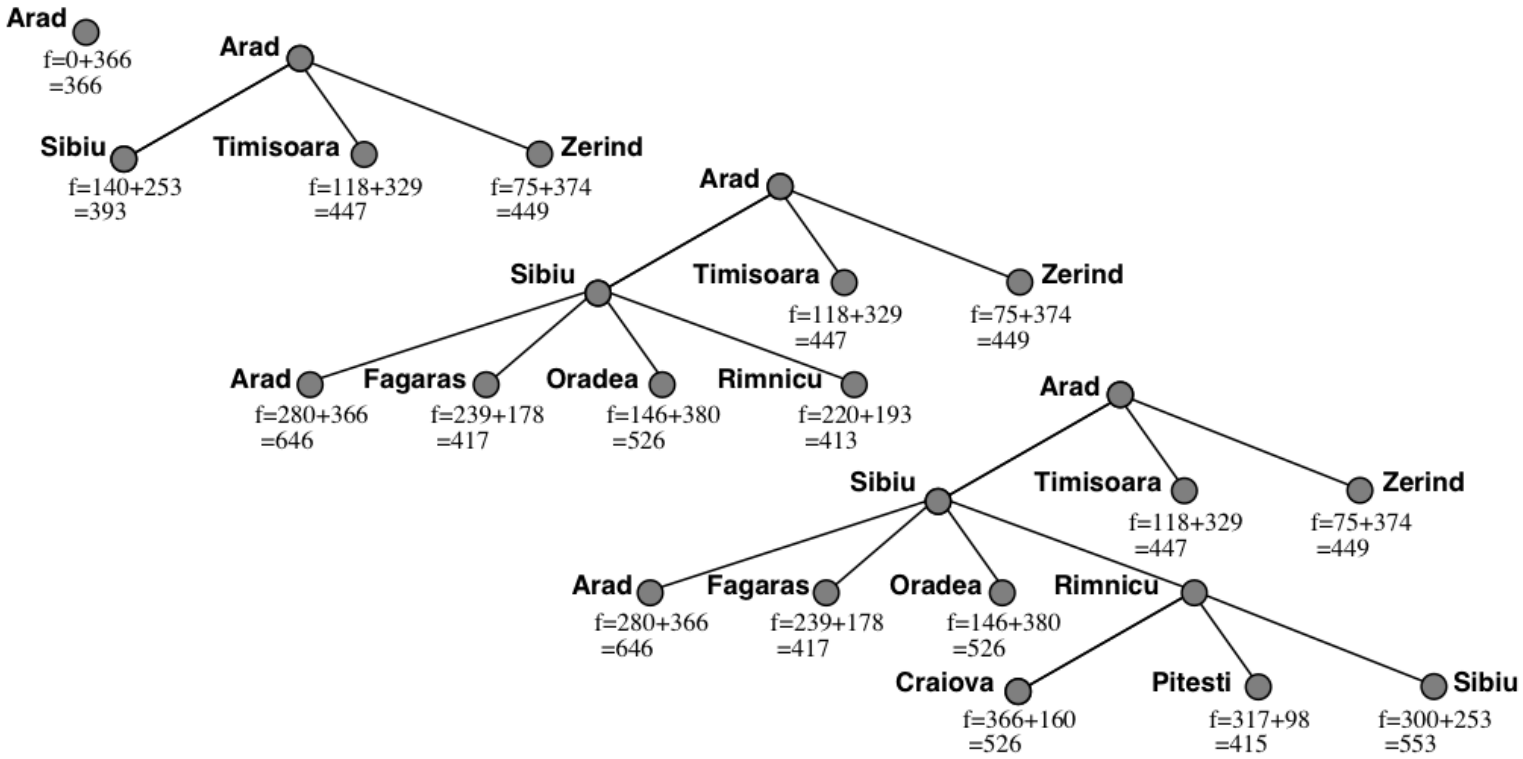
```
                g(s) := g(s_akt) + k(s_akt, s);
                p(s) := s_akt;
                if s ∈ Closed then
                    Entferne s aus Closed;
                    Füge s in die Agenda ein;
                endif
            endif
        endif
    end
end
```

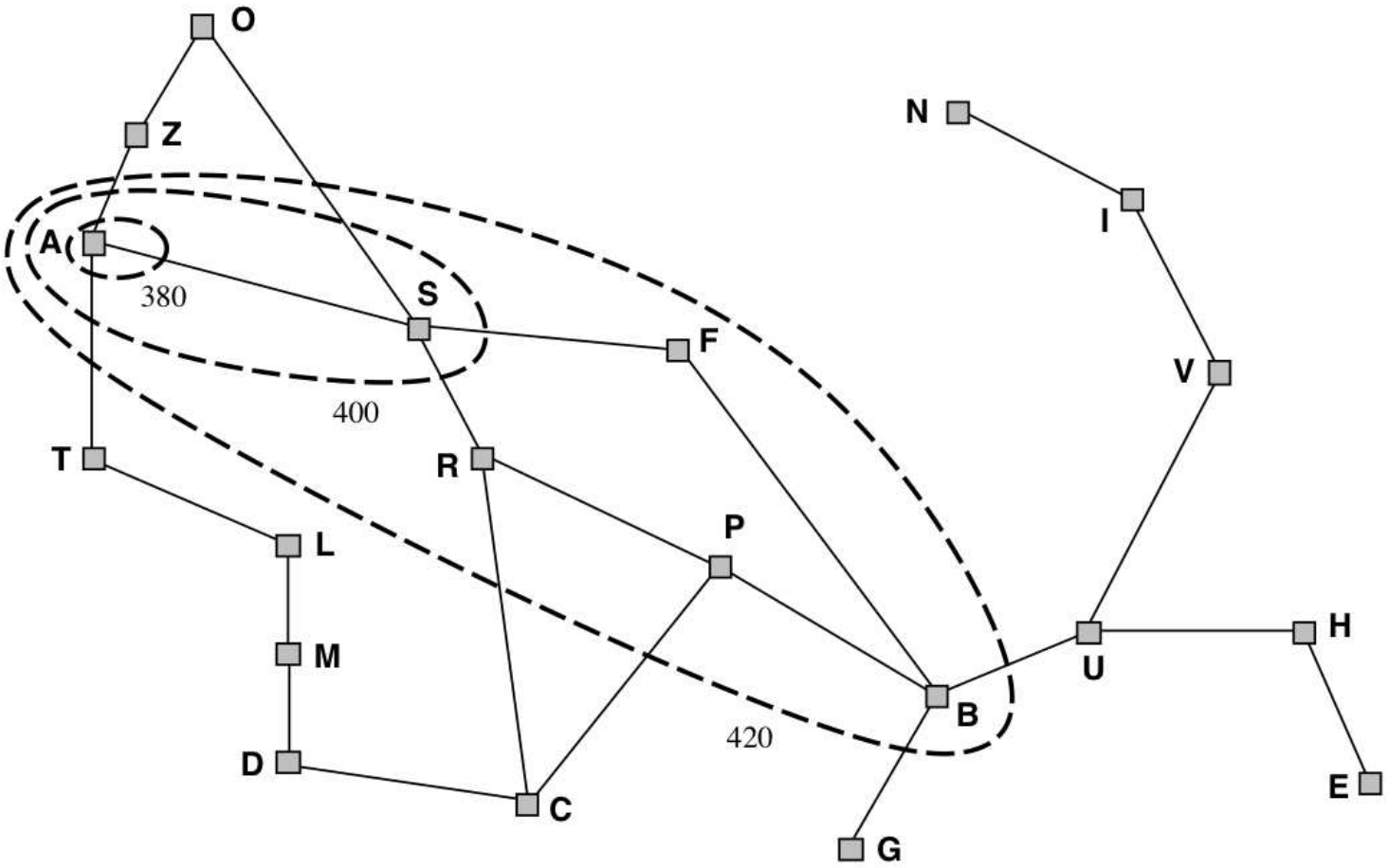
Problem hat keine Lösung; STOP;

- Für einen Knoten s gibt $\mathrm{p}(s)$ den Vorgängerknoten auf dem bisher besten Weg an.
- Den bisher besten Weg zu einem Knoten s erhält man also, in dem man von s sukzessive den Verweisen $\mathrm{p}(.)$ folgt.
- Alternativ kann man an jedem Knoten den kompletten bisher optimalen Pfad speichern.
- Der notwendige Speicherplatzverbrauch für die Pfade ist dann aber quadratisch in der Länge des Suchpfades.

A*-Anwendungsbeispiele

Beispiel 2.13. Beim Routenproblem berechnet der A*-Algorithmus die kürzeste Route.





☞ Durch die Schätzfunktion findet die Suche “zielgerichtet” statt.

Beispiel 2.14. Man benutze den A*-Algorithmus, um eine möglichst kurze Folge von Verschiebeoperationen zu finden, die den Zustand

1	4	2
	8	3
7	6	5

in den Endzustand überführen. Tafel ✎.

Beispiel 2.15. [Rucksackproblem] Das Rucksackproblem lautet:

- Gegeben ist eine Menge $G = \{g_1, \ldots g_n\}$ von Gegenständen.
- Jeder Gegenstand $g \in G$ hat ein Gewicht $w(g)$ und liefert einen Profit $p(g)$.
- Weiterhin ist ein maximales Gesamtgewicht C gegeben.

Es soll nun eine Teilmenge $F \subseteq G$ der Gegenstände bestimmt werden, so daß:

- der Gesamtprofit $\sum_{g \in F} p(g)$ von F maximal ist
- unter der Nebenbedingung, daß das Gesamtgewicht der ausgewählten Gegenstände $\leq C$ ist, d.h. $\sum_{g \in F} w(g) \leq C$.

Zentrale Fragen:

- Wie kann man das Rucksackproblem als Suchproblem formulieren.
- Was muß ein A*-Algorithmus für das Rucksackproblem berücksichtigen?
- Wie erhält man eine Abschätzung des noch erzielbaren Nutzens?

A* und andere Suchverfahren

Bemerkung 2.1. Der A*-Algorithmus enthält die folgenden Algorithmen als Spezialfälle:

- Für $k \geq 0$ und $h \equiv 0$ erhält man den Dijkstra-Algorithmus.
- Für $k \equiv 0$ erhält man die Bestensuche.
- Für $k \equiv 1$ und $h \equiv 0$ erhält man die Breitensuche.
- Für $k \equiv -1$ und $h \equiv 0$ erhält man die Tiefensuche, wenn man Wiederbelebungen verbietet (Übergang von Closed in die Agenda).

Eigenschaften von A*

Satz 2.1. [Terminierung, Fairness] *Es gelte:*

- *Jeder Zustand besitzt nur endlich viele Nachfolgerzustände,*
- *es existiere ϵ, so daß für die Kosten $\mathrm{k}(s, s')$ bei einem Zustandsübergang stets $\mathrm{k}(s, s') \geq \epsilon > 0$ gilt und*
- *es gibt einen erreichbaren Zielzustand.*

$\Rightarrow$ *Dann terminiert A* nach endlich vielen Schritten mit dem Erreichen eines Zielzustandes.*

Bemerkung 2.2. Unter den gegebenen Voraussetzungen endet die Suche u.U. in einem nicht optimalen Zielzustand.

Eigenschaften von A* (2)

Satz 2.2. [Optimalität] *Es gelte:*

- *Gegeben sind die Voraussetzungen von Satz 2.1 und*
- h *ist zulässig.*

$\Rightarrow$ *Dann ist der Zielknoten* z*, mit dem A* terminiert, ein optimaler Zielknoten,*

$\Rightarrow$ *die minimalen Kosten ergeben sich durch* $\mathrm{g}(z)$ *und*

$\Rightarrow$ *ausgehend von* $\mathrm{p}(z)$ *kann eine optimale Zustandsfolge ermittelt werden.*

Korollar 2.3. *Gegeben seien die Voraussetzungen von Satz 2.2. Der gefundene optimale Zielknoten sei* z*. Dann wurden während des Laufs von A* nur Zustände* s *mit* $\Phi(s) \leq \mathrm{g}(z)$ *expandiert.*

Wahl guter Schätzer

- Die Eigenschaften der heuristischen Funktion haben einen wesentlichen Einfluß auf die Performanz der Suche mit A*.
- Eine zulässige heuristische Funktion ist um so besser, je näher sie dem Optimalwert zur Erreichung eines Zielzustandes kommt.

Definition 2.7. Für zwei zulässige Schätzer h und h' heißt:

- h' *besser informiert* als h gdw. $h(s) < h'(s)$ für alle Zustände s gilt.
- h' *nicht schlechter informiert* als h gdw. $h(s) \leq h'(s)$ für alle Zustände s gilt.

Satz 2.4. *Es gelte:*

- *Gegeben sind die Voraussetzungen von Satz 2.2,*
- A *bzw.* A' *seien A*-Algorithmen, die zulässige Schätzer* h *bzw.* h' *verwenden und*
- h' *sei besser informiert als* h*.*

$\Rightarrow$ *Dann wird jeder Zustand* s*, der von* A' *expandiert wird, auch von* A *expandiert.*

Satz 2.5. *Es gelte:*

- *Gegeben sind die Voraussetzungen von Satz 2.1 und*
- $\mathrm{h}(s) \leq (1+\epsilon)\mathrm{h}^*(s)$ *für alle Zustände* s*.*

$\Rightarrow$ *Dann gilt für den vom A*-Algorithms ermittelten Zielzustand* z*:*

$$\mathrm{g}(z) \leq (1+\epsilon)\mathrm{g}(z^*)$$

mit z^* *ist Zielzustand einer optimalen Lösung.*

Monotone Schätzer

Definition 2.8. Gegeben sei eine nichtnegative Kostenfunktion k. Eine heuristische Funktion h heißt *monotoner Schätzer* gdw. gilt:

- $\mathrm{h}(z) = 0$ für alle Zielzustände z.
- Für alle Zustände s und alle Nachfolger s' von s gilt:

$$\mathrm{h}(s) \leq \mathrm{k}(s, s') + \mathrm{h}(s')$$

Beispiel 2.16. Alle Schätzer aus Beispiel 2.12 sind auch monotone Schätzer.

Satz 2.6. *Es gelte:*

- *Gegeben sind die Voraussetzungen von Satz 2.1 und*
- h *sei ein monotoner Schätzer.*

$\Rightarrow$ *Dann ist* h *auch ein zulässiger Schätzer.*

$\Rightarrow$ *Ist der Knoten* s' *durch Expansion des Knotens* s *entstanden, so gilt* $\Phi(s) \leq \Phi(s')$.

$\Rightarrow$ *Es gibt keine Wiederbelebung von Zuständen, d.h. ein Knoten, der expandiert wurde, wird nie mehr selektiert.*

Satz 2.7. h *sei ein zulässiger Schätzer. Dann existiert ein monotoner Schätzer* h'*, der nicht schlechter informiert ist als* h.

Begründung: Betrachte Zustand s mit Vorgänger s_{pre}:

$$\mathrm{h}'(s) := \begin{cases} \mathrm{h}(s) & \text{falls } \mathrm{h}(s_{\mathrm{pre}}) \leq \mathrm{k}(s_{\mathrm{pre}}, s) + \mathrm{h}(s) \\ \mathrm{h}(s_{\mathrm{pre}}) - \mathrm{k}(s_{\mathrm{pre}}, s) & \text{sonst} \end{cases}$$

Dann gilt:

- h' ist monoton.
- h' ist zulässig, weil h zulässig ist.
- $\mathrm{h}(s) \leq \mathrm{h}'(s)$, also ist h' besser informiert als h.

Zusammenfassung des Kapitels

- Zustandsraum: Zustände, Zustandsübergänge, Startzustand, Zielzustände
- Systematische Suche im Zustandsraum: Breitensuche, Tiefensuche
- Heuristische Funktionen: Schätzung der Entfernung zum Ziel
- Bestensuche garantiert keine Optimalität
- A*: Operatorkosten plus heuristischer Funktion
- A* liefert optimale Lösungen bzgl. Operatorkosten

3. Logikbasierte Wissensrepräsentation und Inferenz

Am Beispiel der Aussagenlogik erklären wir schrittweise wichtige Elemente eines logischen Systems.

- Zunächst benötigt ein logisches System ein Vokabular,
- d.h. eine Menge von Namen für Aussagen über die reale Welt.
- Eine derartige Menge von Namen nennen wir *Signatur*. Solch eine Signatur wird üblicherweise durch Σ gekennzeichnet.

Aussagenlogische Signatur

Definition 3.1. Eine *aussagenlogische Signatur* Σ ist eine Menge von Bezeichnern, den *Aussagenvariablen*.

Beispiel 3.1. Die Menge

$$\Sigma_{\mathrm{AL}} := \{\texttt{hatFieber}, \texttt{istKrank}, \texttt{istArbeitsunfähig}\}$$

ist eine aussagenlogische Signatur, die drei Aussagenvariablen zur Verfügung stellt.

Im folgenden benutzen wir häufig Großbuchstaben als Aussagenvariablen.

Formeln

- Formeln ermöglichen es, Beziehungen zwischen Aussagen zu beschreiben.
- Formeln sind gemäß einer gewissen Syntax aufgebaut (sie sind *wohlgeformt*). Diese Syntax legt eine Wissensrepräsentationssprache fest.
- Die Formalsyntax ist üblicherweise rekursiv aufgebaut.
- Die *atomaren Formeln* ergeben sich aus der Signatur.
- Mit logischen Verknüpfungsoperatoren (den *Junktoren*) werden aus Formeln rekursiv komplexere Formeln aufgebaut.

Aussagenlogische Formeln

Definition 3.2. Für eine aussagenlogische Signatur Σ ist die Menge $\texttt{Formel}(\Sigma)$ der *aussagenlogischen Formeln* wie folgt definiert:

- Die Elemente der Menge Σ sind aussagenlogische Formeln, die sogenannten *atomaren Formeln*.
- Falls F und G aussagenlogische Formeln sind, dann sind auch die folgenden Konstrukte aussagenlogische Formeln:

$(\neg \mathrm{F})$	Negation
$(\mathrm{F} \wedge \mathrm{G})$	Konjunktion
$(\mathrm{F} \vee \mathrm{G})$	Disjunktion
$(\mathrm{F} \rightarrow \mathrm{G})$	Implikation
$(\mathrm{F} \leftrightarrow \mathrm{G})$	Äquivalenz

Bemerkung 3.1. Zur Vereinfachung der Schreibweise verzichten wir i.d.R. auf die Klammerung und benutzen statt dessen die folgenden Bindungsprioritäten:

$$\neg, \wedge, \vee, \rightarrow, \leftrightarrow \ .$$

Durch die Menge $\texttt{Formel}(\Sigma)$ wird die Sprache zur Repräsentation von Wissen definiert.

Interpretation

- Die Syntax einer Logik legt ausschließlich deren äußere Form fest, sie sagt aber nichts über die Bedeutung der Formeln aus.
- Benötigt wird eine Verbindung zwischen den syntaktischen Elementen der Logik und den Objekten der zu repräsentierenden Welt.
- Diese Verbindung wird durch eine sogenannte Interpretation (genauer: Σ-Interpretation) hergestellt.
- Eine *Σ-Interpretation* einer Signatur ist die Zuordnung von den Elementen der Signatur Σ (Namen) zu den Elementen der zu repräsentierenden Welt.

Belegung

In der Aussagenlogik ist die Interpretation ganz einfach: Für jede Aussagenvariable wird ein wahrheitswert festgelegt.

Definition 3.3. Es sei Σ eine aussagenlogische Signatur.

- Eine Abbildung $\mathrm{I} : \Sigma \longrightarrow \{\mathtt{wahr}, \mathtt{falsch}\}$ heißt *aussagenlogische Interpretation* oder *Belegung* für Σ.
- $\mathtt{Int}(\Sigma)$ bezeichnet die Menge der Belegungen für Σ.

Beispiel 3.2. Für die Signatur aus Beispiel 3.1 ist I definiert durch

$$\begin{array}{lcl} \mathrm{I}(\mathtt{hatFieber}) & = & \mathtt{wahr} \\ \mathrm{I}(\mathtt{istKrank}) & = & \mathtt{wahr} \\ \mathrm{I}(\mathtt{istArbeitsunfähig}) & = & \mathtt{falsch} \end{array}$$

eine mögliche Belegung.

Erfüllungsrelation

- Die Interpretation liefert uns nur einen Wahrheitswert für die atomaren Formeln.
- Wir benötigen eine Ausdehnung der Semantik auf alle Formeln $\mathtt{F} \in \mathtt{Formel}(\Sigma)$.
- Dieses stellt uns eine Erfüllungsrelation $\models$ bereit.
- Durch solch eine Erfüllungsrelation ist definiert, ob eine Formel $\mathtt{F}$ in einer Σ-Interpretation $\mathtt{I}$ wahr ist oder nicht, d.h.
- sie ordnet einer Interpretation und einer Formel einen Wahrheitswert zu.
- Eine Erfüllungsrelation definiert hierzu im wesentlichen die Semantik der Junktoren.

Definition 3.4. Es seien $F, G \in \texttt{Formel}(\Sigma)$ (nichtatomare) aussagenlogische Formeln. Durch die folgenden Wahrheitstafel wird eine Σ-Interpretation I von Σ auf die Menge $\texttt{Formel}(\Sigma)$ ausgedehnt:

$I(F)$	$I(\neg F)$
f	w
w	f

$I(F)$	$I(G)$	$I(F \vee G)$	$I(F \wedge G)$	$I(F \rightarrow G)$
f	f	f	f	w
f	w	w	f	w
w	f	w	f	f
w	w	w	w	w

Für $I \in \texttt{Int}(\Sigma)$ und $F \in \texttt{Formel}(\Sigma)$ gelte:

$$I \models F \text{ gdw. } I(F) = \texttt{w}$$

Modell

Definition 3.5. Es seien $\mathtt{I} \in \mathtt{Int}(\Sigma)$ und $\mathtt{F} \in \mathtt{Formel}(\Sigma)$. Gilt $\mathtt{I} \models \mathtt{F}$, so sagen wir

- "$\mathtt{I}$ erfüllt $\mathtt{F}$" und
- bezeichnen $\mathtt{I}$ als *Σ-Modell* für $\mathtt{F}$.

$\mathtt{Mod}_\Sigma(\mathtt{F}) \subseteq \mathtt{Int}(\Sigma)$ bezeichnet die *Menge aller Σ-Modelle* für $\mathtt{F}$.

Für eine Menge $\mathcal{F} \subset \mathtt{Formel}(\Sigma)$ von Formeln gelte $\mathtt{I} \models \mathcal{F}$ gdw. $\mathtt{I} \models \mathtt{F}$ für alle $\mathtt{F} \in \mathcal{F}$.

$\mathtt{I}$ ist dann ein *Modell* für die Formelmenge $\mathcal{F}$.

Beispiel 3.3. Die Interpretation I aus Beispiel 3.2 ist ein Modell für die Formel

$$\texttt{hatFieber} \rightarrow \texttt{istKrank}$$

Dagegen ist I kein Modell für die Formel

$$\texttt{istKrank} \rightarrow \texttt{istArbeitsunfähig}$$

Beweis mit Wahrheitstafeln ✎.

Erfüllbarkeit

Besonders interessant sind Formeln, die für alle Interpretationen wahr bzw. falsch sind.

> "Kräht der Hahn auf dem Mist, ändert sich das Wetter oder es bleibt wie es ist."

Definition 3.6. Eine Formel F heißt

- *erfüllbar* gdw. es ein Modell für die Formel gibt.
- *unerfüllbar (Kontradiktion)* gdw. es kein Modell für die Formel gibt.
- *allgemeingültig (Tautologie)* gdw. jede Interpretation ein Modell für die Formel ist.
- *falsifizierbar* gdw. es eine Interpretation gibt, die kein Modell für die Formel ist.

Die Begriffe werden in analoger Weise für Formelmengen $\mathcal{F} \subset \mathtt{Formel}(\Sigma)$ verwendet.

Beispiel 3.4. Wichtige Tautologien sind:

- Modus Ponens
$$(F \wedge (F \rightarrow G)) \rightarrow G$$
- Modus Tollens
$$((F \rightarrow G) \wedge \neg G) \rightarrow \neg F$$
- Und-Elimination
$$(F \wedge G) \rightarrow F$$
- Oder-Introduktion
$$F \rightarrow (F \vee G)$$
- Resolutionsregel
$$((F \rightarrow G) \wedge (\neg F \rightarrow H)) \rightarrow (G \vee H)$$

Semantische Folgerung

- In einem wissensbasierten System wollen wir Fakten aus anderen Fakten und Regeln herleiten.
- Wir können eine Wissensbasis als eine Menge $\mathcal{F} \subset \texttt{Formel}(\Sigma)$ betrachten.
- Eine solche Menge $\mathcal{F} = \{F_1, \ldots, F_n\}$ entspricht der Konjunktion $F_1 \wedge \ldots \wedge F_n$.
- Unser übliches Verständnis von Folgerung läßt sich so ausdrücken:

  Ist eine Formel G immer dann wahr, wenn alle Formeln aus $\mathcal{F}$ wahr sind, dann folgt G aus $\mathcal{F}$.
- Damit können wir die Erfüllungsrelation $\models$ auf eine Beziehung zwischen Formeln und Formelmengen ausdehnen.

Definition 3.7. Es seien $\mathsf{F}, \mathsf{G} \in \mathtt{Formel}(\Sigma)$ aussagenlogische Formeln.

- G heißt *semantische Folgerung* von F gdw. jedes Modell für F auch ein Modell für G ist.
- In diesem Fall schreiben wir $\mathsf{F} \models \mathsf{G}$.
- Wir sagen auch "G folgt logisch aus F" bzw. "aus F folgt semantisch G".
- Für eine Formelmenge $\mathcal{F}$ gelte $\mathcal{F} \models \mathsf{G}$ gdw. jedes Modell für $\mathcal{F}$ auch ein Modell für G ist.
- Für Formelmengen $\mathcal{F}, \mathcal{G}$ gelte $\mathcal{F} \models \mathcal{G}$ gdw. $\mathcal{F} \models \mathsf{G}$ für alle $\mathsf{G} \in \mathcal{G}$ gilt.

Beispiel 3.5. Gegeben sei die Formelmenge $\mathcal{F}$

$$\mathcal{F} = \left\{ \begin{array}{lcl} \texttt{hatFieber} & \rightarrow & \texttt{istKrank}, \\ \texttt{istKrank} & \rightarrow & \texttt{istArbeitsunfähig}, \\ \texttt{hatFieber} & & \end{array} \right\}$$

Kann aus $\mathcal{F}$ die Aussage `istArbeitsunfähig` gefolgert werden, d.h. gilt $\mathcal{F} \models$ `istArbeitsunfähig`?

Ja! Beweis mit Wahrheitstafeln ✎.

Beispiel 3.6. Wir wollen uns ein Haustier anschaffen und machen folgende Überlegungen:

1. Es sollte nur ein Hund (H), eine Katze (K) oder ein Hamster (M) sein.
2. Besitzer wertvoller Möbel (W) sollten keine Katze anschaffen, da diese die Möbel zerkratzen würde.
3. Ein Hund erfordert ein freistehendes Haus (F), damit sich kein Nachbar durch das Bellen gestört fühlt.

Wir vermuten: Für einen Besitzer wertvoller Möbel ohne freistehendes Haus kommt nur ein Hamster in Frage.

Die Aussagen lauten als Aussagenlogische Formeln:

1. $\mathrm{H} \vee \mathrm{K} \vee \mathrm{M}$
2. $W \rightarrow \neg \mathrm{K}$
3. $\mathrm{H} \rightarrow \mathrm{F}$

Hyp. $W \wedge \neg F \rightarrow M \wedge \neg H \wedge \neg K$

Nr.	I(H)	I(K)	I(M)	I(W)	I(F)	1. ∧ 2. ∧ 3.	Hyp.
1.	f	f	f	f	f	f	w
2.	f	f	f	f	w	f	w
3.	f	f	f	w	f	f	f
4.	f	f	f	w	w	f	w
5.	f	f	w	f	f	w	w
6.	f	f	w	f	w	w	w

Nr.	I(H)	I(K)	I(M)	I(W)	I(F)	1. ∧ 2. ∧ 3.	Hyp.
7.	f	f	w	w	f	w	w
8.	f	f	w	w	w	w	w
9.	f	w	f	f	f	w	w
10.	f	w	f	f	w	f	w
11.	f	w	f	w	f	f	f
12.	f	w	f	w	w	f	w
13.	f	w	w	f	f	w	w
14.	f	w	w	f	w	w	w
15.	f	w	w	w	f	f	f
16.	f	w	w	w	w	f	w
17.	w	f	f	f	f	f	w
18.	w	f	f	f	w	f	w
19.	w	f	f	w	f	f	f

Nr.	I(H)	I(K)	I(M)	I(W)	I(F)	1. ∧ 2. ∧ 3.	Hyp.
20.	w	f	f	w	w	w	w
21.	w	f	w	f	f	f	w
22.	w	f	w	f	w	f	w
23.	w	f	w	w	f	f	f
24.	w	f	w	w	w	w	w
25.	w	w	f	f	f	f	w
26.	w	w	f	f	w	w	w
27.	w	w	f	w	f	f	f
28.	w	w	f	w	w	f	w
29.	w	w	w	f	f	f	w
30.	w	w	w	f	w	f	w
31.	w	w	w	w	f	f	f
32.	w	w	w	w	w	f	w

Fazit:

- Es gibt zehn Modelle für die Formelmenge $\{1., 2., 3.\}$. Dies sind die Interpretationen mit den Nummern 5, 6, 7, 8, 9, 13, 14, 20, 24, 26.
- Jedes dieser Modelle ist auch ein Modell für die Hypothese.
- Somit folgt die Hypothese semantisch aus $\{1., 2., 3.\}$.

Satz 3.1. *Es seien* F, G *aussagenlogische Formeln. Dann gilt:*

- F *ist Tautologie gdw.* $\neg\mathrm{F}$ *ist unerfüllbar.*
- $\mathrm{F} \models \mathrm{G}$ *gdw.* $\mathrm{F} \rightarrow \mathrm{G}$ *ist Tautologie.*
- $\mathrm{F} \models \mathrm{G}$ *gdw.* $\mathrm{F} \wedge \neg\mathrm{G}$ *ist unerfüllbar.*

Bemerkung 3.2. Die Äquivalenzen können auf Formelmengen $\mathcal{F}, \mathcal{G}$ ausgedehnt werden.

Kalkül

- Schon das kleine Beispiel 3.6 verdeutlichte, daß Inferenz auf Basis der Definition der semantischen Folgerung ineffizient ist.
- Allgemein müssen für eine Formelmenge $\mathcal{F}$ mit k verschiedenen Aussagevariablen 2^{k} Belegungen getestet werden.
- Daher benutzt man für die maschinelle Inferenz Techniken, die allein auf der Syntax der Formeln beruhen.
- Statt alle möglichen Belegungen zu testen, sucht man nach einer Folge von syntaktischen Umformungen, die die Hypothese zu beweisen.

- Ein *Kalkül* besteht aus einer Menge von logischen *Axiomen* und *Inferenzregeln*.
- Die Axiome sind entweder eine Menge von elementaren Tautologien (*positiver Kalkül*) oder
- eine Menge von elementaren Widersprüchen (*negativer Kalkül*).

- Die Inferenzregeln sind Vorschriften, nach denen aus Formeln andere Formeln abgeleitet werden können.
- Sie werden in der folgenden Form notiert:

$$\frac{F_1, \ldots, F_n}{F}$$

  Dies besagt, daß aus den Formeln (der syntaktischen Form) $F_1, \ldots, F_n$ (Bedingungen) eine Formel der Form F (Schlussfolgerung) abgeleitet werden kann.
- So können aus den Tautologien von Beispiel 3.4 Inferenzregeln gebildet werden. Aus dem Modus Ponens ergibt sich die Inferenzregel:

$$\frac{F, F \rightarrow G}{G}$$

- Ist eine Formel F aus den Formeln $F_1, \ldots, F_n$ durch eine Folge von Anwendungen der Inferenzregeln ableitbar, so schreibt man

$$F_1, \ldots, F_n \vdash F$$

Beispiel 3.7. Gegeben sei die Formelmenge $\mathcal{F}$ aus Beispiel 3.5. Mit der Inferenzregel Modus Ponens leiten wir ab:

$$\frac{\texttt{hatFieber}, \texttt{hatFieber} \rightarrow \texttt{istKrank}}{\texttt{istKrank}}$$

Nochmals angewandt ergibt sich:

$$\frac{\texttt{istKrank}, \texttt{istKrank} \rightarrow \texttt{istArbeitsunfaehig}}{\texttt{istArbeitsunfaehig}}$$

Also gilt: $\mathcal{F} \vdash \texttt{istArbeitsunfähig}$.

Eigenschaften von Kalkülen

- Ein Kalkül ist *korrekt* gdw. alle syntaktischen Ableitungen auch semantische Folgerungen sind, d.h. für Formeln F und G gilt:

$$F \vdash G \text{ impliziert } F \models G$$

- Ein Kalkül ist *vollständig* gdw. alle semantischen Folgerungen auch syntaktisch abgeleitet werden können, d.h. für Formeln F und G gilt:

$$F \models G \text{ impliziert } F \vdash G$$

- Ein Kalkül ist *widerlegungsvollständig* gdw. aus allen semantischen Folgerungen eine unerfüllbare Formel $\square$ abgeleitet werden kann, d.h. für Formeln F und G gilt:

$$F \models G \text{ impliziert } F \wedge \neg G \vdash \square$$

Semantische Äquivalenz

Beispiel 3.8. Syntaktisch unterschiedliche Formel können identische Wahrheitswerte haben. Man betrachte die Formeln $\neg(F \vee G)$ und $\neg F \wedge \neg G$:

F	G	$\neg(F \vee G)$	$\neg F \wedge \neg G$
f	f	w	w
f	w	f	f
w	f	f	f
w	w	f	f

Definition 3.8. Zwei aussagenlogische Formeln $F, G \in \texttt{Formel}(\Sigma)$ heißen *semantisch äquivalent* gdw. $I(G) = I(F)$ für jede Belegung $I \in \texttt{Int}(\Sigma)$ gilt.

Wenn F und G semantisch äquivalent sind, schreiben wir hierfür $F \equiv G$.

Lemma 3.2. *Wichtige semantische Äquivalenzen sind:*

$F \to G$	$\equiv$	$\neg F \vee G$	Implikation
$\neg(F \vee G)$	$\equiv$	$\neg F \wedge \neg G$	DeMorgan
$\neg(F \wedge G)$	$\equiv$	$\neg F \vee \neg G$	
$\neg\neg F$	$\equiv$	F	Dop. Negation
$F \vee F$	$\equiv$	F	Idempotenz
$F \wedge F$	$\equiv$	F	
$F \wedge (F \vee G)$	$\equiv$	F	Absorption
$F \vee (F \wedge G)$	$\equiv$	F	
$F \vee G$	$\equiv$	$G \vee F$	Kommutativität
$F \wedge G$	$\equiv$	$G \wedge F$	
$F \wedge (G \wedge H)$	$\equiv$	$(F \wedge G) \wedge H$	Assoziativität
$F \vee (G \vee H)$	$\equiv$	$(F \vee G) \vee H$	
$F \wedge (G \vee H)$	$\equiv$	$(F \wedge G) \vee (F \wedge H)$	Distributivität
$F \vee (G \wedge H)$	$\equiv$	$(F \vee G) \wedge (F \vee H)$	

Normalformen

Für die maschinelle Inferenz ist die Darstellung einer Formel in einer standardisierten und möglichst einfachen Form wichtig.

Definition 3.9.

- Eine Formel F ist ein *Literal* gdw. F eine atomare Formel oder die Negation einer atomaren Formel ist.
- Eine Formel F ist in *konjunktiver Normalform (KNF)* gdw. F eine Konjunktion von Disjunktionen von Literalen ist, d.h.

$$F = ((L_{1,1} \vee \ldots \vee L_{1,m_1}) \wedge \ldots \wedge (L_{n,1} \vee \ldots \vee L_{n,m_n}))$$

- Eine Formel F ist in *disjunktiver Normalform DNF* gdw. F eine Disjunktion von Konjunktionen von Literalen ist, d.h.

$$F = ((L_{1,1} \wedge \ldots \wedge L_{1,m_1}) \vee \ldots \vee (L_{n,1} \wedge \ldots \wedge L_{n,m_n}))$$

Beispiel 3.9. Die Formeln

$$(F \vee \neg G \vee H) \wedge J \text{ und } \neg F \wedge G$$

sind in KNF.

Die Formeln

$$(\neg F \wedge G) \vee (\neg H \wedge \neg J) \text{ und } F \vee \neg G$$

sind in DNF.

Transformation in Normalform

Umformungsregeln für KNF/DNF-Transformation:

Schritt 1	$F \rightarrow G$	$\mapsto$	$\neg F \vee G$
	$\neg\neg F$	$\mapsto$	F
Schritt 2	$\neg(F \wedge G)$	$\mapsto$	$\neg F \vee \neg G$
	$\neg(F \vee G)$	$\mapsto$	$\neg F \wedge \neg G$
Schritt 3 (KNF)	$F \vee (G \wedge H)$	$\mapsto$	$(F \vee G) \wedge (F \vee H)$
	$(F \wedge G) \vee H$	$\mapsto$	$(F \vee H) \wedge (G \vee H)$
Schritt 3 (DNF)	$F \wedge (G \vee H)$	$\mapsto$	$(F \wedge G) \vee (F \wedge H)$
	$(F \vee G) \wedge H$	$\mapsto$	$(F \wedge H) \vee (G \wedge H)$

Klauselform

Für die maschinelle Inferenz benutzt man eine Mengendarstellung der KNF, die sogenannte Klauselform.

Definition 3.10.

- Eine *Klausel* ist eine Menge von Literalen $\{L_1, \ldots, L_n\}$, die der Disjunktion $L_1 \vee \ldots \vee L_n$ entspricht.
- Die Klausel $\{\}$ ist die *leere Klausel*. Sie wird in der Form $\square$ geschrieben und entspricht dem Wahrheitswert `falsch` $(f, 0)$.
- Die *Klauselform* einer Formel F in KNF mit

$$F = ((L_{1,1} \vee \ldots \vee L_{1,m_1}) \wedge \ldots \wedge (L_{n,1} \vee \ldots \vee L_{n,m_n}))$$

ist die Menge

$$F = \{\{L_{1,1}, \ldots, L_{1,m_1}\}, \ldots, \{L_{n,1}, \ldots, L_{n,m_n}\}\}$$

Resolution

Beispiel 3.10. *Resolution* basiert auf folgendem Schema:

- Wenn es regnet (R), gehe ich ins Kino (K), also $R \rightarrow K$.
- Wenn es nicht regnet ($\neg R$), gehe ich ins Schwimmbad (S), also $\neg R \rightarrow S$.
- Hieraus folgt, daß ich ins Kino oder ins Schwimmbad gehe, also

$$\{R \rightarrow K, \neg R \rightarrow S\} \models K \vee S$$

Als Inferenzregel geschrieben lautet die Resolution wie folgt:

$$\frac{F \rightarrow G, \neg F \rightarrow H}{G \vee H}$$

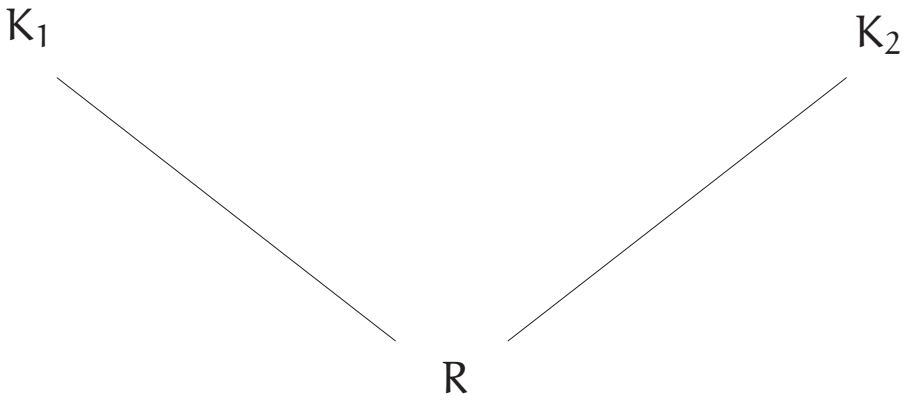
Für die maschinelle Inferenz benutzt man Resolution in Verbindung mit Klauselform.

Definition 3.11. Seien K_1, K_2 Klauseln und sei A eine atomare Formel mit $A \in K_1$ und $\neg A \in K_2$. Dann heißt die Klausel R mit

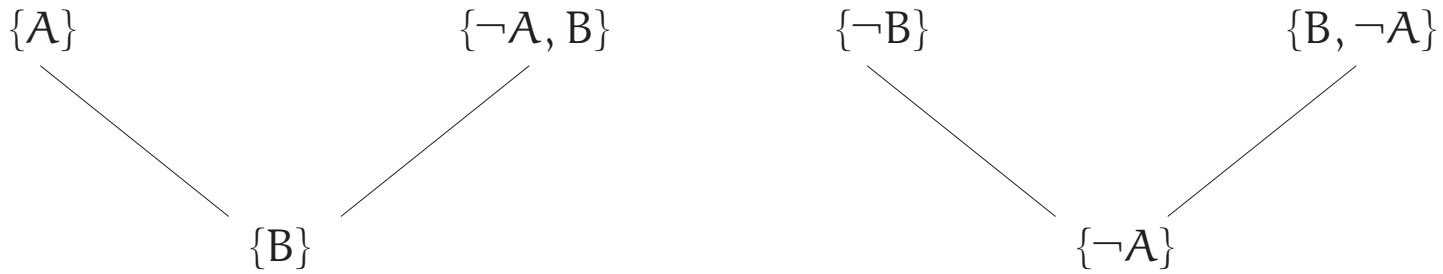
$$R = (K_1 \setminus \{A\}) \cup (K_2 \setminus \{\neg A\})$$

Resolvente von K_1 und K_2.

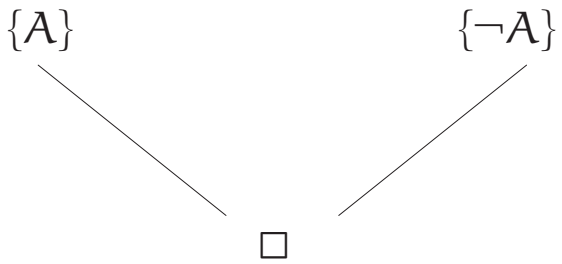
Ein Resolutionsschritt wird wie folgt dargestellt:



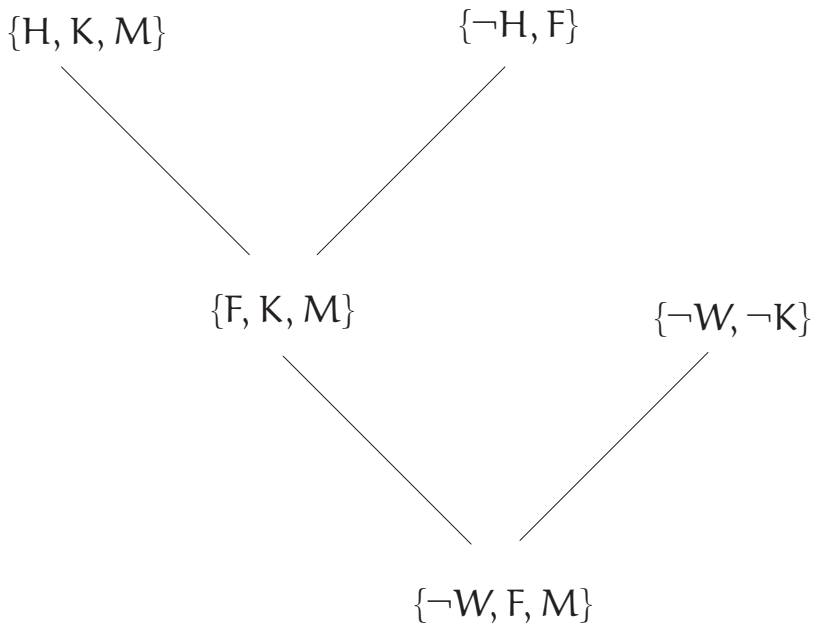
Beispiel 3.11. Modus Ponens und Modus Tollens können als Spezialfall der Resolution dargestellt werden:



Die Resolvente zweier widersprüchlicher Klauseln ist die leere Klausel:



Beispiel 3.12. Herleitung der Aussage aus Beispiel 3.6 mit der Resolutiosregel:



- Das letzte Beispiel zeigt den direkten Beweis einer Formel mit Hilfe der Resolutionsregeln.
- Beim Resolutionskalkül führt man stattdessen einen Widerspruchsbeweis.
- D.h., man beweist $\mathrm{F} \models \mathrm{G}$, in dem man zeigt, daß $\mathrm{F} \wedge \neg \mathrm{G}$ unerfüllbar ist (vgl. Satz 3.1).
- Dies bedeutet, man leitet aus den Klauseln von F vereinigt mit den Klauseln, die sich aus $\neg \mathrm{G}$ ergeben, die leere Klausel ab.

Satz 3.3. *Es sei* $\mathbb{F}$ *eine Klauselmenge und es seien* $K_1, K_2 \in \mathbb{F}$*. Für eine Resolvente* R *von* K_1 *und* K_2 *gilt* $\mathbb{F} \models R$*.*

Insbesondere ist $\mathbb{F}$ *genau dann erfüllbar, wenn* $\mathbb{F} \cup \{R\}$ *erfüllbar ist.*

- Satz 3.3 sagt aus, daß durch die Hinzunahme von Resolventen die Erfüllbarkeitseigenschaft einer Klauselmenge nicht beeinträchtigt wird.
- Dies nutzt man im Resolutionskalkül aus. Um zu zeigen, daß eine Klauselmenge $\mathbb{F}$ unerfüllbar ist, bildet man solange Resolventen und fügt sie der Klauselmenge hinzu, bis irgendwann eine Menge $\mathbb{F}'$ entsteht, die die leere Klausel enthält.
- Diese Klauselmenge $\mathbb{F}'$ ist unerfüllbar, also muß auch die ursprüngliche Klauselmenge $\mathbb{F}$ unerfüllbar sein.

Beispiel 3.13. Herleitung der Aussage aus Beispiel 3.6 mit dem Resolutionskalkül:

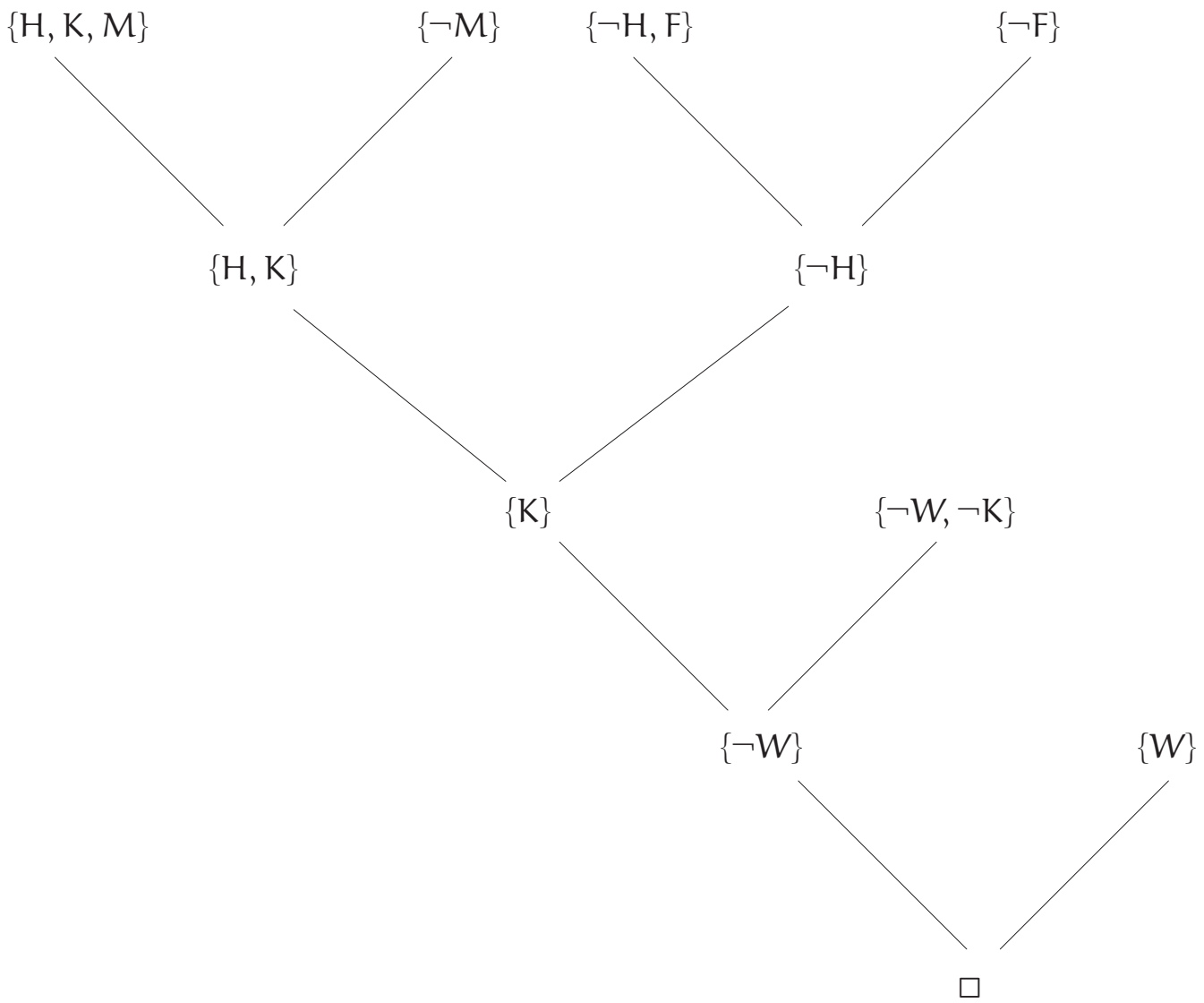
Klauselmenge V der Voraussetzungen:

$$\{\{\mathrm{H}, \mathrm{K}, M\}, \{\neg W, \neg \mathrm{K}\}, \{\neg \mathrm{H}, \mathrm{F}\}\}$$

Klauselmenge A der negierten zu beweisenden Aussage:

$$\{\{W\}, \{\neg \mathrm{F}\}, \{\neg M\}\}$$

Es gilt, aus $V \cup A$ die leere Klausel abzuleiten.


{H, K, M}
{¬M}
{¬H, F}
{¬F}
{H, K}
{¬H}
{K}
{¬W, ¬K}
{¬W}
{W}
□

Eigenschaften der Resolution

Satz 3.4. *Eine Klauselmenge* $\mathbb{F}$ *ist unerfüllbar genau dann, wenn die leere Klausel* $\square$ *mit einer endlichen Anzahl von Resolutionsschritten aus* $\mathbb{F}$ *abgeleitet werden kann.*

Bemerkung 3.3. Aus Satz 3.4 folgt die Korrektheit und (Widerlegungs)-Vollständigkeit des Resolutionskalküls:

- Die leere Klausel kann nur dann abgeleitet werden, wenn die ursprüngliche Klauselmenge unerfüllbar ist $\Longrightarrow$ *Korrektheit*
- Das Resolutionskalkül findet für jede unerfüllbare Klauselmenge eine Widerlegung, d.h. die leere Klausel wird abgeleitet $\Longrightarrow$ *Vollständigkeit*

- Im Fall der Aussagenlogik ist es entscheidbar, ob die leere Klausel abgeleitet werden kann.
- Für n Aussagenvariablen gibt es höchstens 4^n verschiedene Klauseln, die aus diesen Aussagenvariablen gebildet werden können.
- Der Prozess der Resolventenbildung ist also endlich, d.h. irgendwann können keine neuen Resolventen mehr gebildet werden.

Lemma 3.5. *Es sei* $\mathbb{F}$ *eine Klauselmenge.* $\mathbb{F}'$ *sei eine Klauselmenge,*

- *die durch sukzessive Resolventenbildung aus* $\mathbb{F}$ *entstanden ist.*
- $\mathbb{F}'$ *enthalte nicht die leere Klausel und*
- *aus* $\mathbb{F}'$ *kann keine neue Resolvente erzeugt werden.*

Dann ist $\mathbb{F}'$ *und somit auch* $\mathbb{F}$ *erfüllbar.*

Fazit zur Aussagenlogik

- Eine Signatur legt die Variablen der Sprache fest.
- Aus den Variablen entsteht durch Festlegung einer Syntax eine Wissensrepräsentationssprache (Menge der Formeln).
- Eine Interpretation gibt den Variablen eine Bedeutung.
- Die Erfüllungsrelation dehnt diese Bedeutung auf alle Formeln aus
- Über die Erfüllungsrelation wird der Begriff der semantischen Folgerung festgelegt.
- Ein Kalkül stellt die Äquivalenz zwischen semantischer Folgerung und syntaktischen Operationen her.

Prädikatenlogik

In der Aussagenlogik ist es nicht möglich, Aussagen über ganze Klassen von Objekten zu machen, so daß Schlußfolgerungen für individuelle Objekte möglich sind.

Es sei gegeben:

> Martin ist ein Informatiker. Peter ist ein Informatiker.
> Jeder Informatiker kann programmieren.

Wir wollen folgern:

> Martin kann programmieren. Peter kann programmieren.

Syntax der Prädikatenlogik

Mit der Prädikatenlogik (1. Stufe) wollen wir Sachverhalte beschreiben, die folgendes enthalten können.

- Objekte, z.B. Personen oder Sachen
- Funktionen auf den Objekten, z.B. Größe, Gewicht, Hochzeitstag
- Eigenschaften von Objekten
- Beziehungen zwischen Objekten
- Aussagen über Objekte, auch quantifizierende

Wie in der Aussagenlogik beschreiben wir zunächst die Syntax der Wissensrepräsentationssprache.

Wir haben in der Prädikatenlogik folgende syntaktischen Elemente:

- Eine *Konstante* repräsentiert ein spezifisches Element aus einer Menge von Objekten. Wir stellen Konstanten durch Bezeichner dar, die mit einem Kleinbuchstaben beginnen.
  Beispiel: $peter, martin$
- Eine *Variable* repräsentiert ein unspezifisches Element aus einer Menge von Objekten. Wir stellen Variablen durch Bezeichner dar, die mit einem Großbuchstaben beginnen.
  Beispiel: $X, Person$
- Eine *Funktion* repräsentiert einen funktionalen Zusammenhang zwischen Objekten. Solche Funktionen haben eine feste Stelligkeit. Wir stellen Funktionen durch kleingeschriebene Bezeichner dar mit den Argumenten in Klammern.
  Beispiel: $alter(martin), f(X, Y, peter)$

- *Prädikate* repräsentieren Eigenschaften oder Beziehungen zwischen Objekten. Solche Prädikate haben eine feste Stelligkeit. Wir stellen Prädikate durch großgeschriebene Bezeichner dar mit den Argumenten in Klammern.

  Beispiel: $\mathrm{Informatiker}(\mathrm{martin}), \mathrm{Programmieren}(\mathrm{X}), \mathrm{Vater}(\mathrm{lars}, \mathrm{peter})$

  Bemerkung: Im Gegensatz zu Funktionen kann einem Prädikatenausdruck ein Wahrheitswert zugeordnet werden.
- Die *logischen Junktoren* wie in der Aussagenlogik: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$
- *Quantoren*, die mit einer Variable verbunden sind, dienen dazu, die Gültigkeit von Prädikaten zu quantifizieren. Hierzu stehen der Allquantor $\forall$ und der Existenzquantor $\exists$ zur Verfügung.

  Beispiel: $\exists \mathrm{X}\ \mathrm{Programmieren}(\mathrm{X})$

Prädikatenlogische Signatur

Definition 3.12. Eine *(PL1-)Signatur* $\Sigma = (\mathtt{Func}, \mathtt{Pred})$ besteht aus

- einer Menge $\mathtt{Func}$ von *Funktionssymbolen* und
- einer Menge $\mathtt{Pred}$ von *Prädikatensymbolen*.

Jedes Symbol $s \in \mathtt{Func} \cup \mathtt{Pred}$ hat eine feste *Stelligkeit* ≥ 0.

Ein Funktionssymbol mit der Stelligkeit 0 heißt *Konstante*.

PL1-Terme

Definition 3.13. Es sei V eine Menge von Variablensymbolen und $\Sigma = (\mathtt{Func}, \mathtt{Pred})$ sei eine PL1-Signatur. Dann ist die Menge $\mathtt{Term}_\Sigma(V)$ der *(PL1-)Terme* wie folgt definiert:

1. Jede Konstante $c \in \mathtt{Func}$ ist ein PL1-Term.
2. Jedes Variablensymbol $X \in V$ ist ein PL1-Term.
3. Ist $f \in \mathtt{Func}$ ein n-stelliges Funktionssymbol ($n \geq 1$) und sind $t_1, \ldots, t_n$ PL1-Terme, so ist auch $f(t_1, \ldots, t_n)$ ein PL1-Term.

Beispiel 3.14. $f(g(X), Y), jahreVerheiratet(klaus, marie)$

PL1-Formeln

Definition 3.14. Es sei V eine Menge von Variablensymbolen und $\Sigma = (\mathtt{Func}, \mathtt{Pred})$ sei eine PL1-Signatur. Dann ist die Menge $\mathtt{Formel}_\Sigma(V)$ der *PL1-Formeln* wie folgt definiert:

1. Ist $p \in \mathtt{Pred}$ ein n-stelliges Prädikatensymbol und sind $t_1, \ldots, t_n$ Terme, so ist

$$p(t_1, \ldots, t_n)$$

   eine *(atomare) PL1-Formel*.
2. Sind F und G PL1-Formeln, dann sind auch

$$\neg F, F \wedge G, F \vee G, F \rightarrow G, F \leftrightarrow G$$

   PL1-Formeln.

3. Ist F eine PL1-Formel und $X \in \mathcal{V}$, dann sind auch

$$\exists X\, \mathsf{F} \text{ und } \forall X\, \mathsf{F}$$

PL1-Formeln.

Beispiel 3.15. Den anfangs dargestellten Sachverhalt könnten wir durch folgende Formeln ausdrücken.

$$\begin{aligned}
&\texttt{Informatiker}(martin)\\
&\texttt{Informatiker}(peter)\\
&\forall X\ \texttt{Informatiker}(X) \rightarrow \texttt{Programmieren}(X)
\end{aligned}$$

Die Frage, ob Martin und Peter programmieren können, würde dann als PL1-Formel lauten:

$$\texttt{Programmieren}(martin) \wedge \texttt{Programmieren}(peter)$$

Bemerkung 3.4.

- In PL1 ist nur eine Quantifizierung über Objekt-Variablen (also 0-stellige Funktionen) zulässig. Somit wäre die Formel

$$\exists \mathrm{X}\, \mathrm{X}(\mathrm{peter})$$

  keine PL1-Formel.
- Wir verlangen zusätzlich, daß alle Variablen quantifiziert werden, d.h. eine Formel der Art

$$\mathrm{p}(\mathrm{X}) \vee \mathrm{q}(\mathrm{Y})$$

  wäre verboten. Formeln, die ausschließlich quantifizierte Variablen enthalten, bezeichnet man als *geschlossene Formeln*.

Prädikatenlogik und PROLOG

In der Programmiersprache PROLOG können (gewisse) PL1-Formeln direkt repräsentiert werden.

Beispiel 3.16. Der bekannte Sachverhalt als PROLOG-Programm:

```
programmieren(X) :- informatiker(X).

informatiker( martin ).
informatiker( peter ).
```

An solch ein PROLOG-Programm kann man eine Anfrage stellen:

```
?- programmieren( peter ),programmieren( martin ).
Yes
```

Prädikatenlogische Interpretation

Bisher haben wir wieder nur Syntax. Wir müssen nun die Funktions- und Prädikatensymbole mit einer Bedeutung belegen.

Definition 3.15. Es sei $\Sigma = (\mathtt{Func}, \mathtt{Pred})$ eine PL1-Signatur. Eine *Σ-Interpretation* $I = (U_I, S_I)$ besteht aus:

- einer nichtleeren Menge U_I, der *Trägermenge (Grundbereich, Universum)*,
- sowie einer Abbildung S_I, die jedem n-stelligen Funktionssymbol $f \in \mathtt{Func}$ eine n-stellige Funktion $S_I(f) : U_I^n \to U_I$ zuordnet und
- jedem n-stelligen Prädikatensymbol $p \in \mathtt{Pred}$ eine n-stellige Relation $S_I(p) \subseteq U_I^n$ zuordnet.

Termauswertung

Es sind noch nicht alle sprachlichen Symbole der PL mit einer Bedeutung belegt, es fehlen noch die Variablen.

Definition 3.16. Es sei $I = (U_I, S_I)$ eine Σ-Interpretation und V eine Menge von Variablensymbolen. Dann ist eine Variablenbelegung α eine Funktion $\alpha : V \to U_I$.

Definition 3.17. Gegeben sein ein Term $t \in \mathtt{Term}_\Sigma(V)$, eine Σ-Interpretation $I = (U_I, S_I)$ und eine Variablenbelegung $\alpha : V \to U_I$.

Die *Termauswertung* von t in I unter α ist die wie folgt definierte Funktion $[\![\ ]\!]_{I,\alpha} : \mathtt{Term}_\Sigma(V) \to U_I$:

$$\begin{aligned} [\![X]\!]_{I,\alpha} &= \alpha(X) \text{ für } X \in V \\ [\![f(t_1, \ldots, t_n)]\!]_{I,\alpha} &= S_I(f)([\![t_1]\!]_{I,\alpha}, \ldots, [\![t_n]\!]_{I,\alpha}) \end{aligned}$$

PL1-Semantik

Definition 3.18. Es sei $I = (U_I, S_I)$ eine Σ-Interpretation, V eine Menge von Variablensymbolen und $\alpha : V \longrightarrow U_I$ eine Variablenbelegung.

$\alpha_{X/a} : V \longrightarrow U_I$ bezeichne die Modifikation von α an der Stelle X zu a, d.h:

$$\alpha_{X/a}(t) = \begin{cases} \alpha(t) & \text{für } t \neq X \\ a & \text{für } t = X \end{cases}$$

Dann ist der Wahrheitswert einer Formel $F \in \texttt{Formel}_\Sigma(V)$ in I unter α (geschrieben $[\![F]\!]_{I,\alpha}$) wie folgt definiert:

- Für eine atomare Formel $p(t_1, \ldots, t_n)$ gilt:

$$[\![p(t_1, \ldots, t_n)]\!]_{I,\alpha} \text{ ist wahr} :\Leftrightarrow ([\![t_1]\!]_{I,\alpha}, \ldots, [\![t_n]\!]_{I,\alpha}) \in S_I(p)$$

- Für $F \wedge G, F \vee G, F \rightarrow G$ und $\neg F$ gelten die selben Regeln wie in der Aussagenlogik.
- Für eine Formel $\forall X\, F$ gilt:

$$[\![\forall X\, F]\!]_{I,\alpha} \text{ ist wahr} :\Leftrightarrow \text{für jedes } a \in U_I \text{ gilt:} [\![F]\!]_{I,\alpha_{X/a}} \text{ ist wahr}$$

- Für eine Formel $\exists X\, F$ gilt:

$$[\![\exists X\, F]\!]_{I,\alpha} \text{ ist wahr} :\Leftrightarrow \text{es gibt ein } a \in U_I \text{ mit:} [\![F]\!]_{I,\alpha_{X/a}} \text{ ist wahr}$$

Bemerkung 3.5.

- Gemäß Bemerkung 3.4 betrachten wir nur geschlossene Formeln. Dies sind Formeln, in denen alle Variablen gebunden sind.
- Der Wahrheitswert einer geschlossenen Formel ist aber unabhängig von einer Variablenbelegung α.
- Für geschlossene Formeln können wir deshalb auch einfach $[\![\mathrm{F}]\!]_{\mathrm{I}}$ schreiben.
- Die Begriffe *Modell* und *semantische Folgerung* werden analog zur Aussagenlogik definiert.

Beispiel 3.17. Durch die Mengen

$$\begin{aligned} \mathrm{Programmieren} &= \{\mathrm{peter}, \mathrm{martin}\} \text{ und} \\ \mathrm{Informatiker} &= \{\mathrm{peter}, \mathrm{martin}\} \end{aligned}$$

ist ein Modell für unser Beispiel gegeben. Es ist sogar das “kleinste Modell” für diese Formelmenge. Solche Modelle nennt man *Herbrand-Modell*.

Die Interpretation

$$\begin{aligned} \mathrm{Programmieren} &= \{\mathrm{peter}, \mathrm{martin}\} \text{ und} \\ \mathrm{Informatiker} &= \{\mathrm{peter}, \mathrm{martin}, \mathrm{klaus}\} \end{aligned}$$

ist dagegen kein Modell für diese Formelmenge.

Äquivalenzen für PL1-Formeln

Definition 3.19. Zwei PL1-Formeln F und G heißen *semantisch äquivalent* gdw. für alle Σ-Interpretationen I gilt:

$$[\![\mathrm{F}]\!]_{\mathrm{I}} = [\![\mathrm{G}]\!]_{\mathrm{I}}$$

Lemma 3.6.

$$\begin{array}{rcl}
\neg\forall \mathrm{X}\mathrm{F} & \equiv & \exists \mathrm{X}\neg\mathrm{F} \\
\neg\exists \mathrm{X}\mathrm{F} & \equiv & \forall \mathrm{X}\neg\mathrm{F} \\
(\forall \mathrm{X}\mathrm{F}) \wedge (\forall \mathrm{X}\mathrm{G}) & \equiv & \forall \mathrm{X}(\mathrm{F} \wedge \mathrm{G}) \\
(\exists \mathrm{X}\mathrm{F}) \vee (\exists \mathrm{X}\mathrm{G}) & \equiv & \exists \mathrm{X}(\mathrm{F} \vee \mathrm{G}) \\
\forall \mathrm{X}\forall \mathrm{Y}\mathrm{F} & \equiv & \forall \mathrm{Y}\forall \mathrm{X}\mathrm{F} \\
\exists \mathrm{X}\exists \mathrm{Y}\mathrm{F} & \equiv & \exists \mathrm{Y}\exists \mathrm{X}\mathrm{F} \\
\forall \mathrm{X}\mathrm{F} & \equiv & \forall \mathrm{Y}\mathrm{F}[\mathrm{X}/\mathrm{Y}] \\
\exists \mathrm{X}\mathrm{F} & \equiv & \exists \mathrm{Y}\mathrm{F}[\mathrm{X}/\mathrm{Y}]
\end{array}$$

Normalformen

Definition 3.20. Eine Formel F, in der alle Quantoren außen stehen, heißt *Pränexform*.

Eine Pränexform, die als Junktor nur noch Konjunktion, Disjunktion und Negation enthält, wobei die Negation nur unmittelbar vor Atomen auftritt, heißt *vereinigungstechnische Normalform (VNF)*.

Mit folgenden Schritten kann jede PL1-Formel in eine äquivalente Formel in VNF überführt werden:

1. Umbennung von Variablen derart, daß keine Variable sowohl frei als auch gebunden auftritt und hinter allen vorkommenden Quantoren stehen verschiedene Variablen.
2. Beseitigung der Junktoren $\rightarrow$ und $\leftrightarrow$.

3. Die Negation wird ganz nach innen gezogen, so daß sie nur noch unmittelbar vor Atomen auftritt.
4. Die Quantoren werden ganz nach außen geschoben.

Skolemisierung

Skolemisierung dient der Elimination von Existenzquantoren. Wir betrachten eine Pränexform:

$$\forall X_1 \ldots \forall X_k \exists Y \, F(\ldots, Y, \ldots)$$

Zum Ersatz von $\exists Y$ wählen wir ein neues Funktionszeichen f der Stelligkeit k und führen folgende Operation aus:

1. Streiche $\exists Y$ aus der Pränexform.
2. Ersetze in der verbleibenden Formel Y an allen Stellen durch $f(X_1, \ldots, X_k)$.

Wir erhalten somit

$$\forall X_1 \ldots \forall X_k \, F(\ldots, f(X_1, \ldots, X_k), \ldots)$$

Die Funktion f heißt *Skolemfunktion* bzw. für $k = 0$ *Skolemkonstante*.

Normalformen (2)

5. Alle Existenzquantoren werden durch Skolemisierung entfernt.
6. Jetzt können alle Allquantoren entfernt werden.
7. Mit Hilfe der de Morganschen Regeln können disjunktive Normalform DNF bzw. konjunktive Normalform KNF analog zur Aussagenlogik erzeugt werden.

Ebenfalls analog zur Aussagenlogik kann eine PL1-Formel in KNF in Klauselform dargestellt werden.

Substitution

Für die maschinelle Inferenz mit PL wird noch ein Mechanismus zur Instanziierung von Variablen benötigt.

Definition 3.21. Es sei $\mathcal{X} \subseteq V$ eine endliche Teilmenge von Variablensymbolen von V. Dann ist eine Abbildung $\sigma : \mathcal{X} \longrightarrow \mathtt{Term}_\Sigma(V)$ eine *Substitution*.

Durch

$$\sigma(X) = \begin{cases} \sigma(X) & \text{für } X \in \mathcal{X} \\ X & \text{für } X \in V \setminus \mathcal{X} \end{cases}$$

wird der Definitionsbereich von σ zunächst auf V ausgedehnt.

Durch $\sigma(c) = c$ für Konstanten und die rekursive Anwendung von σ in der Form

$$\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$$

erhält man eine Abbildung $\sigma : \mathtt{Term}_\Sigma(V) \longrightarrow \mathtt{Term}_\Sigma(V)$.

Eine Substitution σ geben wir in der Form $\{X_1/t_1, \ldots, X_k/t_k\}$ an.

Unifikation

Die Unifikation ist eine Substitution, bei der Terme gleichgemacht werden.

Definition 3.22. Eine Substitution σ heißt *Unifikator* der Terme s und t gdw. $\sigma(s) = \sigma(t)$ gilt. s und t sind dann *unifizierbar*.

Beispiel 3.18. Sind X, Y, Z Variablensymbole und a, b, c, Konstanten, so sind die Terme $f(X, b)$ und $f(a, c)$ nicht unifizierbar.

Die Terme $f(X, b)$ und $f(a, b)$ sind unifizierbar mit $\sigma = \{X/a\}$.

Die Substitutionen $\sigma = \{X/b, Y/a, Z/g(a, a)\}$ und $\mu = \{X/b, Z/g(a, Y)\}$ sind Unifikatoren für die Terme $f(X, g(a, Y))$ und $f(b, Z)$.

Allgemeinster Unifikator

Definition 3.23. Eine Sustitution σ heißt *allgemeinster Unifikator* für die Terme s und t gdw.

- σ ist Unifikator von s und t und
- Für jeden Unifikator λ von s und t existiert eine Substitution τ, so daß $\lambda = \tau \circ \sigma$ gilt.

Bemerkung 3.6. Es gibt einen Algorithmus, der für zwei Terme s und t entscheidet, ob sie unifizierbar sind und gegebenenfalls einen allgemeinsten Unifikator berechnet.

Gegeben seien zwei Terme s und t. Die Berechung des allgemeinsten Unifikators geschieht wie folgt: Man startet mit $\sigma = \{\}$ als allgemeinsten Unifikator und wendet sukzessive die folgenden Regeln an:

(a) Sind s und t Konstanten, so sind sie unifizierbar gdw. s gleich t ist.

(b) Ist s eine Variable und t eine Konstante, so sind s und t unifizierbar. Man erweitere hierzu σ um $\{s/t\}$.

(c) Ist s eine Variable und t ein Term der Form $f(t_1, \ldots, t_n)$, so sind sie unifizierbar gdw. s nicht in t vorkommt. Man erweitere hierzu σ um $\{s/t\}$.

(d) Sind s und t zusammengesetzte Terme, so sind sie unifizierbar gdw. $s = f(s_1, \ldots, s_n)$, $t = f(t_1, \ldots, t_n)$ und wenn jeweils s_i und t_i unifizierbar sind. Man erweitere σ um die Substitutionen, die sich aus der Unifikation der s_i und t_i ergeben.

Beispiel 3.19. Es seien X, Y, Z Variablensymbole.

Sind die Terme $f(X, h(Y), Y)$ und $f(g(Z), Z, a)$ unifizierbar?

Regel	Ungelöste Unifikationen	σ
	$f(X, h(Y), Y) ? f(g(Z), Z, a)$	$\{\}$
(d)	$X ? g(Z), h(Y) ? Z, Y ? a$	$\{\}$
(c)	$h(Y) ? Z, Y ? a$	$\{X/g(Z)\}$
(c)	$Y ? a$	$\{X/g(h(Y)), Z/h(Y)\}$
(b)		$\{X/g(h(a)), Z/h(a), Y/a\}$

Resolution in PL1

Bemerkung 3.7. Die Begriffe *Klausel*, *Klauselform* und *Literal* werden in PL1 analog zur Aussagenlogik definiert.

Bemerkung 3.8. Wir dehnen die Unifikation auf atomare PL1-Formeln aus.

Zwei atomare Formeln $P(s_1, \ldots, s_n)$ und $P(t_1, \ldots, t_n)$ sind unifizierbar gdw. s_i und t_i für $i = 1, \ldots, n$ unifizierbar sind.

Definition 3.24. Es seien K_1, K_2 PL1-Klauseln. Die Klausel R heißt *PL1-Resolvente* von K_1 und K_2 gdw. folgendes gilt:

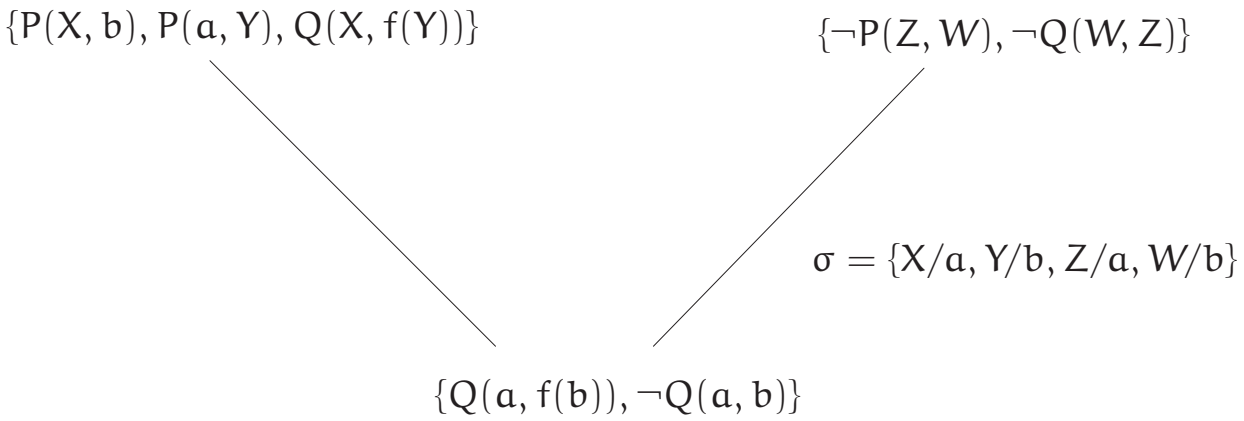
(a) K_1 und K_2 haben keine gemeinsamen Variablen.

(b) Es gibt positive Literale $A_1, \ldots, A_k \in K_1$, ein negatives Literal $\neg A \in K_2$ und einen allgemeinsten Unifikator σ von $A, A_1, \ldots A_k$.

(c) R hat die folgende Form:

$$\sigma((K_1 \setminus \{A_1, \ldots, A_k\}) \cup (K_2 \setminus \{\neg A\}))$$

Beispiel 3.20. Darstellung der Resolution für PL1:



Beispiel 3.21. Überführung einer PL1-Formel in Klauselform.

$$\forall X((\forall Y P(X,Y)) \rightarrow \neg(\forall Y Q(X,Y) \rightarrow R(X,Y)))$$

Für den zweiten an Y gebundenen Allquantor ersetzen wir Y durch Z:

$$\forall X((\forall Y P(X,Y)) \rightarrow \neg(\forall Z Q(X,Z) \rightarrow R(X,Z)))$$

Der Junktor $\rightarrow$ wird eliminiert:

$$\forall X(\neg(\forall Y P(X,Y)) \vee \neg(\forall Z \neg Q(X,Z) \vee R(X,Z)))$$

Die Negationen werden ganz nach innen gezogen:

$$\forall X(\exists Y \neg P(X,Y)) \vee (\exists Z Q(X,Z) \wedge \neg R(X,Z)))$$

Quantoren nach aussen ziehen:

$$\forall X \exists Y \exists Z(\neg P(X,Y) \vee (Q(X,Z) \wedge \neg R(X,Z)))$$

Eliminierung von $\exists Y$ durch Skolemfunktion f:

$$\forall X \exists Z(\neg P(X, f(X)) \vee (Q(X, Z) \wedge \neg R(X, Z)))$$

Eliminierung von $\exists Z$ durch Skolemfunktion g:

$$\forall X(\neg P(X, f(X)) \vee (Q(X, g(X)) \wedge \neg R(X, g(X)))$$

Jetzt kann man den Quantor weglassen:

$$\neg P(X, f(X)) \vee (Q(X, g(X)) \wedge \neg R(X, g(X))$$

De Morgan liefert:

$$(\neg P(X, f(X)) \vee Q(X, g(X))) \wedge (\neg P(X, f(X)) \vee \neg R(X, g(X)))$$

Darstellung als zwei Klauseln:

$$\{\neg P(X, f(X)), Q(X, g(X))\} \text{ und } \{\neg P(X, f(X)), \neg R(X, g(X))\}$$

Umbenennung der Variablen so, daß eine Variable in höchstens einer Klausel auftritt:

$$\{\neg P(X, f(X)), Q(X, g(X))\} \text{ und } \{\neg P(Y, f(Y)), \neg R(Y, g(Y))\}$$

Beispiel 3.22. Resolution in PL1:

1. Es gibt einen Patienten, der alle Ärzte mag:

$$\exists X Patient(X) \wedge \forall Y(Arzt(Y) \rightarrow Mag(X, Y))$$

2. Kein Patient mag einen Quacksalber:

$$\forall X Patient(X) \rightarrow \forall Y(Quaksalber(Y) \rightarrow \neg Mag(X, Y))$$

Daraus wollen wir schließen, daß ein Arzt kein Quaksalber ist.

$$\forall Y(Arzt(Y) \rightarrow \neg Quaksalber(Y))$$

Wir formen 1. um:

$$\begin{aligned} & \exists X Patient(X) \wedge \forall Y(Arzt(Y) \rightarrow Mag(X, Y)) \\ \equiv \quad & \exists X Patient(X) \wedge \forall Y(\neg Arzt(Y) \vee Mag(X, Y)) \\ \equiv \quad & \exists X \forall Y(Patient(X) \wedge (\neg Arzt(Y) \vee Mag(X, Y))) \end{aligned}$$

Skolemisierung liefert:

$$\begin{aligned} & \forall Y(Patient(a) \wedge (\neg Arzt(Y) \vee Mag(a, Y))) \\ \equiv \quad & Patient(a) \wedge \forall Y(\neg Arzt(Y) \vee Mag(a, Y)) \end{aligned}$$

Daraus ergeben sich die Klauseln:

$$K_1 : \{Patient(a)\} \text{ und } K_2 : \{\neg Arzt(Y), Mag(a, Y)\}$$

Wir formen 2. um:

$$\begin{aligned} & \forall X Patient(X) \rightarrow \forall Y(Quaksalber(Y) \rightarrow \neg Mag(X, Y)) \\ \equiv \quad & \forall X \neg Patient(X) \vee \forall Y(\neg Quaksalber(Y) \vee \neg Mag(X, Y)) \\ \equiv \quad & \forall X \forall Y \neg Patient(X) \vee \neg Quaksalber(Y) \vee \neg Mag(X, Y) \end{aligned}$$

Daraus ergibt sich mit Umbenennung die Klausel:

$$K_3 : \{\neg Patient(X), \neg Quaksalber(Z), \neg Mag(X, Z)\}$$

Negation der Hypothese:

$$\begin{aligned} & \neg\forall Y(Arzt(Y) \rightarrow \neg Quaksalber(Y)) \\ \equiv \quad & \neg\forall Y(\neg Arzt(Y) \vee \neg Quaksalber(Y)) \\ \equiv \quad & \exists Y\neg(\neg Arzt(Y) \vee \neg Quaksalber(Y)) \\ \equiv \quad & \exists Y(Arzt(Y) \wedge Quaksalber(Y)) \end{aligned}$$

Skolemisierung liefert die beiden Klauseln:

$$K_4 : \{Arzt(b)\} \text{ und } K_5 : \{Quaksalber(b)\}$$

Ableitung:

Klauseln	Resolvente	Unifikator	Nummer
K_2, K_4	$\{Mag(a, b)\}$	$\sigma = \{Y/b\}$	K_6
K_6, K_3	$\{\neg Patient(a), \neg Quaksalber(b)\}$	$\sigma = \{X/a, Z/b\}$	K_7
K_5, K_7	$\{\neg Patient(a)\}$		K_8
K_1, K_8	$\square$		q.e.d.

4. Regelbasierte Systeme

- Eine weitverbreitete Form der Wissensrepräsentation ist die Formulierung von Regeln.
- Regeln sind formalisierte Konditionalsätze der Form:

  Wenn (`if`) F dann (`then`) G.

- Bedeutung:

Wenn	F wahr ist,
dann	schließe, daß auch G wahr ist.

- Dabei sind in einer logischen Interpretation F und G Aussagen oder PL1-Formeln.

Beispiele für Regeln:

Wenn der aktuelle Buchstabe eines Wortes *q* ist und das Wort keine Abkürzung ist,
dann ist der nächste Buchstabe ein *u*.

Wenn die Motortemperatur zu hoch ist,
dann prüfe die Kühlflüssigkeit.

- Die Formel im “Wenn”-Teil einer Regel wird als *Prämisse* oder *Antendenz* bezeichnet.
- Bezeichnung innerhalb regelbasiertes Systeme: left-hand side (LHS)
- Die Formel im “Dann”-Teil heißt *Konklusion* oder *Konsequenz*.
- Bezeichnung innerhalb regelbasiertes Systeme: right-hand side (RHS)
- Ist die Prämisse einer Regel erfüllt, so sagt man, daß *die Regel feuert*.

Die beiden Regeln des vorangegangenen Beispiels weisen Unterschiede auf:

- Die erste Regel entspricht einer logischen Implikation.
- Die Konklusion der zweiten Regel ist dagegen eine Aktion, die ausgeführt wird, wenn die Prämisse erfüllt ist.

Regeln der zweiten Art werden auch *Produktionsregeln* oder *Aktionsregeln* genannt.

Solche Regeln werden gerne in Produktionssystemen zur Steuerung eingesetzt.

Vorteile der Wissensdarstellung in Form von Regeln:

- Regeln stellen einen guten Kompromiß zwischen Verständlichkeit der Wissensdarstellung und formalen Ansprüchen dar.
- Konditionalsätze werden von Menschen seit langer Zeit benutzt, um Handlungsanweisungen oder Prognosen auszudrücken.
- Regeln sind dem Benutzer daher hinreichend vertraut.
- Ein großer Teil von Expertenwissen läßt sich üblicherweise in Regelform ausdrücken.

☞ Regeln haben eine große Nähe zum menschlichen Denken.

Syntaktischer Aufbau und Umformung von Regeln

- Man legt i.d.R. Wert auf eine möglichst einfache syntaktische Form der Regeln.
- Dies ermöglicht eine effizientere Abarbeitung der Regeln und
- es verbessert die Übersichtlichkeit der Wirkung einzelner Regeln.

Üblicherweise stellt man folgende Bedingungen an die Form der Regeln:

$\vee$ (oder) darf nicht in der Prämisse einer Regel auftreten.
Die Konklusion einer Regel sollte nur aus einem Literal bestehen.

- Regeln, die diesen Bedingungen nicht genügen, müssen umgeformt werden.
- Hierbei kommt die klassische Logik mittels der Äquivalenz zum Einsatz.

Beispiel 4.1.

> Wenn es morgen regnet oder schneit, gehen wir ins Kino oder bleiben zu Hause.

Diese Regel läßt sich äquivalent durch die folgenden vier Regeln ausdücken:

1. Wenn es morgen regnet und wir nicht ins Kino gehen, dann bleiben wir zu Hause.
2. Wenn es morgen regnet und wir nicht zu Hause bleiben, dann gehen wir ins Kino.
3. Wenn es morgen schneit und wir nicht ins Kino gehen, dann bleiben wir zu Hause.
4. Wenn es morgen schneit und wir nicht zu Hause bleiben, dann gehen wir ins Kino.

Nun erfüllt jede Regel die gestellten Anforderungen. Die Zahl der Regeln hat sich allerdings erhöht.

Durch folgende Umformungen können wir jede Regel in eine Menge von Regeln der gewünschten Form umwandeln:

1. Bringe die Prämisse in DNF und die Konklusion in KNF.
2. Ersetze die Regel

$$\texttt{if } K_1 \vee \ldots \vee K_n \texttt{ then } D_1 \wedge D_m$$

   durch $n \cdot m$ Regeln: $\texttt{if } K_i \texttt{ then } D_j$
3. Ersetze die Regel

$$\texttt{if } K \texttt{ then } L_1 \vee \ldots \vee L_k$$

   durch k Regeln

$$\texttt{if} K \wedge (\bigwedge_{i \neq i_0} \neg L_i) \texttt{ then } L_{i_0}$$

Bemerkungen zur Negation (I):

- Fakten stellen Wissen über konkrete Sachverhalte dar. Die Fakten werden in den Prämissen der Regeln verwendet.
- Prinzipiell ist es möglich, sowohl positive als auch negative Fakten darzustellen.
- Wie negative Fakten zu interpretieren sind, hängt von der Form der verwendeten Negation ab.
- Geht man davon aus, daß das gesamte Wissen über den Anwendungsbereich in der Datenbasis enthalten ist, kann man ein Faktum als falsch ansehen, wenn es nicht in der Datenbasis enthalten ist.
- Man nennt diese grundlegende Annahme *closed-world-assumption*.

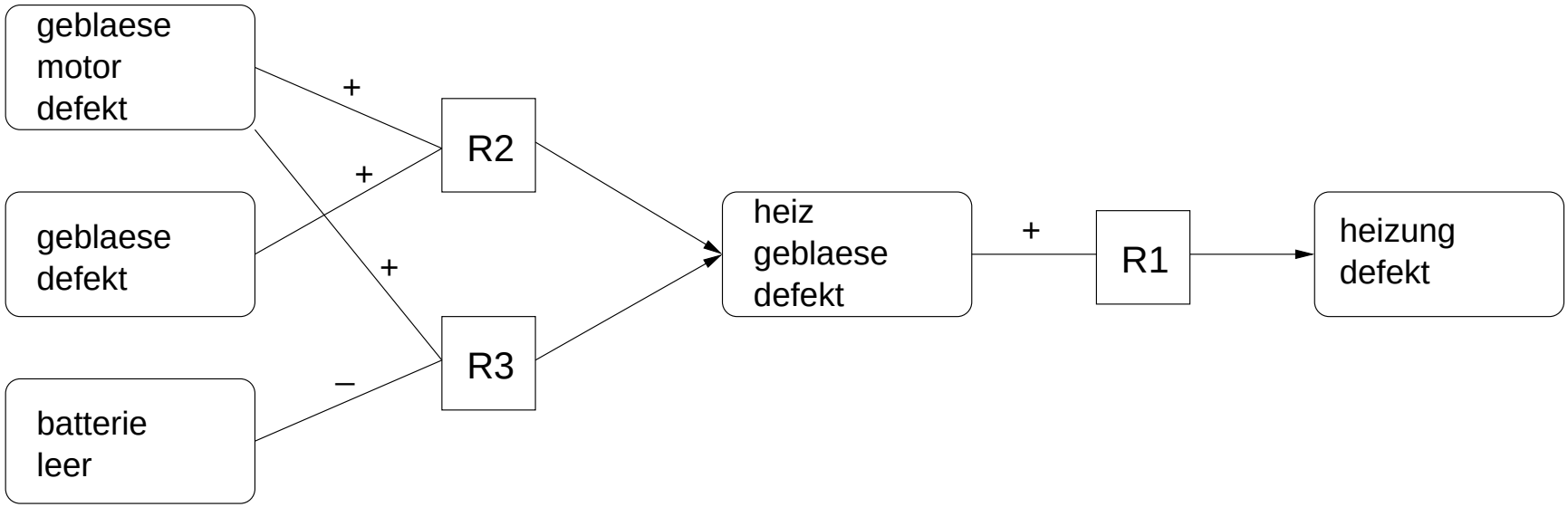
- Ein negatives Faktum f wird in der Form $-f$ geschrieben,
- ein positives in der Form $+f$.
- Die Fakten können noch von weiteren Argumenten abhängen.
- Die Regeln werden benannt und in der Form

$$R : b_1 \wedge \ldots \wedge b_n \rightarrow h$$

geschrieben. Beispiel:

$$\begin{aligned} R_1 &: +rot(ampel) \rightarrow stop \\ R_2 &: +gelb(ampel) \rightarrow stop \\ R_3 &: +gruen(ampel) \rightarrow stop \end{aligned}$$

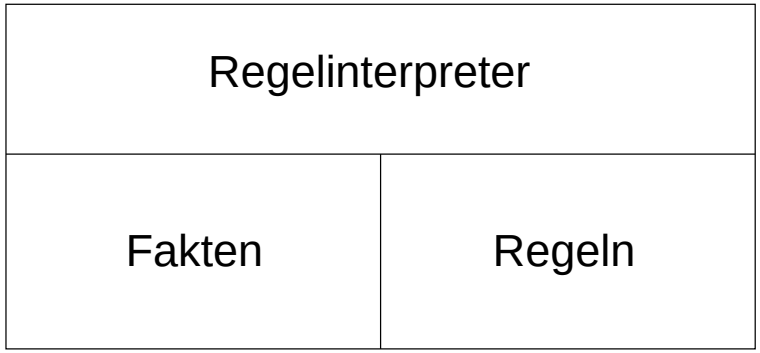
Regeln können auch in graphischer Form als *Regelnetze* repräsentiert werden.

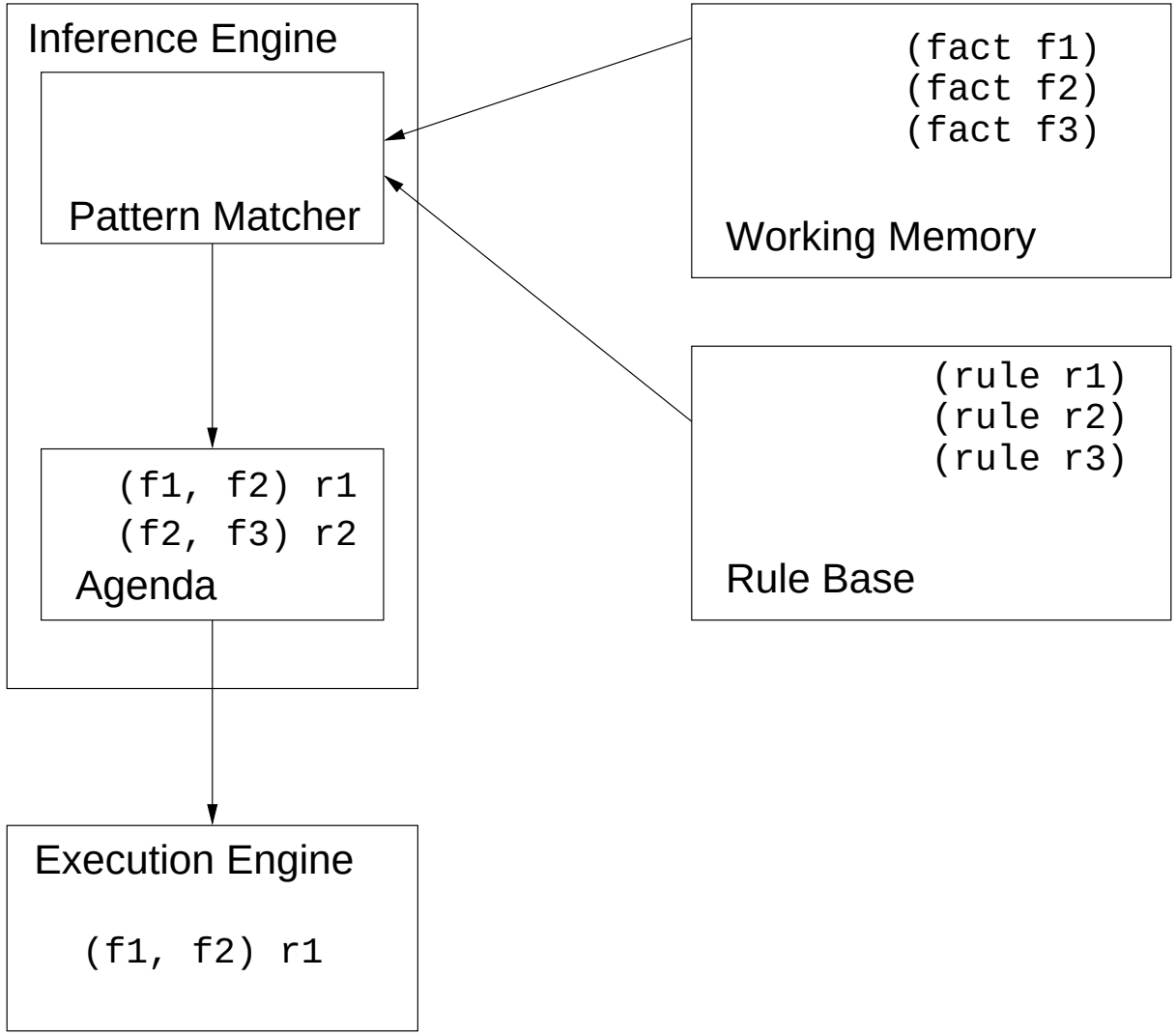


$$R1 : +heizgeblaesedefekt \rightarrow heizungdefekt$$
$$R2 : +geblaesemotordefekt \wedge geblaesedefekt \rightarrow heizgeblaesedefekt$$
$$R3 : +geblaesemotordefekt \wedge -batterieleer \rightarrow heizgeblaesedefekt$$

Architektur regelbasierter Systeme

- Ein *regelbasiertes System (rule-based system, rule engine)* besteht aus Fakten, Regeln und einem Regelinterpretierer (inference engine).
- Die Fakten und Regeln bilden zusammen die *Wissensbasis*.
- Der Teil der Wissensbasis, der die Fakten enthält, wird als *Faktenbasis* bezeichnet.
- Ein sehr einfaches regelbasiertes System ist Prolog.




Inference Engine
Pattern Matcher
(f1, f2) r1
(f2, f3) r2
Agenda
(fact f1)
(fact f2)
(fact f3)
Working Memory
(rule r1)
(rule r2)
(rule r3)
Rule Base
Execution Engine
(f1, f2) r1

Inference Engine

Die Inference Engine ist die wichtigste Komponente eines regelbasierten Systems.

Sie kontrolliert die Anwendung der Regeln auf die Fakten.

1. Die Inference Engine bestimmt mit Hilfe des Pattern Matchers, welche Regeln anwendbar sind.
   Die Menge der anwendbaren Regeln heißt *Konfliktmenge (conflict set)*.
2. Die Konfliktmenge wird geordnet, die entstehende geordnete Liste heißt *Agenda*.
   Die Bestimmung der Ordnung und damit die Auswahl einer Regeln nennen wir *Konfliktlösung*.
3. Die erste Regel der Agenda wird ausgeführt (feuert). Danach wiederholt sich der Prozeß.

Rule Base

- Die *Regelbasis (rule base)* enthält die Regeln des Systems.
- Die Definition der Regeln erfolgt mit Hilfe einer Regelsprache.
- Üblicherweise wird ein *Regel-Compiler* eingesetzt, um die Regeln in eine kompaktere Form zu überführen.
- Durch die Compilierung erfolgt der Aufbau eines Rete-Netzwerks, hierzu später mehr.

Working Memory

- Das sogenannte *working memory* enthält die aktuell gültigen Fakten.
- Üblicherweise werden Indexe eingesetzt, um schnell ermitteln zu können, welche Fakten für welche Regelprämissen relevant sind.
- Das *working memory* ist i.d.R. ähnlich einer relationalen Datenbank strukturiert.
- Schnittstellen zur Repräsentation von Fakten durch Objekte (z.B. in Java) sind üblich (z.B. Jess, ILOG JRules)

Pattern Matcher

- Zweck des pattern matchers ist es, die Menge der anwendbaren Regeln, die Konfliktmenge, zu ermitteln.
- Hierzu müssen Kombinationen von Fakten und Regeln betrachtet werden.
- Hierbei hilft das Rete-Netzwerk.
- Nach einer Änderung des Working Memory wird die Konfliktmenge nicht neu berechnet.
- Stattdessen wird ermittelt, welche Auswirkungen die Änderungen auf die Konfliktmenge haben.

Agenda

- Nach der Ermittlung der Konfliktmenge muß entschieden werden, welche der anwendbaren Regeln ausgeführt werden soll.
- Hier gibt es eine Reihe von Strategien, um Konflikte aufzulösen.
- Prioritäten
- dynamisches Ein- und Ausschalten von Regelmodulen
- spezifischere Regeln zuerst
- zuletzt aktivierte Regeln zuerst

Execution Engine

- Ausführung der ausgewählten Regel
- Änderungen am Working Memory möglich
- Programmierung von Seiteneffekte
- Integration mit einer Programmiersprache/-umgebung

Inferenz in Regelsystemen

Die grundlegende Inferenzregel in einem regelbasierten System ist der Modus Ponens:

if A **then** B	(Regel)
A wahr	(Faktum)
B wahr	(Schlußfolgerung)

- Der Modus Ponens benutzt jeweils eine Regel zur Inferenz.
- Die eigentliche Leistung eines regelbasierten Systems zeigt sich aber in seiner Fähigkeit, komplexe Informationen durch die Verkettung von Regeln zu verarbeiten.
- Dabei spielt die Richtung des Inferenzprozesses eine große Rolle.
- Man unterscheidet dabei zwischen *Vorwärtsverkettung (forward chaining)* bzw. *datengetriebene Inferenz (data driven inference)* und
- *Rückwärtsverkettung (backward chaining)* bzw. *zielorientierte Inferenz (goal-oriented inference)*.

Beispiel 4.2. Die Faktenbasis enthalte die Objekte

$$A, B, C, D, E, F, G, H, I, J, K, L, M,$$

jeweils mit einem zugehörigen Wahrheitswert.

Die Regelbasis enthalte die folgenden Regeln:

R1 : **if** $A \wedge B$ **then** H
R2 : **if** $C \vee D$ **then** I
R3 : **if** $E \wedge F \wedge G$ **then** J
R4 : **if** $H \vee I$ **then** K
R5 : **if** $I \wedge J$ **then** L
R6 : **if** $K \wedge L$ **then** M

Wir werden dieses Beispiel später für die Veranschaulichung von Vorwärts- und Rückwärtsverkettung benutzen.

Es gibt einen wesentlichen Unterschied zwischen regelbasierten Systemen und der klassischen Logik bei der Inferenz:

☞ Regeln sind *gerichtet*, d.h. sie können nur dann angewandt werden (feuern), wenn ihre Prämisse erfüllt ist.

Die beiden Regeln

$$\textbf{if } A \textbf{ then } B \qquad \text{und} \qquad \textbf{if } \neg B \textbf{ then } \neg A$$

sind zwar logisch äquivalent, als Regeln in einer Regelbasis können Sie aber zu unterschiedlichen Ableitungen führen.

Wird nur der Modus Ponens als Inferenzregel verwendet, feuert bei Vorliegen des Faktums A nur die erste Regel, bei Nichtvorliegen des Faktums B nur die zweite Regel.

Legt man auf die Erhaltung der logischen Äquivalenz Wert, gibt es die folgenden Möglichkeiten:

- Man nimmt beide Regeln in die Wissensbasis auf oder
- man implementiert zusätzlich den Modus Tollens.

if A **then** B	(Regel)
B falsch	(Faktum)
A falsch	(Schlußfolgerung)

Rückwärtsverkettung

- Bei der *Rückwärtsverkettung* geht man von einem Zielobjekt aus, über dessen Zustand der Benutzer Informationen wünscht.
- Das System durchsucht dann die Regelbasis nach geeigneten Regeln, die das Zielobjekt in der Konklusion enthalten.
- Die Objekte der Prämissen werden zu Zwischenzielen.
- Die Regeln werden also vom Folgerungsteil zum Bedingungsteil hin ausgewertet.
- Dies entspricht der Vorgehensweise eines Prolog-Interpreters.

Beispiel 4.3. Wir greifen Beispiel 4.2 auf. Gegeben sei die Faktenmenge $\mathcal{F}$ durch:

$$\mathrm{H} = \mathrm{true}, \mathrm{C} = \mathrm{true}, \mathrm{E} = \mathrm{true}, \mathrm{F} = \mathrm{true}, \mathrm{G} = \mathrm{true}$$

Das Zielobjekt sei M.

Durch die Rückwärtsverkettung werden in dieser Reihenfolge die Werte

$$\mathrm{K} = \mathrm{true}, \mathrm{I} = \mathrm{true}, \mathrm{J} = \mathrm{true}, \mathrm{L} = \mathrm{true}, \mathrm{M} = \mathrm{true}$$

abgeleitet.

Steuerung der Regelinterpretation

Die Reihenfolge, in der die Regeln bei der Rückwärtsverkettung verarbeitet werden, hat Einfluß auf

- die Performanz des Systems und
- die Anzahl der Fragen, die an den Benutzer gestellt werden.

Bisher wird die Reihenfolge durch die Reihenfolge der Regeln in der Wissensbasis bestimmt.

Dieses Schema ist im allgemeinen zu unflexibel.

- Wir gehen davon aus, daß gewisse Regeln mit größerer Wahrscheinlichkeit als andere Regeln zutreffen.
- In diesem Fall sollte man die Regeln mit höherer Wahrscheinlichkeit zuerst abarbeiten.
- Gründe für unterschiedliche Wahrscheinlichkeiten:
  - Es werden Fakten abgeleitet, die andere Hypothesen weniger wahrscheinlich machen.
    Beispiel: Ist das Geschlecht einer Person bekannt, so können gewisse nicht zutreffende geschlechtsspezifische Krankheiten als sehr unwahrscheinlich angesehen werden.
  - Im Verlauf längerer Erfahrungen mit einem Regelsystem kann sich herausstellen, daß bestimmte Regeln generell häufiger erfüllt sind als andere.

Steuerungswissen, Metawissen

- Wissen dieser Form heißt *Steuerungswissen* oder *strategisches Wissen*. Es dient dazu, die Inferenz zu steuern. Steuerungswissen ist eine Form von *Metawissen*.
- Es unterscheidet sich damit von dem *Objektwissen*, welches das Wissen über die Objekte des Anwendungsbereichs darstellt.
- Wir gehen davon aus, daß die Auswahl einer Regel unabhängig von der Reihenfolge der Regeln in der Regelbasis getroffen werden kann.
- Eine Situation, in der mehrere Regeln anwendbar sind, nennt man *Regelkonflikt*.
- Die Menge der anwendbaren Regeln ist die *Konfliktmenge*.
- Die Auswahl einer Regeln nennt man *Konfliktlösung*.

Konfliktlösung mit statischen Prioritäten

Es kann sehr schwierig sein, die “richtige” Regel auszuwählen und von dieser Regeln kann viel abhängen.

☞ Die Konflikte sind der natürliche Preis der Annehmlichkeiten des deklarativen Programmierens.

- Jede Regel erhält eine Priorität.
- Aus der Konfliktmenge wird die Regel mit der höchsten Priorität ausgewählt. Dies entspricht einer partiellen Ordnung der Regelmenge.
- Ist die Regel nicht eindeutig bestimmt, wird der Konflikt wiederum anders aufgelöst (z.B. durch die Reihenfolge).

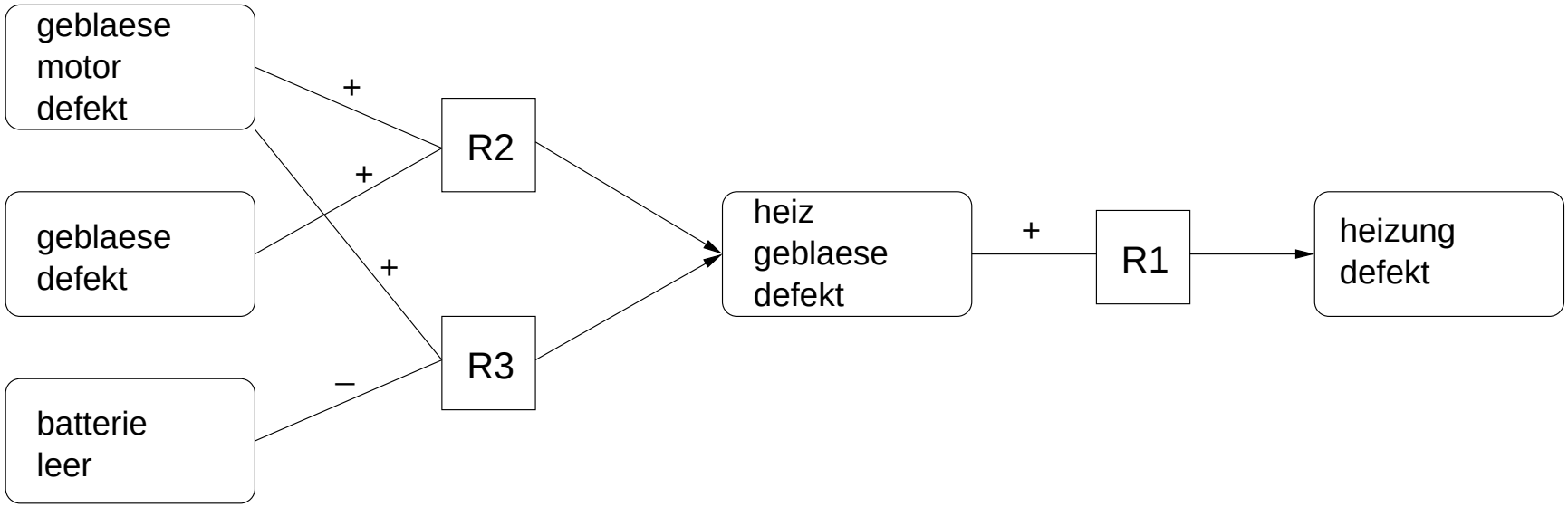
Konfliktlösung mit dynamischen Prioritäten

- Wir führen zusätzlich *heuristische Regeln* ein. Heuristische Regeln repräsentieren das Steuerungswissen.
- Die Prämisse einer heuristischen Regel hat die gleiche Form wie die einer normalen Regel.
- Die Konklusion einer heuristischen Regel ist eine Aktion, die eine Änderung einer Regelpriorität bewirkt.
- Schreibweise:

$$\mathfrak{h} : \mathfrak{a}_1 \wedge \ldots \wedge \mathfrak{a}_n \rightarrow \mathfrak{add}(R, \delta)$$

- Semantik solch einer heuristischen Regel: Ist die Prämisse $\mathfrak{a}_1 \wedge \ldots \wedge \mathfrak{a}_n$ erfüllt, so addiere δ zu der Priorität der Regel R.

Regelnetzwerk



$R1 : +heizgeblaesedefekt \rightarrow heizungdefekt$

$R2 : +geblaesemotordefekt \wedge geblaesedefekt \rightarrow heizgeblaesedefekt$

$R3 : +geblaesemotordefekt \wedge -batterieleer \rightarrow heizgeblaesedefekt$

Beispiel 4.4. Wir betrachten diese Regeln. Die Faktenbasis sei leer. Eine mögliche Ableitung ohne Prioritäten lautet:

Hypothesen	Regeln	Faktenbasis	Eingabe
Heizung defekt?	R_1	-	
Heizgebläse defekt?	R_2, R_3	-	
Gebläsemotor defekt?	-	-	Ja
Gebläse defekt?	-	Gebläsemotor defekt	Nein
Heizgebläse defekt?	R_3	Gebläsemotor defekt	
Gebläsemotor defekt?	-	Gebläsemotor defekt	
Batterie leer?	-	Gebläsemotor defekt	Nein

Wir nehmen nun eine heuristische Regel hinzu:

$$h_1 : +licht \rightarrow add(R_3, 5)$$

Verlauf Tafel ✎

Vorwärtsverkettung

Rückwärtsverkettung ist nur unter den folgenden Bedingungen sinnvoll:

- Es muß möglich sein, realistische Hypothesen zielgerichtet zu generieren.
- Bei der Rückwärtsverkettung wird der Benutzer aufgefordert, Beobachtungen zu machen, Tests durchzuführen oder Werte einzugeben. Sind im voraus alle Fakten bekannt, ist diese Vorgehensweise weniger sinnvoll.

- Kernstück der *Vorwärtsverkettung* ist die wiederholte Auswahl einer Regel, deren Prämisse vollständig erfüllt ist.
- Der Aktionsteil dieser Regel wird ausgeführt, die Faktenbasis wird entsprechend angepaßt.
- Anschließend wird eine neue Regel ausgeführt.

Terminierung:

☞ Die Rückwärtsverkettung endet, wenn die Hypothese bewiesen wurde oder alle Beweismöglichkeiten ausgeschöpft sind.

☞ Die Vorwärtsverkettung endet, wenn keine neuen Fakten mehr abgeleitet werden können oder ein Zielfaktum generiert wurde.

- Man kann sich ein vorwärtsverkettetes System wie ein Spiel vorstellen.
- Die Regeln entsprechen den Spielregeln.
- Die Fakten beschreiben die aktuelle Spielstellung.
- Die Regeln geben nur einen formalen Rahmen für den Ablauf vor. Sie geben keine Auskunft darüber, wie das Spiel zu gewinnen ist.
- Nichtsdestotrotz können die Regeln natürlich gewisse Gewinnstrategien implementieren.

Beispiel 4.5. Das Spiel besteht aus zwei Regeln:

R1 : Wenn sich im Korb zwei schwarze Chips befinden, dann nimm sie heraus und lege einen Weißen hinein.

R2: Wenn sich im Korb ein schwarzer und ein weißer Chip befinden, dann nimm den Weißen heraus.

Das Spiel ist beendet, wenn kein Zug mehr möglich ist.

Ziel ist es, am Ende so wenig Chips wie möglich im Korb zu behalten.

Eine Spielstellung beschreiben wir durch ein Faktum $\mathtt{status}(s, w)$, wobei s die Anzahl der schwarzen und w die Anzahl der weißen Chips angibt.

Die Regeln lauten für das Chipspiel formal:

$$R_1 \quad : \quad +status(s, w) \wedge s \geq 2 \rightarrow status(s - 2, w + 1)$$

$$R_2 \quad : \quad +status(s, w) \wedge s \geq 1 \wedge w \geq 1 \rightarrow status(s, w - 1)$$

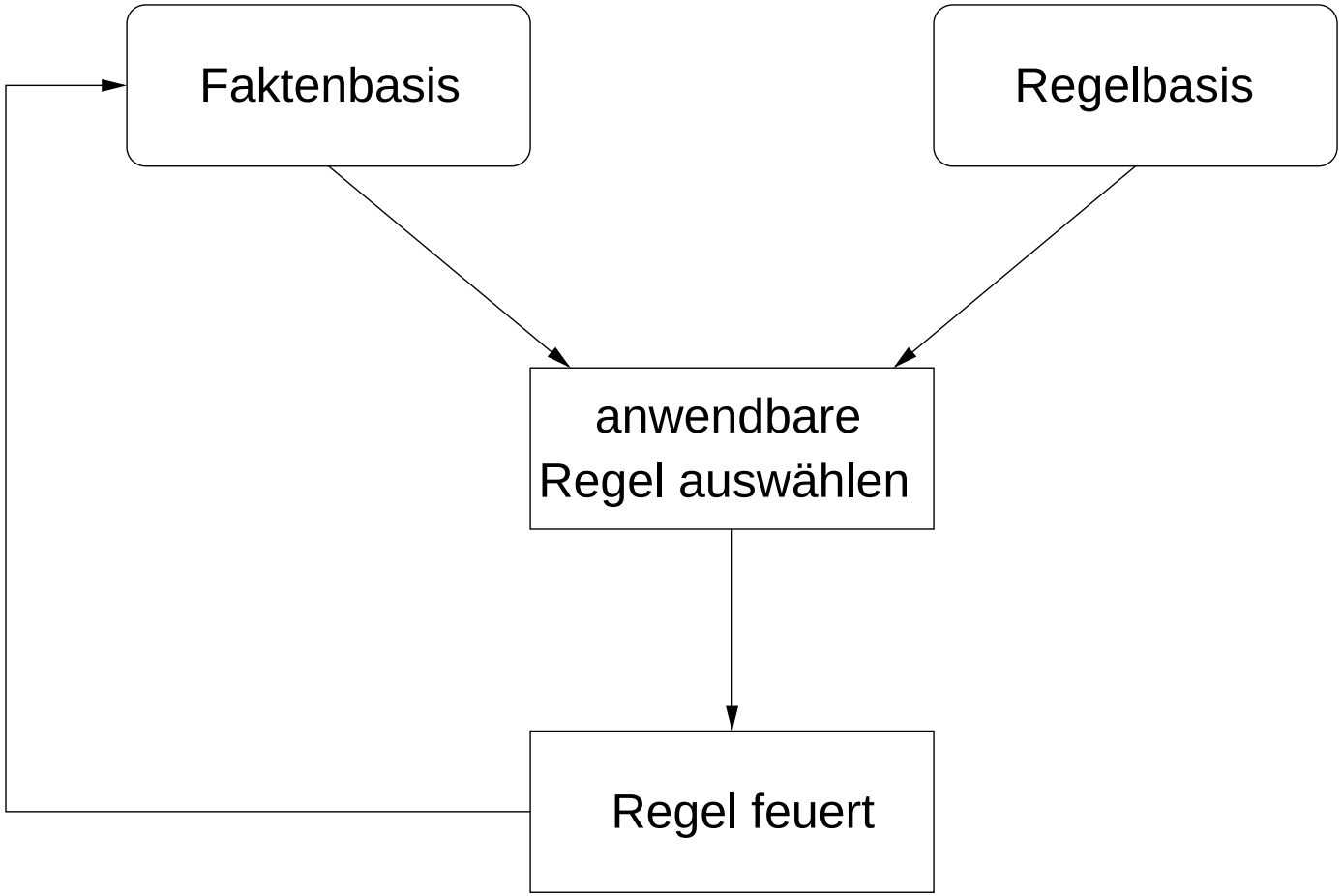
Als Anfangsstellung könnte beispielsweise

$$\{status(2, 2)\}$$

vorliegen.

- Die Regelinterpretation verläuft dann in einer Schleife.
- In jedem Zyklus wird eine anwendbare Regel ausgewählt.

Einfaches Schema für die Vorwärtsverkettung:



Definition 4.1.

- Es sei b eine variablenfreie Prämisse.
  Ist $b = +a$, so ist b *erfüllt* gdw. a in der Faktenbasis ist.
  Ist $b = -a$, so ist b *erfüllt* gdw. a nicht in der Faktenbasis enthalten ist.
  Ist b eine prozedurale Prämisse, so ist b *erfüllt* gdw. b zu `true` ausgewertet wird.
- Sei $r = b_1 \wedge \ldots \wedge b_n \rightarrow h$ eine Regel und σ eine Substitution. Wir definieren

$$\sigma(r) := \sigma(b_1) \wedge \ldots \wedge \sigma(b_n) \rightarrow \sigma(h)$$

  $\sigma(r)$ heißt eine *Instanz* von r.
- Eine Regelinstanz $\sigma(r)$ heißt *anwendbar* in einer Faktenbasis gdw. alle Prämissen von $\sigma(r)$ in der Faktenbasis erfüllt sind.

Bemerkungen:

- Ist eine anwendbare Regelinstanz ausgewählt, so kann diese Regel feuern, d.h. die Konklusion wird als neues Faktum in die Faktenbasis eingetragen.

- Im Verlauf der Interpretation werden so neue Fakten erzeugt, die alten werden aber nicht gelöscht.
- Würde man die Fakten löschen, könnte man keine Alternativen im Suchbaum ausprobieren.

Schreibweise: Eine Regelinstanz $\sigma(r)$ schreiben wir auch in der Form $r(x_1, \ldots, x_n)$, wobei die x_1 bis x_n die in der Regel verwendeten Variablenwerte sind.

Beispiel 4.6. Ein möglicher Verlauf des Chipsspiel:

Iteration	Konfliktmenge	neues Faktum
1	$R_1(2,2)*, R_2(2,2)$	$status(0,3)$
2	$R_1(2,2), R_2(2,2)*$	$status(2,1)$
3	$R_1(2,2), R_2(2,2), R_1(2,1), R_2(2,1)*$	$status(2,0)$
4	$R_1(2,2), R_2(2,2), R_1(2,1), R_2(2,1), R_1(2,0)*$	$status(0,1)$

Die jeweils ausgewählte Regel ist mit * markiert.

Der RETE-Algorithmus

Der *Rete-Algorithmus* wurde 1979 von Charles Forgy an der Carnegie Mellon University im Rahmen der Entwicklung von OPS5 (Expertensystem-Shell) entwickelt.

> Forgy, Charles, *Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem*, Artificial Intelligence, 19, pp. 17-37, 1982.

☞ Der Rete-Algorithmus bildet die Basis für viele neue Entwicklungen im Bereich von Expertensystemen (Jess, Drools, ILog JRules, IBM CommonRules)

Ziel des Rete-Algorithmus:

- Für eine Menge von Mustern und sich eine ändernde Objektmenge auf effiziente Weise die Muster zu bestimmen, die durch die aktuelle Objektmenge erfüllt werden.

In der Anwendung für Regelsysteme sind die Muster die Produktionsregeln.

Der Rete-Algorithmus erreicht die Effizienzsteigerung durch:

- ☞ Einschränkung der zu überprüfenden Bedingungen
  Regelprämissen enthalten teilweise identische Teile. Die mit diesen Teilen verbundenen Überprüfungen werden nur einmal durchgeführt.
- ☞ Einschränkung der zu überprüfenden Datenbasiseinträge
  Im nächsten Zyklus werden nur die Änderungen an der Faktenbasis daraufhin überprüft, ob sich eine Änderung an der Konfliktmenge ergibt.

Rete-Netzwerk

- “rete” heißt im Lateinischen Netz.
- Der Rete-Algorithmus erzeugt aus einer Menge von Regelprämissen ein Entscheidungsnetzwerk in Form eines Datenflussgraphen.
- Der Datenflussgraph besteht im wesentlichen aus zwei Typen von Knoten, die die eigentlichen Operationen repräsentieren.

Bezeichnungen im Umfeld von Rete:

- Die Regelprämissen werden auch als *LHS (left hand side)* bezeichnet.
- Die Konklusionen bzw. Aktionen einer Regel heißen *RHS (right hand side)*.
- Die Faktenbasis bezeichnet man als *working memory*.

Typen von Knoten im Rete-Netzwerk

α-Knoten: repräsentieren Selektionsbedingungen, die sich auf einzelne Objekte (Elemente, Tupel) des working memory beziehen.

β-Knoten: repräsentieren Bedingungen, die über Junktoren (und Variablenbindungen) miteinander verknüpft sind.

Beispiel 4.7.
Wenn X Elternteil von Y ist und X weiblich ist
Dann ist X Mutter von Y.

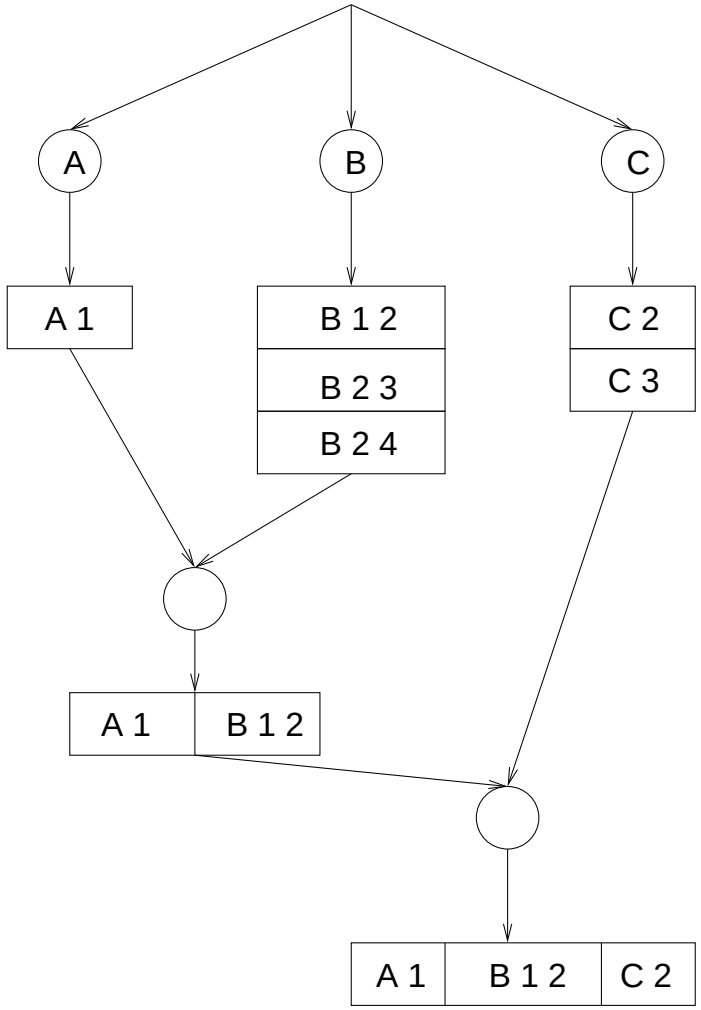
α-Knoten: X ist weiblich
α-Knoten: X ist Elternteil von Y
β-Knoten: Das “und” in Verbindung mit der gebundenen Variablen X

Beispiel 4.8. Wir betrachten die Regel:

$$A(X) \wedge B(X,Y) \wedge C(Y) \rightarrow \mathit{Aktion}$$

α-Knoten: drei Stück für die Einzelbedingungen $A(X), B(X,Y), C(Y)$
β-Knoten: zwei Stück für die Verbindung von $A(X)$ mit $B(X,Y)$ und $B(X,Y)$ mit $C(Y)$.

Working Memory:
(A 1)
(B 1 2)
(B 2 3)
(B 2 4)
(C 2)
(C 3)
(D 2)
(D 4)



Selektion
α −Knoten

Verbund
β −Knoten

β −Knoten
Verbund

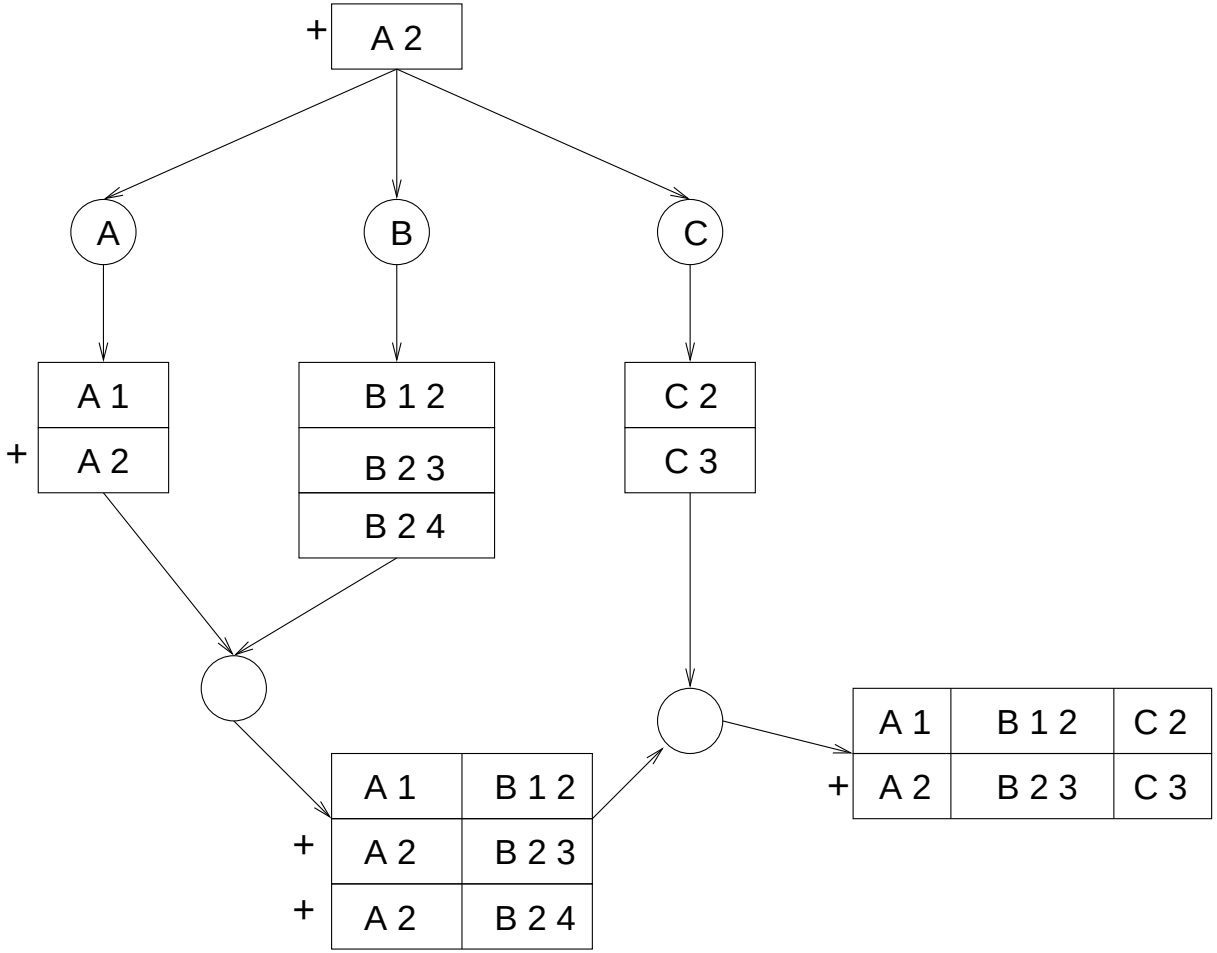
Bemerkungen:

- Zu den einzelnen Knoten werden die Ergebnisse abgespeichert, d.h.:
  - zu einem α-Knoten die Objekte/Tupel, die die Selektionsbedingung erfüllen und
  - zu einem β-Knoten die Tupelkombinationen, die die Verbundbedingung erfüllen.
- Wenn die beiden “Eingänge” eines β-Knotens nicht über eine gemeinsame Variablen verbunden sind, liefert der Knoten das Kreuzprodukt der Eingangstupelmengen als Ergebnis.
- Das Ergebnis des letzten Knotens einer Regel stellt die Menge der Regelinstanzen dar, die diese Regel erfüllen.

Inkrementelle Änderungen in einem Rete-Netzwerk

- Änderungen am Working Memory werden durch das Netzwerk propagiert.
- Wird in das Working Memory ein Tupel eingefügt, wandert dieses als Plus-Tupel (+) durch das Rete-Netz, wobei die Knotenspeicher aktualisiert werden.
- Analoges gilt für Tupel, die aus dem Working Memory gelöscht werden (–).
- Änderungen an der Ergebnismenge des letzen Knotens stellen Änderungen an der Konfliktmenge dar.

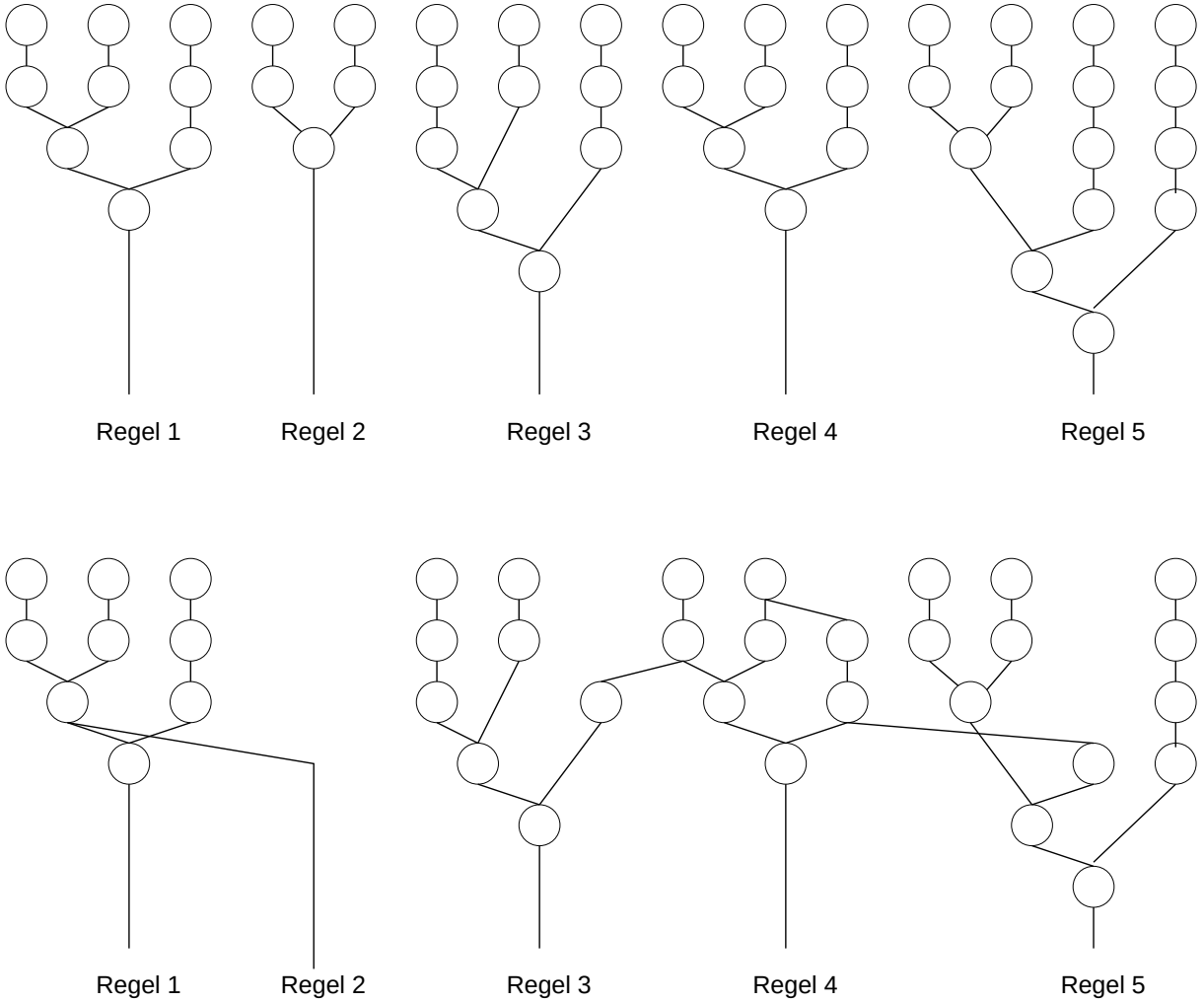
Beispiel 4.9. Das Tupel (A 2) wird zusätzlich in das Working Memory aufgenommen:



Auswertung mehrerer Regeln

- Wenn man die Rete-Netze naiv auf Regelmengen erweitert, würde jede Regel in ein separates Netz übersetzt.
- Das Gesamtnetz für die Regelbasis würde dann aus einer Menge von Einzelnetzen bestehen.
- Der Rete-Algorithmus ist intelligenter: Er verschmilzt gleiche Teile der Einzelnetze.
- Genauer: Er verschmilzt gleiche Anfangsstücke von den Einzelnetzen.

Prinzip:

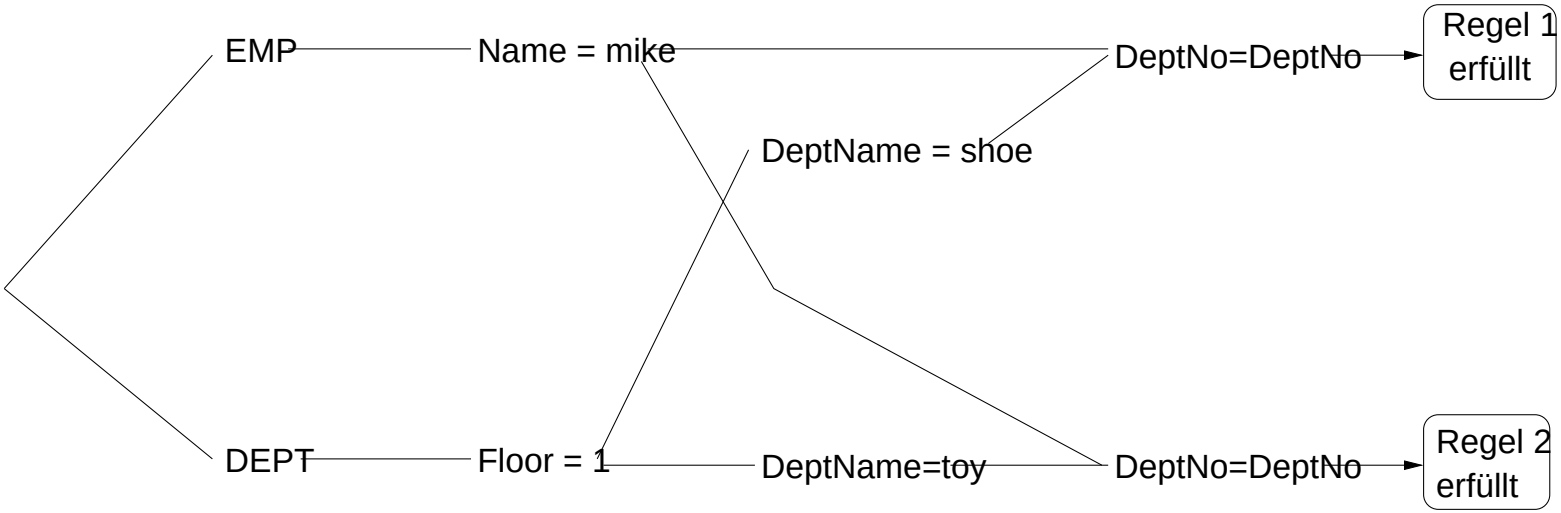

Regel 1
Regel 2
Regel 3
Regel 4
Regel 5
Regel 1
Regel 2
Regel 3
Regel 4
Regel 5

Beispiel 4.10. Gegeben seien die Relationen:
$emp(EmpName, Salary, DeptNo)$,
$dept(DeptNo, DeptName, Floor, Mgr)$ sowie die Regeln:

$$R1 : emp(mike, Salary, DeptNo) \wedge dept(DeptNo, shoe, 1, Mgr) \rightarrow Aktion1$$

$$R2 : emp(mike, Salary, DeptNo) \wedge dept(DeptNo, toy, 1, Mgr) \rightarrow Aktion2$$

Rete-Netzwerk:



Beispiel 4.11.

Wenn der Status der Wettervorhersage *aktuell* ist und
die Quelle der Wettervorhersage die *Rundfunknachrichten* sind und
Regen vorhergesagt wird und
dunkle Wolken beobachtet werden,
dann gebe eine Regenwarnung aus.

Wenn der Status der Wettervorhersage *aktuell* ist und
die Quelle der Wettervorhersage eine *Bauernregel* ist und
Sonnenschein vorhergesagt wird und
der heutige Wochentag ein *Freitag* ist und
der heutige Monat *Juni, Juli, August oder September* ist,
dann gebe eine Ausflugsempfehlung aus.

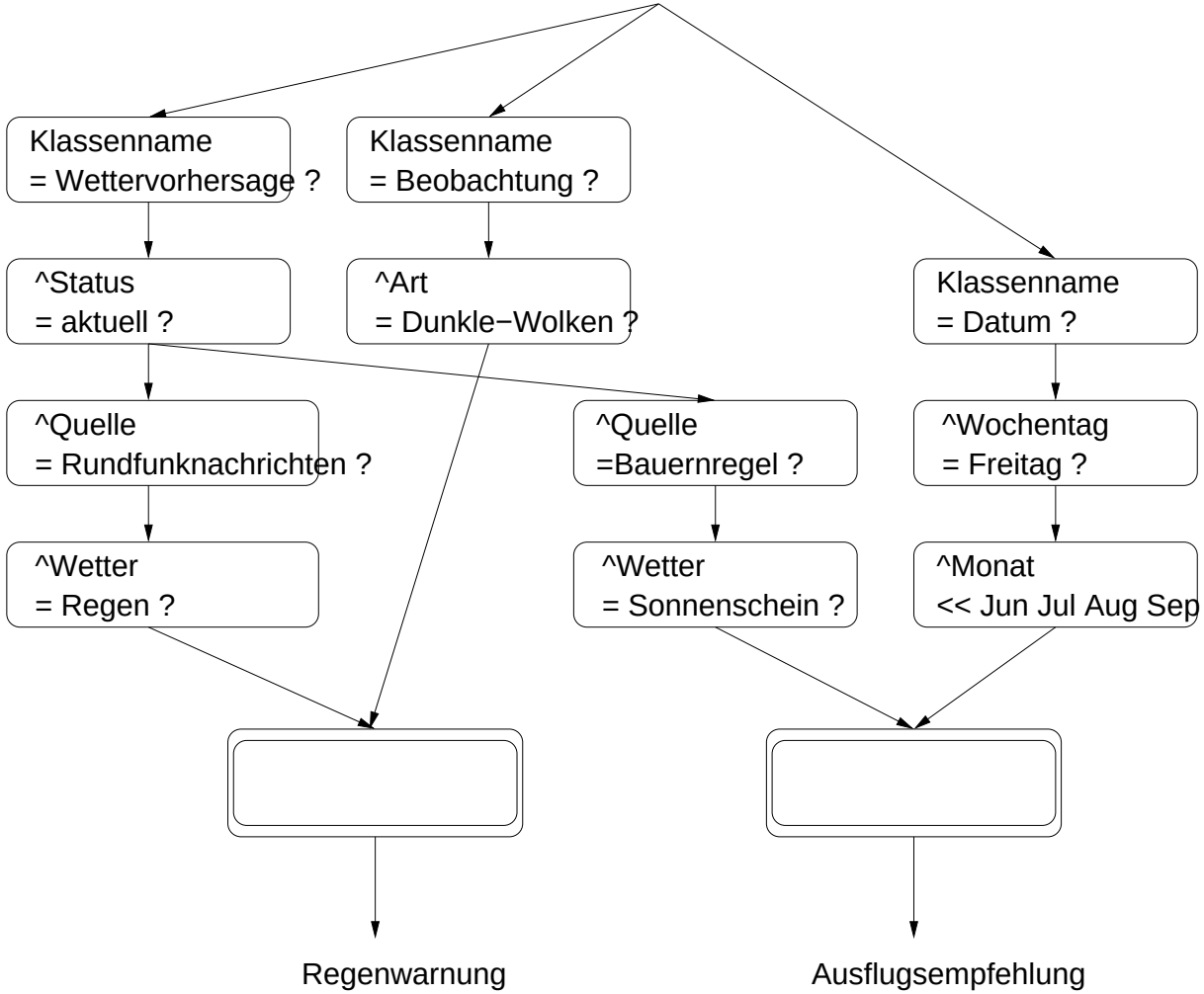
Strukturierung in Klassen (Relationenschema) und Attribute:

Klasse	**Attribute**
Wettervorhersage	Status, Quelle, Wetter
Beobachtung	Art
Datum	Wochentag, Monat

Darstellung der Regeln in OPS5 Syntax:

```
(P Regenwarnung
   (Wettervorhersage
      ^Status aktuell
      ^Quelle Rundfunknachrichten
      ^Wetter Regen)
   (Beobachtung
      ^Art    dunkle-Wolken)
--> (WRITE |Es wird Regen geben.| (CRLF)
           |Ich empfehle, einen Schirm mitzunehmen.|)
)
```

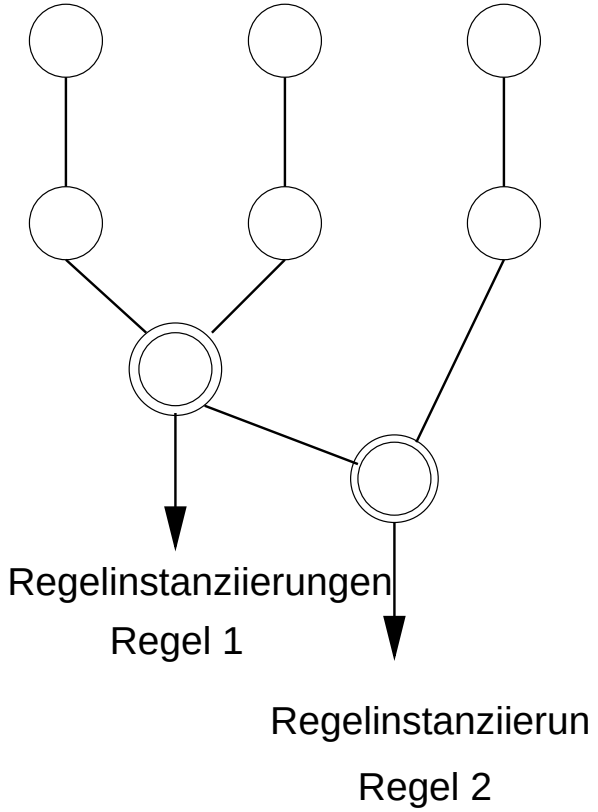
```
(P Ausflugsempfehlung
   (Wettervorhersage
      ^Status aktuell
      ^Quelle Bauernregel
      ^Wetter Sonnenschein)
   (Datum
      ^Wochentag Freitag
      ^Monat << Juni Juli August September >>)
--> (WRITE |Das Wetter wird prima.| (CRLF)
           |Ich empfehle, dieses Wochenende ans Meer zu fahren.|)
)
```


Klassenname
= Wettervorhersage ?
Klassenname
= Beobachtung ?
^Status
= aktuell ?
^Art
= Dunkle−Wolken ?
Klassenname
= Datum ?
^Quelle
= Rundfunknachrichten ?
^Quelle
=Bauernregel ?
^Wochentag
= Freitag ?
^Wetter
= Regen ?
^Wetter
= Sonnenschein ?
^Monat
<< Jun Jul Aug Sep
Regenwarnung
Ausflugsempfehlung

Es kann vorkommen, daß die Regelinstanziierungen am Ausgang eines Knotens gleichzeitig Input für eine anderen Knoten sind.

```
(Regel1
   (Bedingung1 ... )
   (Bedingung2 ... )
--> (Aktionsteil ... )
)

(Regel2
   (Bedingung1 ... )
   (Bedingung2 ... )
   (Bedingung3 ... )
--> (Aktionsteil ... )
)
```



Optimierungen für RETE

Zur Verkleinerung der Zwischenergebnisse an Knoten sollte man die Regeln so spezifisch wie möglich formulieren.

Beispiel 4.12. Regel, die zu Personen den Ort des Arbeitsplatzes ausgibt:

```
(P Ort_des Arbeitsplatzes
   (Person
      ^Name              <Name>
      ^beschaeftigt_bei <Arbeitgeber>)
   (Firma
      ^Name              <Arbeitgeber>
      ^Ort               <Ort>)
--> (WRITE <Name> |arbeitet in| <Ort>))
```

Wenn wir zusätzlich wissen, daß Personen unter 16 und über 65 sowie Schüler und Studenten keinen Arbeitgeber haben, so können wir durch eine spezifischere Regel die Eingabe für den β-Knoten verkleinern:

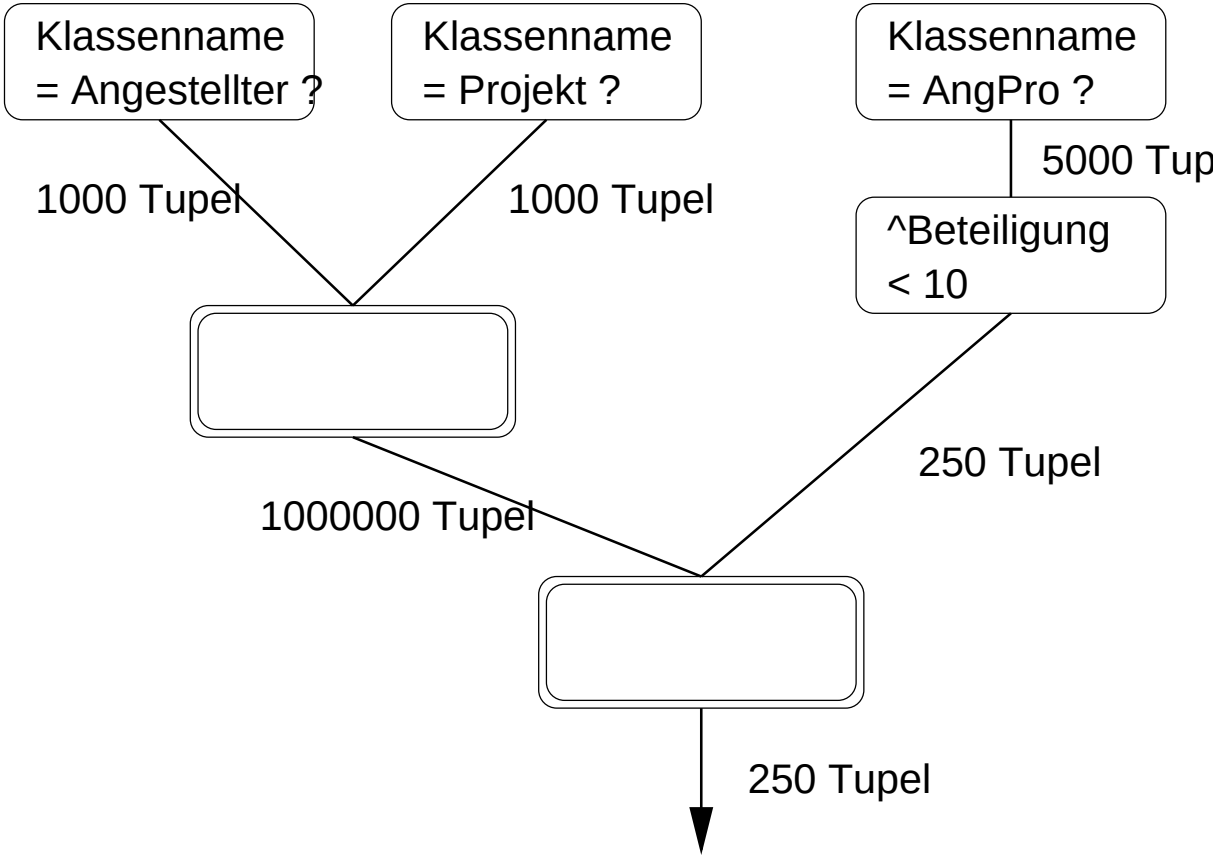
```
(P Ort_des Arbeitsplatzes
   (Person
      ^Name             <Name>
      ^Alter            { >= 16 <= 65 }
      ^Beruf            { <> Schueler <> Student }
      ^beschaeftigt_bei <Arbeitgeber>)
   (Firma
      ^Name             <Arbeitgeber>
      ^Ort              <Ort>)
--> (WRITE <Name> |arbeitet in| <Ort)
)
```

Zwischenergebnisse können sehr groß werden, wenn Bedingungen, die nicht über eine Variable verbunden sind, an einem β-Knoten zusammengeführt werden.

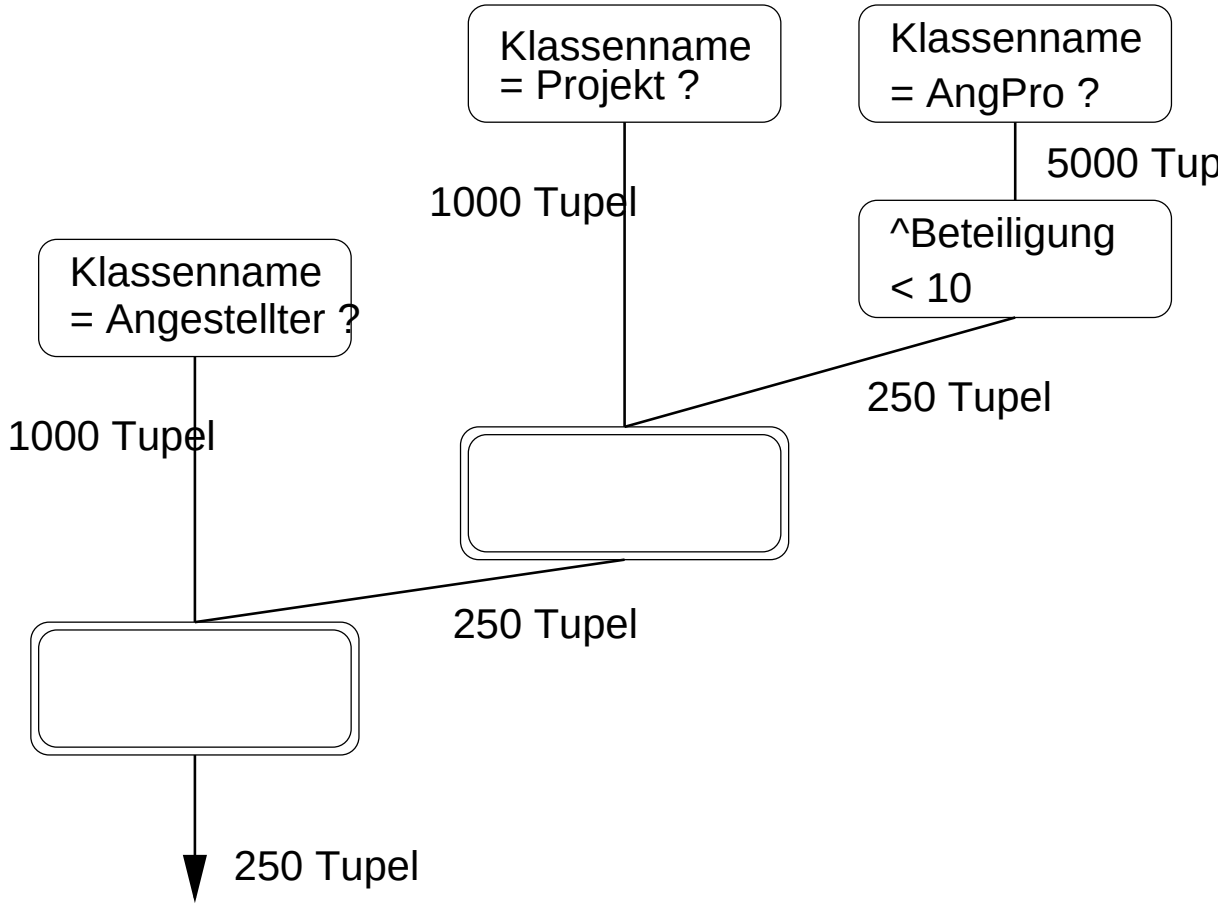
Beispiel 4.13. Die Namen und Projektbezeichnungen zu Angestellten, die weniger als 10% ihrer Arbeitszeit an einem Projekt beteiligt sind, sollen ermittelt werden:

```
(P EliminiereKleineProjektbeteiligung
   (Angestellter
      ^PersNr       <persNr>
      ^Name         <name>)
   (Projekt
      ^ProjNr       <projNr>
      ^Bezeichnung  <bezeichnung>)
   (AngPro
      ^PersNr       <persNr>
      ^ProjNr       <projNr>
      ^Beteiligung  < 10)
--> (WRITE |Angestellter| <name> (CRLF)
           |sollte aus Projekt| (CRLF)
           <bezeichnung> (CRLF)
           |abgezogen werden.|)
)
```

Ein ungünstiges RETE-Netzwerk für diese Regel:



Besser ist das folgende RETE-Netzwerk:



Integration von Regeln in Anwendungen

Bei ILOG JRules besteht eine Regel aus den Komponenten:

- Name
- Priorität
- Menge von Bedingugnen
- Menge von Aktionen

Syntax der Regelsprache:

```
rule ruleName { [priority = value; ]
   [ packet = packetName; ]
   when { conditions ... }
   then { [actions ... ] }
};
```

Beispiel 4.14.

```
rule OccupancyStatus {
   packet = mortage;
   when {
      ?loanApp: Loan( firstHome == true; lienType == FIRSTMORTGAGE );
      Property( occupancyStatus != PRINCIPALRESIDENCE );
   }
   then {
      execute ?loanApp.GenerateMessage(
```

```
        "Occupancy status must be principal residence." );
   }
};
```

- `Loan` und `Property` sind Java-Klassen.
- `firstHome`, `lienType` und `occupancyStatus` sind Instanz-Attribute dieser Java-Klassen.
- Durch `?loanApp` wird eine Variablenbindung eingeführt. Die Variable ist mit dem entsprechenden Java-Objekt verbunden.
- Durch `execute` wird auf dem qualifizierenden Java-Objekt die Methode `GenerateMessage` aufgerufen.

Bedingungen in Regeln:

- Zugriff auf Attribute von Java- oder C++-Objekten
- Java- oder C++-Methodenaufrufe
- boolsche Ausdrücke

- Existenzquantor
- Existentielle Negation: Bedingungung ist war, wenn kein Objekt existiert, das den positiven Teil erfüllt (impliziter Allquantor)
- Anzahlen von Objektinstanzen die eine Bedingung erfüllen

Aktionen:

- Update von Objekt-Attributen
- Anlegen oder Löschen von Objekten im bzw. aus dem Working Memory
- Aktivierung oder Deaktivierung von Regelpaketen
- Lokale Variable deklarieren und initialisieren
- Ausführung von Java- oder C++-Anweisungen

Entwicklungszyklus: