

Applications of Artificial Intelligence

Sebastian Iwanowski
FH Wedel

Chapter 1:
Introduction and Survey

Survey of this course

Prerequisites of knowledge:

Discrete Mathematics (including applications), Programming I and II

helpful: Object oriented programming

Targets of this course:

Raising interest for AI applications and technology

Survey knowledge of most AI technologies

Knowledge of several application fields for AI

Which are the applications and technologies?

Wait a second ...

What is AI ?

Turing's test



A software is intelligent, if a human cannot distinguish its behaviour from the behaviour of a human.

Application: Medical Diagnosis

Psychoanalysis: Eliza 1966: Joseph Weizenbaum, MIT

Computer performs a psychoanalysis session and acts „as one thinks a psychoanalyst would act“.

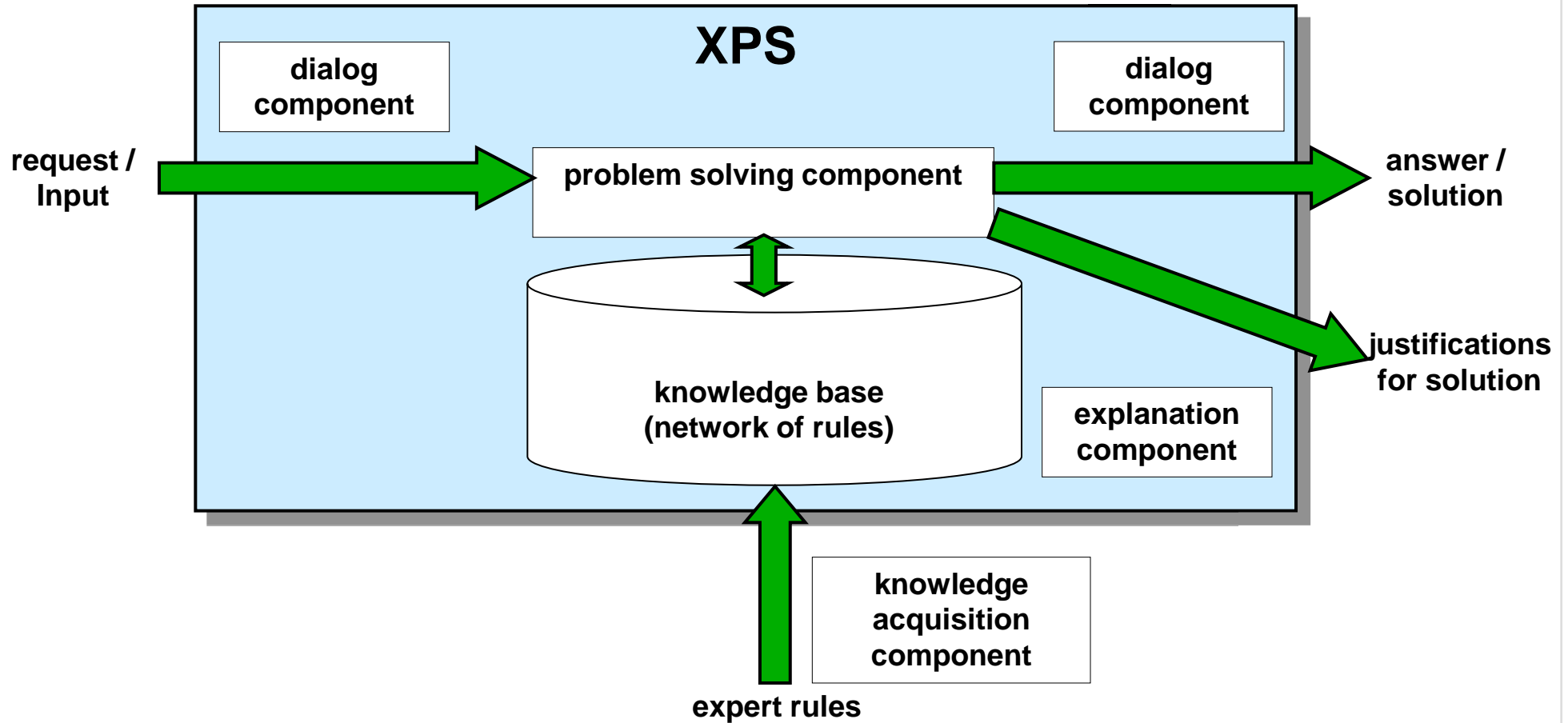
- **passed Turing's test with a lot of people**
- **built-in language assembler and composer**
- **response rules**

Medical Diagnosis: Mycin 1972: University of Stanford

- **for diagnosis and treatment of infectious diseases**
- **worked with probabilistic rules**
- **got high hit scores**
- **little acceptance among physicians due to distrust to computers**

Base Technology: Expert System

Expert System Architecture



Application: Technical Diagnosis

What is **technical** diagnosis?

Input:

- Technical system (e.g. car, train)
- Observations (e.g. measurements, fault codes, driver's complaint), out of order.

Task:

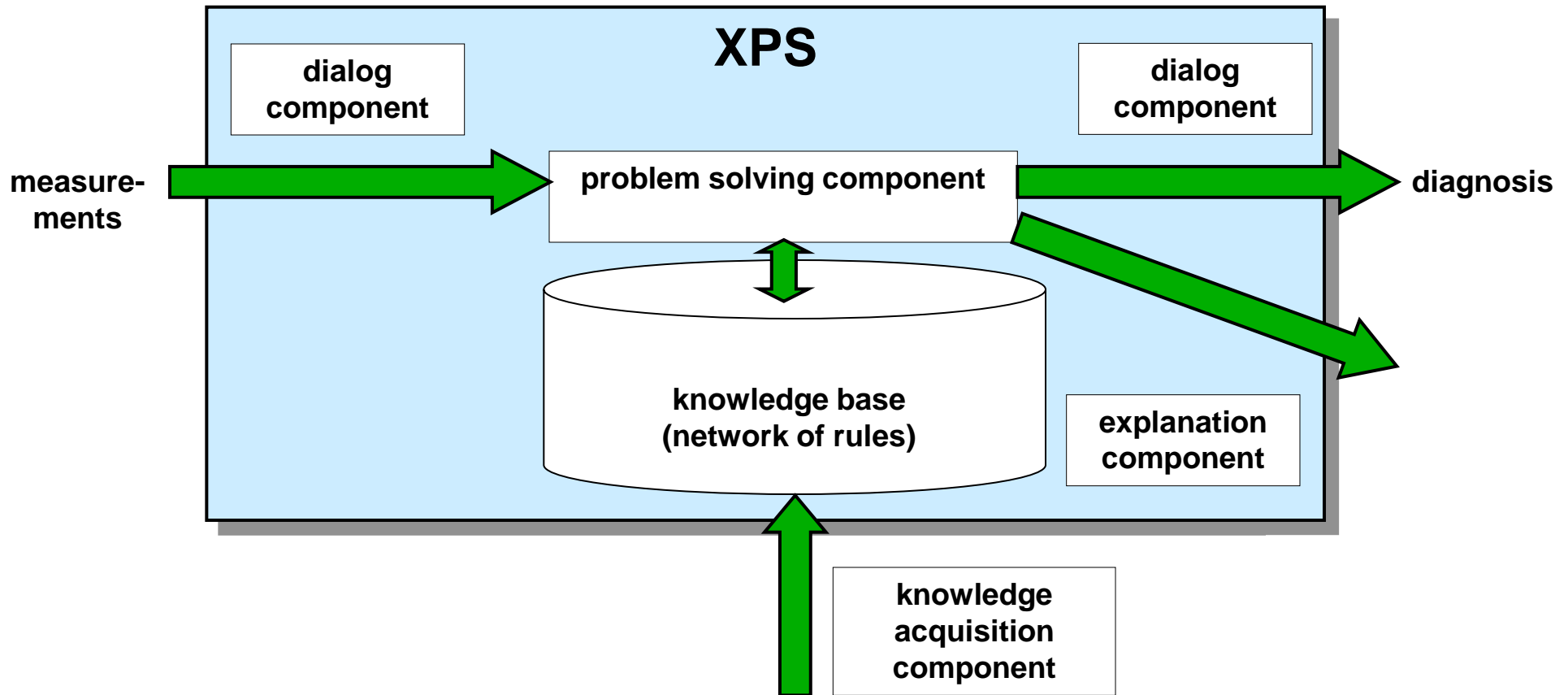
Detect,

- for which reasons the system is out of order
- exactly enough to recover the proper function of the system.



Application: Technical Diagnosis

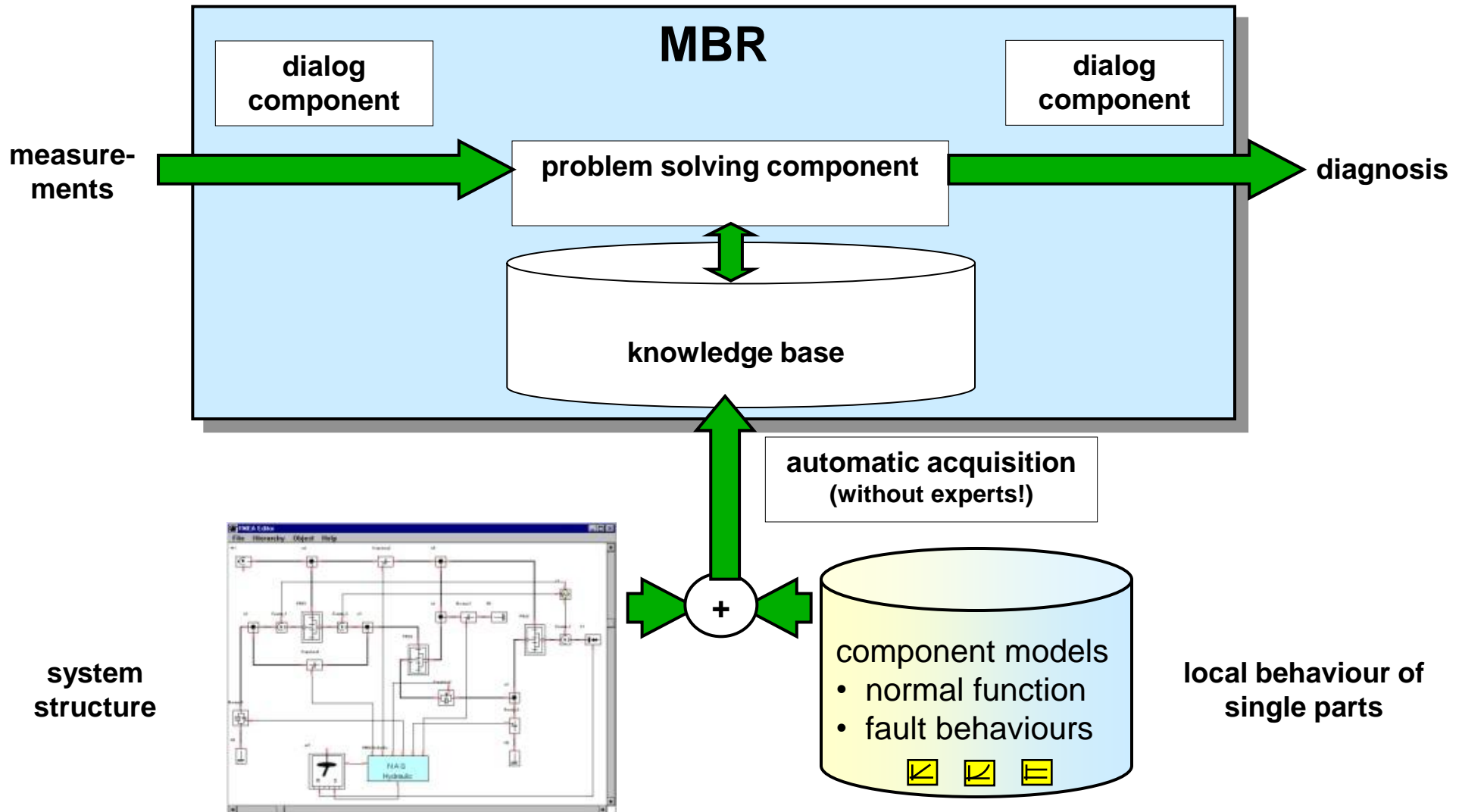
1970-1980s: diagnosis = heuristic classification



knowledge about structure and function:
specific symptom \Rightarrow specific diagnosis

Application: Technical Diagnosis

1990s: diagnosis = model-based reasoning

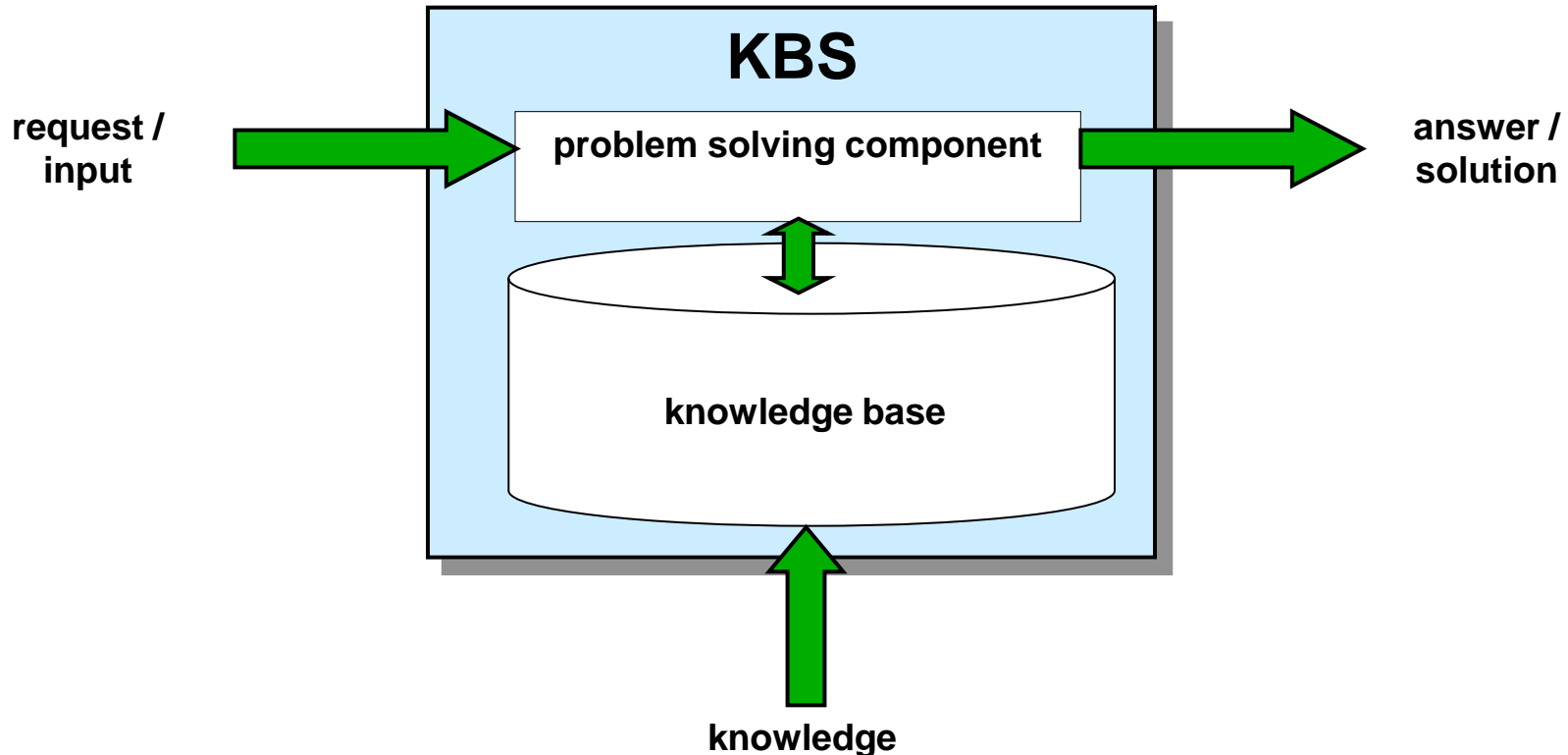


Base Technology: Knowledge-Based Systems

knowledge + problem solving component = KBS

data processing rules

Architecture KBS (joint generalisation of XPS and MBR)



Application: Image recognition

Goals:

- Identifying persons of a certain group (gender, appearance, attitude, etc.)
- Identity control for admission / authorisation
- Identifying certain persons if they are in a certain area
- Forensic analysis
- Identifying street signs
- Identifying arbitrary objects for certain purposes
- ...

Base Technology: Machine Learning (CBR)

Knowledge Acquisition Technique: Training by examples

1. approach: vector-based using a similarity measure

- classical approach already used in the early days of AI
- modern method: Support Vector Machines

2. approach: Neural networks

- modern method: deep learning

➡ ***Machine learning techniques are a current hype due to impressing success stories***

➡ ***This is why nowadays many people identify AI with Machine Learning***

➡ ***“Algorithms” in this context are understood the algorithms how to adjust the parameters of the neural network from the training samples***

Note!

- Neural network algorithms are purely statistical and have no causal justification
- Algorithms investigated in “Algorithmics” do always have a causal justification which can be proven.

Applications using Machine Learning:

Recent graduation theses supervised by iw:

Master Thesis Shwetha Mohan Kumar: *Computation of Delays in the Public Transportation of Hamburg*, WS2021/22

Master thesis Thimo Tollmien: *Optimizations of Delay Predictions in Local Public Transport Using Deep Learning*,
SS 2018

➡ **traffic advice, big data**

Master thesis Frederik Schnoegel: *Einsatz von Natural Language Processing im IT Support*, SS 2020

➡ **semantic categorisation**

Bachelor thesis Henning Brandt: *Implementation of a model for determining the concentration of organic molecules in a multicapillary gas chromatograph using machine learning*, WS 2019/20

➡ **technical diagnosis**

Bachelor thesis Dennis Maas: *Transformation invariant bar code recognition using neural networks*, SS 2019

➡ **logistics**

Bachelor thesis Michel Belde: *Improvement of a consulting app for the sales department using image recognition*,
WS 2018/19

➡ **sales**

Bachelor thesis Lasse Karls: *Graph-based feature extraction to improve machine learning in predicting the business affiliation of a Signal Iduna customer*, WS 2018/19

➡ **customer maintenance**

Application: Passenger Information System

Task:

For two points A and B, find the shortest path between A and B using exclusively segments of the traffic network.

Solution:

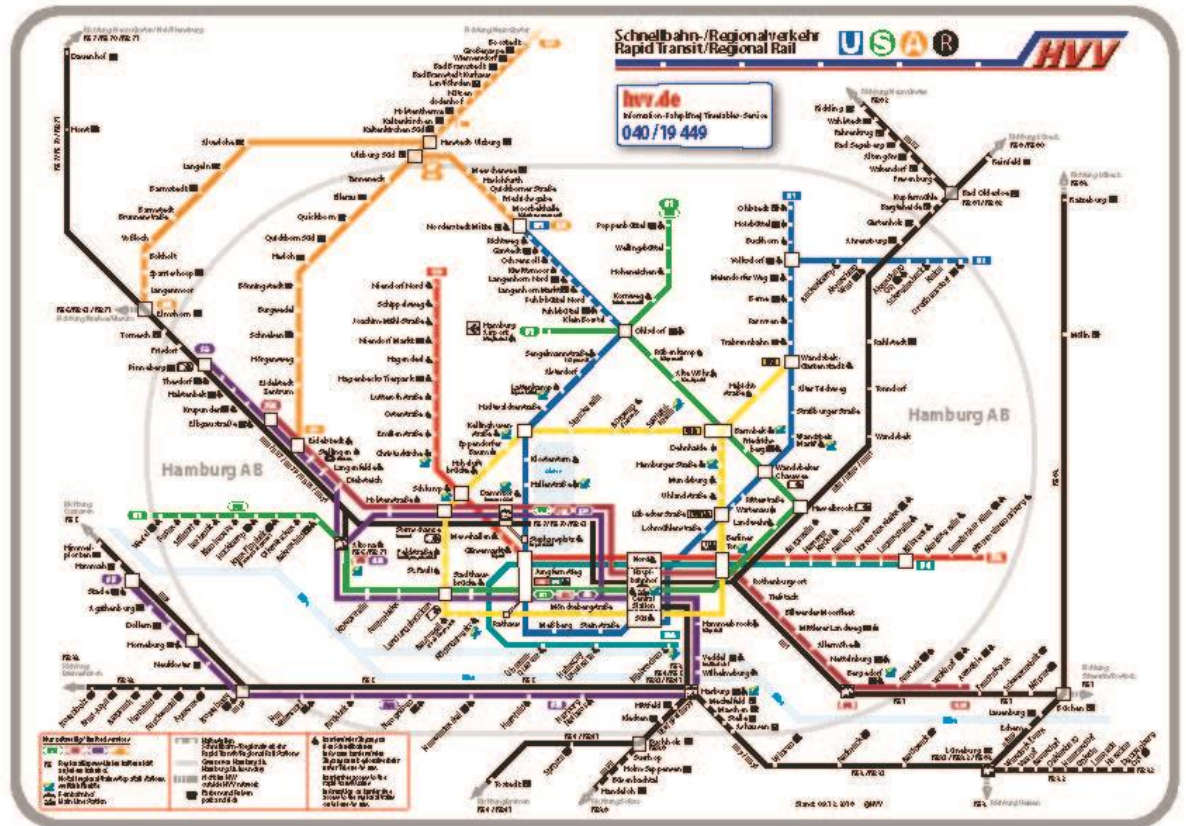
Dijkstra's algorithm

(cf. Discrete Mathematics, ch. 7, graph theory)

A* algorithm

Optimisation with further heuristics (e.g. Geofox system für Hamburg Transportation Network)

Optimisation with preprocessing (e.g. Hafas for German Railways)



several seminars, projects and graduation theses at FH Wedel on routing

Application: Passenger Information System

Passenger information for HVV with smartphones:

Development and implementation of actual prototypes:

- iPhone
- Android smartphones

Diploma thesis Sebastian Hammes (eos-uptrade, SS 2010)

- results used in HVV App

Bachelor thesis Henning Reimer (HBT, SS 2010)

- results used in Geofox App

Master thesis Josias Polchau (HBT, SS 2014)

- Innovation award of Rotary Club Wedel

Speed-up of routing computation:

This is NOT typical AI !

Master thesis Nicolas Mönch: *Shortest paths in dynamic graphs*, WS 2015/16

Bachelor thesis Christian Binder: *Optimisation of a public transport routing algorithm*, SS 2017

Master thesis Lukas Müller: *Hierarchical Algorithms in Public Transport*, SS 2018

Application: Passenger Information System

*does not contain AI techniques
as defined in a classical way*

Mobile passenger assistant:

A „navigation device“ for public short-distance traffic

Master thesis by Josias Polchau (SS 2014)

Funktion	Nutzen	Realisierungsaufwand	
Informationen zur aktuellen Fahrt	██████████	██████████	implemented
Aktualisierung der Route	██████████	██████████	implemented
Erinnerungen	██████	███	implemented
Verbesserung der Fußwegnavigation	██████████	██████████	implemented
Füllstands-Anzeige	██████	██████████	
Stau-Karte	█	██████	implemented
Anzeige: Einstieg vorne/hinten	██████	derzeit nicht möglich	
Fahrradmitnahme	██████████	derzeit nicht möglich	
Routenpause	██████	██████████	
Lautsprecheransagen	██████	derzeit nicht möglich	

Example for a typical AI solution in this context:

Master Thesis Shwetha Mohan Kumar: *Computation of Delays in the Public Transportation of Hamburg*, WS2021/22

Master thesis Thimo Tollmien: *Optimizations of Delay Predictions in Local Public Transport Using Deep Learning*,

SS 2018

Application: Road Navigation

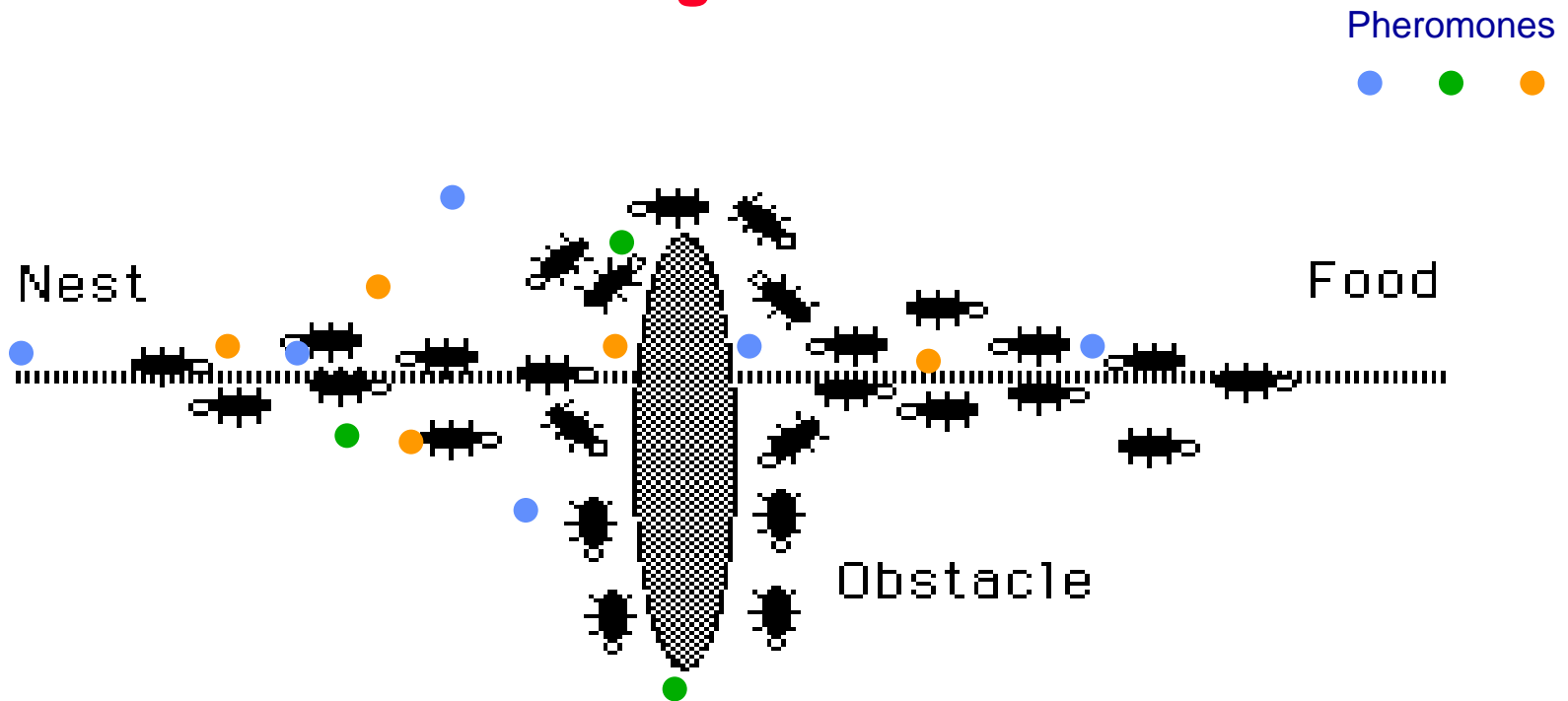
Differences to be considered for adaptation to road networks:

- **Road network is much denser.**
- **no time-tables or opening hours**
- **Traveling time depends very much on traffic density.**
- **Traffic devices are not controlled centrally.**

Application: Road Navigation

Swarm Intelligence: Pheromone-Based Approach

Ants seeking for food

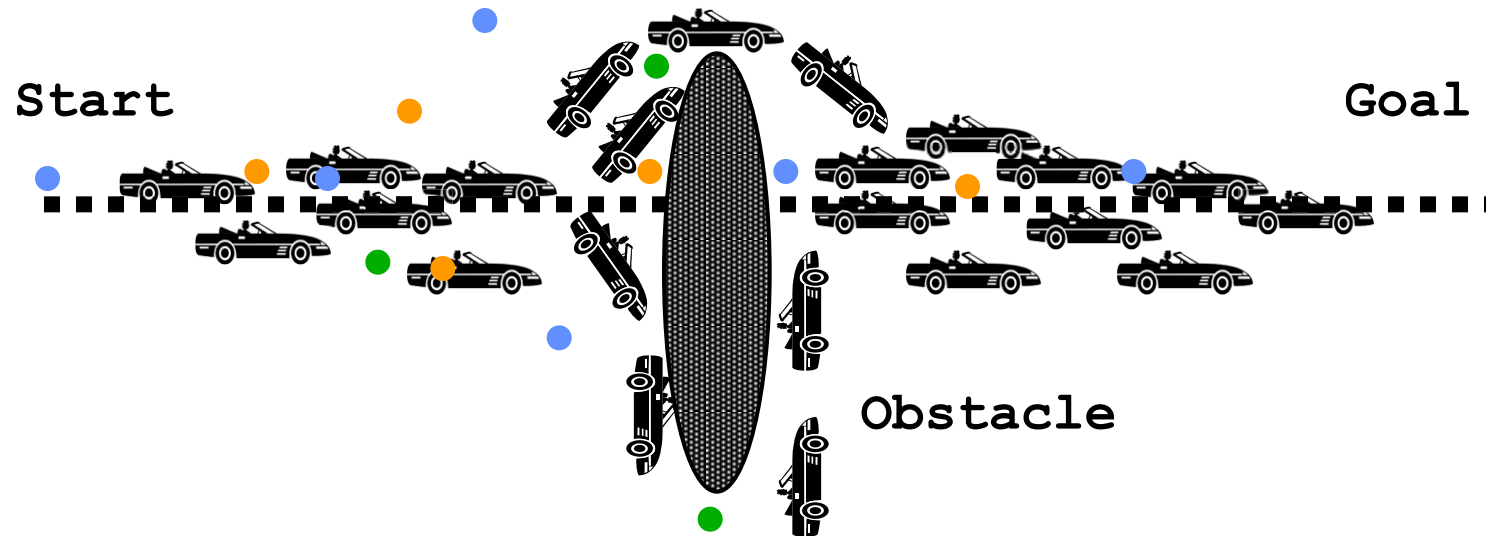


Application: Road Navigation

Swarm Intelligence: Pheromone-Based Approach

Analogue:
Cars seeking for routes

Pheromones



Base Technology: Swarm Intelligence

- **a lot of small autonomous units, each with limited ability**
- **total organism has a higher ability than the sum of the units**
(“emergent behaviour”)
- **determined rule system for total organism**
- **anytime ability**

Research focus at FH Wedel by iw:

Several projects, graduation theses and publications since 2006

Application: Game AI

Chess computer

(Ex. for a turn-based game)

First Milestone 1997:

Kasparov 2.5 – Deep Blue 3.5



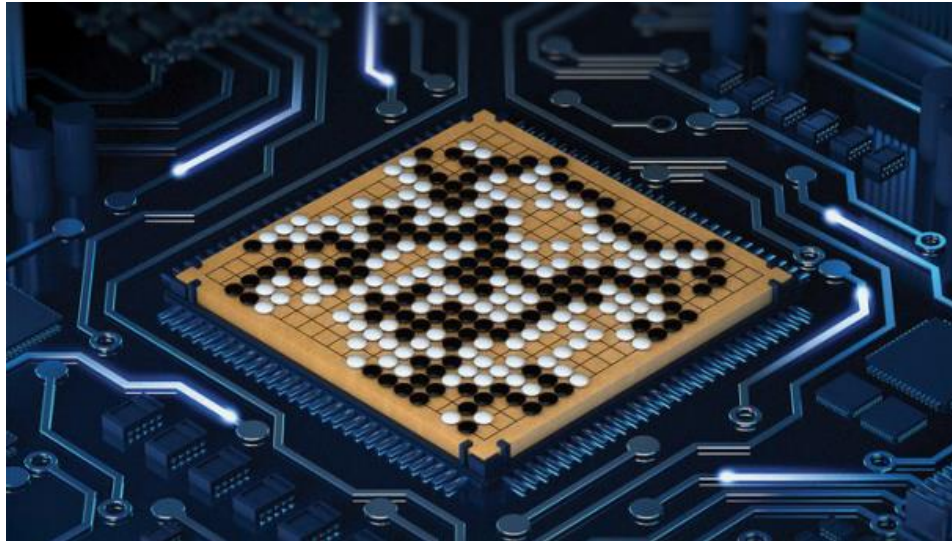
Further infos: <http://www.research.ibm.com/deepblue>



Application: Game AI

Go computer

(a much harder turn-based game)



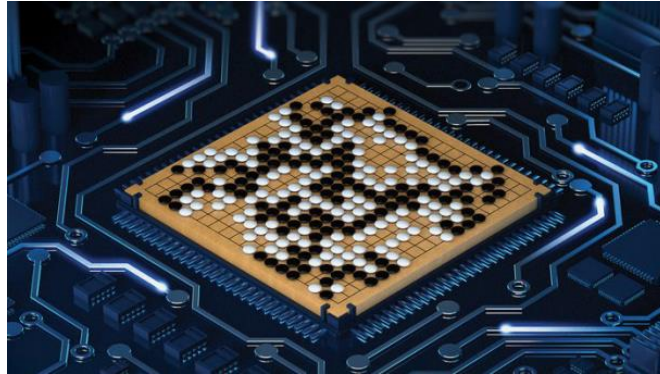
Second Milestone 2015:

- Google's Deep Mind developed Alpha Go.
- Alpha Go used Machine Learning and was trained by experienced Go players.
- In 2015 Alpha Go beat several world famous Go players.

Application: Game AI

Go computer

(a much harder turn-based game)



Third Milestone 2017:

- In 2017, Deep Mind developed the update version Alpha Go Zero.
- Alpha Go Zero started by playing against itself and was not trained by humans at all.
- Within 3 days of continuous training, Alpha Go Zero reached a stage, experienced Go players need years for.
- Alpha Go Zero played 100 matches against Alpha Go and won them all.
- By now, DeepMind developed improved versions, e.g. AlphaZero which can also play other games like chess.

Application: Game AI

Fang den Fox - Mozilla Firefox

Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe

http://catchfox.hbt.de/fdf/Fox.html

Meistbesuchte Seiten Windows Windows Media Kostenlose Hotmail Links anpassen FTP-Tutorial by Fabien... Recursion Software, In... Familienbilder2006

Suche Lesezeichen Rechtschreibprüfung Übersetzen Senden an Einstellungen

Fang den FOX mit GEOFOX

Spiel Fänger Ansicht Hilfe Freitag, 12.36 Uhr

Figuren

FOX

Die letzte bekannte Position des FOX ist:
Eppendorfer Baum

Diese Figur hat bereits diese Runde gezogen.

Fänger 1 +

Fänger 2 +

Fänger 3 +

Fänger 4 +

Fänger 5 +

Karte Satellit Hybrid

©2008 DigitalGlobe, GeoContent, AeroWest, GeoEye, Kartendaten ©2008 Tele Atlas - Nutzungsbedingungen

Status

For informational stuff, maybe status bar
Begin loading stations: 11:35:20
Data received: 11:41:07
StartingStations made: 11:41:97

Fertig

Application: Game AI

Turn-based game „Catch the fox“

- Diploma thesis 2009 at HBT (operator of Geofox)
- 3. prize of Hochbahn award
- Computer controls the fox
which should be caught by human-controlled avatars
- Game uses real time information of HVV
- Originally programmed on GoogleMaps, then transferred to licensed map
- License reasons forced to switch off the online game.
- A new implementation is only possible with OpenStreetMap.



Project work possible at HBT

Application: Game AI

Real-time strategy games



Source Age of Empires 2, screenshot of Nils van Kan



Application: Game AI

Real-time strategy games

Typical AI requirements:

- Path finding and location analysis
- Resource planning
- Policies and strategies

Base Technology: Search Strategies

- **Construction of search spaces**
- **Uninformed search strategies**
 - **breadth-first search**
 - **depth-first search**
 - **combined search**  **Special case: Dijkstra's algorithm**
- **Informed search strategies**
 -  **Special case: A* algorithm**

is used in navigation products as well

Application: Game AI

Realtime strategy games

Requirements in modern games:

- Pathfinding and terrain analysis in environments changing dynamically

Algorithmic techniques:

- Construction of way graph for navigation
- Learning from suboptimal paths
- Working with unsafe information

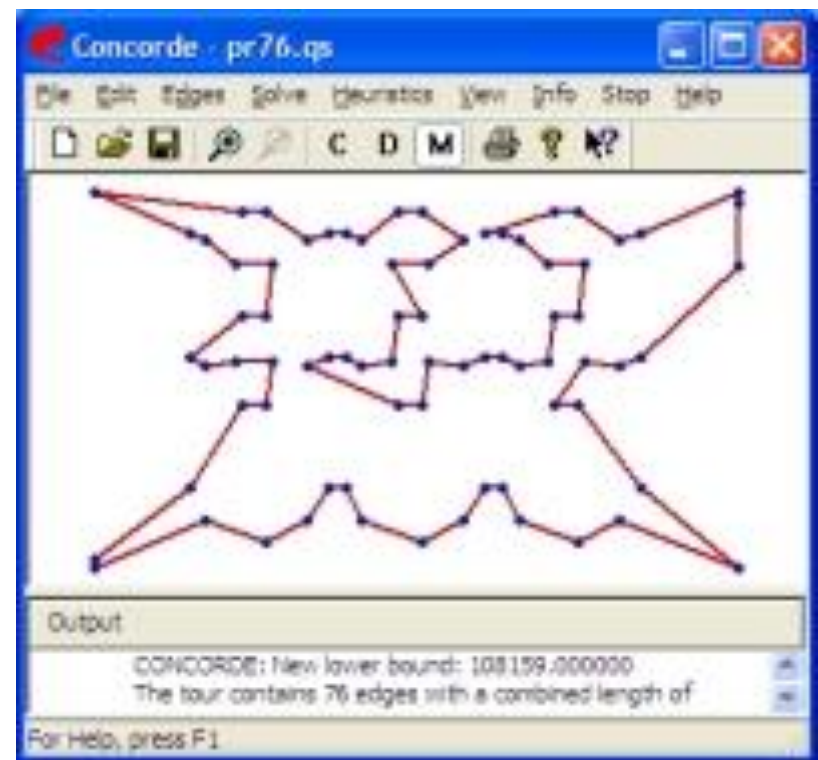
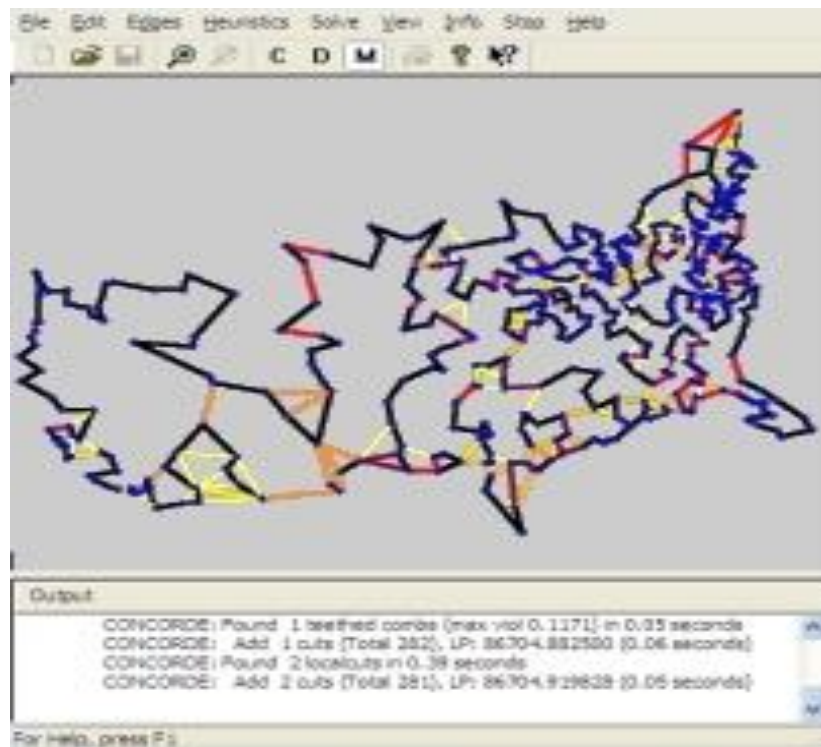
*does not always
include classical AI*

*but is always
considered Game AI!*

Application: Traveling Salesman Problem (TSP)

Master example for an NP-hard problem:

For a given set of cities with known mutual distances, find the shortest round trip passing each city at least once.



Source: <http://www.tsp.gatech.edu//index.html>

Application: Traveling Salesman Problem (TSP)

Generalisations in logistic applications:

- **considering time restrictions (time windows)**
- **considering load capacities for delivery problems**
- **further system-specific requirements**

Examples for graduation theses in companies:

implico: Tour planning for oil and gas delivery (SS 2010, SS 2011, SS 2013)

Long-term development project: Tourist Information System

**Christoph Forster / Thomas Kresalek / Felix Döppers:
Master project Hamburg Tourist Information (since 2009)**

<http://vsrv-studprojekt2.fh-wedel.de:8080/touristinformationsystem/home>

Solution of dynamic problems via ant systems

Example for a graduation thesis in a company:

Christopher Blöcker: Dynamic optimisation of tour delivery using an ant system (SS 2011)

Application: Class Scheduling

Given finite sets Courses, Rooms, Time slots

Task: Generate an injective (one-to-one) function $C \rightarrow R \times T$

Strict Constraints (must be fulfilled in any case):

- **Certain courses must not take place at the same time**
- **For some courses, certain time slots are not admitted**
- **For some courses, certain rooms are not admitted**

Soft constraints (may be violated):

- **Certain courses should not take place at some times**
- **Certain courses should take place successively**
- **Certain courses should not take place on the same day**

Optimisation function:

- **fewest violations of soft criteria**
- **fewest free periods for certain study programmes**
- **most uniform distribution on different days for ...**

Bachelor Thesis Timm Hoffmann:

Autonomous Planning System for Generating a Timetable for FH Wedel, WS 2013/2014

Base Technology: Constraint Satisfaction Problem (CSP)

Specification of a CSP:

- **set of variables**
- **domains of definition**
- **constraints: relations between variables (strict or soft)**
(normally, equations and inequalities)
- **optimisation criterion**
(normally, a real-valued function on the variables which has to be minimised or maximised)

valid solution:

assignment of all variables with values such that all strict constraints are satisfied

optimal solution:

valid solution optimising the optimisation criterion

Manifold application scenarios in various problems of logistics

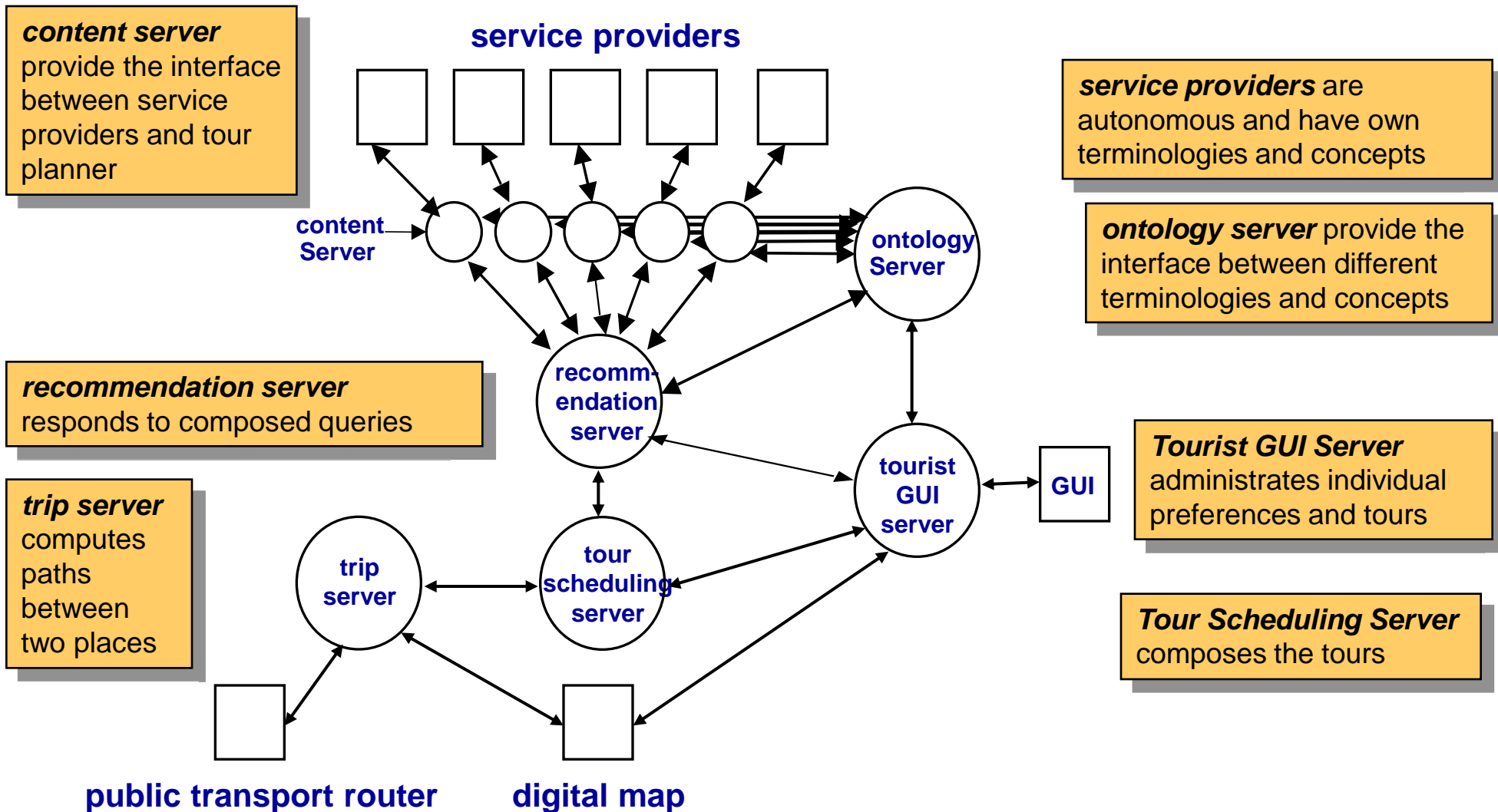
Application: Tourist Information System

Requirements:

- Tourist gets the final control.
- Service provider is autonomous and takes responsibility for all information
- Independent broking between several providers
- Flexible response to requirement changes even during the tour
- Fault tolerance for single provider failure
- **Arbitrary** service providers should be subject to be added or withdrawn automatically during system operation.

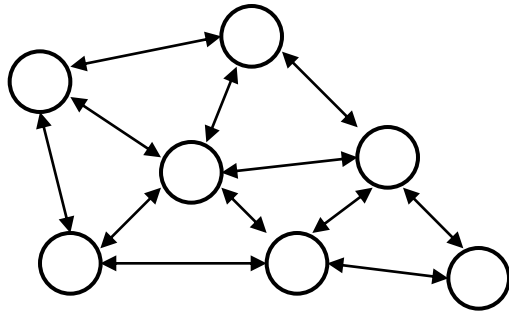
Application: Tourist Information System

Architecture of tour planning system: prototype of a SOA

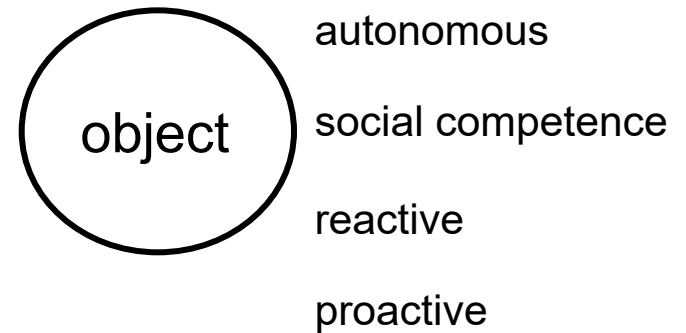


Base Technology: Agent-Oriented Software

Multi-agent system:



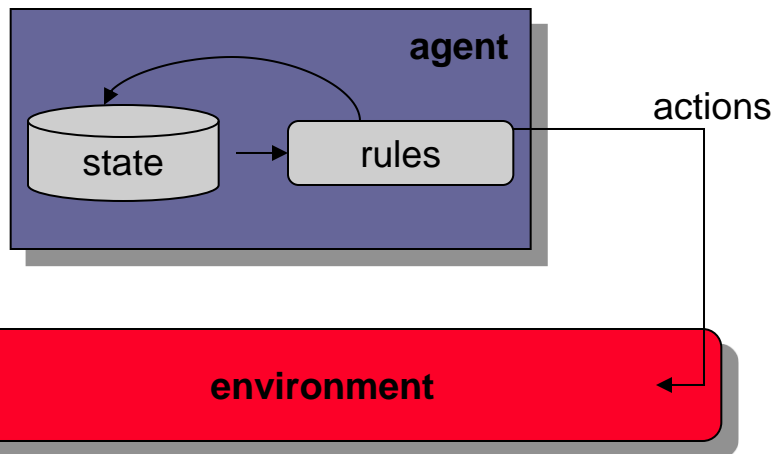
Software agent:



Weitere Infos: Seminarvortrag und Ausarbeitung von Matthias Rohr, SS 2004, Nr. 4,
<http://www.fh-wedel.de/~si/seminare/ss04/Termine/Themen.html>, erreichbar über [archiv/iw](#)

Base Technology: Agent-Oriented Software

Agent property: Proactivity (goal oriented)



Agents do not only react to stimuli of the environment, but also depend on an internal state and have the capability to pursue own plans and actions.

=> They are taking **initiatives**

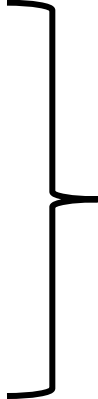
*„The difference between an automation and an agent is a somewhat like the difference between a dog and a butler. If you send your dog to buy a copy of the New York Times every morning, it will come back with its mouth empty if the news stand happens to have run out one day. In contrast, the butler will probably take the **initiative** to buy you a copy of the Washington Post, since he knows, that sometimes you read it instead.“*

Le Du

Quelle: Seminarvortrag und Ausarbeitung von Matthias Rohr, SS 2004, Nr. 4

Base Technology: Semantic Network

- **ontology management**
- **description language**
- **description logics**



developed in the 1990s based
on AI syntax standards of the
1980s

Modern adaptation (2001): *Semantic Web standards*

Initiator: Tim Berners-Lee

Ontology management, description language and description logics
in XML or comparable standards

Common feature:

Universally valid definitions in a syntax readable by engines and browsers

Defining AI

Thinking Humanly

“The exciting new effort to make computers think . . . *machines with minds*, in the full and literal sense.” (Haugeland, 1985)

“[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning . . .” (Bellman, 1978)

Acting Humanly

“The art of creating machines that perform functions that require intelligence when performed by people.” (Kurzweil, 1990)

“The study of how to make computers do things at which, at the moment, people are better.” (Rich and Knight, 1991)

Thinking Rationally

“The study of mental faculties through the use of computational models.”
(Charniak and McDermott, 1985)

“The study of the computations that make it possible to perceive, reason, and act.”
(Winston, 1992)

Acting Rationally

“Computational Intelligence is the study of the design of intelligent agents.” (Poole *et al.*, 1998)

“AI . . . is concerned with intelligent behavior in artifacts.” (Nilsson, 1998)

Figure 1.1 Some definitions of artificial intelligence, organized into four categories.

Definitions from Russell / Norvig

Defining AI

AI deals with problems which

- **are relevant in practical applications.**
- **may not be specifiable in a mathematical way.**
- **are NP-hard if they can be specified in a mathematical way.**

Definition iw

Features of classical AI solutions

The classical controversy between different research communities in computer science:

AI vs. Algorithmics

- **flexible solutions**
- **human customer oriented solutions**
- **exact solutions**
- **efficient solutions**

This need not be contradictory!

Features of classical AI solutions

Intelligent creatures are able to process very general knowledge: The more general, the more intelligent.

The ability to process general knowledge needs general description languages for data and processes.

The most general description language is the language of mathematical logics.

This is why traditional AI implementations work with logic description languages.

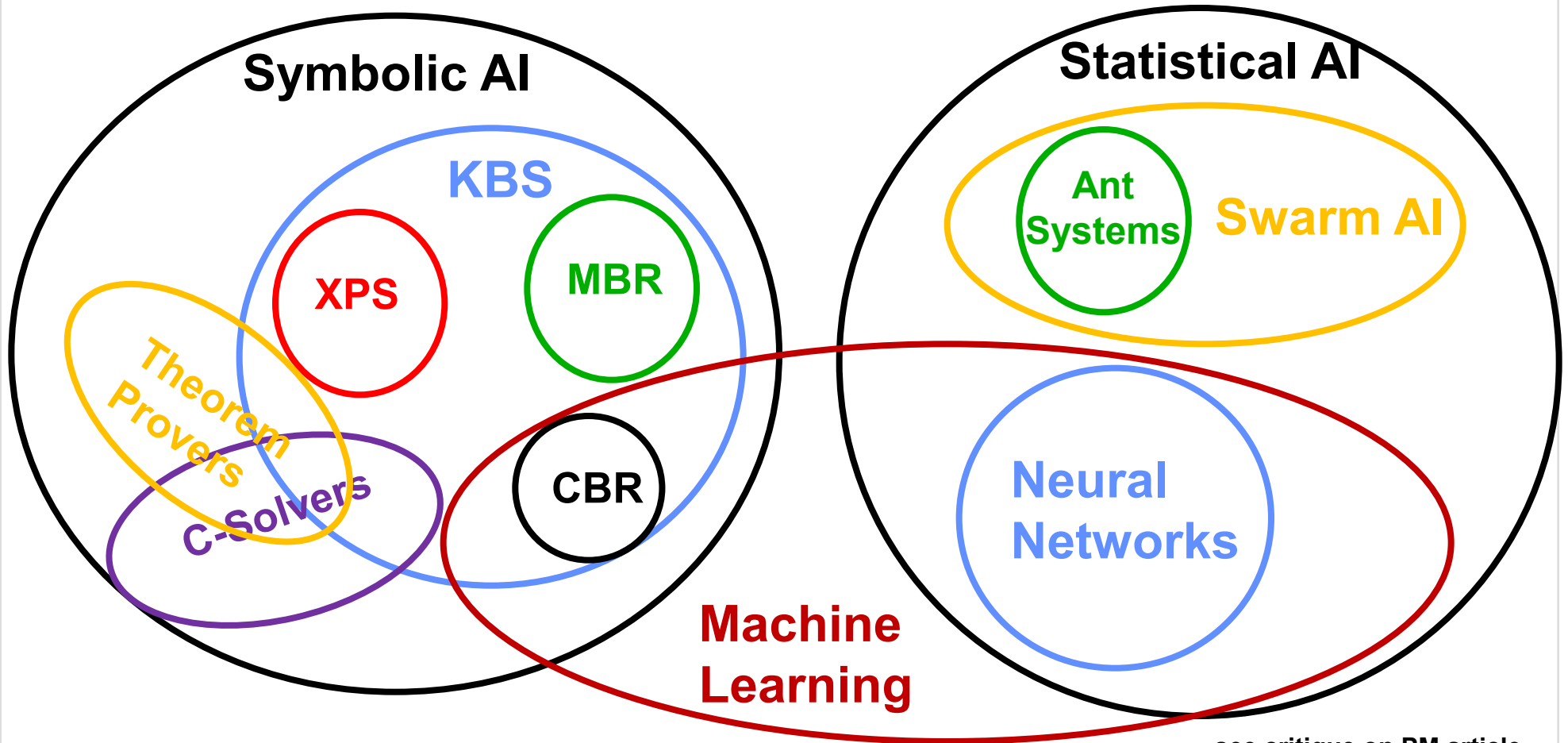
- Problems:**
- **The tasks are usually formulated in a different way.**
 - **There is a trade-off between generality and efficiency.**

Base Technology: Logic Programming Language

- **Input:**
Specification of the problem with a logical description language
- **Output:**
Response in a logical description language
- **Automatically (without specifying algorithms!):**
Generation of output from input
- **For improvement of efficiency:**
Different specifications of the problem are possible and may influence the output if the automatic generation procedure is well-understood

Summary Chapter 1

The set of AI techniques



see critique on PM article

Good applications share several techniques

Summary Chapter 1

AI goals for software solutions

- **generality**
- **flexibility, extensibility**
- **justification of answers** (only for symbolic AI)

Tools and methods invented and applied in AI

- **Logic programming languages (PROLOG)**
- **Object-oriented programming languages (Smalltalk)**
- **Functional programming languages (Lisp)**
- **Distributed technology (neural networks, multi-agent-systems, swarm intelligence)**
- **Concept descriptions (ontologies)**

Summary Chapter 1

Applications of AI:

- **Diagnosis**
 - Medical diagnosis
 - Technical diagnosis
- **Optimisation problems with dynamic parameters**
 - Passenger information systems
 - Road navigation
 - Logistics (TSP, Scheduling)
- **Resource allocation**
 - Allocation problems with manifold constraints (e.g. class schedule, tourist information system)
- **Flexible management of distributed knowledge**
 - Tourist information system
- **Games where a machine simulates a human player**
 - turn-based
 - real-time

Summary Chapter 1

Base Technologies of AI:

- **Knowledge-based systems (generalisation of expert systems)**
 - Separation of knowledge and inference engine
 - Intelligent knowledge acquisition and representation
 - Main focus: Reusability
- **Neural networks**
 - Special case of knowledge-based systems, but without explanation component
- **Swarm intelligence**
 - distributed
 - statistic
 - concurrent updating
- **Agent oriented software**
 - distributed
 - autonomous
 - proactive

Summary Chapter 1

Base Technologies of AI:

- **Semantic network**
 - **Ontologies: Generation and administration of terminology and concepts**
- **Search strategies**
 - **Uninformed vs. informed**
- **Constraint satisfaction problem (CSP)**
 - **Search for valid solutions**
 - **Search for optimal solutions**
- **Logic programming languages**
 - ***What* is specified by man**
 - ***How* is generated automatically**

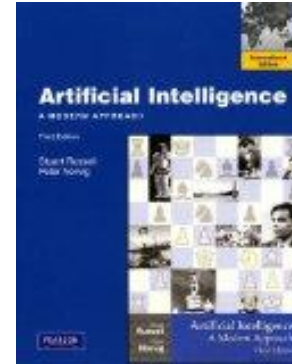
Literature

Symbolic AI in general:



Günter Görz / Josef Schneeberger / Ute Schmid:
Handbuch der Künstlichen Intelligenz
Oldenbourg 2013 (5. Auflage), ISBN 978-3-486-71307-7

Wolfgang Ertel / Josef Schneeberger: *Grundkurs Künstliche Intelligenz*
Vieweg 2009 (2. Auflage), ISBN 987-3-8348-0783-0



Stuart Russell / Peter Norvig:
Artificial Intelligence: A Modern Approach,
Pearson 2010 (3. edition),
ISBN 0-13-207148-7

Machine Learning:

Ian Goodfellow, Yoshua Bengio, Aaron Courville: *Deep Learning*, MIT Press 2016,
available via <http://www.deeplearningbook.org/> and FH Wedel handout server (via my website)

for special fields of AI:

see my current website and comments

Applications of Artificial Intelligence

Sebastian Iwanowski
FH Wedel

Chapter 2:
Logic- and Rule-Based Programming
Using the Example of Prolog

Literature for Prolog

Textbooks:

Ivan Bratko: *PROLOG, Programming for Artificial Intelligence*,
2nd Edition, Pearson 1990, ISBN 0-201-41606-9
3rd Edition, Pearson 2001, ISBN 0-201-40375-6
4th Edition, Pearson 2011, ISBN 0-321-41746-6
Companion website with Prolog code: www.pearsoned.co.uk/bratko

P. Blackburn, J. Bos, K. Striegnitz: *Learn Prolog Now!*,
Texts in Computing Vol. 7, King's College Publications. 2006, ISBN 1-904987-17-6.
Companion website with on-line version: www.learnprolognow.org

Peter Bothner / Wolf-Michael Kähler: Programmieren in *PROLOG (in German)*,
Eine umfassende praxisgerechte Einführung,
Vieweg 1991, ISBN 3-528-05158-2

Seminar presentation (in German):

Max Rohde: *Eignung logischer Programmiersprachen für Spiele-KI am Beispiel Prolog*,
FH Wedel, Iwanowski, SS 2007, Informatik-Seminar zur Spiele-KI

↳ gibt auch einen Überblick über Prolog und enthält weiterführende Literaturliste

Elements of PROLOG

Elementary components:

- **numbers**

Integer and real numbers are distinguished ($1 \neq 1.0$).

- **atoms**

name where the first character is a small literal

- **variables**

name where the first character is a capital literal, exception: `_`

- **lists**

`[]` or `[term | list]`

short notation: `[1,2,3,4]` for `[1 | [2 | [3 | [4 | []]]]]`

- **terms**

numbers, atoms, variables, lists or expressions like `atom(term)`, `atom(term,term)` or ...

- **predicates**

terms of the type `atom(term)`, `atom(term,term)` or ...

2 predicates are equal, if their name is the same atom and the number of parameters is the same.

Elements of PROLOG

Logic operators between predicates:

- **conjunction**
a , b corresponds to: $a \wedge b$
- **implication**
a :- b corresponds to: $b \rightarrow a$
- **equivalence**
a = b corresponds to: $b \leftrightarrow a$
- **antivalence (exor)**
a \= b corresponds to: $b \nleftrightarrow a$
- **version-specific operators for comfort**
member, length, ...

Elements of PROLOG

Arithmetic operators

- **+, -, *, /, div, mod**

Arithmetic expressions are always formed in infix notation.

Evaluation of arithmetic expressions

- **not automatically!**
- **when a variable is assigned an expression**

`varname is arithmetic expression`

Result of the arithmetic expression is assigned to the variable.

- **using special logic operators with evaluation capability**

`<, =<, > >=, =:=, =\=` evaluate arithmetic expressions on either side.
(in some implementations only on one side)

Elements of PROLOG

Knowledge in form of **clauses**

- **facts**

`predicate.`

Such predicates are assumed to be true in the knowledge base.

- **rules**

`predicate :- conjunction of predicates.`

The concluding predicate (on the left) is considered true if the proposition (on the right) has to be assumed true.

For the same concluding predicate there may be different rules.

- **queries**

`?- conjunction of predicates.`

Prolog tries to derive the truth of a query from the known facts and rules.

If this derivation is successful, the answer is `yes` and the values necessary to bind on a variable for the verification are output.

Otherwise the answer is `no`.

Elements of PROLOG

Prolog's **special** handling of *not*

- **Most versions of Prolog provide a concept for negation**

`not Term`

`\+ Term`

`Term1 =\= Term2`

Prolog evaluates these predicates to true if it cannot prove that Term is true resp. Term1 = Term2.

Warning:

This is not the same as that Prolog can prove that Term is false resp. Term1 \neq Term2

Consequence:

Strict mathematical problem solvers better avoid using negation.

Functionality of a PROLOG interpreter

PROLOG is knowledge-based:

- **Knowledge base**

Facts and rules, dynamically extensible

- **Inference engine**

deriving facts and rules automatically using the inference techniques **resolution** und **unification**

- **Dialog component**

Input: Query

Output: yes / no, Specification of used unification in case of success, write as a „side effect“

Yes: The predicate of the query can be concluded from knowledge base.

No: The predicate of the query cannot be concluded from knowledge base.

No does not imply that it can be concluded that the predicate is false.

Functionality of a PROLOG interpreter

How the inference engine works:

- **Decomposition of a goal into subgoals**

First goal is the original query.

Prolog tries to achieve the goal with unifications of the predicates of the knowledge base.

This makes the predicates to subgoals.

- **Order of evaluation**

All data of the knowledge base are evaluated **from top to bottom**.

Conjunctions of rule propositions are evaluated **from left to right**.

The evaluation order does *not* distinguish between facts and rules.

- **Instantiation of variables**

Variables are instantiated with values only for the sake of unification.

The current instantiation is removed after definite success or failure of unification with this value.

- **Backtracking**

Failure of a unification automatically initiates a new instantiation.

Deep backtracking: Try the verification with a different value in the proposition for the same rule.

Shallow Backtracking: Try to verify a different rule implying the same predicate.

PROLOG: Simple example

- **Predicate world from first semester:**

Knowledge base:

```
father(sven,georg).  
brother(holger,anna).  
married(sven, anna).
```

```
male(X) :- father(X,Y).  
male(X) :- brother(X,Y).
```

```
uncle(X,Y) :- father(Z,Y), brother(X,Z).  
uncle(X,Y) :- mother(Z,Y), brother(X,Z).  
mother(X,Y) :- father(Z,Y), married(X,Z).  
female(X) :- married(X,Z), male(Z).  
married(X,Y) :- married(Y,X).
```

Queries:

```
?-female(anna).  
?-male(georg).  
?-uncle(holger,georg).  
?-male(X).  
?-married(holger,X).
```

Declarative alternative
without problems with
symmetric predicates: XSB
<http://xsb.sourceforge.net/>

In ISO-Prolog this does not work!

better:

```
isMarried(X,Y) :- married(X,Y).  
isMarried(X,Y) :- married(Y,X).
```

```
?- isMarried(holger,X).
```

PROLOG: More complicated example

- 8 queens problem (1st solution of Bratko)

Knowledge base:

queens1([]).

```
queens1([X/Y | Others]) :-  
  queens1(Others),  
  member(Y,[1,2,3,4,5,6,7,8]),  
  conflictFree(X/Y,Others).
```

```
conflictFree(_,[]).
```

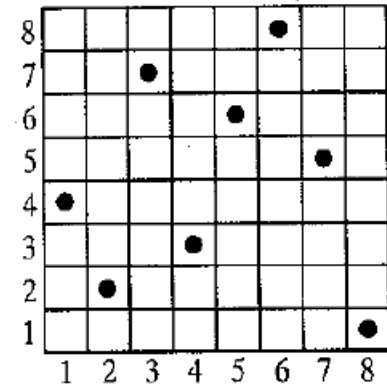
```
conflictFree(X/Y, [HeadX/HeadY | Others]) :-  
  Y =\= HeadY,  
  DiffY is HeadY - Y,  
  DiffY =\= HeadX - X,  
  DiffY =\= X - HeadX,  
  conflictFree(X/Y,Others).
```

```
template([1/Y1,2/Y2,3/Y3,4/Y4,5/Y5,6/Y6,7/Y7,8/Y8]).
```

Query:

query for a single answer:
?-template(S), queens1(S).

query for all answers:
?-template(S), queens1(S), write(S), nl, fail.



not: DiffY ::= HeadY-Y

not: HeadY - Y =\= HeadX-X

Base Technology: Logic Programming Language

- **Input:**
Specification of the problem with a logical description language
- **Output:**
Response in a logical description language
- **Automatically (without specifying algorithms!):**
Generation of output from input
- **For improvement of efficiency:**
Different specifications of the problem are possible and may influence the output if the automatic generation procedure is well-understood

Logic programming languages

Task for the interpreter:

Original goal: Construction task

*less than ever not decidable
for arbitrary formulae*

Given a set \mathcal{F} of logic formulae. Determine all formulae that can be logically derived from \mathcal{F} .

Easier goal: Verification task

not decidable for arbitrary formulae

Given a set \mathcal{F} of logic formulae and a (new) logic formula F .
Find out if F can be derived from \mathcal{F} .

Problems equivalent to the verification task:

- 1) Given a set \mathcal{F} of logic formulae and a (new) formula F . Find out if the set $\{\neg F\} \cup \mathcal{F}$ is contradictory.
- 2) Given a set \mathcal{F} of logic formulae. Find out if it is contradictory.

Corresponds to satisfiability problem: not decidable for arbitrary formulae

Chances to simplify the problem:

Restrict the class of admissible formulae !

Propositional formulae

- A propositional **formula on truth values** is a combination of finitely many literals with operators of propositional logics.
 - The literals are variables which may assume exactly one of two values.
- The **instantiation of a formula** is an assignment of values `true` or `false` to all literals such that the same literals achieve the same value.
- A formula is **satisfiable** if there is an instantiation such that the formula evaluates to true.
 - The satisfiability problem of propositional logics is always solvable because there are only finitely many combinations in the potential solution space which may be tested successively.
 - Unfortunately, successive testing takes very long time (exponential in the number of literals). Until now no more efficient algorithm is known.

Problem is NP-complete !

Predicate logics (first order)

Predicate logics extends propositional logics by the following:

- **predicates**
 - propositions depending on variables.
If a proposition depends on k variables, it is called k -ary.
- **variables**
 - correspond to the literals of propositional logics,
but may assume one out of a set of arbitrarily many values
- **functions**
 - unique assignments depending on variables
(if a function depends on k variables, it is called k -ary)
 - 0-ary functions are constants.
- **quantors**
 - existence quantor (\exists) und all quantor (\forall)
 - Quantors must be applied to variables only (otherwise not first order)

Predicate logics (first order)

A predicate logic **formula** (**first order**) is built by the following rules:

- A term is a variable or a k-ary function (using any symbol for the function name)
- A formula is a k-ary predicate with arbitrary terms as input or the conjunction, disjunction or negation thereof.
- A formula may also contain quantors **applied to variables**

Ex.: formula $\varphi = \forall x (R(f(\mathbf{y}), g(\mathbf{z},\mathbf{y})) \wedge \exists y (\neg P(g(\mathbf{y},\mathbf{z}), \mathbf{x}) \vee R(\mathbf{y}, \mathbf{z})))$

Green occurrences of y and z are **free**.

Red occurrences of variables are **bound**.

Closed formulae (constants): Formulae not containing any free variable.

Open formulae (without quantors): Formulae not containing any bound variable.

Atomic formulae: Formulae consisting of one predicate involving terms only (no disjunctions, conjunctions or negations)

Predicate logics (first order)

- The **instantiation of a formula** is an assignment of *values to the free variables from predefined domains of definition* such that the same variables achieve the same values.
- A formula is **satisfiable** if there is an instantiation such that the formula evaluates to true.



- In predicate logics, the satisfiability problem is **not decidable**, i.e. no algorithm may ever exist to decide for an arbitrary formula as input if the formula is satisfiable or not.

The general problem is unsolvable !

Is there a work-around ?

Yes, solve a more specific problem !

Power of Prolog

PROLOG does not accept arbitrary predicate formulae:

- *Domains for variables and functions are arbitrary.*
- *no quantors*
- *In CNF, all clauses must be Horn clauses:*

$$\neg p \vee \neg q \vee \dots \vee \neg r \vee x$$

At most one literal is positive

Rule-based notation of Horn clauses:

$$p \wedge q \wedge \dots \wedge r \rightarrow x$$

Rule (Horn clause)

In the assumption there may be a conjunction of positive literals only.

Proposition (Completeness of Horn clause calculus):



For each set of old Horn clauses and a given new Horn clause, Prolog may decide after finite time if the new clause can be concluded from the old clauses **or not**.



Remark „Finite time“ includes „very long“ !

Use of Prolog

Didactic use:

- **good exercise for dealing with formal logics**
- **exercising recursive formulations of problems and algorithms**

Practical use:

- **good for a quick test of concepts (rapid prototyping)**
- **relatively comfortable for simple problems for which no other solution exists than exhaustive search of all possibilities**
- **suitable for successive and systematic output of all possible solutions of a search problem**

Limits:

- **Rather a toy than a tool of commercial use, too far from practical needs**
- **totally useless if efficiency of solution is relevant**

Applications of Artificial Intelligence

Sebastian Iwanowski
FH Wedel

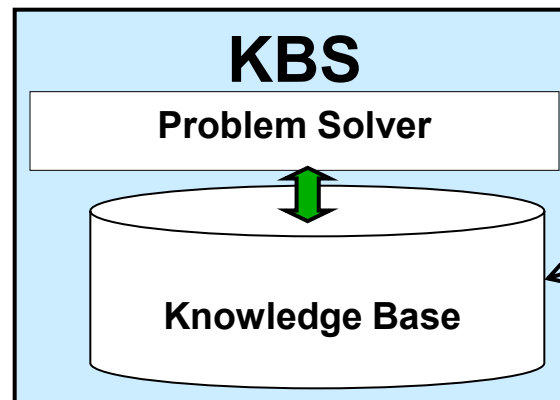
Chapter 3:
Algorithmic Methods of AI

Search Strategies

Relevance of search strategies for logic problems:

Search for a solution of the satisfiability problem

Relevance of search strategies for knowledge-based systems:



if the knowledge
is not case-based

The problem solver nearly always has to solve
a satisfiability problem for constraints of the knowledge base!

➔ **All problem solvers search**

Example for a knowledge-based search engine: PROLOG

PROLOG is knowledge-based:

- **Knowledge base**

Facts and rules, dynamically extensible

- **Inference engine („Problem Solver“)**

deriving facts and rules automatically

- **Dialog component**

Input: Query

Output: yes / no, specification of unification applied in case of success,
write as a „side effect“

Yes: The predicate of the query can be concluded from knowledge base.

No: The predicate of the query cannot be concluded from knowledge base.

„No“ does not imply that it can be concluded that the predicate is false.

Application: Class Scheduling

Given finite sets Courses, Rooms, Time slots

Task: Generate an injective (one-to-one) function $C \rightarrow R \times T$

Strict Constraints (must be fulfilled in any case):

- **Certain courses must not take place at the same time.**
- **For some courses, certain time slots are not admitted.**
- **For some courses, certain rooms are not admitted.**

Soft constraints (may be violated):

- **Certain courses should not take place at some times.**
- **Certain courses should take place successively.**
- **Certain courses should not take place on the same day.**

Optimisation function:

- **fewest violations of soft criteria**
- **fewest free periods for certain study programmes**
- **most uniform distribution on different days for ...**

Application: Traveling Salesman Problem (TSP)

Given: Graph with node set V and weighted edges between the nodes

Task: Find a round trip traversing the graph edges reaching each node at least once.

Constraints:

- Only edges of the graph are to be used.

Optimisation function:

- Minimise the global edge costs !

Generalisation in logistic applications:

Constraints:

- Load and distribute goods obeying capacity restrictions !
- Consider time windows in which delivery may take place !

Soft criteria (may be violated):

- Certain edges have to be avoided.
- Certain time windows are unfavourable.

Application: Shortest Path Problem

Given: Graph with node set V and weighted edges between the nodes

Task: For two selected nodes S and T , find a path through the graph.

Constraints:

- Only edges of the graph are to be used.

Optimisation function:

- Minimise the global edge costs !

Generalisation in transport applications (public or individual):

Constraints:

- Edge costs depend on the time used.
- Travelers are subject to individual constraints that may value certain edges in a different way or make them even unusable.

Soft criteria (may be violated):

- Certain edges have to be avoided
- Certain time windows are unfavourable

Constraint Satisfaction Problem (CSP)

Specification of a CSP:

- **set of variables**
- **domains of definition**
- **constraints: relations between variables (strict or soft)**
(normally, equations or inequalities)
- **optimisation criterion**
(normally, a real-valued function on the variables which has to be minimised or maximised)

valid solution:

assignment of values to all variables such that all strict constraints are satisfied

optimal solution:

valid solution optimising the optimisation criterion

Constraint Solvers are programs which find a valid or even optimal solution for a given CSP automatically.

Traversing search graphs

1. search method: Find a global solution via partial solutions

- **Node: describes state in search domain**
 - **State: Assigning values to variables**
Each state has got an evaluation.
- **Edge: transition of a state into a subsequent state**
(usually feasible in one direction only)
 - **Subsequent state: Assign a value to a new variable**
keeping the values for the already assigned variables
- **Initial node: initial state**
(is always unique)
 - **Initial node: No variable has got a value.**
- **Final node: final state wanted (problem solution)**
(several ones are admissible)
 - **Final node: All specified variables have got admissible values.**

Traversing search graphs

Different search goals are possible:

- 1) Find some solution or detect that there is none.
 - 2) Find further solutions or detect that there are none.
 - 3) Find all solutions.
 - 4) Find an optimal solution or at least a rather good one.
- Expansion of a node: Compute all subsequent resp. adjacent nodes

Different search strategies differ in:

Which node has to be expanded next?

Special case:

- **Search graph is a search tree**
(makes the path from initial node to each final node unique)

Example for search trees in CSP

Constraint system:

- 1) $(2 < x < 4)$
- 2) $(0 < y < 6)$
- 3) $(x + y > 7)$
- 4) $(x \cdot y < 10.5)$

**Domain of definition
for valid solutions:**

$x, y \in \mathbf{Q}$,
at most k positions after the decimal point

**Optimisation
criterion:**

Minimise $|y - x|$

for bounded k :

- finite search space
- several valid solutions
- always 1 optimal solution

for unbounded k :

- infinite search space
- infinitely many valid solutions
- no optimal solution

Example for search trees in CSP

Constraint system:

- 1) $(2 < x < 4)$
- 2) $(0 < y < 6)$
- 3) $(x + y > 7)$
- 4) $(x \cdot y < 10.5)$

Domain of definition for valid solutions:

$x, y \in \mathbf{Q}$,
at most k positions after the decimal point

Optimisation criterion:

Minimise $|y - x|$

CSP variables:

- Assign values to the 8 variables x_0, x_1, x_2, x_3 and y_0, y_1, y_2, y_3
where $x = x_0 . x_1 x_2 x_3$ and $y = y_0 . y_1 y_2 y_3$
and x_i and y_i are the respective decimal digits (integer numbers between 0 and 9).

Nodes and successor definitions:

- Each node in the search network assigns either the same number of digits for x as for y with values (type 1) or one digit more for y than for x (type 2).
- A successor of type 1 is a node of type 2 with the same values as the predecessor plus one more value for a digit for y .
- A successor of type 2 is a node of type 1 with the same values as the predecessor plus one more value for a digit for x .

Example for search trees in CSP

Constraint system:

- 1) $(2 < x < 4)$
- 2) $(0 < y < 6)$
- 3) $(x + y > 7)$
- 4) $(x \cdot y < 10.5)$

Domain of definition for valid solutions:

$x, y \in \mathbf{Q}$,
at most k positions after the decimal point

Optimisation criterion:

Minimise $|y - x|$

Nodes and successor definitions:

- Each node in the search network assigns either the same number of digits for x as for y with values (type 1) or one digit more for y than for x (type 2).
- A successor of type 1 is a node of type 2 with the same values as the predecessor plus one more value for a digit for y .
- A successor of type 2 is a node of type 1 with the same values as the predecessor plus one more value for a digit for x

**depends on
good luck**

Optimum expansion strategy for this problem:

- The initial node assigns $x_0=2$, $y_0=4$ (type 1). All other digits are not yet assigned.
- Expand the initial node and the successors such that you come to the optimal solution $(x=2.176, y=4.825)$ fastest possible.

Uninformed Search Strategies

In general, only *blind (uninformed) search* is possible:

There is no information about good search directions (the target is only recognised on arrival)

Possible expansion strategies:

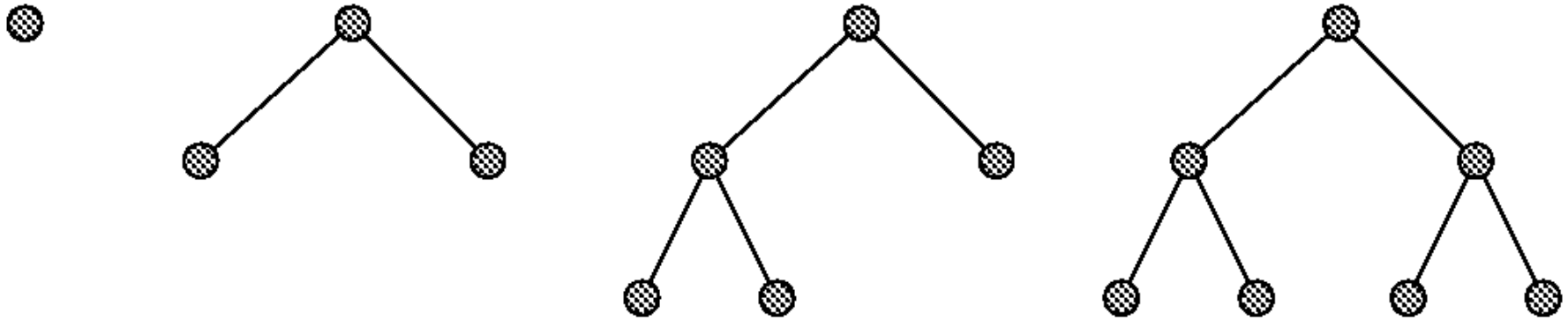
- Valid nodes are expanded first.
- The rightmost valid node on the next level is expanded.
- ...

Systematic search strategies:

1. breadth first search
2. depth first search
3. best first search

Uninformed Search Strategies

breadth first search:



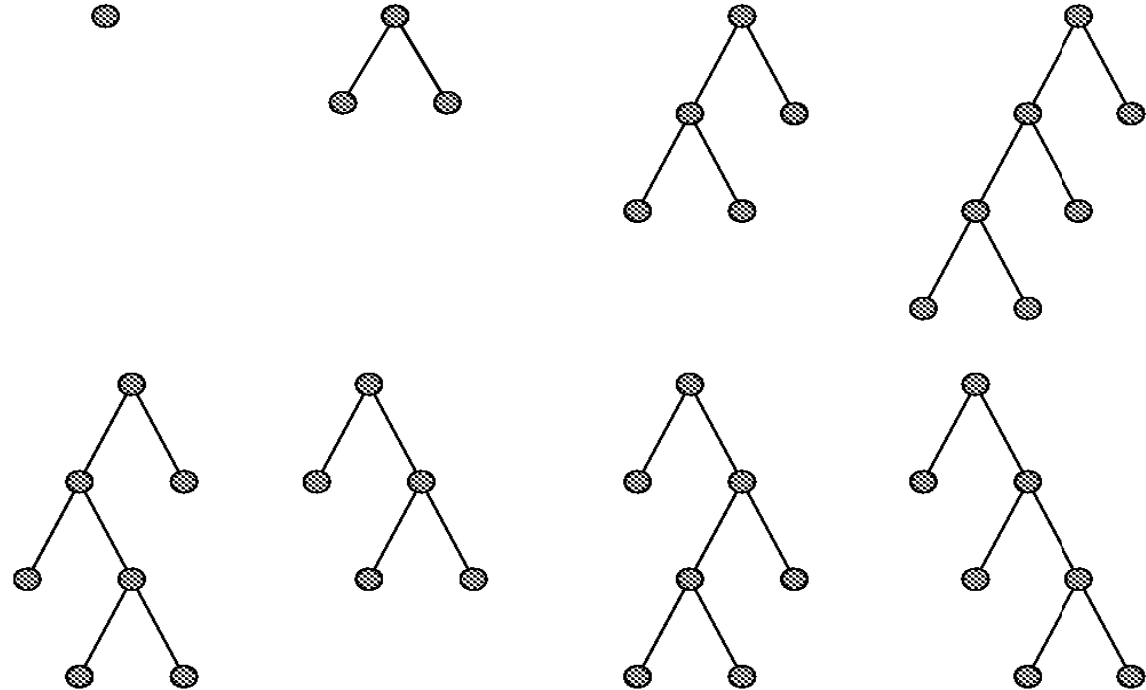
problem size: depth of search tree

Exponential time and space

for AI search procedures not relevant in most cases

Uninformed Search Strategies

depth first search:



Exponential time

problem size: depth of search tree

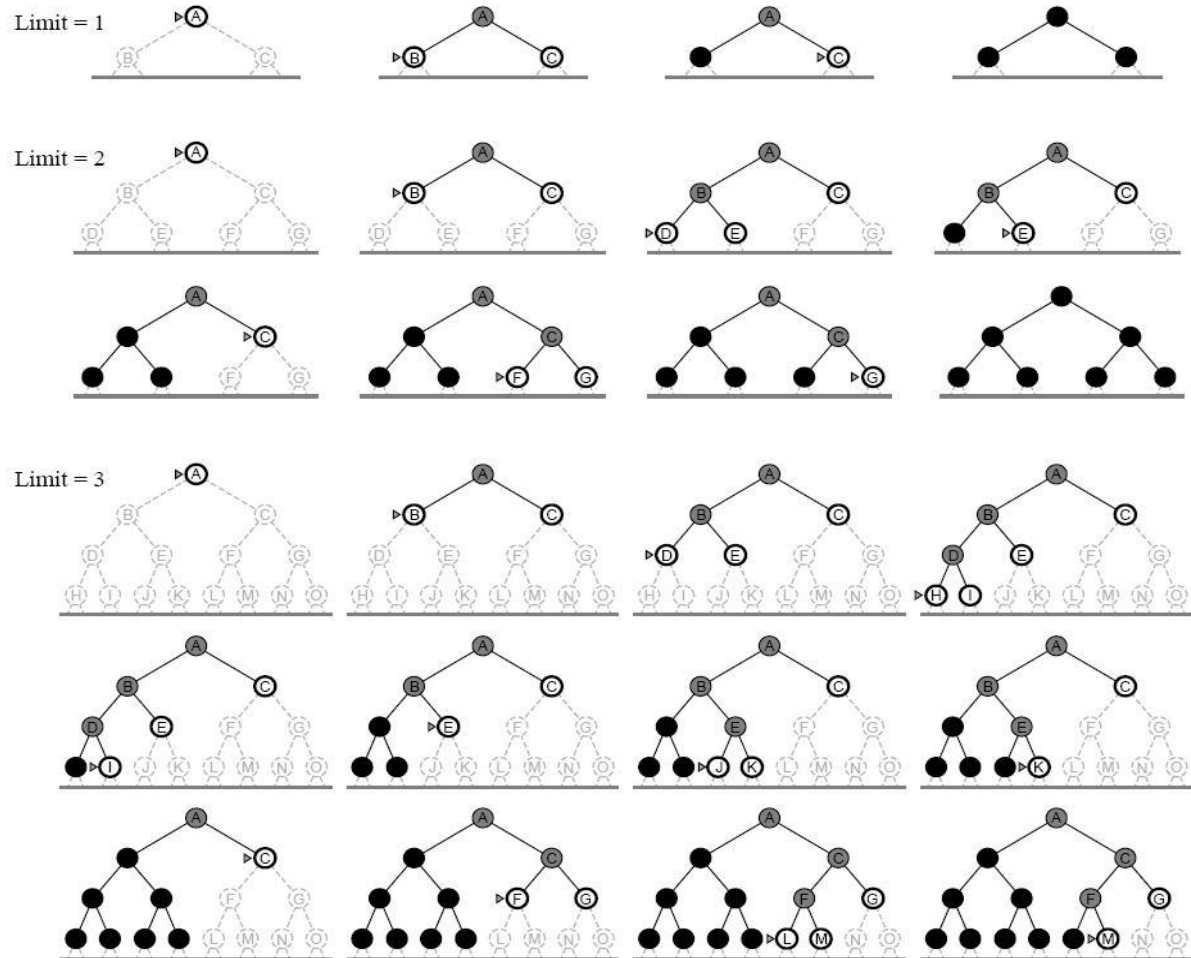
Linear space

The „normal case“ for standard AI procedures

Uninformed Search Strategies

bounded depth first search:

- Execute depth first search only up to limited search level.
- If not successful, increase limit for search level and start depth first search again.



Uninformed Search Strategies

best first search:

- Additional information: Evaluation label for the nodes.
- Search target: Find the best solution first (and the others later).
- Expand the node with best evaluation first.

→ *Mixture of depth first and breadth first searches*

In the *worst case* this is no better than breadth first search:

Exponential effort for time and space

Problem size:

Depth of search tree

For good evaluation functions, *the average case* is much better!

For special problems, even the *worst case* is much better:

Example: Special case „Shortest Path Problem“:

Dijkstra's algorithm (**quadratic** effort for time, **linear** for space)

Problem size: Number of nodes

Uninformed Search Strategies

Dijkstra's algorithm for weighted graphs

(special case of best first search)

↓
For all edges (u,v) there is a weight function:
 $length(u,v) :=$ length of an edge from node u to node v

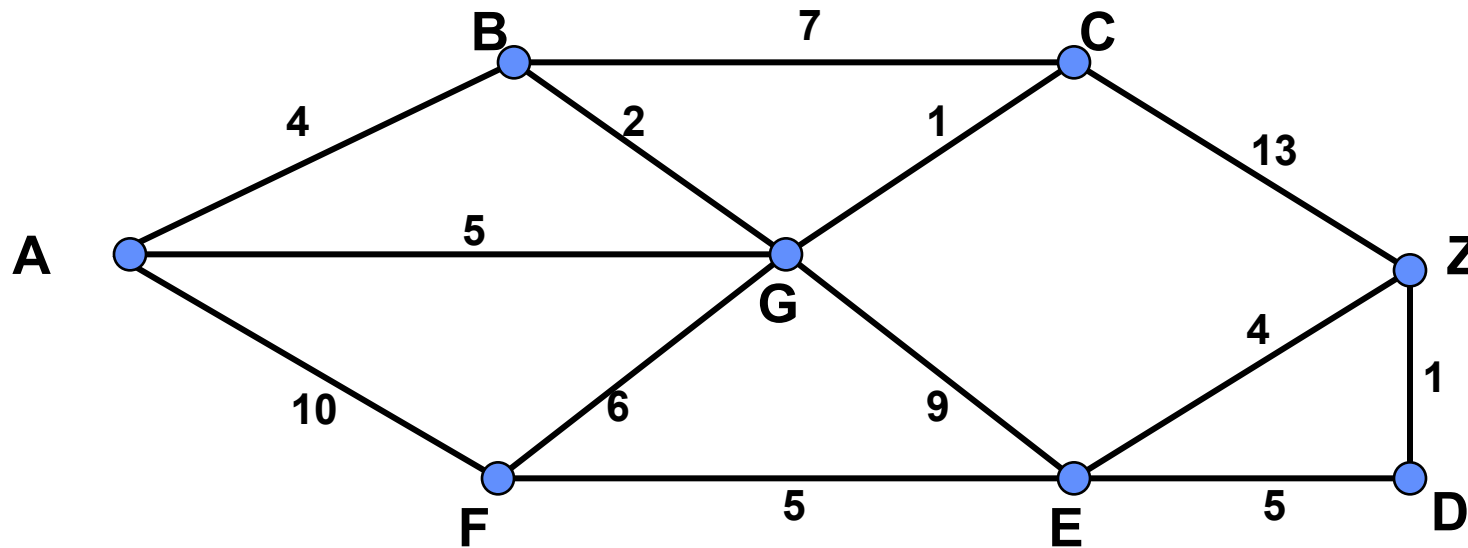
Requirement for edge weights: All lengths have to be nonnegative.

Algorithm for the search of a path from S to T having minimal global edge length:

- Put S into the set **Done**. Label S by $distance(S) := 0$.
Put all other nodes into the set **YetToCompute**.
Label all neighbors N of S by $distance(N) := length(S,N)$
and all other nodes by $distance(V) := \infty$.
- Repeat:
 Choose node V from **YetToCompute** with minimum $distance(V)$
 and shift V to the set **Done**.
 Update all neighbors N of V that are still in **YetToCompute**:
 $distance(N) := \min \{distance(N), distance(V) + length(V,N)\}$.
until $V = T$

Proposition: This algorithm expands all nodes with a path length shorter than to T.

Example for Dijkstra's algorithm



Shortest path from G to Z: $G \rightarrow E \rightarrow Z$ (13 units)

Node (distance from G, direct predecessor):

A(5,G)

A(5,G)

A(5,G)

B(2,G)

B(2,G)

C(1,G)

D(∞)

→

D(∞)

→

D(∞)

→

D(∞)

→

D(∞)

→

D(14,E)

E(9,G)

E(9,G)

E(9,G)

E(9,G)

E(9,G)

F(6,G)

F(6,G)

F(6,G)

F(6,G)

Z(14,C)

Z(∞)

Z(14,C)

Z(14,C)

Z(14,C)

Z(14,C)

Z(13,E)

Informed (Heuristic) Search Strategies

Given the following kind of information for weighted graphs:

Distance function $h(\text{state})$ being an *estimated* measure for the real distance to the target

- easily computable
- but accurate enough not to lead the search procedure to the wrong target

$h()$ provides a nonnegative value: The smaller the value, the closer the target

Application: „Hill climbing“

- Informed add-on to **depth first search**:
- Among the possible candidates, expand the node with best heuristic value.
- In case of backtracking expand the next best node respectively.

Main problem: Long halt in local maxima

Informed (Heuristic) Search Strategies

Given the following kind of information for weighted graphs:

Distance function $h(\text{state})$ being an *estimated* measure for the real distance to the target

- easily computable
- but accurate enough not to lead the search procedure to the wrong target

$h()$ provides a nonnegative value: The smaller the value, the closer the target

Application: Optimistic hill climbing

- Special case of informed add-on to **depth first search**
- Expand only the node with best heuristic value.
- Backtracking is omitted: If heuristic value was wrong, the best result will not be found.

Main problem: Getting stuck in local maxima

Informed (Heuristic) Search Strategies

Given the following kind of information for weighted graphs:

Distance function $h(\text{state})$ being an *estimated* measure for the real distance to the target

- easily computable
- but accurate enough not to lead the search procedure to the wrong target

$h()$ provides a nonnegative value: The smaller the value, the closer the target

Application: A* algorithm

- Informed add-on to **best first search**
- Expand the node where the sum of node label **plus** heuristic function is minimum.

Weitere Infos für die Anwendung von A* in öffentlichen Verkehrsnetzen:

Seminarvortrag und Ausarbeitung von Stefan Görlich, SS 2005, Nr. 5

<http://www.fh-wedel.de/archiv/iw/Lehrveranstaltungen/SS2005/SeminarKI.html>

Informed (Heuristic) Search Strategies

A* algorithm for weighted graphs

(Generalisation of Dijkstra's algorithm)

(State evaluation = Node evaluation)

Requirement for edge weights:

All edge lengths must be nonnegative.

Requirement for heuristic function $h_T(u)$ for estimating the real distance $d_T(u)$ to target node T:

Admissibility: $h_T(u) \leq d_T(u)$

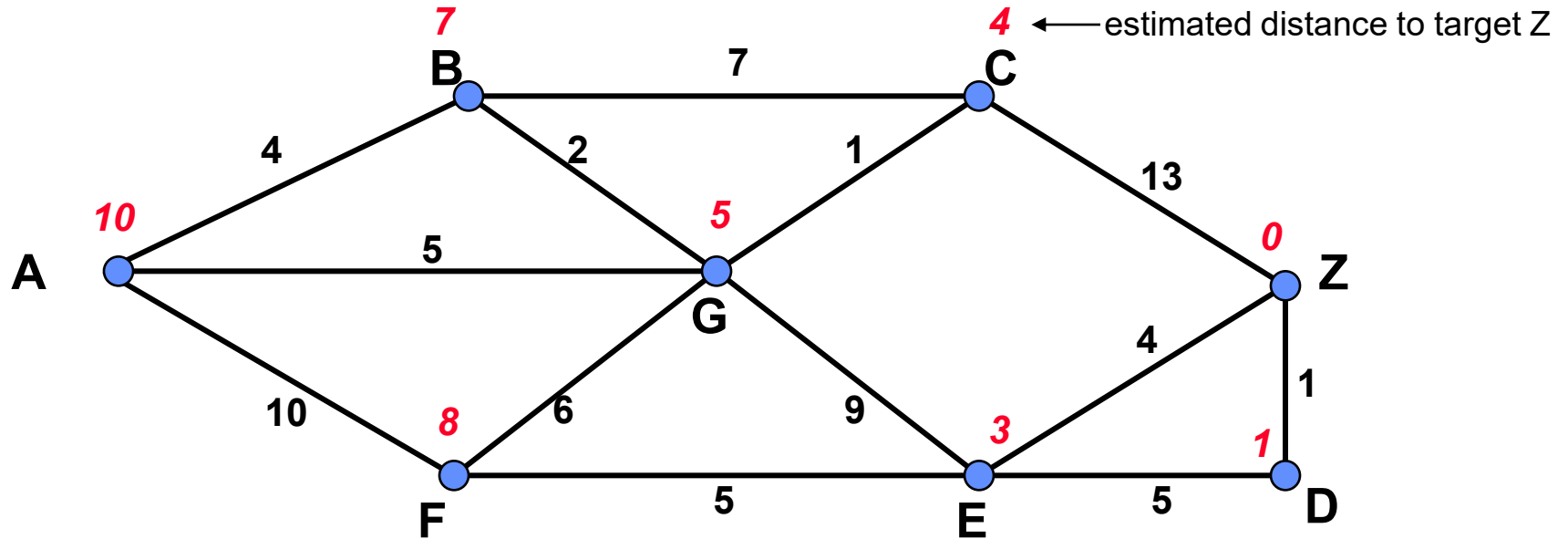
Monotonicity: $h_T(u) \leq h_T(v) + \text{length}(u,v)$

Algorithm for the search of a path from S to T having minimal global edge length:

- Put S into the set **Done**. Label S by $\text{distance}(S) := 0$.
Put all other nodes into the set **YetToCompute**.
Label all neighbors N of S by $\text{distance}(N) := \text{length}(S,N)$ and
 $\text{estimatedTotal}(N) := \text{distance}(N) + h_T(N)$
and all other nodes by $\text{distance}(V) := \infty$ and $\text{estimatedTotal}(V) := \infty$.
 - Repeat:
 - Choose node V from **YetToCompute** with minimum $\text{estimatedTotal}(V)$
and shift V to the set **Done**.
 - Update all neighbors N of V that are still in **YetToCompute**:
 $\text{distance}(N) := \min \{ \text{distance}(N), \text{distance}(V) + \text{length}(V,N) \}$.
 $\text{estimatedTotal}(N) := \text{distance}(N) + h_T(N)$ (if update is necessary).
- until V = T

Proposition: This algorithm expands all nodes with an **estimatedTotal** shorter than to T.

Example for A* algorithm



Shortest path from G to Z: $G \rightarrow E \rightarrow Z$ (13 units)

Node (real distance from G, direct predecessor, estimated total to target):

A(5,G,15)		A(5,G,15)		A(5,G,15)		A(5,G,15)
B(2,G,9)		B(2,G,9)				
C(1,G,5)						
D(∞)	→	D(∞)	→	D(∞)	→	D(14,E,15)
E(9,G,12)		E(9,G,12)		E(9,G,12)		
F(6,G,13)		F(6,G,14)		F(6,G,14)		F(6,G,14)
Z(∞)		Z(14,C,14)		Z(14,C,14)		Z(13,E,13)

Informed (Heuristic) Search Strategies

A* algorithm for weighted graphs

(Generalisation of Dijkstra's algorithm)

Requirement for edge weights: All edge lengths must be nonnegative.

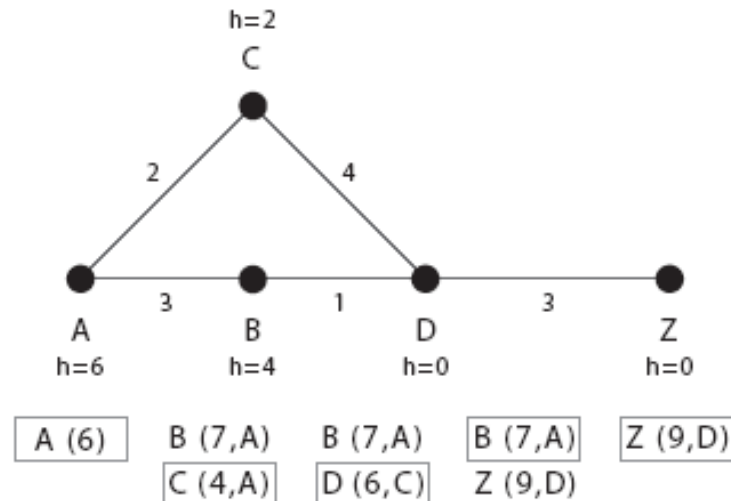
Requirement for heuristic function $h_B(u)$ for estimating the real distance $d_B(u)$ to target node B:

Admissability: $h_B(u) \leq d_B(u)$

What happens if monotonicity is abandoned ?

$$h_B(u) \leq h_B(v) + \text{length}(u,v)$$

Example:



Aus: Diplomarbeit Andre Keller (SS 2008)

Error: D will not be updated anymore because it is already in **Done**

Informed (Heuristic) Search Strategies

A* algorithm for weighted graphs

(Generalisation of Dijkstra's algorithm)

(State evaluation = Node evaluation)

Requirement for edge weights: All edge lengths must be nonnegative.

Requirement for heuristic function $h_T(u)$ for estimating the real distance $d_T(u)$ to target node T:

Admissability only: $h_T(u) \leq d_T(u)$

Algorithm for the search of a path from S to T having minimal global edge length:

- Put S into the set **Done**. Label S by $distance(S) := 0$.
Put all other nodes into the set **YetToCompute**.
Label all neighbors N of S by $distance(N) := length(S, N)$ and
 $estimatedTotal(N) := distance(N) + h_T(N)$
and all other nodes by $distance(V) := \infty$ and $estimatedTotal(V) := \infty$.
- Repeat:
Choose node V from **YetToCompute** with minimum $estimatedTotal(V)$
and shift V to the set **Done**.
Update all neighbors N of V from **Done and YetToCompute**:
 $distance(N) := \min \{distance(N), distance(V) + length(V, N)\}$.
 $estimatedTotal(N) := distance(N) + h_T(N)$ (if update is necessary).
If an update occurred to a neighbor N* of Done: Shift N* back to YetToCompute
until $V = T$

Pruning search space strategies

For the 1. search method introduced so far:

Approaching global solutions via partial solutions:

Any strategy must **backtrack** to earlier assignment stages in the search tree when no solution can be found with the current assignments. It should be avoided to do this only when all assignments have been explicitly performed.

Strategies for pruning the search space:

1. Partial Testing

- Test constraints having variables only that have already assigned values.
- States in which some constraints are violated already *may* not be expanded further, but rather traced back. If there are no descendant nodes anymore and no solution is found, the inference **must** trace back.

2. Forward Checking

- Reduce all domains for variables not assigned such that the future assignment still has a chance to be feasible.
- Trace back if this leads to empty domains.

Pruning search space strategies

Example for not pruning at all:

8-queens-problem (solution by Bratko, 2nd method)

Knowledge base:

```
queens2(YList) :-  
  permutation([1,2,3,4,5,6,7,8], YList),  
  admissible(YList).
```

```
permutation([], []).  
permutation([First|Tail], ResultList) :-  
  permutation(Tail, ResultTail),  
  del(First, ResultList, ResultTail).
```

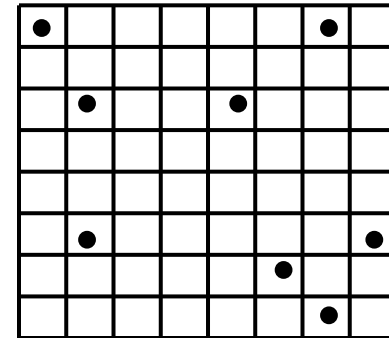
```
admissible([]).  
admissible([Y1|Others]) :-  
  admissible(Others),  
  conflictFree(Y1, Others, 1).
```

```
conflictFree(_, [], _).
```

```
conflictFree(Y, [Y1|YTail], XDiff) :-  
  YDiff is Y1-Y,  
  YDiff =\= XDiff,  
  YDiff =\= -XDiff,  
  XDiff1 is XDiff + 1,  
  conflictFree(Y, YTail, XDiff1).
```

Query:

```
?-queens2(YList).
```



Pruning search space strategies

Example for partial testing:

8-queens-problem (solution by Bratko, 1st method)

Knowledge base:

queens1([]).

queens1([X/Y | Others]) :-
 queens1(Others),
 member(Y,[1,2,3,4,5,6,7,8]),
 conflictFree(X/Y,Others).

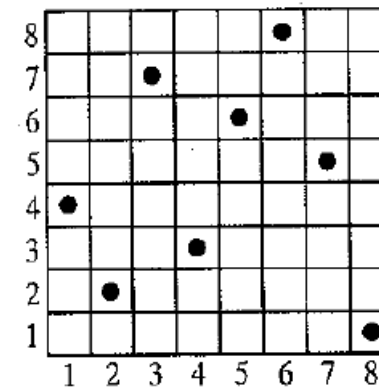
conflictFree(_,[]).

conflictFree(X/Y, [HeadX/HeadY | Others]) :-
 Y \neq HeadY,
 DiffY is HeadY - Y,
 DiffY \neq HeadX - X,
 DiffY \neq X - HeadX,
 conflictFree(X/Y,Others).

template([1/Y1,2/Y2,3/Y3,4/Y4,5/Y5,6/Y6,7/Y7,8/Y8]).

Query:

?- template(S), queens1(S).



Pruning search space strategies

Example for forward checking:

8-queens-problem (solution by Bratko, 3rd method)

Knowledge base:

queens3(YList) :-

```
sol(YList, [1,2,3,4,5,6,7,8],  
      [1,2,3,4,5,6,7,8],  
      [-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7],  
      [2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]).
```

```
sol([],[], DomainY, DomainU, DomainV).
```

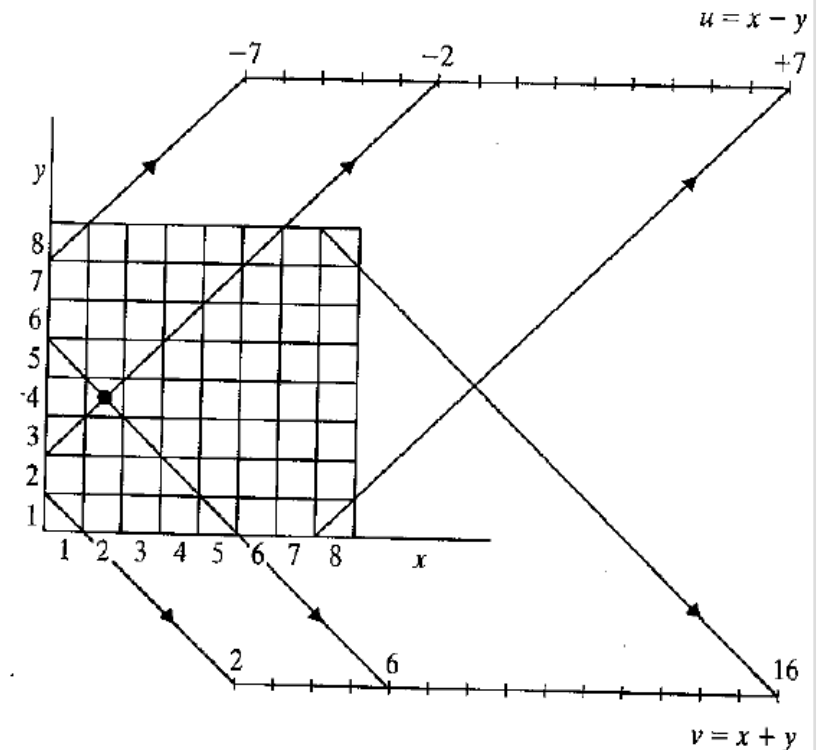
```
sol([Y | YTail], [X | XTail], DomainY, DomainU, DomainV) :-  
  del(Y,DomainY,ReducedDomainY),  
  U is X - Y,  
  del(U,DomainU,ReducedDomainU),  
  V is X + Y,  
  del(V,DomainV,ReducedDomainV),  
  sol(YTail, XTail, ReducedDomainY, ReducedDomainU,  
      ReducedDomainV).
```

```
del(Item, [Item|List], List).
```

```
del(Item, [First|Tail],[First|ResultTail]) :-  
  del(Item,Tail,ResultTail).
```

Query:

?-queens3(YList).



Traversing search graphs

Alternative 2. search method:

Systematic improvement of preliminary (global) solutions

- **Node: describes state in search domain**
 - **State: Assignment of values to all variables**
(not all of them need be admissible)
Each state has got an evaluation.
- **Edge: Transition of a state into an adjacent state**
(usually feasible in both directions)
 - **Adjacent state: New values for certain variables**
keeping all values for the other variables
- **Initial node: initial state** (is always unique)
 - **Initial node: Start with any assignment to the variables.**
(or apply a reasonable starting heuristic)
- **Final node: final state wanted (problem solution)**
(several ones are admissible)
 - **Final node: No adjacent state has got a better evaluation than the present one.**
(or some heuristic function is achieved)

General Optimisation Methods for CSP

For the 2. search method of systematic improvement:

Min-Conflicts procedure:

Idea:

- Start with an arbitrary assignment of values (valid or not).
- Assign new values for certain variables such that the new assignment bares fewer conflicts than the old one.

Advantages:

- happens to show good run time behaviour
- „repair strategy“ if something changes dynamically

Disadvantages:

- „Getting stuck“ in local minima
 - counter measures: random walk, **tabu list**, ...

Weitere Details zum Thema Constraintsysteme:

Seminarvortrag und Ausarbeitung von Stefan Schmidt, SS 2005, Nr. 6,

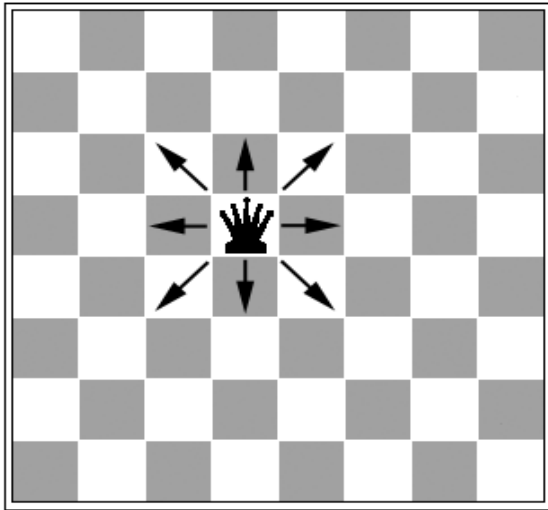
<http://www.fh-wedel.de/archiv/iw/Lehrveranstaltungen/SS2005/SeminarKI.html>

General Optimisation Methods for CSP

For the 2. search method of systematic improvement:

Min-Conflicts procedure:

Application: 8-queens-problem



Quelle: Seminarvortrag von Stefan Schmidt, SS 2005, Nr. 6,

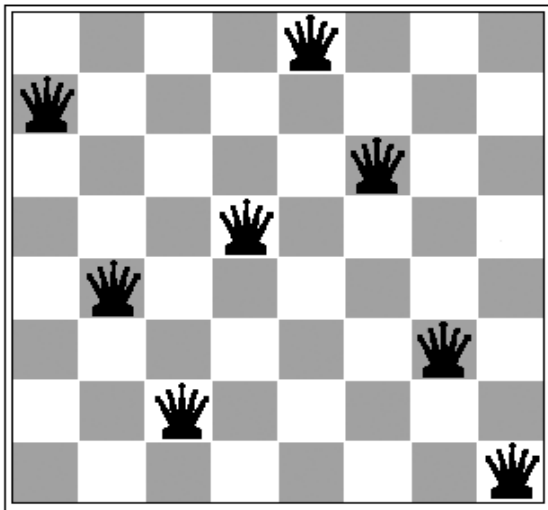
<http://www.fh-wedel.de/archiv/iw/Lehrveranstaltungen/SS2005/SeminarKI.html>

General Optimisation Methods for CSP

For the 2. search method of systematic improvement:

Min-Conflicts procedure:

Application: 8-queens-problem



Quelle: Seminarvortrag von Stefan Schmidt, SS 2005, Nr. 6,

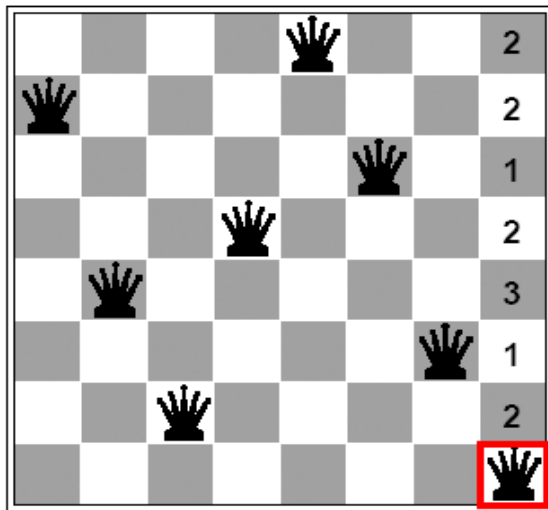
<http://www.fh-wedel.de/archiv/iw/Lehrveranstaltungen/SS2005/SeminarKI.html>

General Optimisation Methods for CSP

For the 2. search method of systematic improvement:

Min-Conflicts procedure:

Application: 8-queens-problem



Quelle: Seminarvortrag von Stefan Schmidt, SS 2005, Nr. 6,

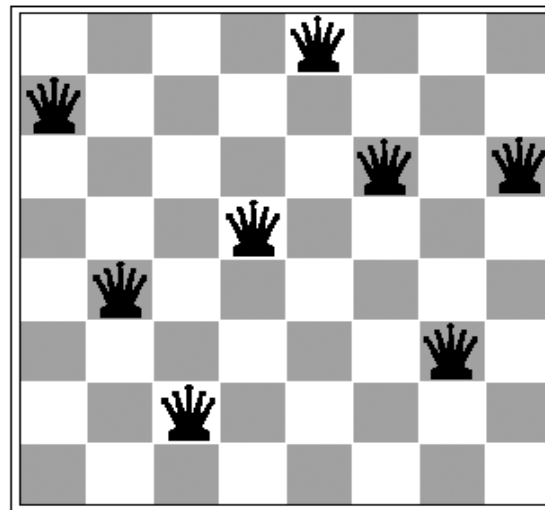
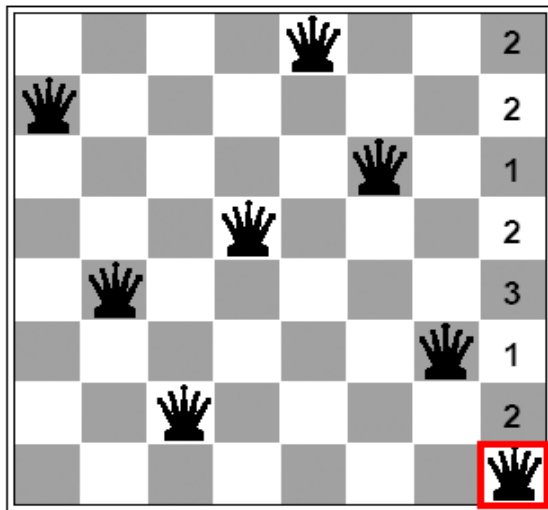
<http://www.fh-wedel.de/archiv/iw/Lehrveranstaltungen/SS2005/SeminarKI.html>

General Optimisation Methods for CSP

For the 2. search method of systematic improvement:

Min-Conflicts procedure:

Application: 8-queens-problem



Quelle: Seminarvortrag von Stefan Schmidt, SS 2005, Nr. 6,

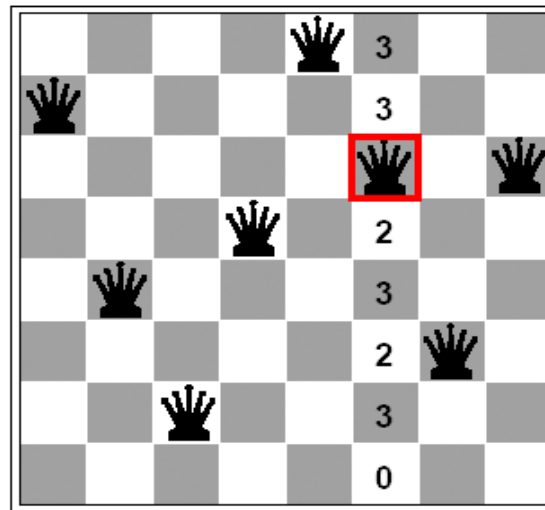
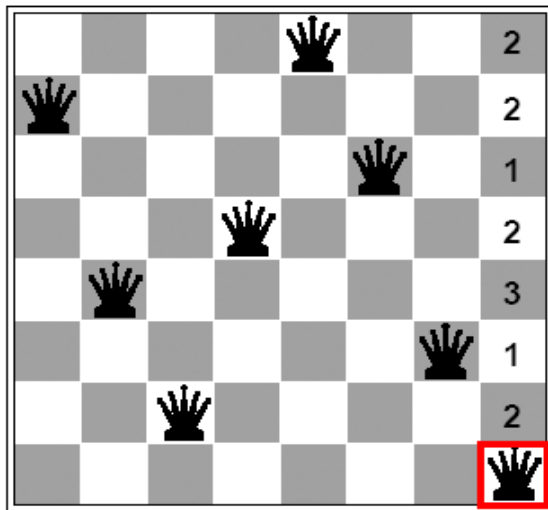
<http://www.fh-wedel.de/archiv/iw/Lehrveranstaltungen/SS2005/SeminarKI.html>

General Optimisation Methods for CSP

For the 2. search method of systematic improvement:

Min-Conflicts procedure:

Application: 8-queens-problem



Quelle: Seminarvortrag von Stefan Schmidt, SS 2005, Nr. 6,

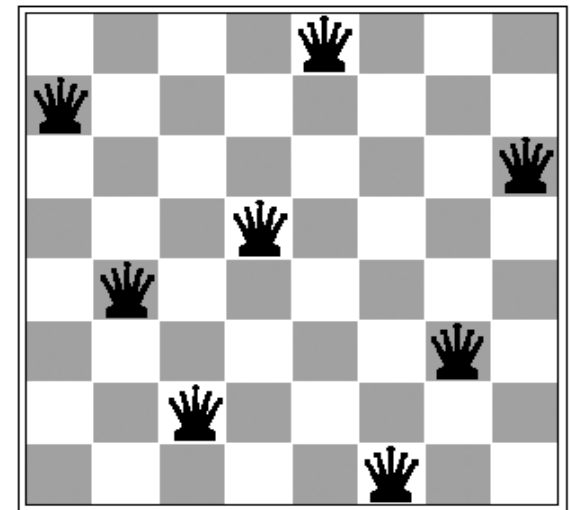
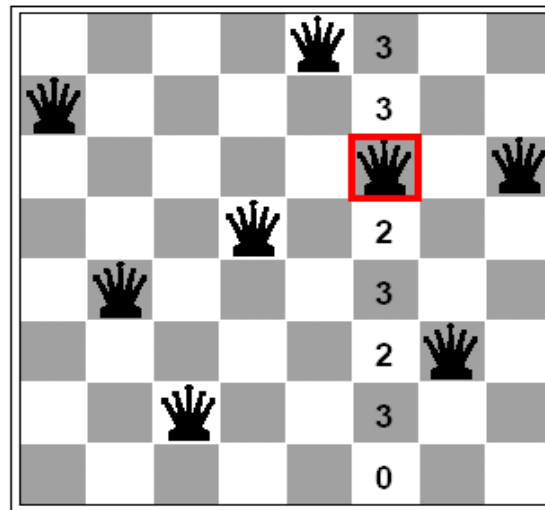
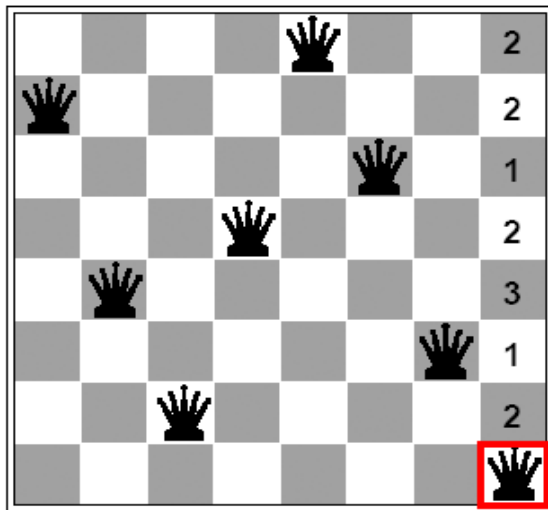
<http://www.fh-wedel.de/archiv/iw/Lehrveranstaltungen/SS2005/SeminarKI.html>

General Optimisation Methods for CSP

For the 2. search method of systematic improvement:

Min-Conflicts procedure:

Application: 8-queens-problem



Quelle: Seminarvortrag von Stefan Schmidt, SS 2005, Nr. 6,

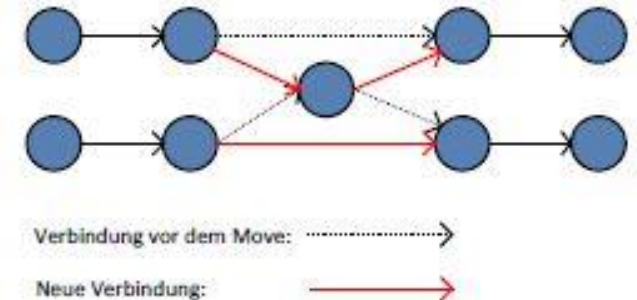
<http://www.fh-wedel.de/archiv/iw/Lehrveranstaltungen/SS2005/SeminarKI.html>

General Optimisation Methods for CSP

For the 2. search method of systematic improvement:

Working with tabu lists in search graphs:

- Determine a certain validity range for the algorithm, e.g. by a given number of operations
- Protocol all edges used in a transition from one state to another
- All edges used within the previous validity range are not to be used again, neither their counterdirection.



Further enhancement: Simulated annealing

- Admit temporary deteriorations.
- Diminish the tolerance bound for deterioration in the course of algorithmic progress gradually.

These methods will mainly be used in improvements of global solutions

- Good results in logistics (TSP generalisations)

Applications of Artificial Intelligence

Sebastian Iwanowski
FH Wedel

Chapter 4: Knowledge-Based Systems

4.1: Representation and Classification of Knowledge

Representation of knowledge: How ?



logic knowledge:

atoms

rules

derivation rules

facts

if ... then ...

resolution, unification

functional knowledge:

data

functions

function evaluation

object-oriented knowledge:

objects

methods

compiler / interpreter

declarative

procedural

control

knowledge

Classification of knowledge: What ?

The following criteria are mutually independent:

- **deep vs. shallow** (consider how a statement is composed of smaller units)
model-based vs. universally valid
- **certain vs. uncertain** (consider the probability of a statement)
deterministically vs. probabilistically
- **exact vs. fuzzy** (consider the accuracy of a statement)
quantitative vs. qualitative

Classification of knowledge: What ?

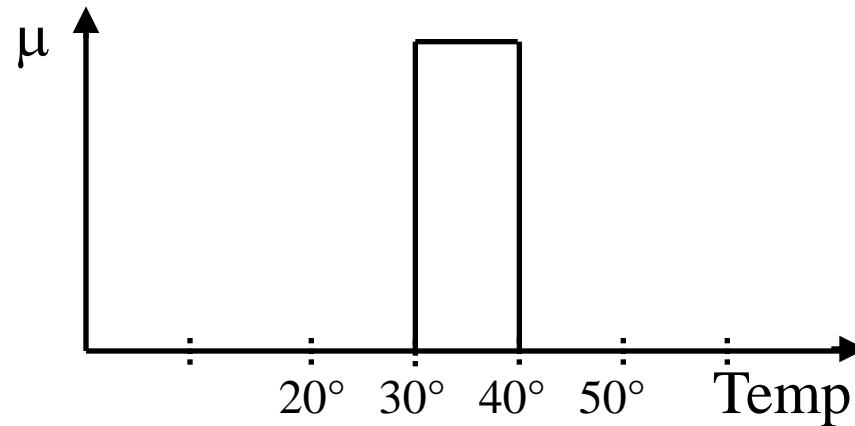
Example for distinguishing probability from accuracy:

- The train comes in 10 minutes. *certain, exact*
- The train comes in about 10 minutes. *certain, fuzzy*
- The train comes probably in 10 minutes. *uncertain, exact*
- The train comes probably in about 10 minutes. *uncertain, fuzzy*
- The probability that the train comes in 10 minutes is 0.9. *uncertain, exact*
- The plausibility range of the hypothesis that the train comes in 10 minutes is in (0,05; 0,95). *uncertain, exact*

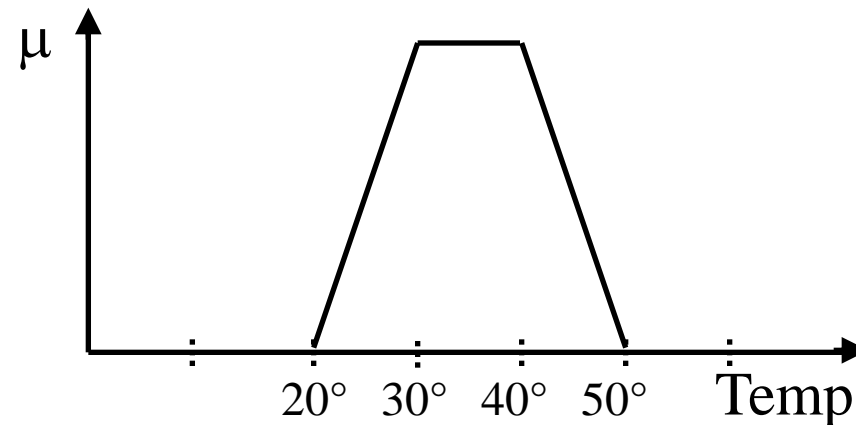
Classification of knowledge: What ?

Fuzzy sets as example for qualitative knowledge

exact set

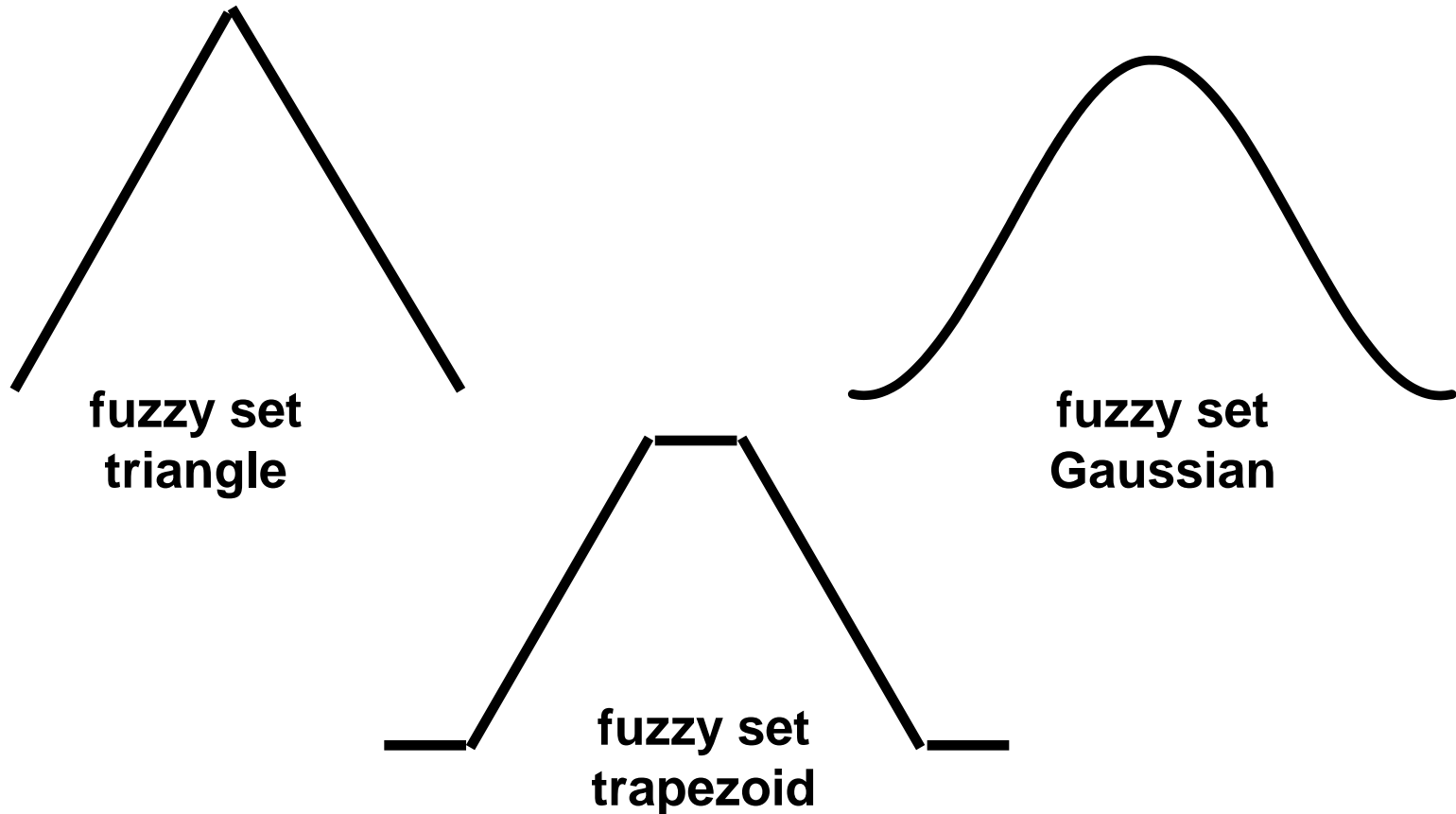


fuzzy set



Classification of knowledge: **What ?**

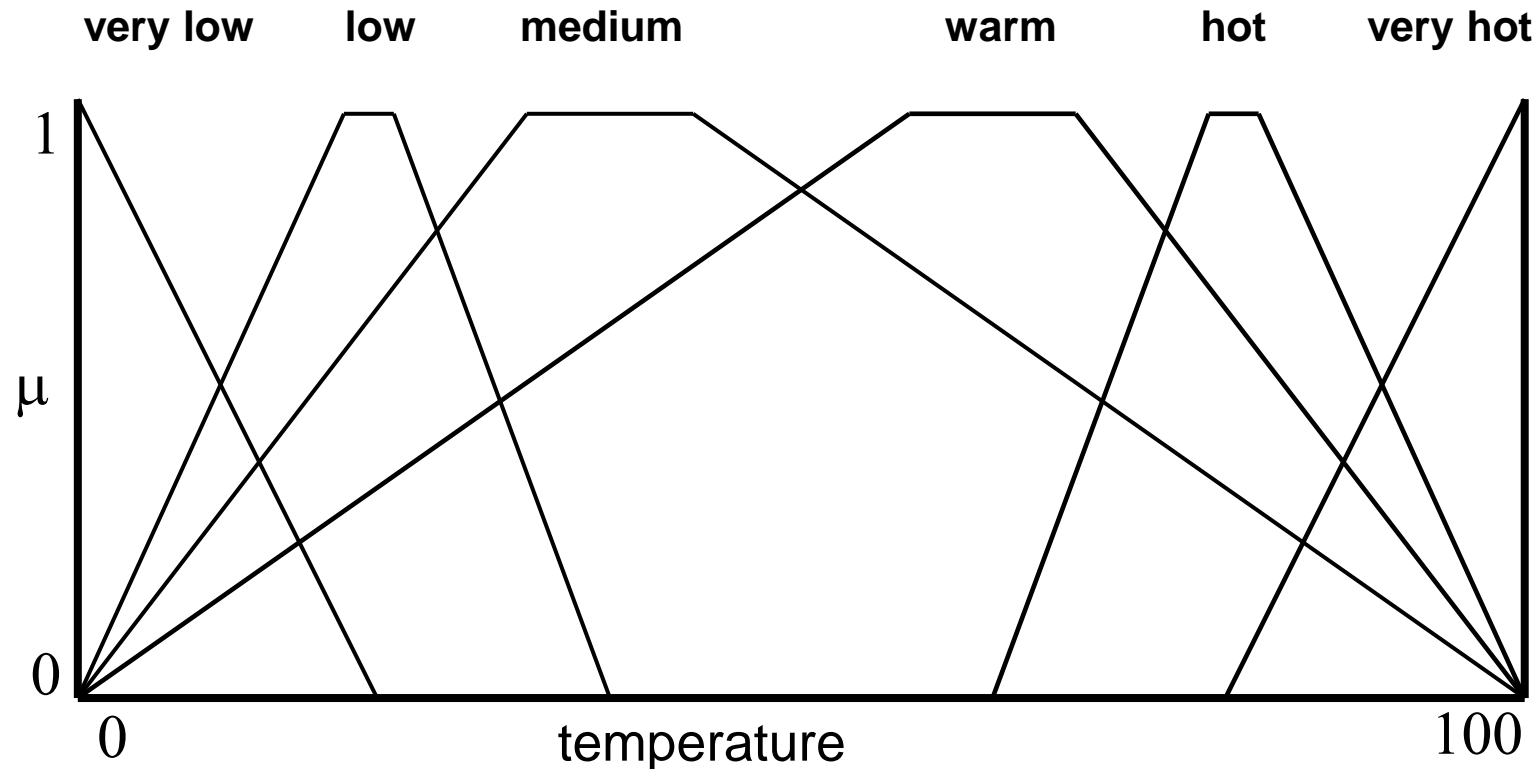
Fuzzy sets as example for qualitative knowledge



Classification of knowledge: What ?

Fuzzy sets as example for qualitative knowledge

The linguistic variable „temperature“



Classification of knowledge: What ?

Fuzzy sets as example for qualitative knowledge

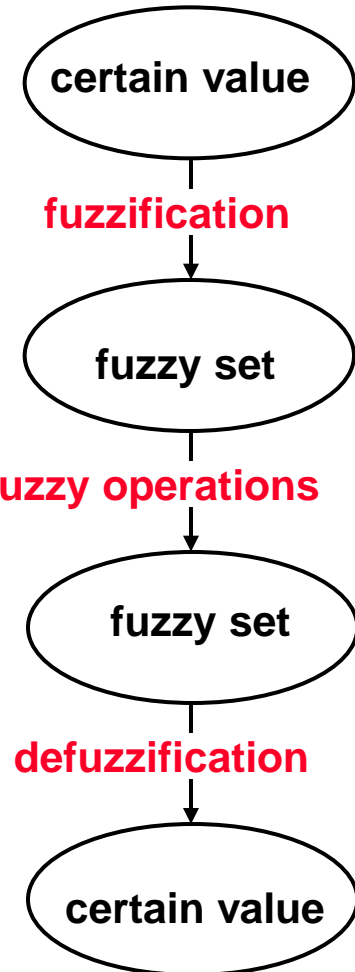
Prinziple of fuzzy technology:

Fuzzy operations:

operators for building new sets from old ones

rules for mapping sets to other sets

measurement



setting

Classification of knowledge: What ?

Fuzzy sets as example for qualitative knowledge

Examples for fuzzy operators:

- $\mu_C(\mathbf{x}) = \min \{ \mu_A(\mathbf{x}), \mu_B(\mathbf{x}) \} \quad \mathbf{x} \in X$

- $\mu_C(\mathbf{x}) = \max \{ \mu_A(\mathbf{x}), \mu_B(\mathbf{x}) \} \quad \mathbf{x} \in X$

- $\mu_C(\mathbf{x}) = 1 - \mu_A(\mathbf{x}) \quad \mathbf{x} \in X$

Classification of knowledge: What ?

Fuzzy sets as example for qualitative knowledge

you can have it more complicated:

- $\mu_C(\mathbf{x}) = \gamma \min\{\mu_A(\mathbf{x}), \mu_B(\mathbf{x})\} + \frac{1}{2} (1 - \gamma)(\mu_A(\mathbf{x}) + \mu_B(\mathbf{x}))$ ($\gamma \in [0,1]$)

What does this function compute ?

Classification of knowledge: **What ?**

Fuzzy sets as example for qualitative knowledge

Example for fuzzy rules:

if (distance = small)
and (velocity = large),
then (braking power = large)

if (distance = medium)
and (velocity = large),
then (braking power = medium)

Classification of knowledge: What ?

Representation of temporal knowledge

Allen's interval logic for the qualitative representation of time intervals

1. STARTS(t_1, t_2)

t_1 starts with t_2 but ends before t_2

2. FINISHES(t_1, t_2)

t_1 ends with t_2 but starts after t_2

3. DURING(t_1, t_2)

t_2 contains t_1 completely

4. BEFORE(t_1, t_2)

t_1 starts before t_2 , and t_1 and t_2 do not overlap or contain each other

5. OVERLAP(t_1, t_2)

t_1 starts before t_2 and ends after the start of t_2 and before the end of t_2

6. MEETS(t_1, t_2)

t_1 starts before t_2 and ends when t_2 starts

7. EQUAL(t_1, t_2)

t_1 and t_2 denote the same interval

Classification of knowledge: What ?

Representation of spatial knowledge

exact knowledge about an object

- geo coordinates in angle and degree (or floating point representation of the angle)
- relative distance of objects in m (with distance between the objects and relative angle)

qualitative knowledge about an object

- geo cell in which the object is located
- relative order (in front of, behind, left of, right of, etc.)

Classification of knowledge: **What ?**

Practical problem for temporal and spatial knowledge:

How exact should the knowledge be ?

- year, month, day, hour, second, millisecond, ...
- country, city, address, exact geo coordinates with certain digits, ...

Summary:

knowledge representation and classification

Various forms of knowledge representation

- **logical:** production rules
- **functional:** constraints
- **object-oriented:** frames, semantic networks

Various qualities of knowledge

- deep vs. shallow (consider how a statement is composed of smaller units)
- certain vs. uncertain (consider the probability of a statement)
- exact vs. fuzzy (consider the accuracy of a statement)

Applications of Artificial Intelligence

Sebastian Iwanowski
FH Wedel

Chapter 4:
Knowledge-Based Systems

4.2: Rule-Based Reasoning

Application from practice: Technical diagnosis

Run time system:

(knowledge-based systems call this **problem solver / inference engine**)

Input:

- Setting certain control inputs
- Observing values depending on this setting

Output:

- A unique instruction how to repair which component

This is where diagnostic systems do **not** differ !

Application from practice: Technical diagnosis

Knowledge-based diagnosis:

1) Knowledge acquisition: Input into knowledge base

- rule-based (symptom-based)
 - model-based
 - case-based
- } as alternatives

2) Knowledge structure

- depends on knowledge acquisition

3) Knowledge processing by the problem solver

- depends on knowledge structure

This is where diagnostic systems may differ !

1. Symptom-Based Diagnosis

Input to knowledge base:

- Causing and manifest faults for the overall system
- Possible symptoms (measurements)
- Relations between faults and symptoms (rules)
 - Symptoms may confirm a fault or even explain it.
 - Symptoms may exclude a fault.

Structure of knowledge base:

- Semantic network
- Feasible structures:
 - Fault networks (trees)
 - Decision trees

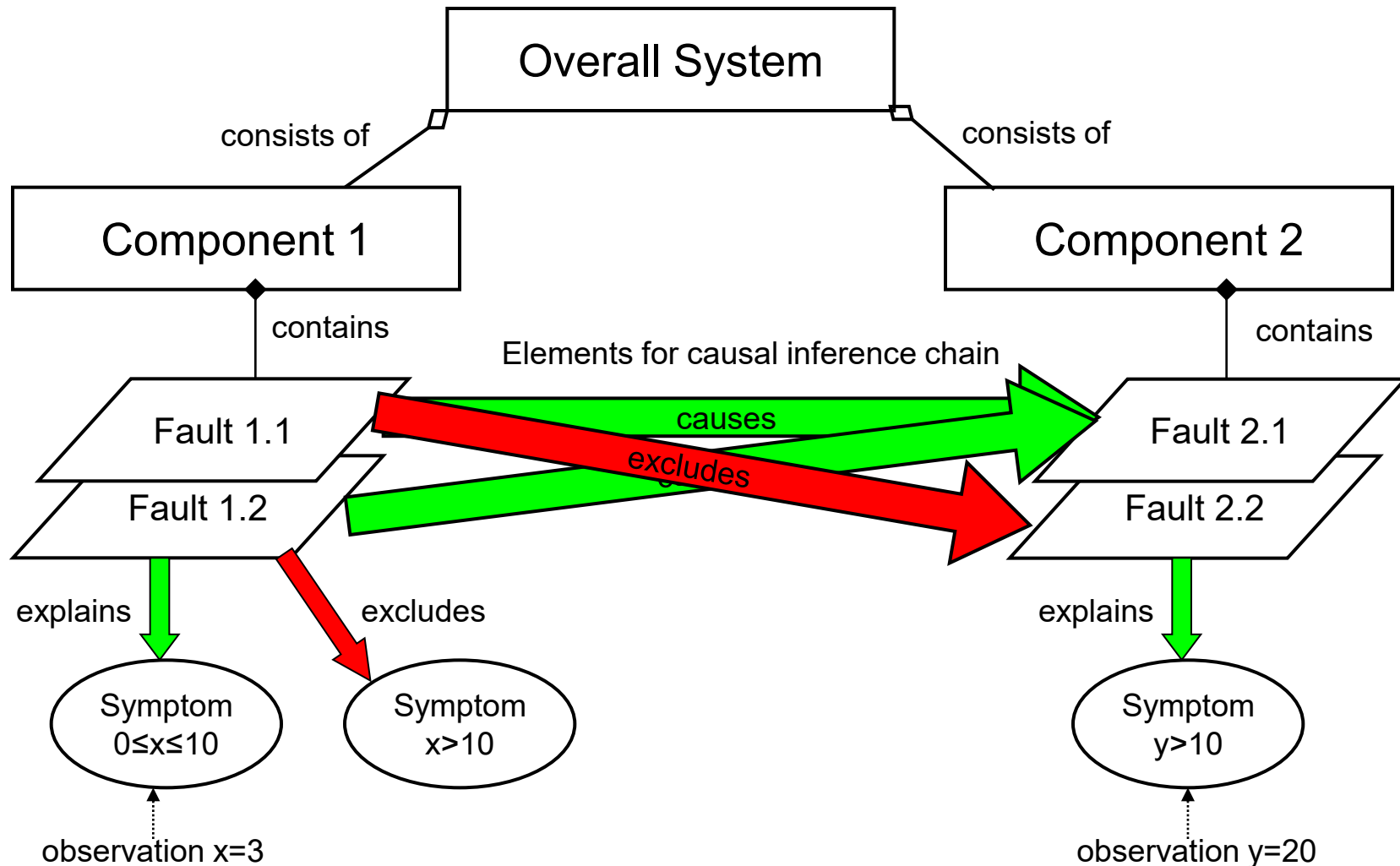
Task of inference engine:

- Navigation in semantic network

This is „classical“ expert system technology

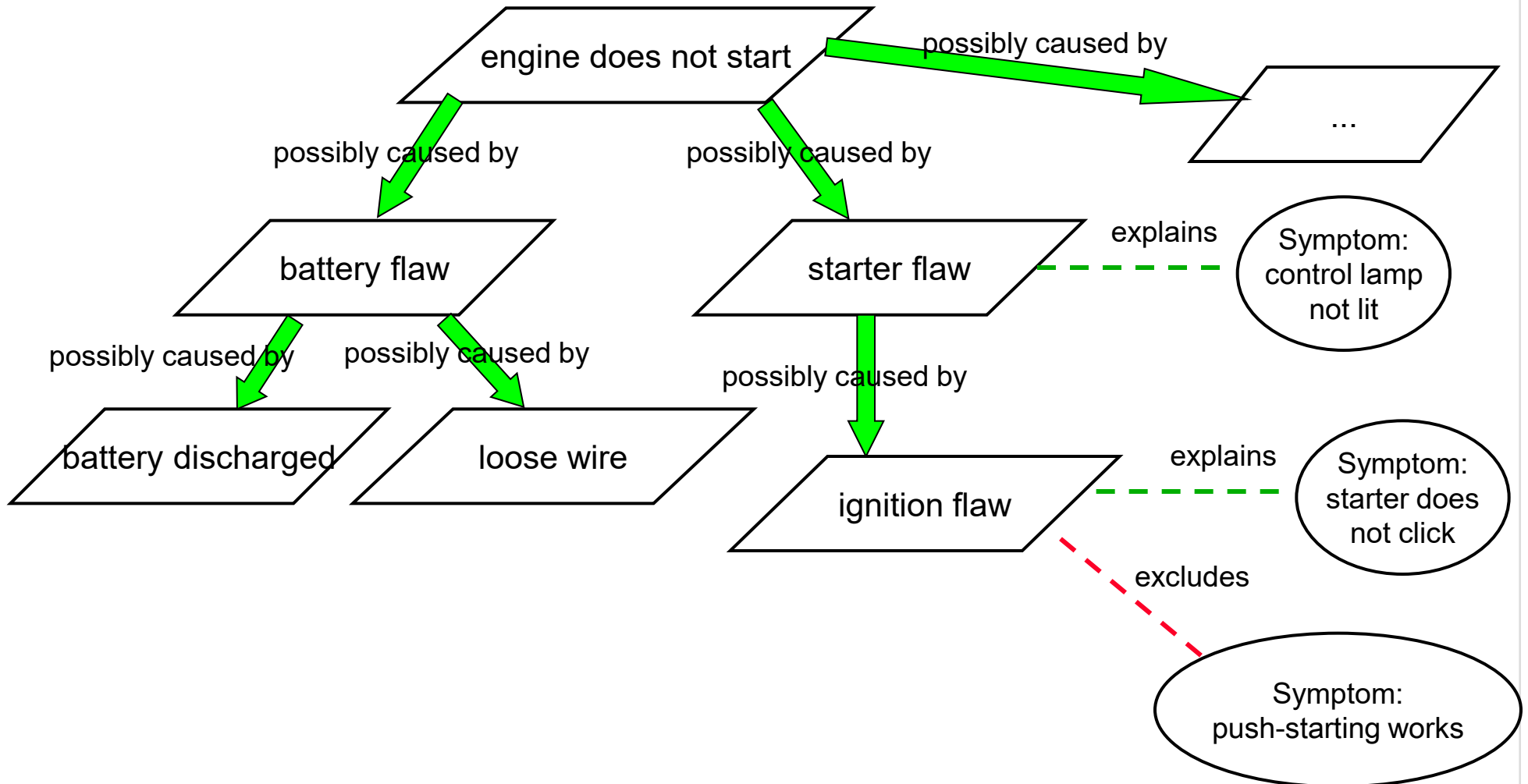
1. Symptom-Based Diagnosis

Example for elements of the knowledge base:



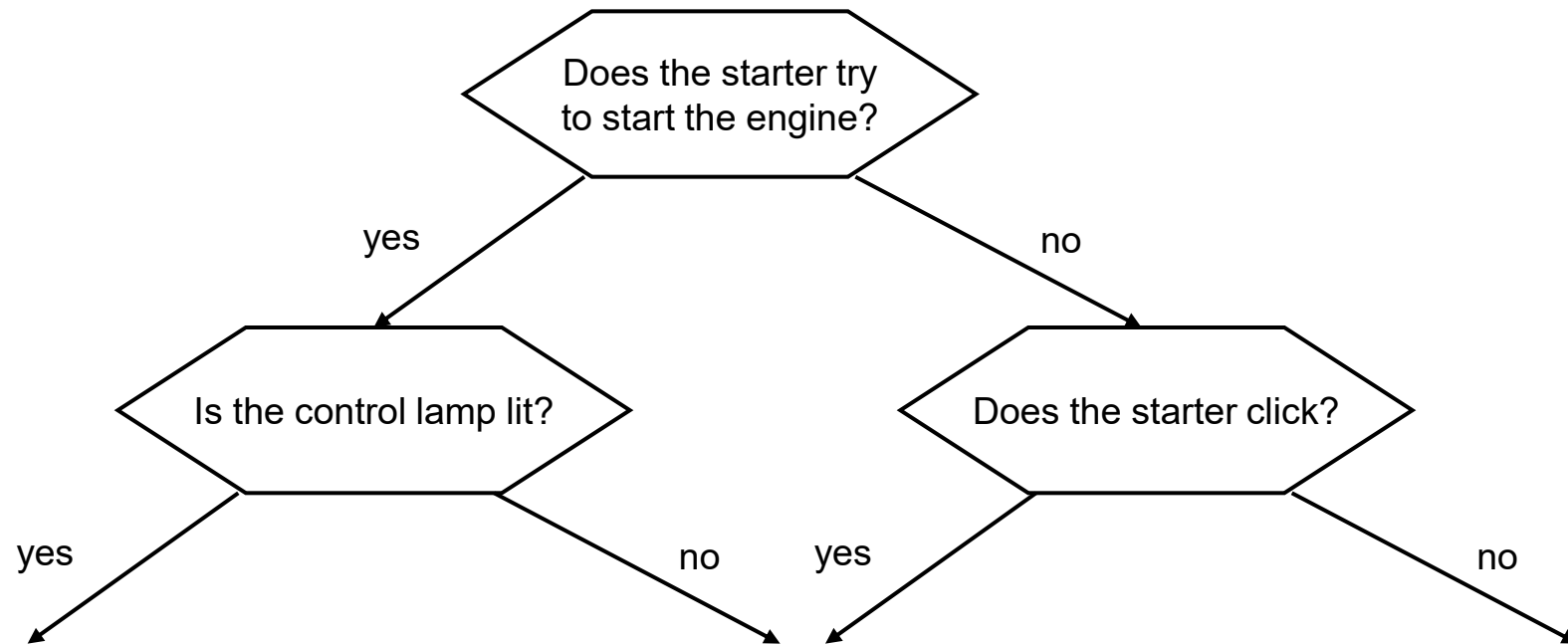
1. Symptom-Based Diagnosis

Example for a fault tree:



1. Symptom-Based Diagnosis

Example for a decision tree:



1. Symptom-Based Diagnosis

Job of inference engine:

- **Navigation in semantic network**
(e.g. fault tree or decision tree)
- **Possible start points of navigation:**
 - Suspected faults
 - Observed symptoms
- **Main task is evaluating and firing of rules:**
 - Insert a concluded result of one rule into the antecedent of another rule.
 - Work with probabilities and fuzzy rules.

Such input must be allowed for knowledge acquisition.

1. Symptom-Based Diagnosis

Advantages and Disadvantages:

- **Knowledge structure complies to terminology of experts.**
 - An expert can easily handle the knowledge acquisition component.
 - Knowledge acquisition costs a lot of time.
- **Knowledge is stored very goal-oriented.**
 - Diagnosis of run time component is fast.
 - Knowledge base may not easily be altered.
 - Reusability is a fundamental problem.
 - There are methods for reusing parts of knowledge though.

1. Symptom-Based Diagnosis

Advantages und Disadvantages:

- **Knowledge base does not contain deep knowledge.**
 - Every application domain is feasible in principle.
 - Knowledge base is often not complete.
 - Knowledge base is confusing and is thus not easily verifiable.

A lot of knowledge bases contain faults.

Applications of Artificial Intelligence

Sebastian Iwanowski
FH Wedel

Chapter 4:
Knowledge-Based Systems

4.3: Model-Based Reasoning

3. Model-Based Diagnosis

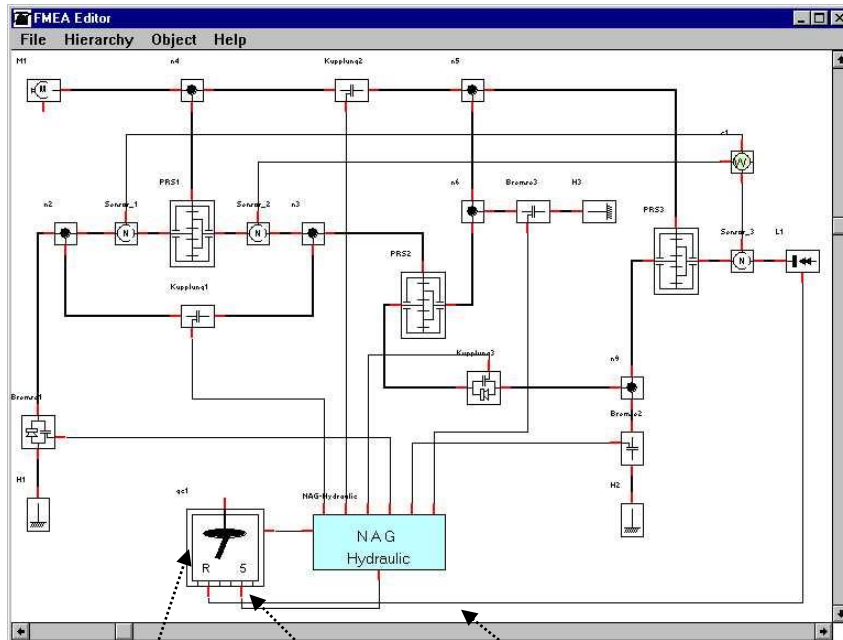
Goal:

- fast knowledge acquisition
- exact and provable solution of problem solver

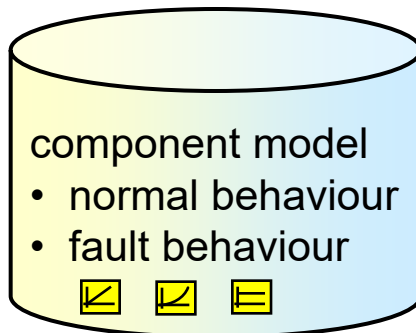
Challenge:

- reasonable response time of problem solver at run time

3. Model-Based Diagnosis



component port link



System model:

Which components of which type are connected in which way?

→ available from CAD data

Component models:

How do values depend on each other lying at ports of the component?

→ to be modeled once per component type

→ Model is reusable for all systems containing components of this type.

3. Model-Based Diagnosis

Input to knowledge base:

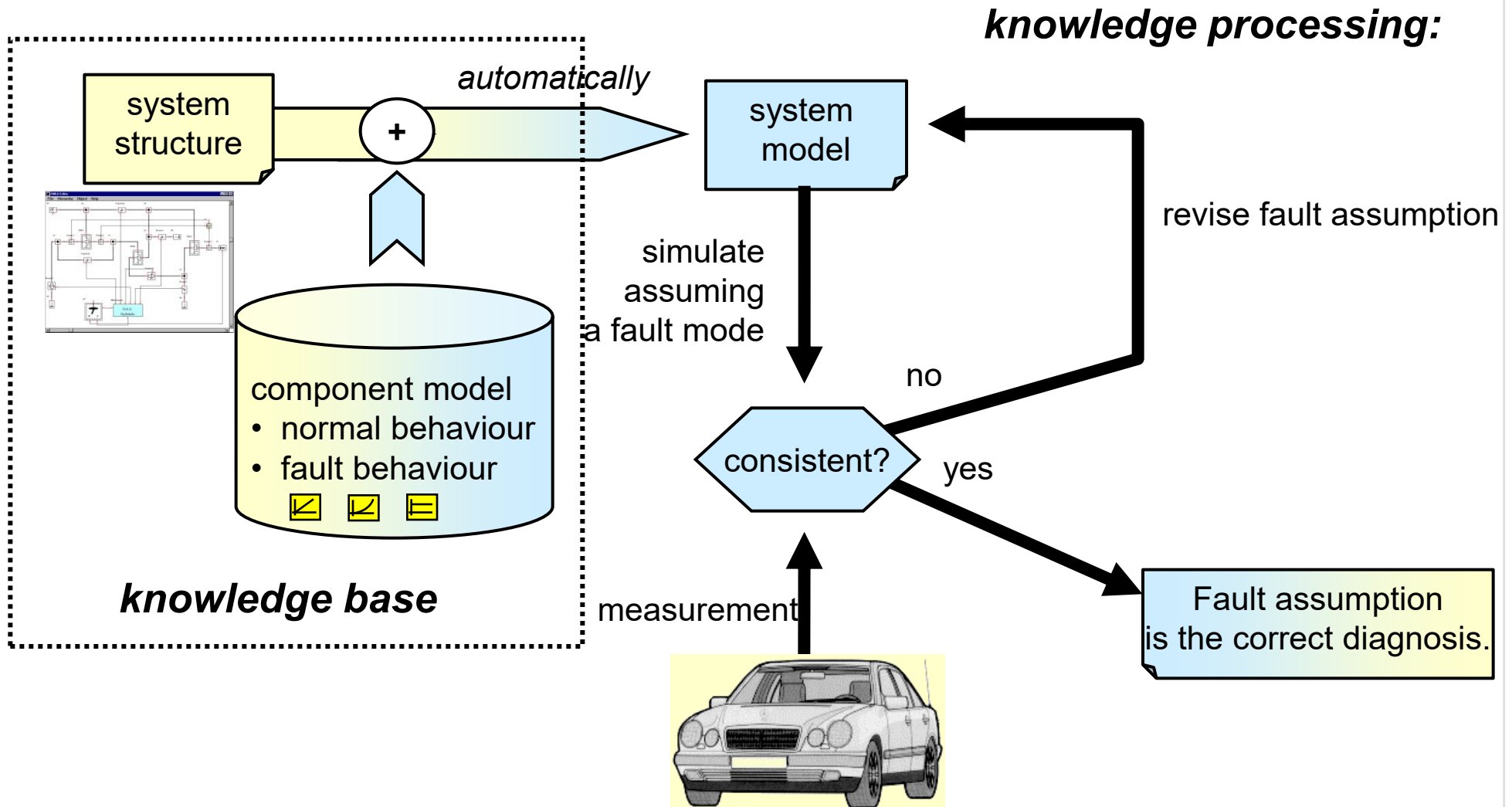
- system model: hierarchical structure of the system (+ how the components are connected)
- component models

Structure of knowledge base:

- constraint network (assembled automatically)
- structured by:
 - assigning constraints to components and ports
 - assigning variables to components and ports

3. Model-Based Diagnosis

Base functionality: Conflict driven search



3. Model-Based Diagnosis

Base functionality: Finding consistent assumptions

GDE 1987: *The* prototype for model-based diagnosis

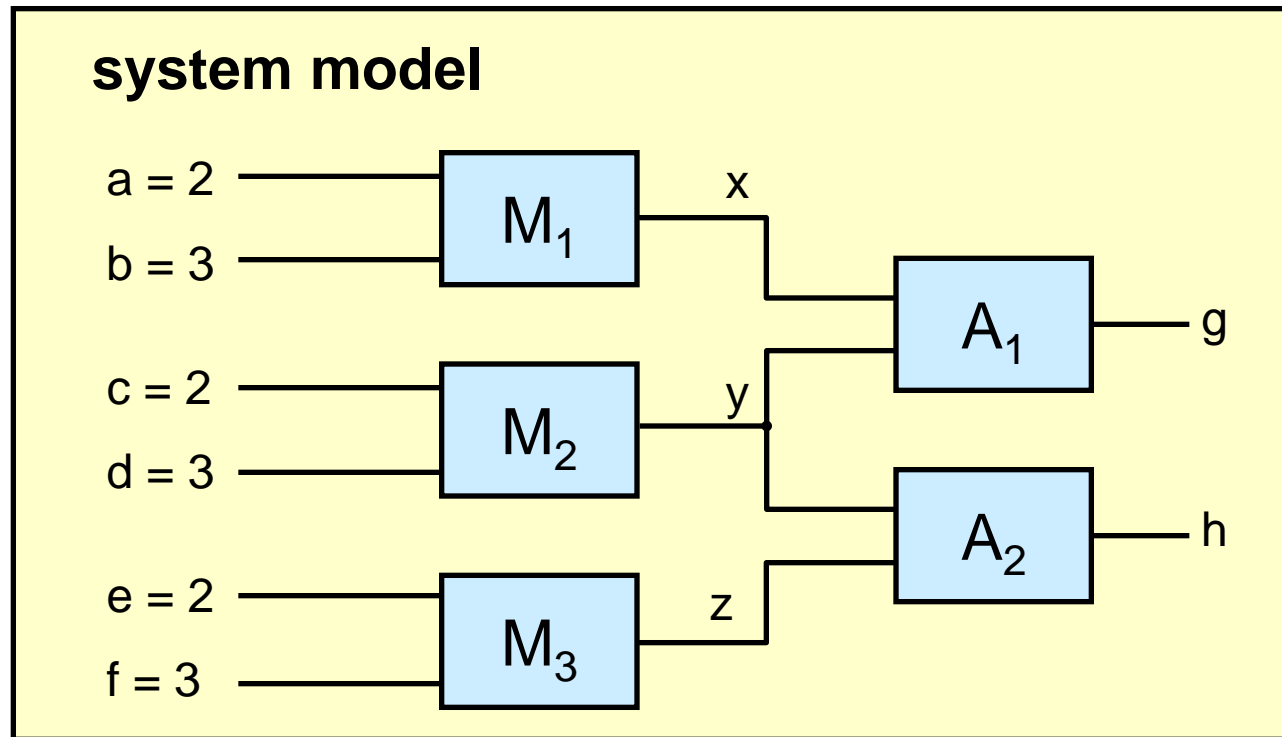
Problem:

- ‚brute-force‘ Simulation of ***all*** fault assumptions combinatorically not feasible

Idea: General Diagnostic Engine GDE, deKleer & Williams 1987

- intelligent search in the space of all fault assumptions
- uses inconsistent assumptions for pruning the search space
- base principle: **conflict-driven search**

GDE - Example

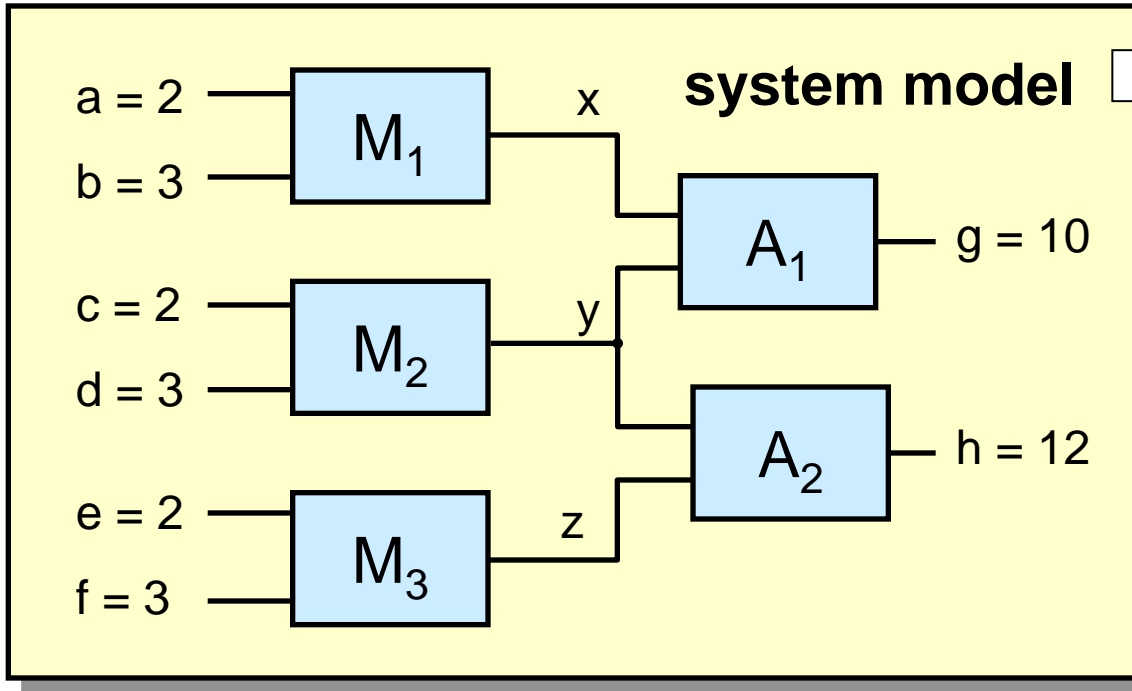


component models

- multiplier: $\text{mode=ok} \Rightarrow \text{out} = \text{in}_1 * \text{in}_2$
- adder: $\text{mode=ok} \Rightarrow \text{out} = \text{in}_1 + \text{in}_2$

measurements: $g = 10 \wedge h = 12$

GDE - Example



simulation

$x = 6$ {M1}

$y = 6$ {M2}

$z = 6$ {M3}

$g = 12$ {M1 M2 A1}, $g = 10$

$y = 4$ {M1 A1}

$h = 10$ {M1 A1 A2 M3}, $h = 12$

$y = 6$ {A2 M3}

two conflicts

diagnoses:

single-fault **M1**

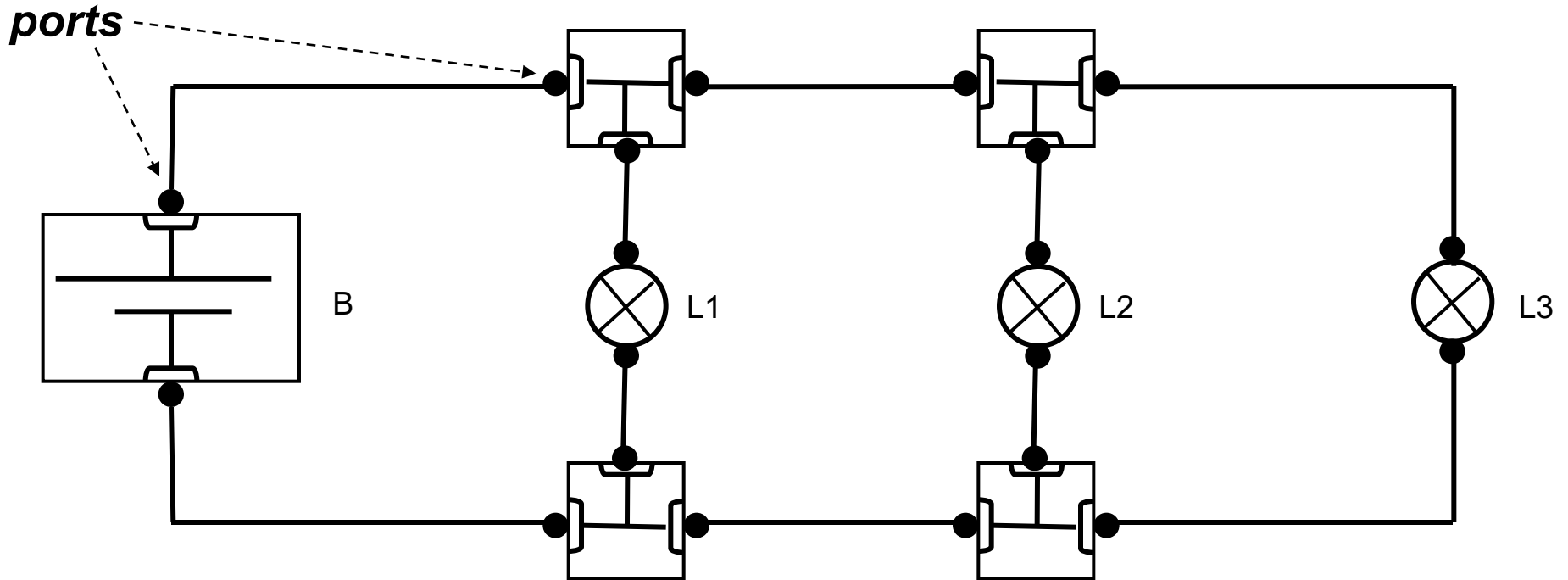
single-fault **A1**

double fault **M2 M3**

:

M1	M2	M3	A1	A2
X	X		X	
X		X	X	X

Modeling a simple electric circuit in a first shot



component types:

Battery

Lamp

Wire

Junction (3)

Model-Based Diagnosis: Base functionality

Models of electric components:

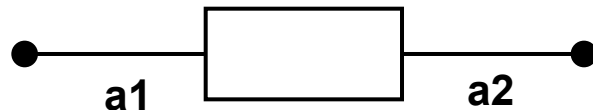
Battery:



value ranges: $\text{minus, plus} \in \{ \text{ground, supply voltage} \}$

rules:
 $\text{ok} \Rightarrow (\text{minus} = \text{ground})$
 $\text{ok} \Rightarrow (\text{plus} = \text{supply voltage})$

Wire:



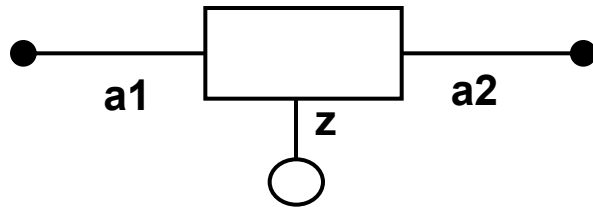
value ranges: $\text{a1, a2} \in \{ \text{ground, supply voltage} \}$

rules:
 $\text{ok} \wedge (\text{a1} = \text{ground}) \Rightarrow (\text{a2} = \text{ground})$
 $\text{ok} \wedge (\text{a1} = \text{supply voltage}) \Rightarrow (\text{a2} = \text{supply voltage})$
 $\text{ok} \wedge (\text{a2} = \text{ground}) \Rightarrow (\text{a1} = \text{ground})$
 $\text{ok} \wedge (\text{a2} = \text{supply voltage}) \Rightarrow (\text{a1} = \text{supply voltage})$

Model-Based Diagnosis: Base functionality

Models of electric components:

Lamp:



value ranges:

$a1, a2 \in \{ \text{ground, supply voltage} \}$

$z \in \{ \text{lit, dark} \}$

rules:

$ok \wedge (a1 = \text{supply voltage}) \wedge (a2 = \text{ground}) \Rightarrow (z = \text{lit})$

$ok \wedge (a2 = \text{supply voltage}) \wedge (a1 = \text{ground}) \Rightarrow (z = \text{lit})$

$ok \wedge (a1 = \text{supply voltage}) \wedge (a2 = \text{supply voltage}) \Rightarrow (z = \text{dark})$

$ok \wedge (a1 = \text{ground}) \wedge (a2 = \text{ground}) \Rightarrow (z = \text{dark})$

$ok \wedge (a1 = \text{ground}) \wedge (z = \text{lit}) \Rightarrow (a2 = \text{supply voltage})$

$ok \wedge (a1 = \text{supply voltage}) \wedge (z = \text{lit}) \Rightarrow (a2 = \text{ground})$

$ok \wedge (a1 = \text{ground}) \wedge (z = \text{dark}) \Rightarrow (a2 = \text{ground})$

$ok \wedge (a1 = \text{supply voltage}) \wedge (z = \text{dark}) \Rightarrow (a2 = \text{supply voltage})$

$ok \wedge (a2 = \text{ground}) \wedge (z = \text{lit}) \Rightarrow (a1 = \text{supply voltage})$

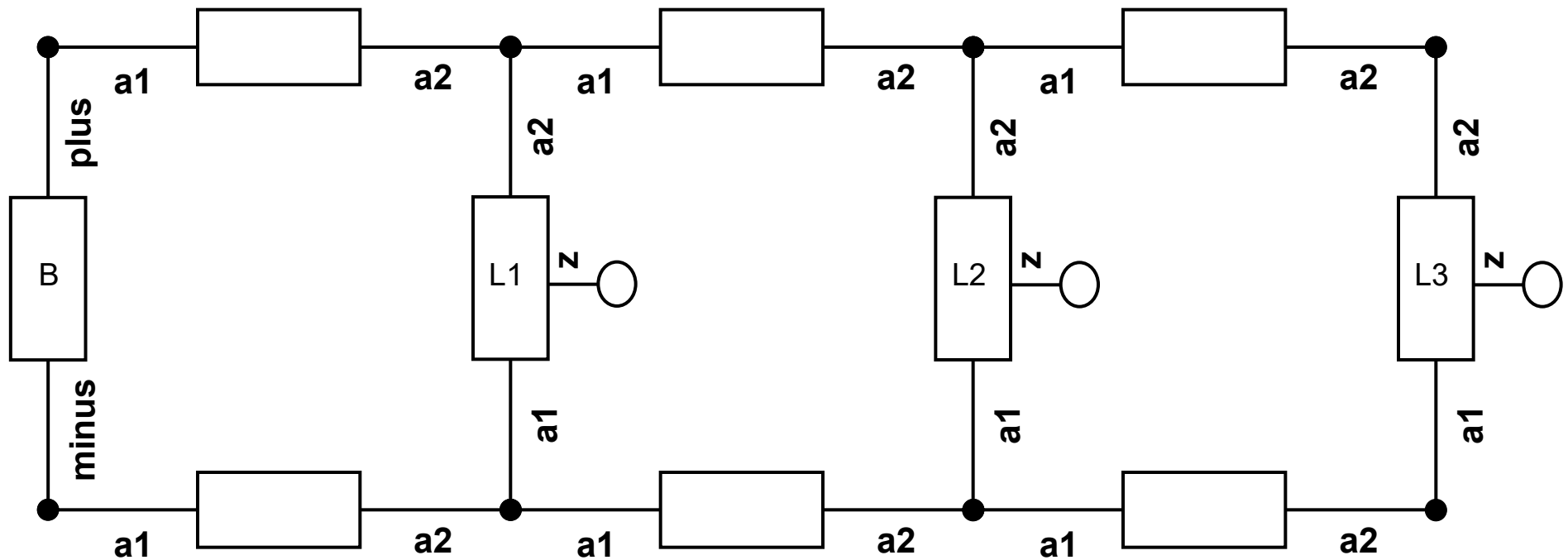
$ok \wedge (a2 = \text{supply voltage}) \wedge (z = \text{lit}) \Rightarrow (a1 = \text{ground})$

$ok \wedge (a2 = \text{ground}) \wedge (z = \text{dark}) \Rightarrow (a1 = \text{ground})$

$ok \wedge (a2 = \text{supply voltage}) \wedge (z = \text{dark}) \Rightarrow (a1 = \text{supply voltage})$

Model-Based Diagnosis: Base functionality

Composing the system model from the component models:



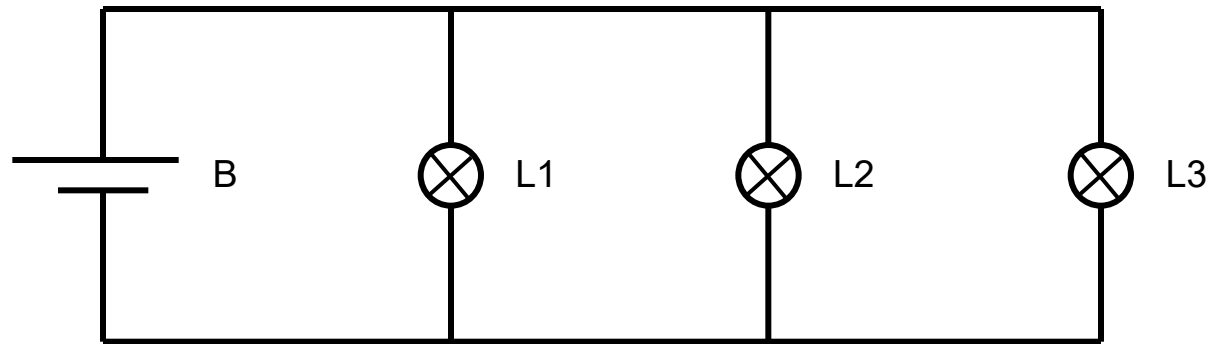
Values at connecting ports must be the same from both sides.

In case of contradiction: Conflict between the behavioural modes predicting the resp. values

Diagnoses are sets of behavioural modes not containing any conflict.

Model-Based Diagnosis: Base functionality

Example why the adder/multiplier example does not reveal all difficulties for practice:



Observation:

L1, L2 are not lit, L3 is lit

GDE diagnoses:

1. (B ok, L1 faulty, L2 faulty, L3 ok)
2. (B faulty, L1 ok, L2 ok, L3 faulty) ???
3. (B faulty, L1 ok, L2 ok, L3 ok) ???

Model-Based Diagnosis: Base functionality

Conclusion from this modeling:

There is no logic contradiction to the following diagnosis:

2. (B faulty, L1 ok, L2 ok, L3 faulty)

Reason:

L3 may be lit in fault mode even if there is no voltage difference.

Incomplete knowledge base !

Even worse:

If a behavioural rule is only evaluated when its antecedents assume actual values, then no contradiction can be found to the following diagnosis:

3. (B faulty, L1 ok, L2 ok, L3 ok)

Reason:

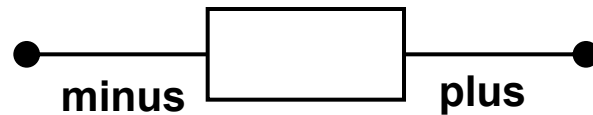
There is no voltage value computed anywhere in the system.

Incomplete inference ability of the problem solver !

Model-Based Diagnosis: Base functionality

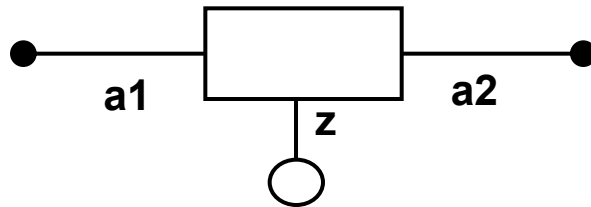
Additional rules for the exclusion of diagnoses 2 / 3:

Battery:



faulty \Rightarrow (minus = ground)

Lamp:



faulty \wedge (a1 = supply voltage) \wedge (a2 = supply voltage) \Rightarrow (z = dark)

faulty \wedge (a1 = ground) \wedge (a2 = ground) \Rightarrow (z = dark)

There must be models for faulty behaviour, too, in order to exclude diagnoses that are physically impossible.

Model-Based Diagnosis: Extended functionality

Base functionality:

Input:

- Setting certain control inputs
- Observing values depending on this setting

Output:

- Several diagnoses of the following kind:
 - Each diagnosis assigns a behavioural mode to each component: ok or a defined fault mode
 - The rules of all behavioural modes assigned agree with all set and observed values.

What does the user need ?

Input: see above

Output: • A unique instruction how to repair which component

Model-Based Diagnosis: Extended functionality

Extended functionality:

1) Suggestion of setting certain control inputs

- Setting certain values at certain places in the system
(such that the observations to be expected differ such that the diagnoses valid so far may be distinguished best)

2) Suggestion of observation points

- Selecting observation points
(such that the observations to be expected differ such that the diagnoses valid so far may be distinguished best)

Test

Requirement for the modeling:

- Definition of test points
- Definition of test values to be set at the test points
- Definition of observation points to be measured

Control actions

Observations

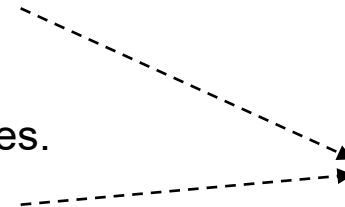
Modeling the components in a proper way

Behavioural modes

- modes of the component to be searched for in the diagnostic process
- Domain of definition must be finite (normal less than 10 values)

Variables

- containing values
- The variable values are used in the constraints.
- The constraints compute new values for other variables.



*Distinguish
internal variables
from port variables !*

Ports

- containing variables to be identified at the connections to adjacent components

Constraints

- set of behavioural rules connecting the variables of the same component
- Normally, a constraint is only valid under the assumption of a certain behavioural mode.

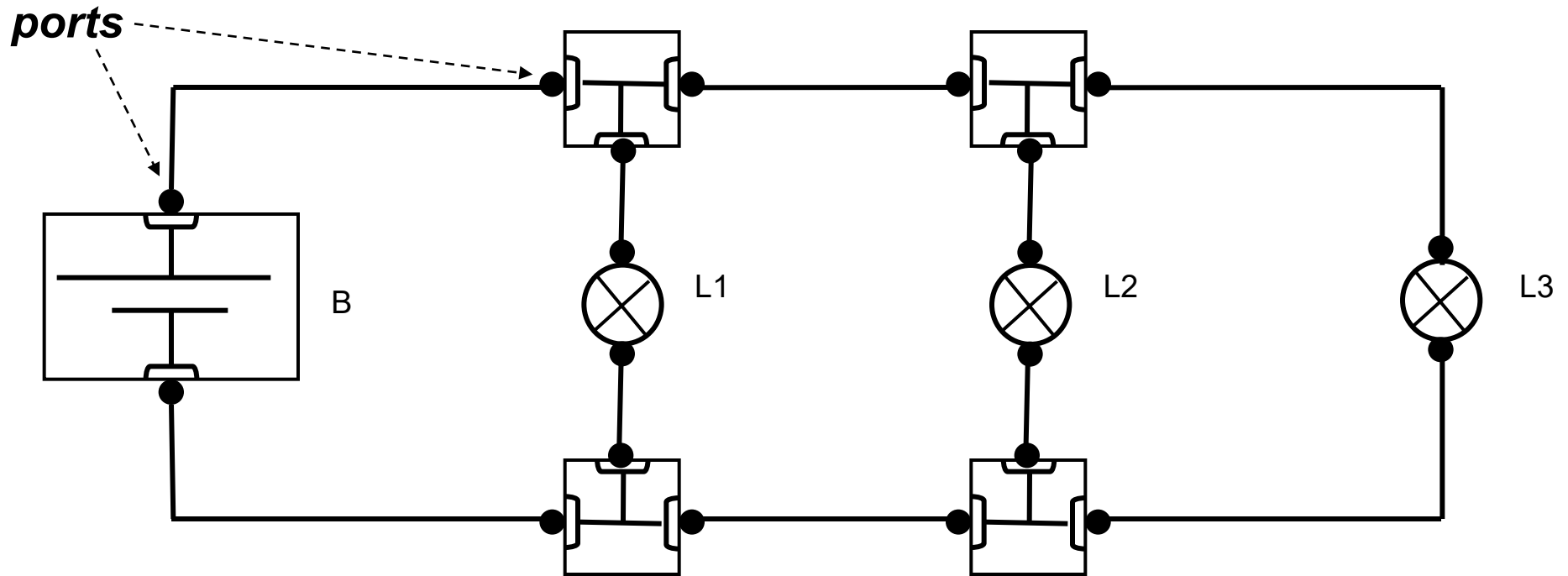
Control actions

- variables and values to be set
- measure of accessibility and the difficulty to set certain values.

Observations

- variables
- measure for accessibility

Modeling a simple electric circuit in a proper way



component types:

Battery

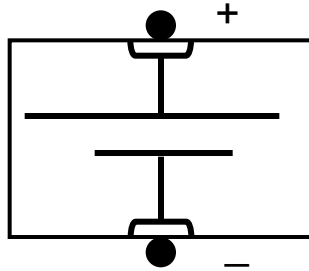
Lamp

Wire

Junction (3)

Modeling a simple electric circuit

Battery



fault modes:

discharged

contact gap at +

contact gap at -

loose contact at +

loose contact at -

corroded

ports: +, -

control actions:

open connector at +

open connector at -

close connector at +

close connector at -

constraints:

cf. slides 10, 15

observations:

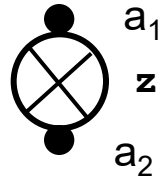
inspect connectors

measure voltage at +

measure voltage at -

Modeling a simple electric circuit

Lamp



ports: a_1, a_2

internal variables: z

constraints:

cf. slides 11, 15

fault modes:

blown

lamp is not inserted

loose contact

corroded

control actions:

remove lamp

insert lamp

observations:

inspect lamp

Wire



ports: a_1, a_2

constraints:

cf. slides 10

fault modes:

broken

shorted to ground

shorted to voltage

corroded

control actions:

observations:

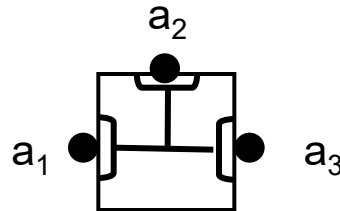
measure voltage at a_1

measure voltage at a_2

inspect wire

Modeling a simple electric circuit

Junction (3)



ports: a_1, a_2, a_3

constraints:

exercise

(related to wires)

fault modes:

contact gap at a_1

contact gap at a_2

contact gap at a_3

loose contact at a_1

loose contact at a_2

loose contact at a_3

control actions:

close contact at a_1

close contact at a_2

close contact at a_3

open contact at a_1

open contact at a_2

open contact at a_3

observations:

inspect contacts

Applications of Artificial Intelligence

Sebastian Iwanowski
FH Wedel

Chapter 4:
Knowledge-Based Systems

4.3: Model-Based Reasoning
Details

Model-Based Diagnosis (MDS)

Terminology of the GDE approach:

Component:

Unit of which behaviour should be classified („diagnosed“)
usually enumerated from 1 to n

Component type:

collects components of same behaviour

Behavioural mode:

represents a specific behaviour of all components of that type
usually enumerated from 1 to k:

1 represents ok

2 thru k are the fault modes (ordered by probability)

(Diagnosis) Candidate:

Assignment of exactly one behavioural mode to each component of the system

Model-Based Diagnosis (MDS)

Terminology of the GDE approach:

Candidate:

(2 1 3 1 1 2 1) means:

- Component Nr. 1 is in behavioural mode 2
- Component Nr. 2 is in behavioural mode 1
- Component Nr. 3 is in behavioural mode 3
- Component Nr. 4 is in behavioural mode 1
- Component Nr. 5 is in behavioural mode 1
- Component Nr. 6 is in behavioural mode 2
- Component Nr. 7 is in behavioural mode 1

Conflict:

Assignment of exactly one behavioural mode to some components of the system

(0 1 0 0 0 2 0) means:

- Component Nr. 2 is in behavioural mode 1
- Component Nr. 6 is in behavioural mode 2

About the other components no proposition is made.

Interpretation: It is not consistent that component 2 is in behavioural mode 1 and und component 6 is in behavioural mode 2.

Model-Based Diagnosis (MDS)

Terminology of the GDE approach:

Diagnosis (= consistent candidate):

Candidate not containing any conflict

Examples: $(2\ 1\ 3\ 1\ 1\ 2\ 1)$ contains the conflict $(0\ 1\ 0\ 0\ 0\ 2\ 0)$, i.e., it is not a diagnosis.

If $(0\ 1\ 0\ 0\ 0\ 2\ 0)$ is the only conflict, then $(1\ 1\ 1\ 1\ 1\ 1\ 1)$ is a diagnosis.

If $(0\ 1\ 0\ 0\ 0\ 2\ 0)$ and $(1\ 1\ 0\ 0\ 0\ 0\ 0)$ are the only conflicts, then $(1\ 2\ 1\ 1\ 1\ 1\ 1)$ is a diagnosis

Preference between candidates:

A candidate A is preferred to another candidate B, if A assigns at most the number of the behavioural mode of B for each component.

Example: $(1\ 1\ 1\ 1\ 1\ 1\ 1)$ is preferred to $(1\ 2\ 1\ 1\ 1\ 1\ 1)$

Maximum preferred diagnosis:

A diagnosis is called a maximum preferred diagnosis, if all preferred candidates contain conflicts, i.e. the diagnosis is maximum with respect to the preference relation.

Example: If $(0\ 1\ 0\ 0\ 0\ 2\ 0)$ and $(1\ 1\ 0\ 0\ 0\ 0\ 0)$ are the conflicts, then $(1\ 2\ 1\ 1\ 1\ 1\ 1)$ and $(2\ 1\ 1\ 1\ 1\ 1\ 1)$ are the only two maximum preferred diagnoses.

Model-Based Diagnosis (MDS)

Goal of MDS (Daimler enhancement of the GDE):

- 1) **Base functionality:** Find the best diagnoses
- 2) **Extended functionality:** Repair instruction:
Propose actions and tests in order to distinguish between diagnoses found in 1)

Details of 1): Find the maximum preferred diagnoses.

If there are too many maximum preferred diagnoses, the focus should be restricted to the *most probable* ones among *all* maximum preferred diagnoses.

The remaining maximum preferred diagnoses are to be marked as *pending* and may be inserted into focus at a later time.

Possible focus restriction policies (may be combined):

- a) Determine a maximum number of focus diagnoses
- b) Determine a probability threshold for the gap between focus diagnoses and pending diagnoses.

Model-Based Diagnosis (MDS)

Algorithm for finding the most probable maximum preferred diagnoses
(**Problem 1**):

At any time, all candidates of the focus are maximum preferred.

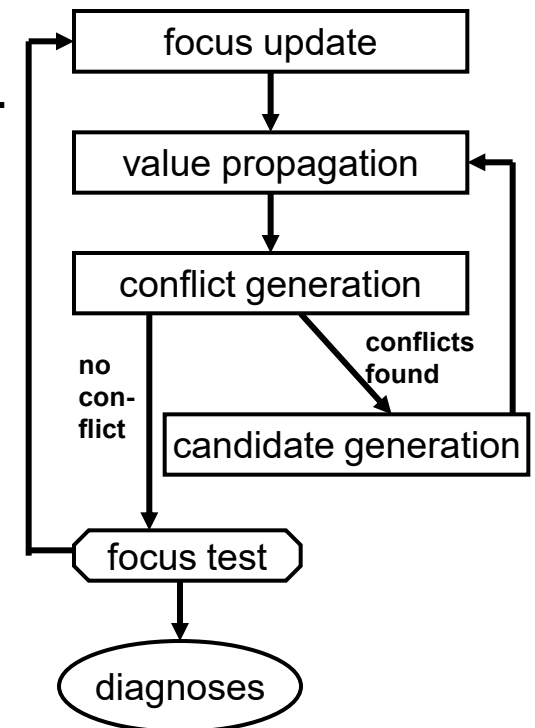
1. **Update the focus candidates: Initialise with 11....1.**
At later stages, pending candidates may be dragged into focus.
2. **Generate and propagate all values resulting from behavioural modes of candidates in focus.**
3. **Find the minimal conflicts from the propagated values.**
4. **Exclude the candidates containing conflicts and compute new maximum preferred candidates not containing any conflict.**
5. **If focus is sufficiently large, the goal is achieved.**
Otherwise continue with 1.

In reality, steps 2 thru 4 are implemented concurrently.

(achieved by event oriented programming)

In the following, the methods for **candidate generation** and **conflict generation** are described separately.

Diagnostic cycle



MDS: Candidate generation

INPUT:

- **Old conflicts and all maximum preferred und consistent diagnoses for these conflicts**
- **New conflicts**

OUTPUT:

- **Set of maximum preferred candidates being consistent for the new conflicts, too**

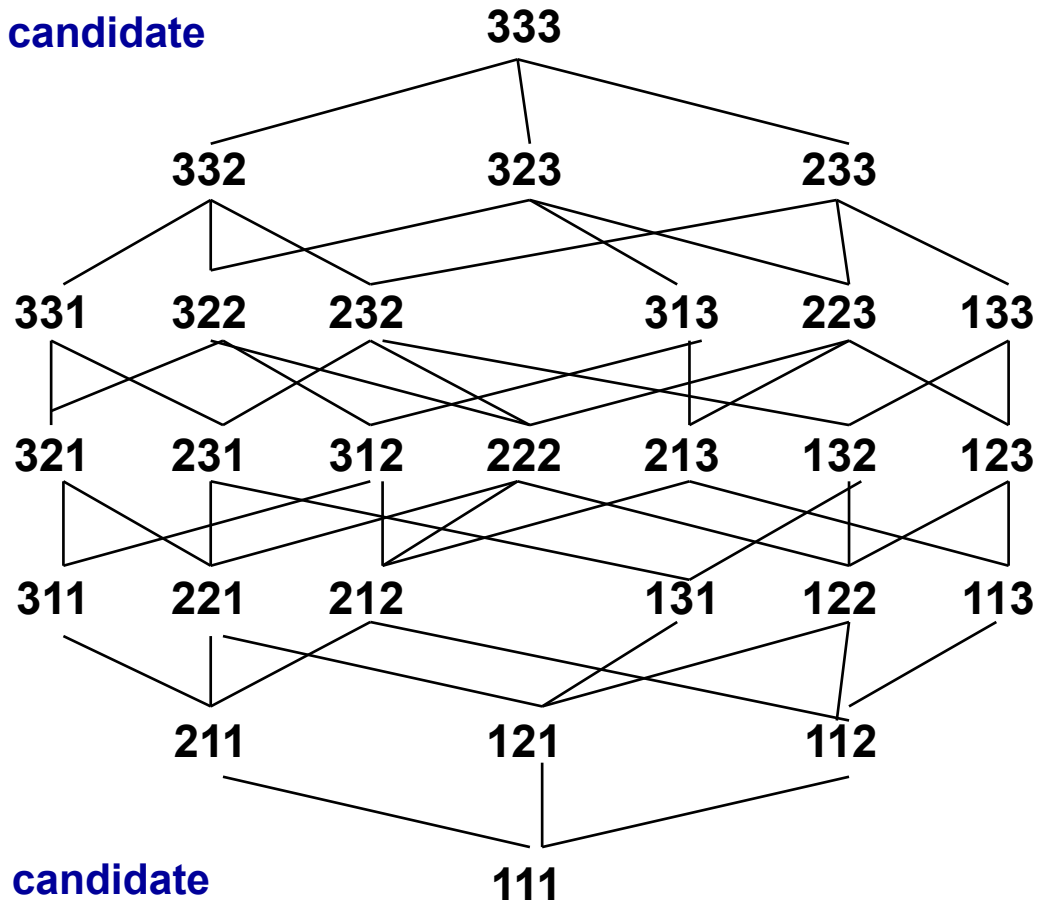
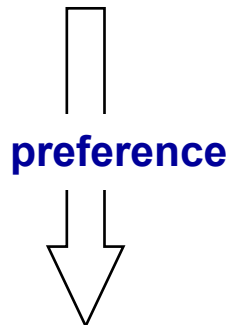
Embedding the candidate generation into the diagnostic process:

- Output of candidate generation will be taken as input in the next diagnostic cycle.
- Value propagation may find new conflicts.
- New conflicts may kick out diagnoses from focus.
- If no new conflicts are found, the diagnostic process is finished.

MDS: Preference web of candidates

Example: 3 components
3 behavioural modes for each of the components

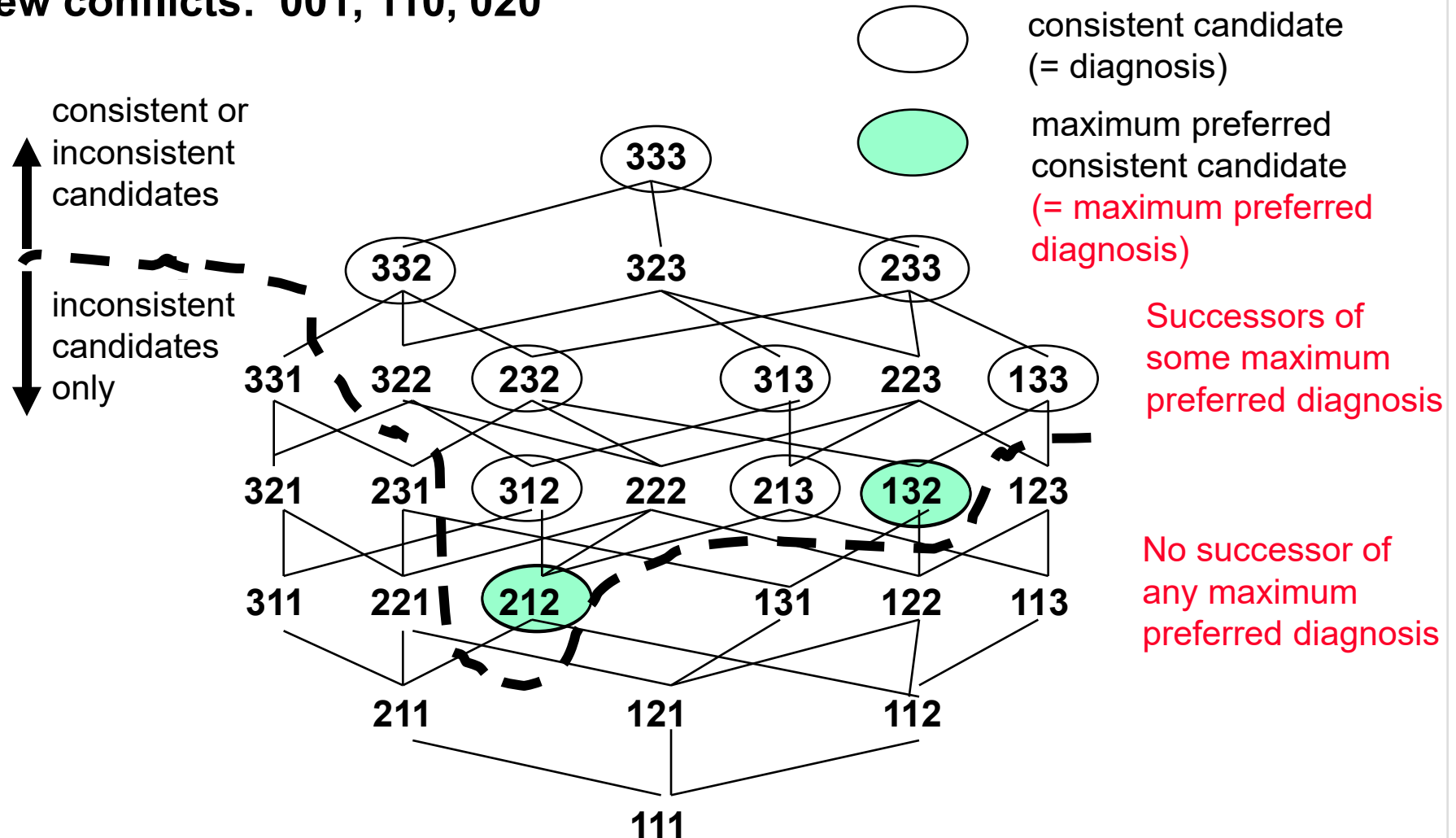
minimum preferred candidate



maximum preferred candidate

MDS: Preference web of candidates

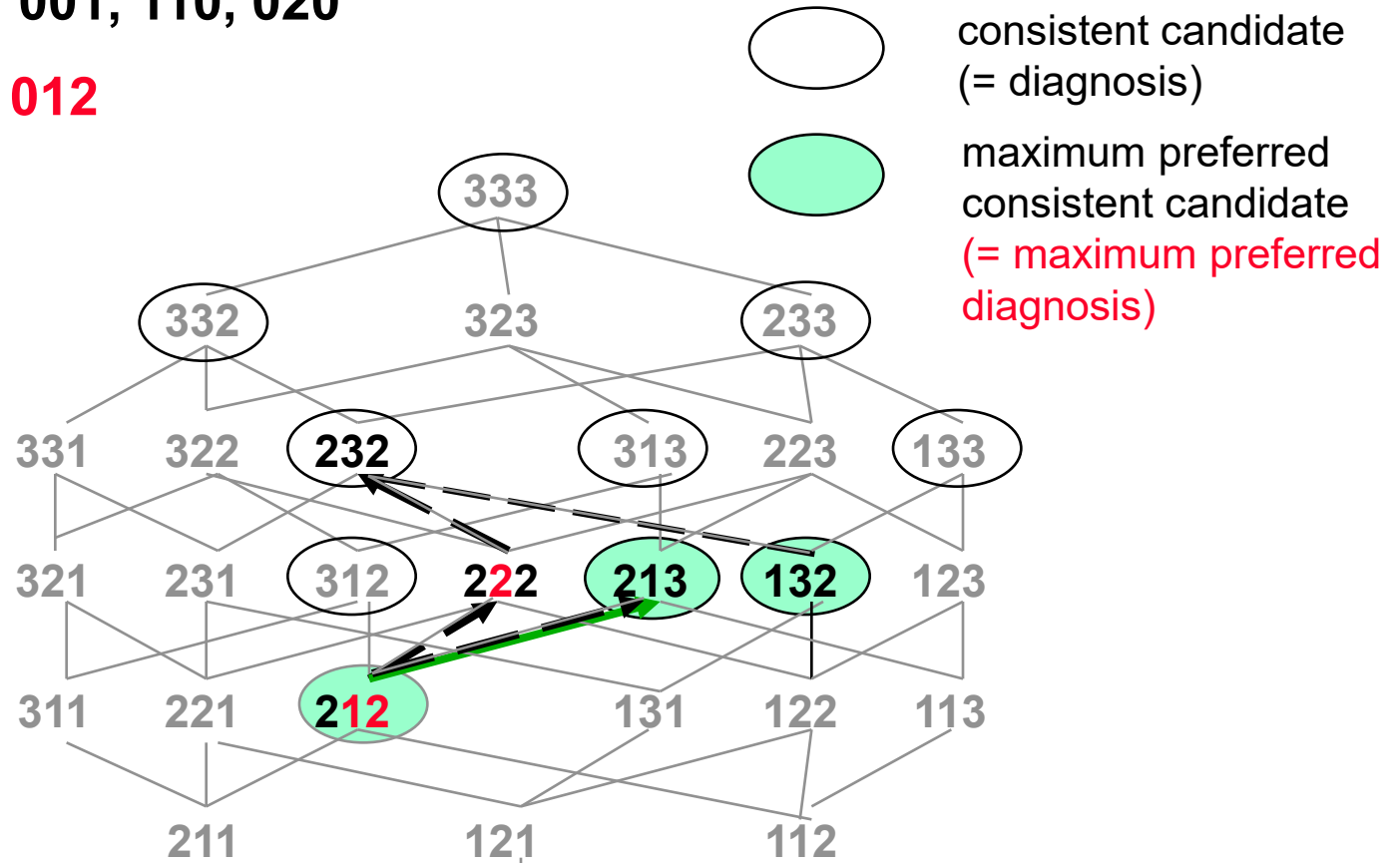
New conflicts: 001, 110, 020



MDS: Candidates update

Old conflicts: 001, 110, 020

New conflict: 012



Former maximum preferred diagnoses: 111

{ 212, 132 }

New maximum preferred diagnoses (stage 1):

{ 222, 213, 132 }

New maximum preferred diagnoses (stage 2):

{ 213, 132 }

MDS: Candidates update

Actions at detection of a new conflict:

- 1) Consistency check of all maximum preferred diagnoses
- 2) Removal of all candidates proven to be inconsistent
- 3) Generation of the preference successors of each candidate just removed
- 4) Adopting the preference successors satisfying the following conditions:
 - The successor is not preferred by a different consistent diagnosis.
 - The successor is consistent itself.

MDS: Candidates update

Actions at detection of a new conflict :

3) Generation of the preference successors of each candidate just removed:

- If C is a conflict contained in an old diagnosis, then generate only successors of C changing the behavioral mode of **just one component contained in C**.

(Generation of direct successors only, directly referring to conflict C)

Remark: This restricted method does not skip any eventual diagnosis

Prop.: Each successor diagnosis not containing C is successor diagnosis of a direct successor not containing C

- If one of the direct successors contains a conflict C', then do not generate this successor, but rather all successors referring directly to C'.

MDS: Optimising the candidate generation

Eliminating **irrelevant** conflicts:

- Conflicts are only relevant, if they may eventually remove a successor of a presently maximum preferred diagnosis.
- For the consistency test, only consider relevant conflicts: Each diagnosis d stores the relevant conflicts. Any successor of d will only be checked for the conflicts of d 's list.

Examples for relevant conflicts:

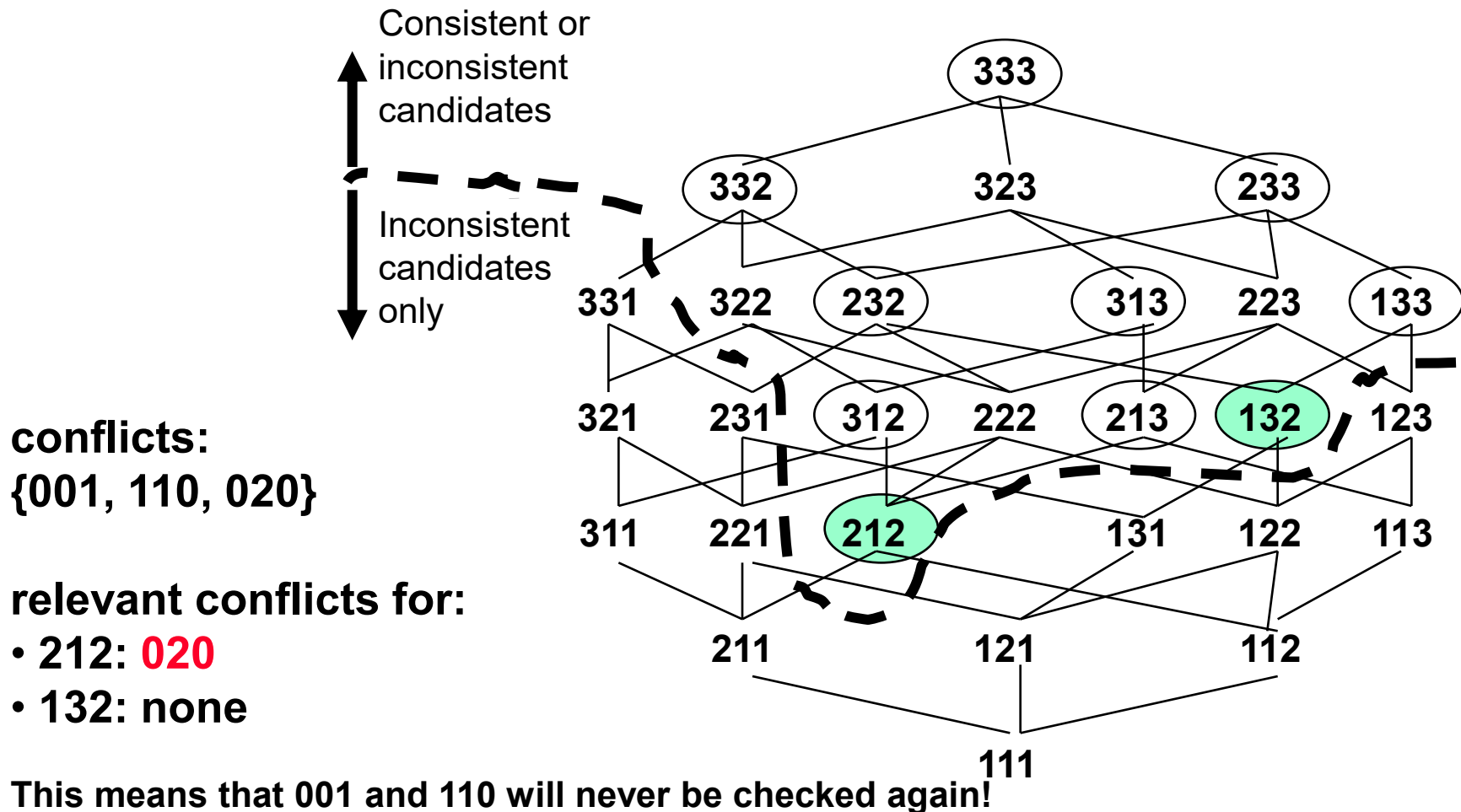
conflict:	0 2 2	2 0 2	2 0 2
candidate:	3 1 2	2 1 1	1 1 3
relevant ?			

Mathematical criterion for the relevance of a conflict (easy to check!)

A conflict c is relevant for a diagnosis d if for all components holds:
 c either assigns no mode (0) or a mode at least as high as the mode in d .

MDS: Optimising the candidate generation

Eliminating **irrelevant** conflicts:



MDS: Optimising the candidate generation

The Daimler product MDS contains a lot of further optimisations for accelerating the candidate generation process which are not mentioned here.

MDS: Conflict generation

Candidate generation solves the following task:

- **Given a set of conflicts: Find the most probable maximum preferred diagnoses taking into account those conflicts.**
- **This reduces the problem of finding the best diagnosis to the following task: Find the set of conflicts !**

What is a conflict ?

- **Assignment of exactly one behavioural mode resp. to some components of a system**
- **Logically, a conflict is a disjunction of negative literals.**
- **Comparing: Logically, a diagnosis is a conjunction of positive literals.**

How is a conflict generated?

- **by values contradicting each other**
- **The contradicting values are backed by different assumptions.**
- **Then one of the assumptions must be false.**

TMS: Truth Maintenance System

Objects of a TMS:

Propositional node:

Represents an arbitrary proposition (may be true or false)

Justification:

$A_1 \wedge A_2 \wedge \dots \wedge A_n \Rightarrow C$ where A_1, A_2, \dots, A_n, C are propositional nodes

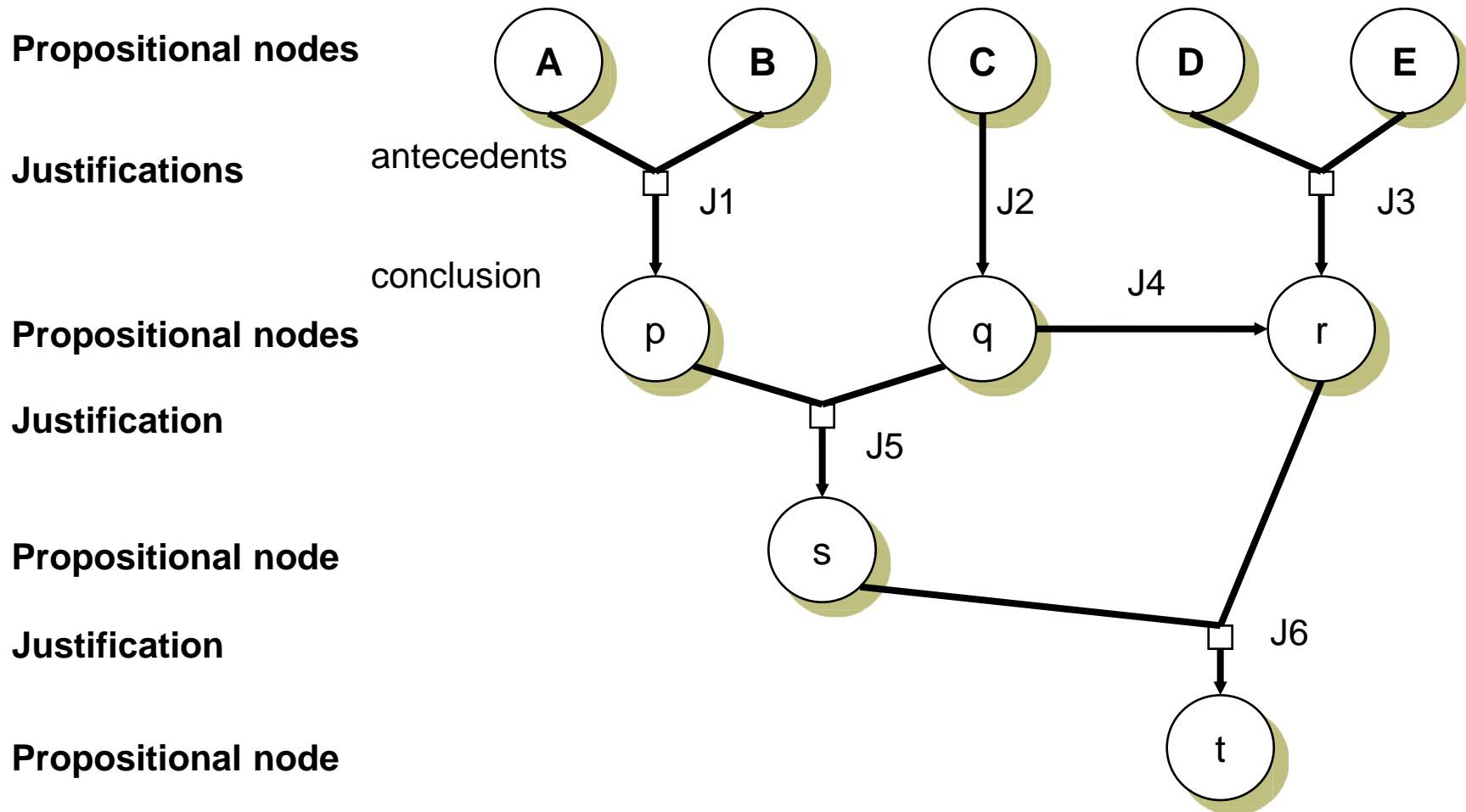
A_1, A_2, \dots, A_n are the **antecedents** of the justification

C is the **conclusion** of the justification

Contradiction node (\perp):

represents a proposition which holds by no means

TMS: Truth Maintenance System



**From the combination of propositions,
a justification makes a new proposition.
The antecedents of a justification are to be considered as conjunction.**

ATMS: Assumption-based Truth Maintenance System

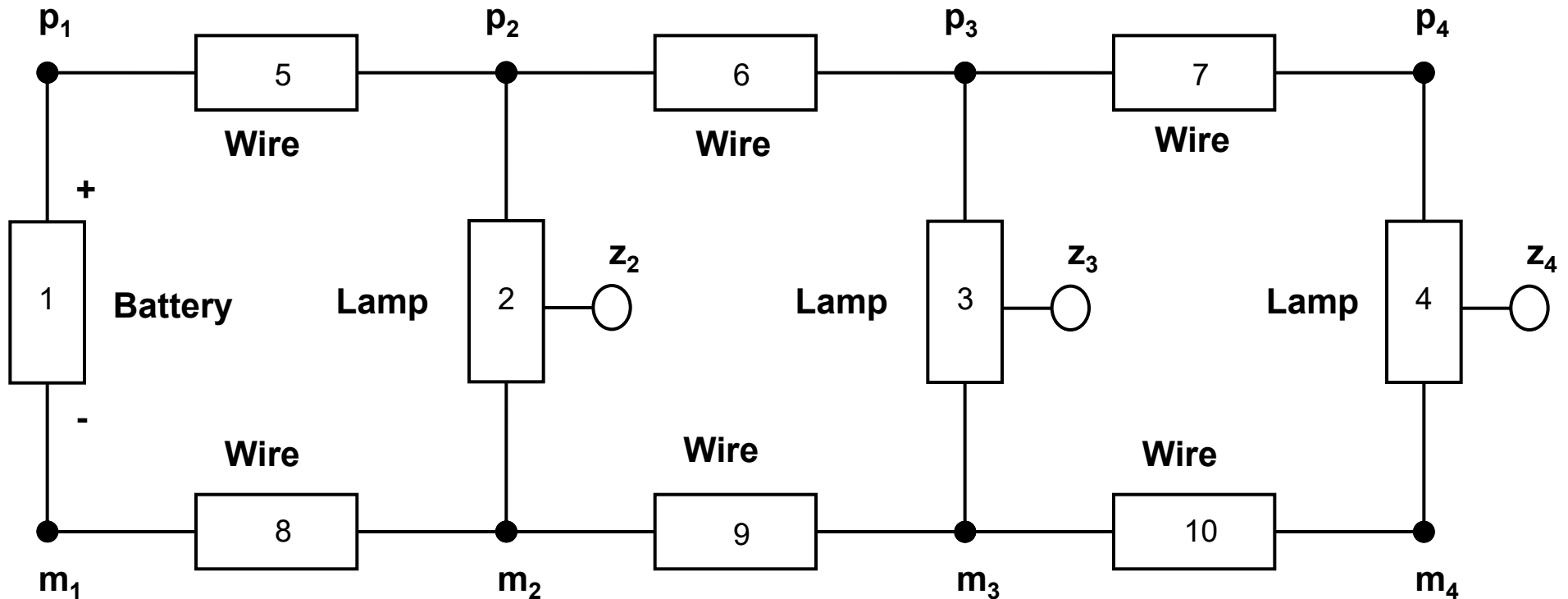
Functionality of a **general** TMS:

- 1) Certain propositional nodes are considered true (beliefs).
- 2) TMS determines by propagation of these assumptions via the justifications which other propositions must also hold then.
- 3) In particular, if the contradiction node must hold, then the assumptions must be contradictory.

Additional functionality of an **A**TMS:

- An ATMS works with *several* assumption sets in parallel: A (context) environment is the set of assumptions that should hold at the same time, but there may be different such environments holding alternatively.
- 1) The propositions are assigned with the assumption environments under which they must hold.
 - 2) The ATMS propagates these assumption environments over the justifications and determines which other propositions must hold then as well.
 - 3) In particular, the environments of the contradiction node reveals which environments are contradictory.

Example for applying an ATMS



Behavioural modes:

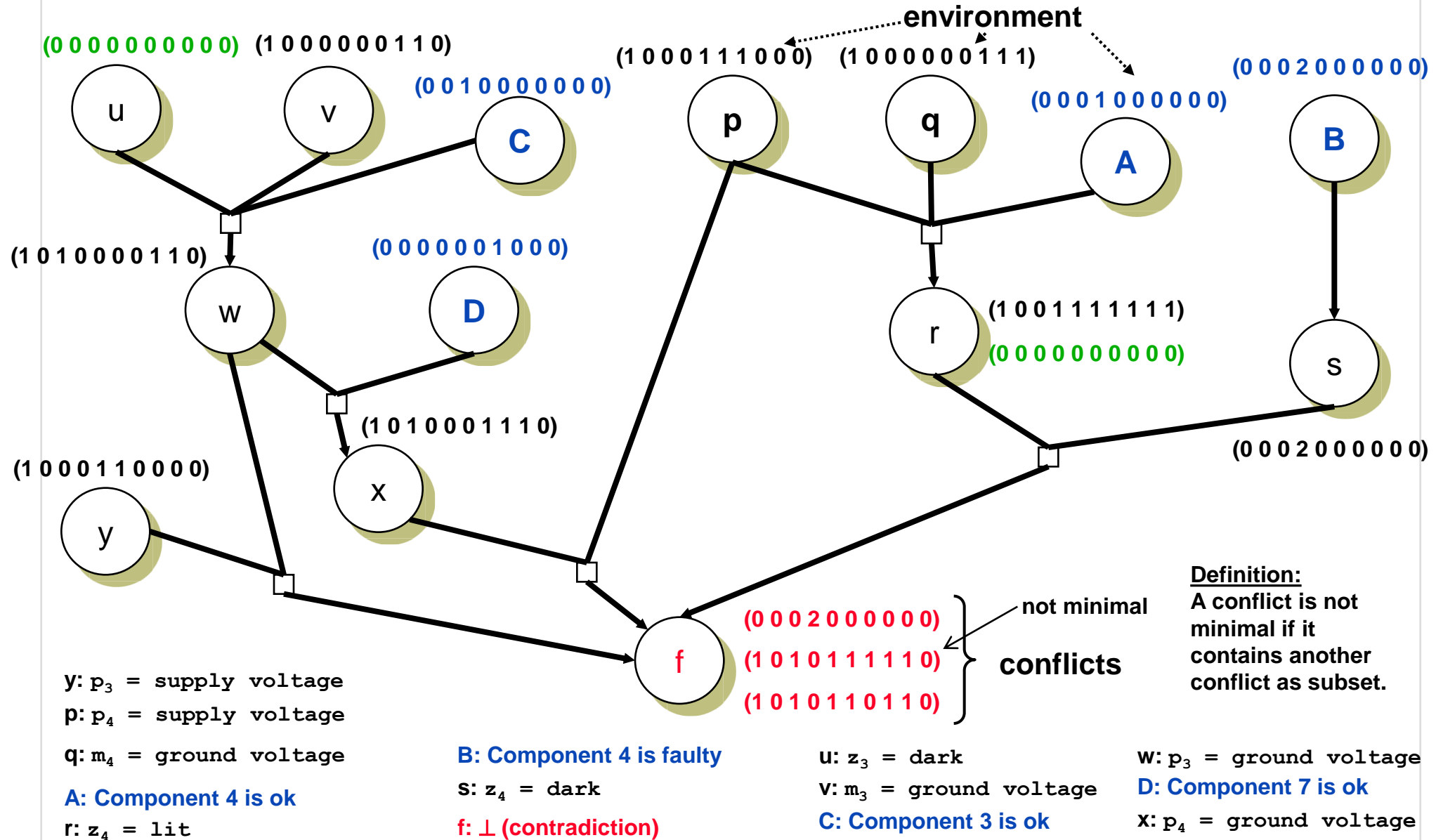
Mode 1 for all component types: normal behaviour

Modus 2 for all component types: unique fault mode

Battery: Modus 2 \Rightarrow (minus = ground voltage)

Lamp: Modus 2 \Rightarrow (z = dark)

Example for applying an ATMS



ATMS: Assumption-based Truth Maintenance System

Terminology of ATMS:

Propositional node:

The propositional nodes distinguish between *normal propositions* and *assumptions*, i.e. the class of assumption nodes is a specialisation of propositional nodes.

Environment:

Context of assumptions: *Conjunction* of assumptions, under which a proposition holds (if all assumptions of this environment are valid)

Label:

Set of different environments for a propositional node. Different environments need not be consistent to each other. The proposition holds already under the *disjunction* of the environments.

conflict (nogood):

Environment of the label of the contradictory node

ATMS: Assumption-based Truth Maintenance System

Application of an ATMS for model-based diagnosis:

Propositional nodes:

- 1) „Normal“ nodes: Assignment of a certain value to a certain position (variable) in the system
- 2) Assumption node: Assignment of a behavioural mode to a component

Justification: Application of a generic behavioural rule to actual values

Environment:

Concurrent (conjunction) assignment of behavioural modes to components under which a proposition would hold. The assignment need not be complete, i.e. it is an arbitrary candidate (like in a conflict).

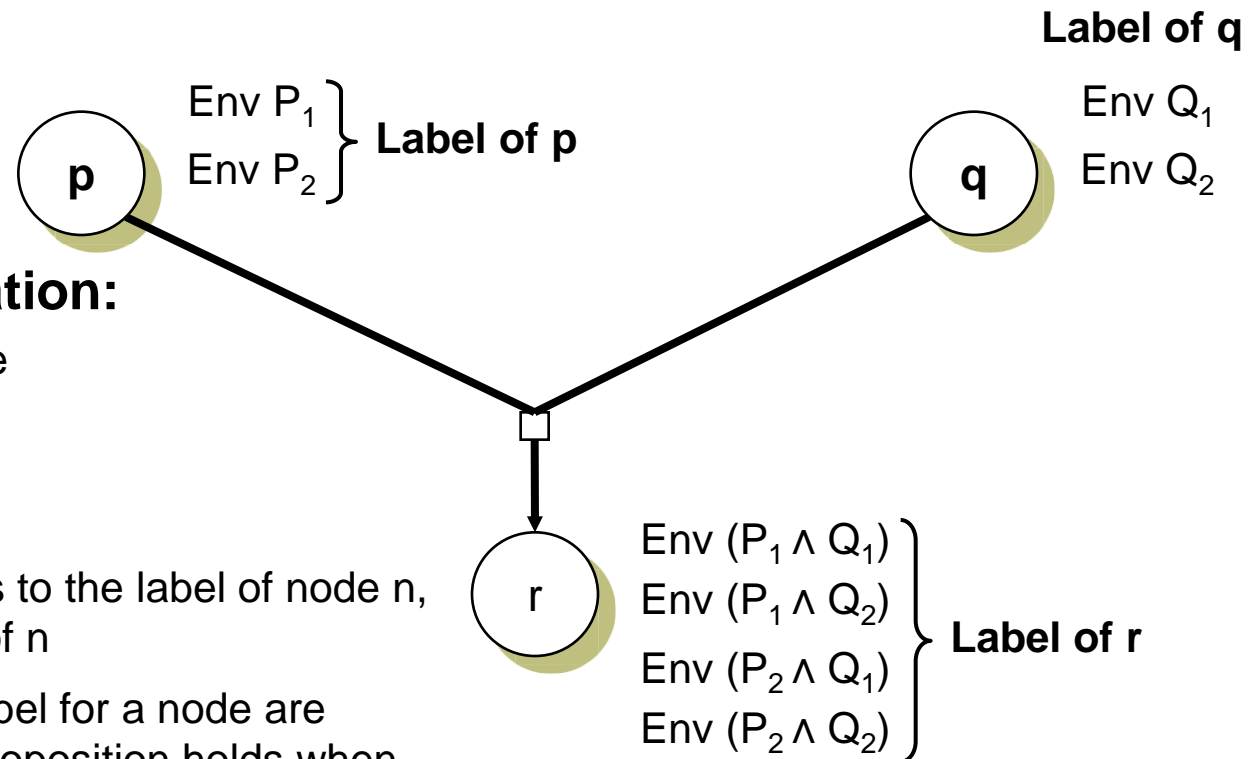
For assumption nodes: Assignment of a behavioural mode to exactly one component

conflict (nogood):

Environment of the label of the contradictory node: Assignment of behavioural modes to components of which at least one must be faulty.

This enables the same notation and meaning of conflicts as in the terminology of the GDE.

Label update in an ATMS



Interpretation of justification:

- r holds when p and q are true (conjunction)

Interpretation of labels:

- When environment e belongs to the label of node n , this means: $e \Rightarrow$ proposition of n
- Several environments of a label for a node are treated as disjunction: The proposition holds when at least one of the environments is true.

Elimination of redundant environments:

- Contradictory environments may be removed.
- This enables the removal of all environments containing conflicts.
- Environments implying other environments of the same label may be omitted as well.

User interface of an ATMS

Input of problem solver:

- Assumption nodes
- “Normal” nodes
- Justifications between the nodes
(they must be obtained from the component library applied to actual values)
- Certain environment assignments to normal nodes,
e.g., observations or other premises as (0 0 ... 0)

Output to the problem solver:

- Set of minimal conflicts (Definition of minimality on slide 21)

The ATMS performs automatically: *These are a lot of operations !*

- Generation of labels for the assumption nodes
- Update of labels for all conclusions where the label of some antecedent has changes.
- Elimination of redundant environments

User interface of an ATMS

Input of problem solver:

- Assumption nodes
- “Normal” nodes
- Justifications between the nodes
(they must be obtained from the component library applied to actual values)
- Certain environment assignments to normal nodes,
e.g., observations or other premises as (0 0 ... 0)

Output to the problem solver:

- Set of minimal conflicts

Candidate generator uses the ATMS as follows:

- Generate all assumption nodes for the focus diagnoses
- Value propagation (simulation):
Compute all values resulting from assumptions of the focus diagnoses,
generate the respective propositional nodes und justifications,
plug this into the ATMS.
- Ask the ATMS for the new conflicts.

This is done by a
separate module
called Value
Propagator (**VP**)

Value propagation and ATMS

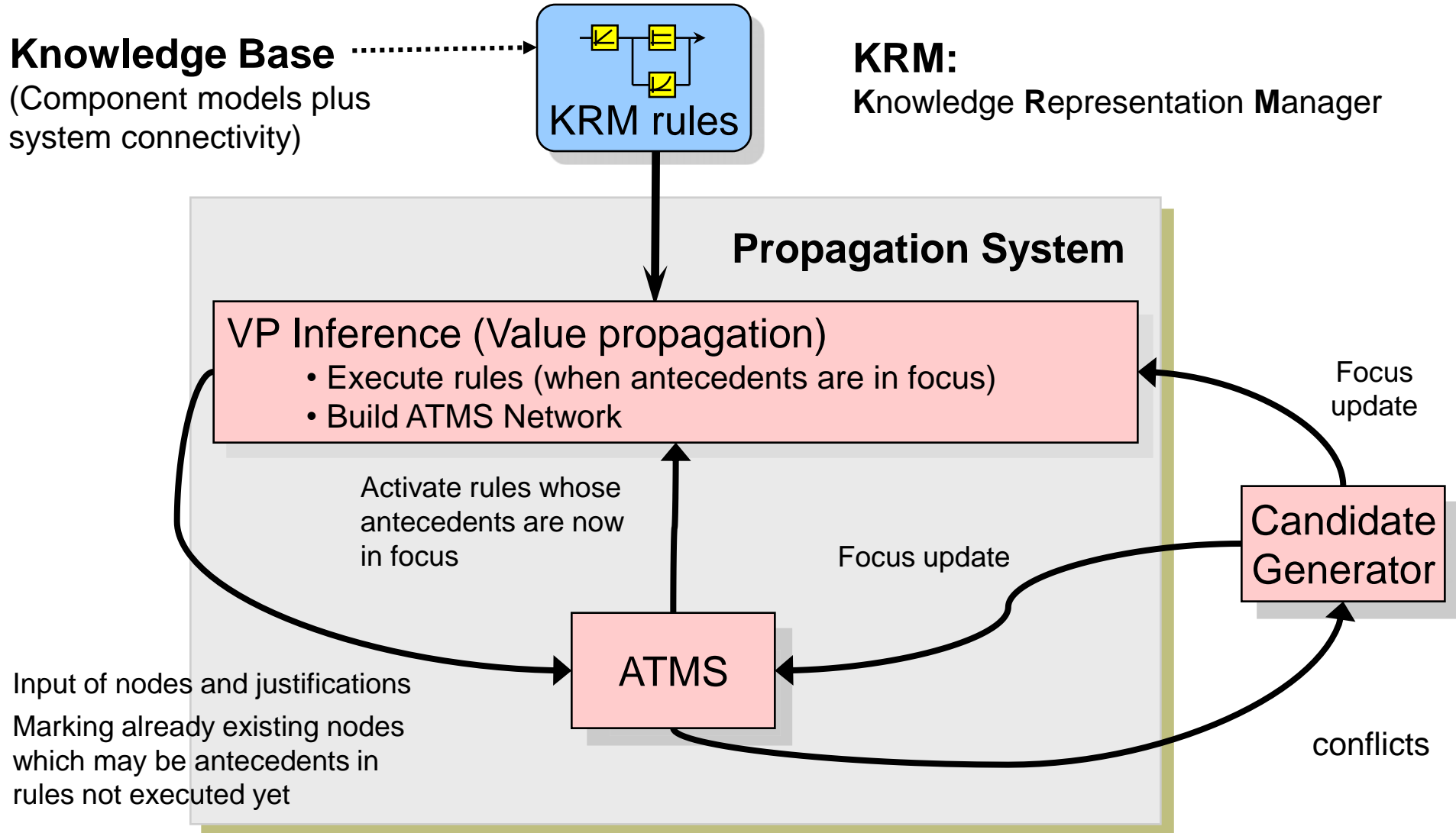
What is propagation in general ?

- Propagation is the distribution of information in a network made of nodes and edges

Separation of value propagation (VP) and ATMS:

- The **ATMS** is responsible for *propagation of environments* in a given network with already determined value dependencies.
- The *propagation of values* is performed by a rule propagator (**VP**) which generates justifications for actual values from the generic values of the behavioural modes of the components. Thus, VP generates the network of value dependencies required by the ATMS.

Value propagation and ATMS



In optimised candidate generators and ATMS's the interface is more complicated.

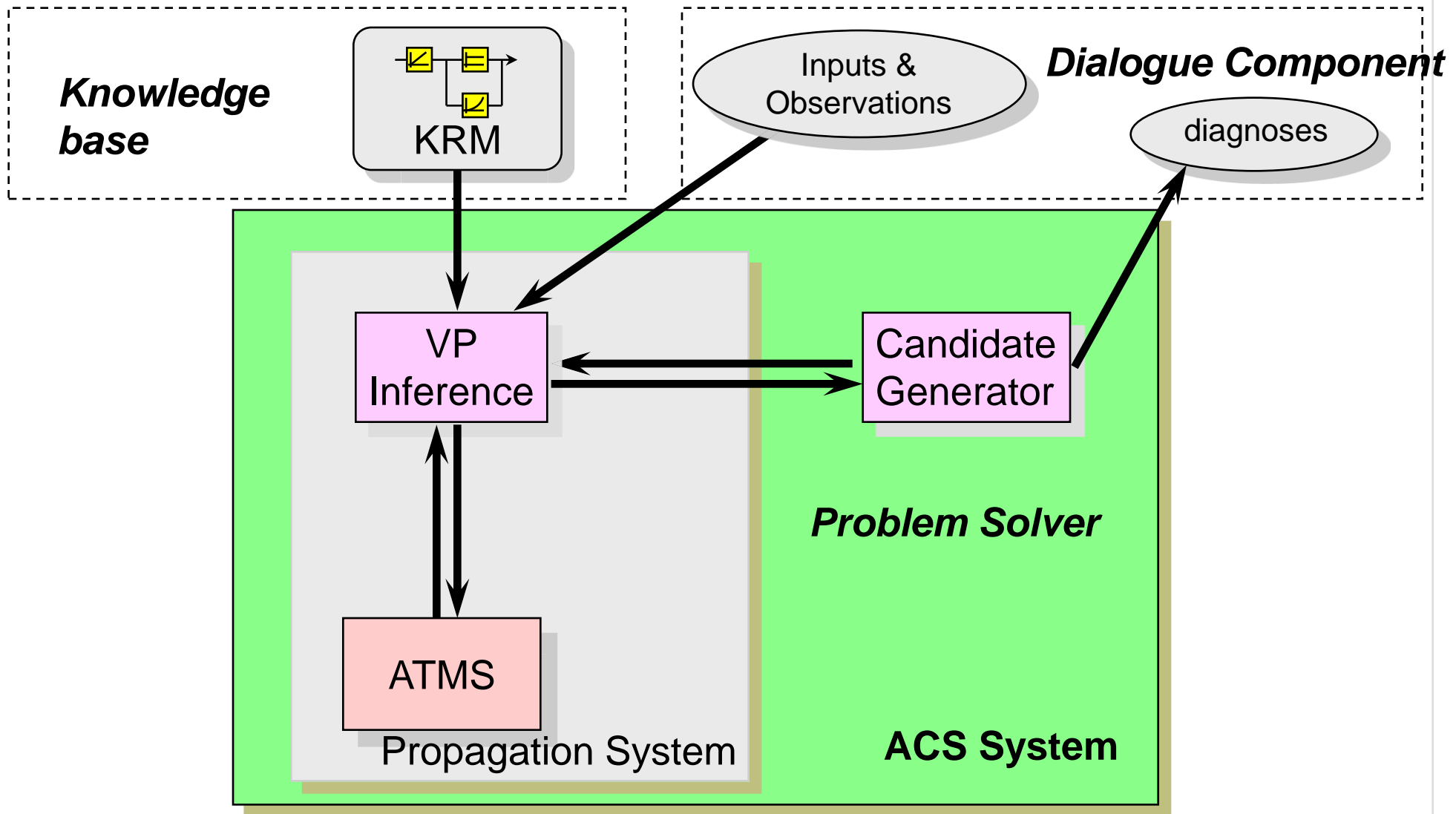
Value propagation and ATMS

What is the benefit of separating value propagation and ATMS?

→ Better software architecture by modularisation

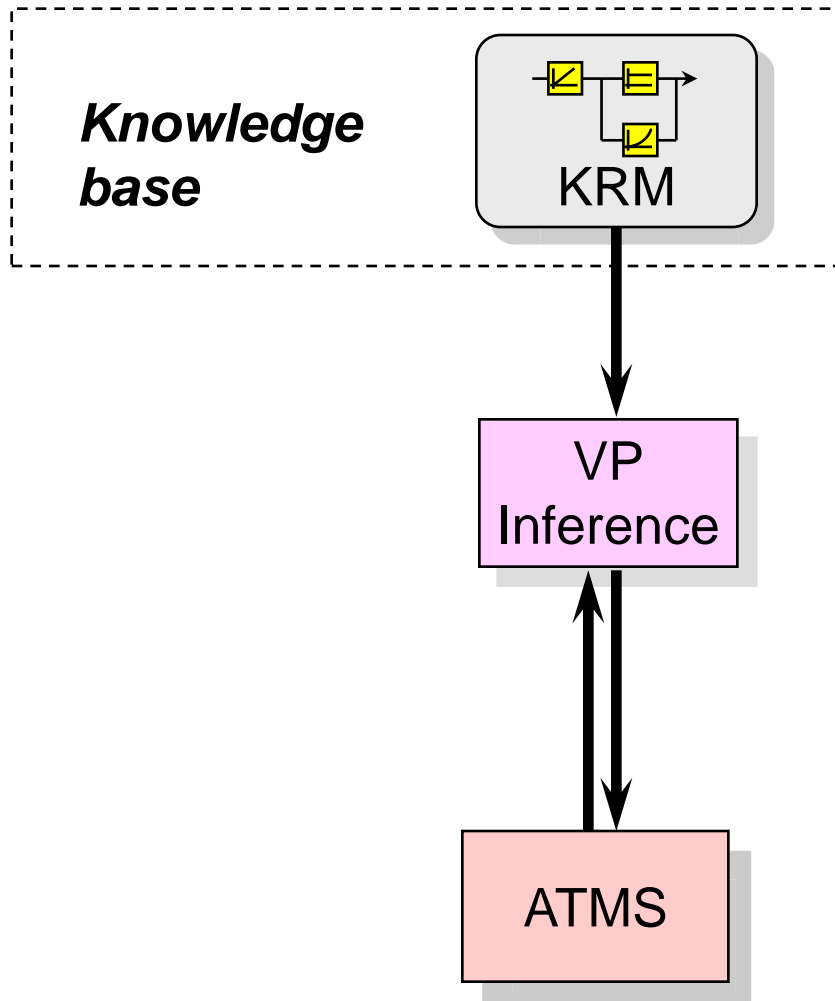
- **Values** are generated mostly from observations (measurements) and intended actions. This is not frequent, thus, there are **not many** values to be considered.
- **Environments** are generated from assumptions about behavioural modes. Of such constructs there exist **a lot of** (even at single faults at least as many as there exist components).
- This makes the update of focus environments much more often to occur than the computation of new values. The update of focus environments may be considered an ATMS internal problem

Interaction of candidate generator, RP and ATMS



ACS: Assumption-based Constraint Solver

Requirement to the knowledge base



What does the knowledge base have to provide to the inference component (problem solver)?

- Rules for the relations of values in each behavioural mode (*component models*)
- Knowledge about the value domains:
When are two values considered contradictory?

Daimler's MDS solves these requirements by offering a constraint language for component models.

Applications of Artificial Intelligence

Sebastian Iwanowski
FH Wedel

Chapter 4: Knowledge-Based Systems

4.4: Machine learning (Case-Based Reasoning)

Case-Based Diagnosis

Input to knowledge base:

- Cases with complete symptom vector and associated faults (classified unambiguously)

a) Classical AI, with similarity measure:

- Similarity measure for incomplete symptom vectors (often weighted between different types of symptoms)

Structure of knowledge base:

- Points in vector space
- Similarity measure

Job of inference engine:

- For a new vector given, find the most similar symptom vector of the knowledge base.
- Assign the same fault to the new vector as associated to the reference vector in the knowledge base (possibly with a probability value).

Case-Based Diagnosis

Input to knowledge base:

- Cases with complete symptom vector and associated faults (classified unambiguously)

b) with neural networks:

- Neural network with input layer (for symptom vector) and output layer (for faults) and (optionally) intermediate layer of nodes and edges, marked by variable weights.

Structure of knowledge base:

- Points in vector space
- Neural network with clearly defined weights (dependent on trained symptom vectors and associated faults)

Job of inference engine:

- Apply new symptom vector to the input layer of the network.
- Read the associated fault from the output layer.

Machine Learning (Case-Based Reasoning)

Generalisation of case-based diagnosis to arbitrary case-based reasoning strategies:

Principle (also called *supervised learning*):

- Given cases as vectors (*complete symptom vectors*): These are “learnt” and build the knowledge base.
- Given new vectors, of which not all parameters are known (*incomplete symptom vectors*): These are to be classified.
- Assign values to the unknown parameters.

Job of inference engine (simple variant):

- For the new vector, find the closest symptom vector learnt by the knowledge base.
- For the unknown parameters of the new vector, assign the same values as in the associated symptom vector learnt by the knowledge base.

This variant only makes sense when the unknown values come from a discrete (better finite) domain !

Machine Learning (Case-Based Reasoning)

Improvement for continuous value domains:

Job of inference engine (better variant):

- For the unknown parameters of the new vector, assign values “in between” values of “nearby” symptom vectors learnt by the knowledge base.

Other mathematical formulation of this method:

- Consider the unknown parameters of the new vectors as function values of the known parameters: Find a continuous function where all vectors learnt by the knowledge base are contained.
- Of this function, assign the function values of the known parameters to the unknown parameters.

Query: How do we get an appropriate function for a given set of reference vectors?

Answer:

- Take a class of functions, each function differing by certain parameters.
- Determine the parameters solving an equation system obtained from the known reference vectors.

Machine Learning (Case-Based Reasoning)

Distinguish between training errors and test errors:

Fixing the polynomial degree for the classification function f will make it impossible that all training examples solve the function f correctly.

→ **training error**

Optimisation goal: Make the training error as small as possible.

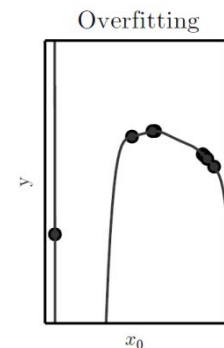
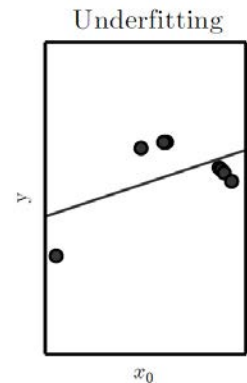
Taking a too small degree for the classification function causes **underfitting**:

Once, a classification function is chosen, this function will classify the test examples.

→ **test error**

Optimisation goal: Make the test error not much bigger than the training error, i.e. minimise the expected difference between test error and training error.

Taking a too high degree for the classification function causes **overfitting**:



graphics from deeplearning book, Goodfellow et al.

Machine Learning (Case-Based Reasoning)

Determining parameters in function classes (regression):

Linear regression:

- Find the weights in a linear function of the form: $f(x_1, x_2, \dots, x_n) = \sum_{i=1}^n w_i x_i$

Generalisation:

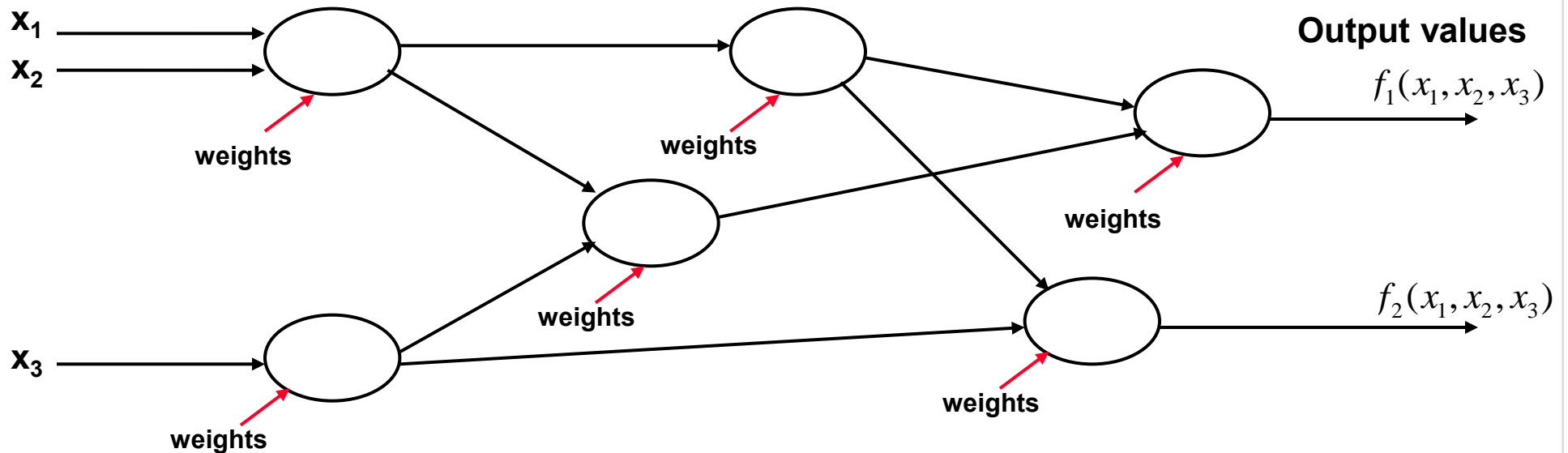
1. Find the weights in a linear equation **system**.
 2. Find the weights in equation systems of higher order.
 3. Find the weights in parametrised inequality systems.
- **Case-based reasoning is designed for systems which *cannot* be modeled easily.**
 - **This is why a higher order equation system does not make sense.**
 - **It is better to work with many weakly connected equation systems and distribute the unknown knowledge.**

Neural Networks

Idea of neural networks:

Given a multi-valued function f (notation: $f_i(x_1, x_2, \dots, x_n)$)

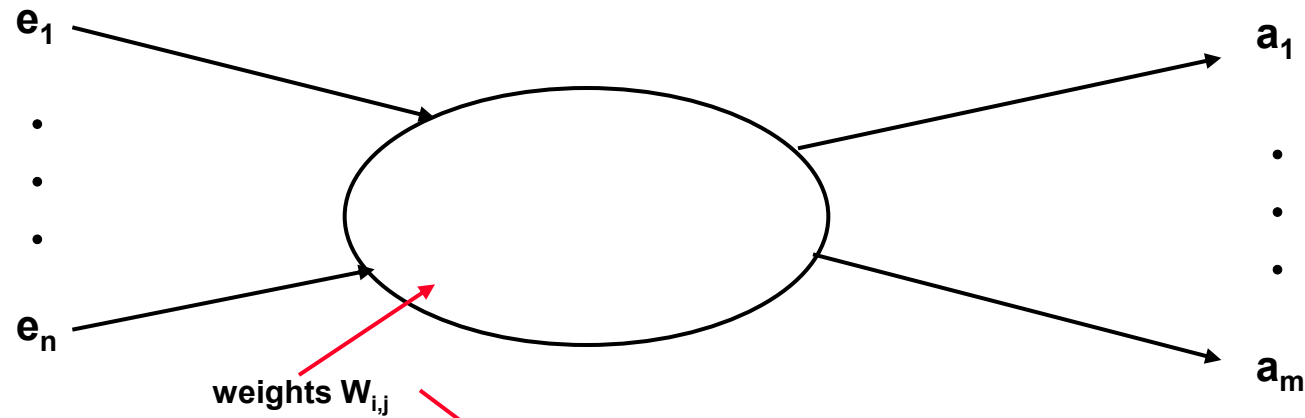
Input values



- The weights may be preset but are adapted to the examples learnt.
- Function values of new inputs are obtained applying the neural network.

Neural Networks

Functionality of a single neuron:

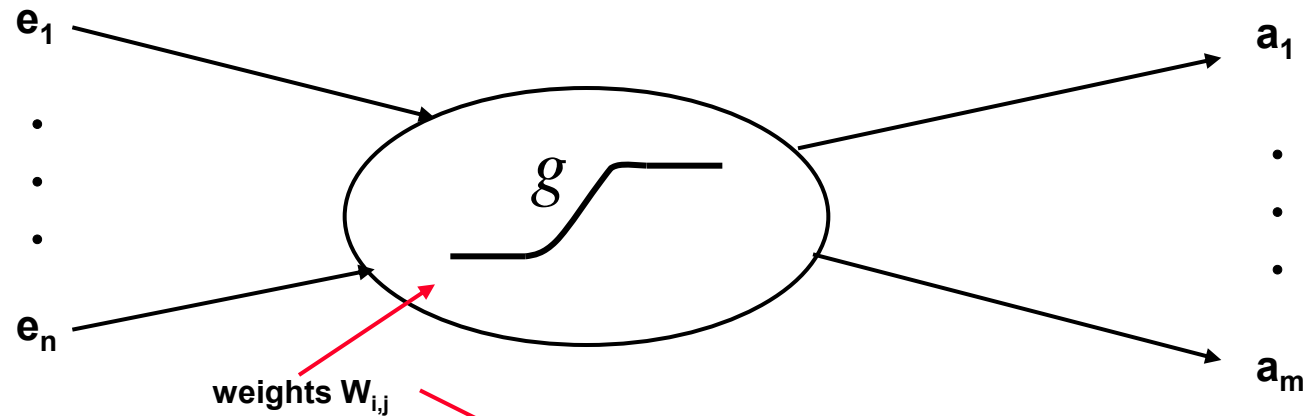


$$a_i(e_1, e_2, \dots, e_n) = \sum_{j=1}^n w_{i,j} \cdot e_j$$

- $A = (a_1, \dots, a_m)$ is a linear function in the e 's. A is represented by an $(m \times n)$ -matrix of the w_{ij} 's.

Neural Networks

Functionality of a single neuron:



$$a_i(e_1, e_2, \dots, e_n) = g\left(\sum_{j=1}^n w_{i,j} \cdot e_j\right)$$

- $A = (a_1, \dots, a_m)$ is a linear function in the e 's. A is represented by an $(m \times n)$ -matrix of the w_{ij} 's.
- g is a generalised threshold function which is the same for all outputs of the same neuron.

Neural Networks

Different layered neural networks:

In standard neural networks, all neurons are placed on certain layers:

- There is an input neuron for each input variable and an output neuron for each output variable. Thus, the input neurons represent linear $(1,n)$ functions and the output neurons linear $(n,1)$ functions.
- Neurons on the same layer have the same distance to input and output.
- Links are only existing between neurons of adjacent layers.
- By default, all neurons of a layer are connected to all neurons of the adjacent layers (one in input, one in output direction). But this may change even dynamically during the learning process.

Neural networks without intermediate layers:

- Neurons of the first layer accepting the inputs are connected to neurons of the second layer providing the outputs.

Neural networks with intermediate layers (*deep learning approach*)

- Input and output layers are connected by further “hidden” intermediate layers.
- The term “deep learning” is usually only applied when there are at least two hidden layers.

Neural networks with feedback (Recurrent Neural Networks, RNN):

- Generation of “memory”

Neural Networks

Basic technique for adjusting the weights (modern approach):

Initialisation of the weights of all neurons:

- Principally, arbitrary weights may be chosen.
- There may be specific initialisation heuristics for special types of networks.

For each (input, output) training sample:

- Forward propagation from input to output
- Comparison between predicted values and real values at the output
- For the error estimate (called „loss function“), different methods may be possible:
e.g. means-squared, cross-entropy
- Backward propagation from output to input: Adjusting the weights considering this sample

„Backpropagation algorithm“ (Rumelhart, 1986)

Details and examples (including implementation) in:

Erik Genthe, „ Backpropagation in neural networks – explained at examples“ (in German), FH Wedel seminar presentation, SS 2020, <https://intern.fh-wedel.de/fileadmin/mitarbeiter/iw/Lehrveranstaltungen/2020SS/Seminar/Ausarbeitung3BackpropagationGenthe.pdf>

Neural Networks

Basic technique for adjusting the weights (modern approach):

Forward and backward propagation in detail:

Forward propagation from input to output:

- The input values are attached to the input layer.
- The output of each neuron is computed with the so far used weights successively (in the order how far the neuron is apart from the input).
- This is repeated until the output layer has got all values.

Backward propagation from output to input:

- The output layer values are compared with the real output values of the sample, and the output error is computed **using the predefined loss function**.
- For each neuron directly connected to the output layer, it is computed how much this contributed to the output error. The more it contributed, the more the corresponding weight is changed.
- This is repeated towards the input layer, i.e. for each input x_i of the latest layer considered (which is the output of the previous layer considered), it is computed how much each neuron of the previous layer contributed to x_i .
- **Note:** The error should not be corrected to zero (danger of overfitting). This is compensated by a **learning rate factor** $\ll 1$ which has to be multiplied with the optimum corrector.
- **Important:** There may be *several* ways to compute the amount of contribution of a neuron.

Neural Networks

Basic technique for adjusting the weights (modern approach):

How to perform backward propagation:

Gradient descent method (the most popular example for backward propagation):

- The impact of a weight to an error is computed by the partial derivative w.r.t. this weight of the error function. The derivative of the function of a neuron corresponds linearly to the input by which the weight is multiplied, but the error function may be nonlinear.
- The easiest formulae describe the impact of the weight of the output layer. Then the partial derivative is rather simple.
- The impact of the weight of earlier layers is obtained by plugging in the dependency of later values from previous weights. Then the derivative becomes more complicated and is highly nonlinear due to the chain rule involving the weights of later neurons.
- If an activation function is involved in a neuron, this must also be considered in the descent function. Then the derivative of the activation function must be considered as well.
Note: Typically, the activation function is not linear but of higher order or even exponential.
- Notation: The derivative of a multidimensional function w.r.t. all involved variables, is called the **Nabla** ∇ **operator** consisting of all single partial derivatives.
- If the function is multi-valued, the Nabla operator is a matrix consisting of the partial derivatives.

Detailed description with example also in:

Michel Belde, Bachelor thesis on „Improvement of a consulting app for the sales department using image recognition“, chapter 3.1, on class website (in German)

Neural Networks

Basic technique for adjusting the weights (modern approach):

Improvements for Backpropagation: Batches

- Consider a set of several input/output samples and feed them together in the network.
- The weights are then adjusted such that the average error is minimised.

Problems with backpropagation in deep networks:

Vanishing / exploding gradient problem:

- If gradients are close to zero, they accumulate in deep networks and vanish until input.
- This is mainly due to unfortunate activation functions.
- Deep networks need activation functions with a derivative not close to zero in all of the domain.
- If gradients are considerably higher than one, they accumulate in deep networks and may explode.
- Deep networks need activation functions with a derivative not much higher than one in all of the domain.

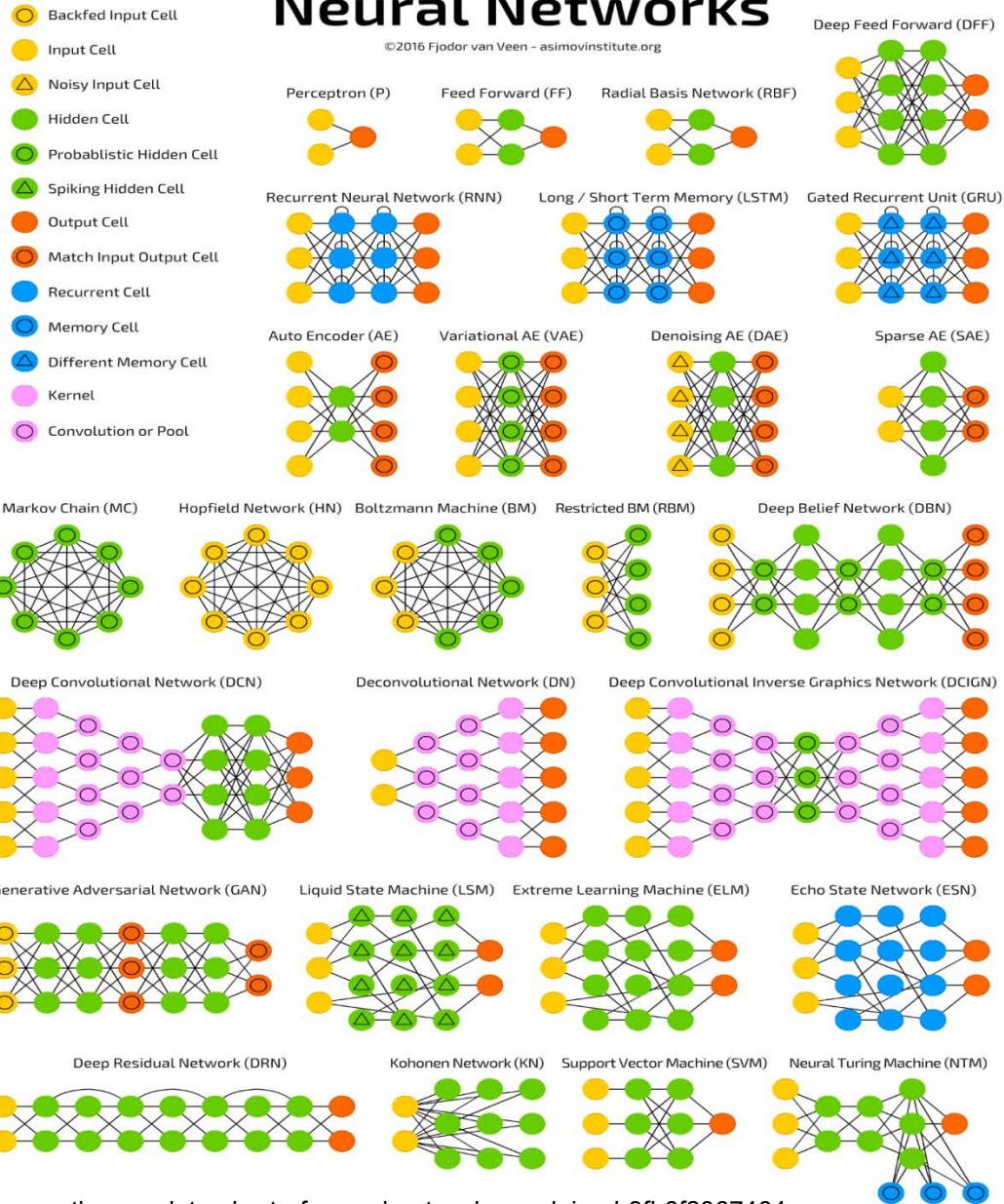
More details in:

Jacob Hansen, „Details of the backpropagation algorithm“, FH Wedel seminar presentation, SS 2019, http://intern.fh-wedel.de/fileadmin/mitarbeiter/iw/Lehrveranstaltungen/2019SS/Seminar/Ausarbeitung1_Backpropagation_Hansen.pdf

Dennis Maas, „Specific methods for training and evaluation of deep neuronal nets“, FH Wedel seminar presentation, SS 2019 (in German), http://intern.fh-wedel.de/fileadmin/mitarbeiter/iw/Lehrveranstaltungen/2019SS/Seminar/Ausarbeitung3_TiefeNetze_Maas.pdf

A mostly complete chart of
Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org



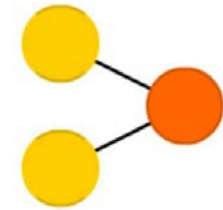
<https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464>

Types of Neural Networks

The first one: Perceptron (1957, Frank Rosenblatt)

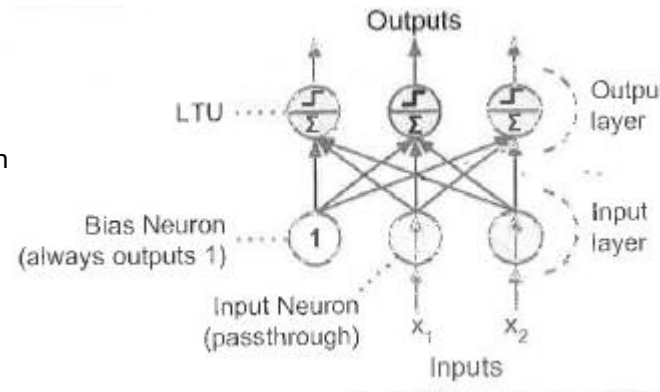
- corresponds to a single regression function. The weights are in the individual input neurons.
- Unique output neuron has a step function only, no further weight.
- cannot compute XOR function (Minsky 1971).
- can be generalised for an n-m function having several outputs.

Perceptron (P)



<https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464>

Aurelien Geron: Hands-On
Machine Learning with SciKit Learn
& Tensorflow, O'Reilly 2017



Weight adjusting technique:

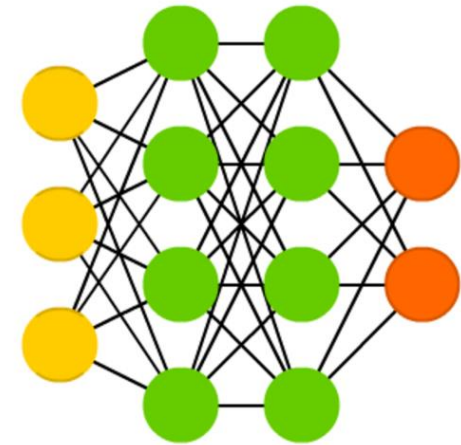
- does not apply backpropagation algorithm as described before.
- Applies Hebb's rule instead: Cells that fire together, wire together.

Types of Neural Networks

Deep Feed Forward Network

- uses backpropagation algorithm as described before (proposed by Rumelhart 1986).
- can compute all logical functions.
- differs from Perceptron not only by internal layers, but also by other activation functions than the step function.
- is nowadays used in a lot of standard learning settings.

Deep Feed Forward (DFF)



<https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464>

Types of Neural Networks

Recurrent Neural Network

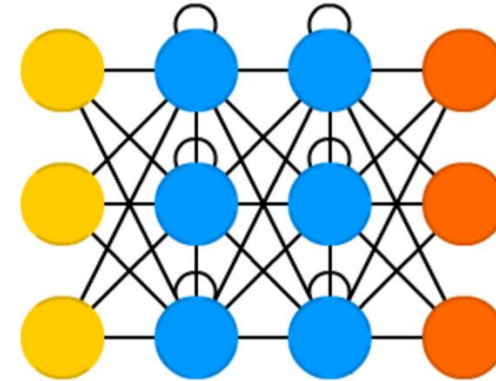
- network with “memory”
- Recurrent neurons receive their own output as an input with a delay.
- suitable for context dependent input: In which order did data occur?
- With the use of simple neurons one can store the order only, by more complicated “memory cells” one can store past information for a given time.
- Rumelhart’s backpropagation algorithm can be adapted when the network is unfolded for several time stamps (only discrete time possible):
backpropagation-through-time algorithm

More details in:

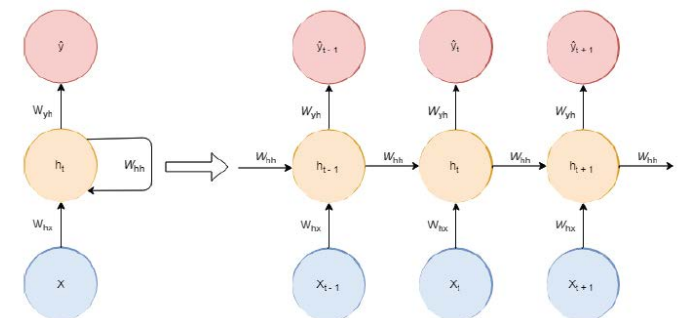
Marcello Attila Messina, „Weight adjustment in neural networks with memory”, FH Wedel seminar presentation, SS 2019 (in German),

http://intern.fh-wedel.de/fileadmin/mitarbeiter/iw/Lehrveranstaltungen/2019SS/Seminar/Ausarbeitung5_RNN_Messina.pdf

Recurrent Neural Network (RNN)



<https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464>

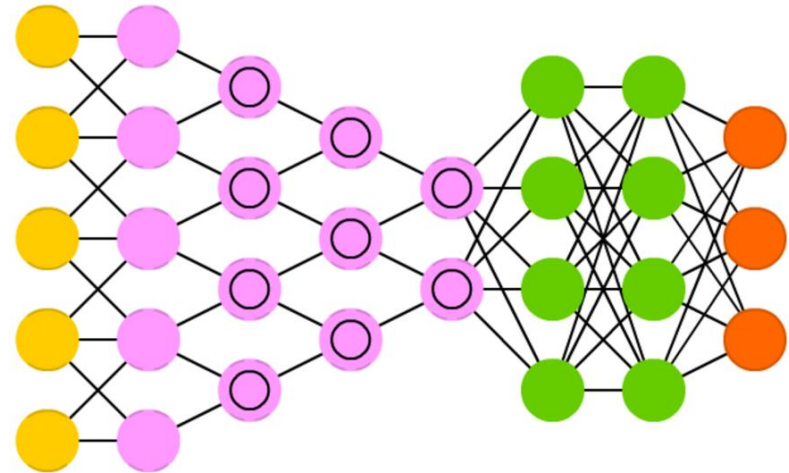


Types of Neural Networks

Deep Convolutional Network

- After the (pink) “convolutional layers”, the (circled) “pooling layers” extract the important features and neglect unimportant ones.
- Unlike in other internal layers, neurons of pooling layers are not connected to all neurons of the adjacent layers.
- This is the modern standard for image recognition and classification.

Deep Convolutional Network (DCN)

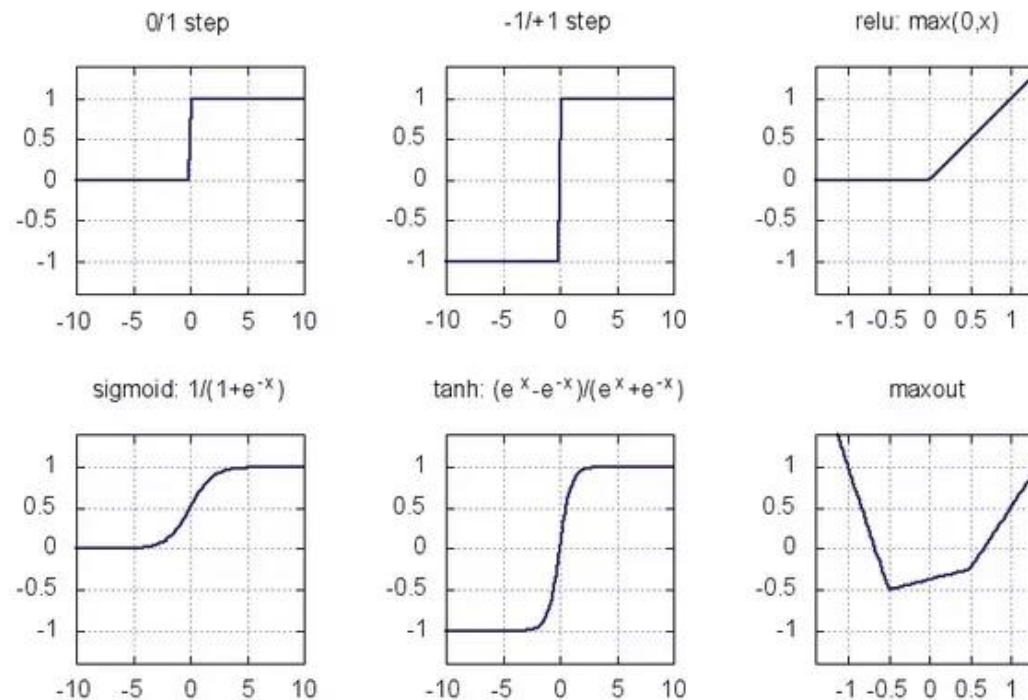


<https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464>

Neural Networks Techniques

Different Activation Functions

- The activation function of a neuron gets the input to that neuron and transforms this into a new value which is really fed into that neuron (applying the weight function).
- It is customary to choose the same activation function for all neurons of a given network.



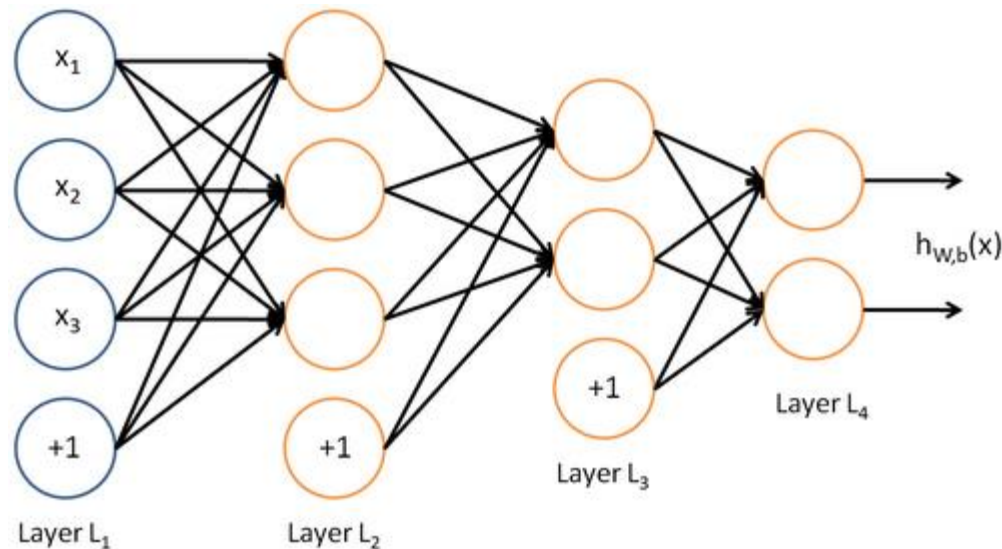
From: Thimo Tollmien, Master thesis on „Optimisation of delay predictions with deep learning“, FH Wedel 2018

Neural Networks Techniques

Special Improvement Techniques

Improvement by bias nodes

- Bias nodes are additional input nodes for each layer of the network. They are not connected with the previous layer. So they feed in extra information.
- Usually they feed in an extra “1” per layer.
- This may help weight adjustment in the training period.



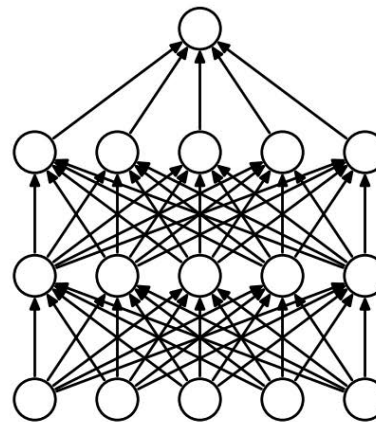
<https://www.quora.com/What-is-bias-in-artificial-neural-network>

Neural Networks Techniques

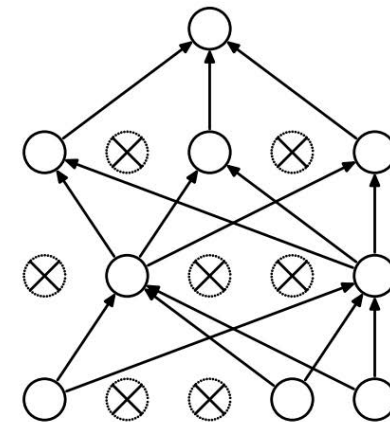
Special Improvement Techniques

Dropout

- avoids overfitting
- Input vectors are fed into the network several times
- Each time a different set of randomly chosen connection is cut.
- If the avoidance of a connection does not increase the training error, then this connection will be omitted in the future.



(a) Standard Neural Net



(b) After applying dropout.

From: Thimo Tollmien, Master thesis on „Optimisation of delay predictions in public transportation with deep learning“, FH Wedel 2018

Neural Networks Techniques

Training Techniques

The quality of a trained neural network decisively depends on the quality of input samples and features it was trained with.

Cross validation

- Split the training set into two (or more) parts: Train only with one part and test the result with the other. Do the same vice versa.

Feature selection

- Add or remove certain features (dimensions) of the training set.

Machine learning without known results

Data mining (unsupervised learning)

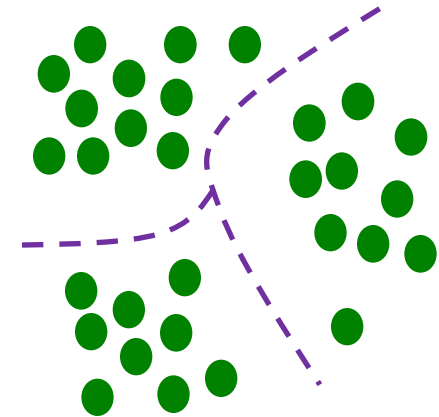
Given some data input. Is it possible to classify this input into different categories?

Different classification methods:

- Cluster detection
- Greatest gap detection

Common clustering techniques:

- nearest neighbor
- k-means (for k clusters)



From: Dirk Lützelberger, NXP, Colloquium talk on Machine Learning FH Wedel 2018

Data mining is an own field with a rich variety of classification techniques.

„**Big data**“

Learning in this context means: Continuous adaptation to changing data

Machine learning without known results

Data mining (unsupervised learning)

Given some data input. Is it possible to classify this input into different categories?

k-means details (basic method):

- 1) Decide for a certain number k and keep it fixed.
- 2) Select k arbitrary clusters and compute their means.
- 3) Determine the variance (sum of the square distances to the mean) for the clusters.
- 4) Reshuffle the k clusters, and repeat Step 3). Store the clusters with the respective minimum.
- 5) Repeat Steps 3) and 4) until the minimum does not “considerably” improve.

There are a lot of refinements for this method.

Note: As long as Step 1) is observed, k-means does not help to detect the „optimum“ k .

Without Step 1) this method can be generalised to find an „optimum“ k . **x-means**

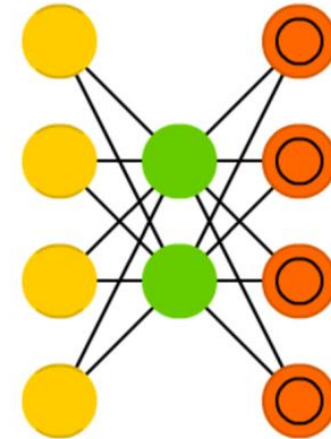
Remark: The problem „Find the best arrangement of k clusters such that the distance to the respective means is minimised“ is NP-hard (i.e. not likely to be computable efficiently)

Types of Neural Networks

Autoencoder

- suitable for unsupervised learning
- Input and output layers should have the same number of neurons.
- Hidden layers should have fewer neurons than input and output layers.

Auto Encoder (AE)



<https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464>

Machine Learning in Practice

Graduation theses in Machine Learning supervised by iw

Tjark Smalla (WS 2016): Implementation of a Neural Network for Order Prediction and Comparison with an Existing Logistic Regression

Otto GmbH

Bronislav Koch (WS 2017): Determination of a set of clients with maximum probability for project success in a multivariate model,

Sven Mahn IT GmbH

Lasse Karls (WS 2017): Graph-based feature extraction to improve machine learning in predicting the business affiliation of a Signal Iduna customer,

Signal Iduna

Thimo Tollmien (SS 2018): Optimizations of Delay Predictions in Local Public Transport Using Deep Learning,

Master thesis in cooperation with **HBT**

Michel Belde (WS 2018): Improvement of a consulting app for the sales department using image recognition,

akquinet Engineering GmbH

Dennis Maas (SS 2019): Transformation invariant bar code recognition using neural networks,

Opus//G GmbH

Machine Learning in Practice

Graduation theses in Machine Learning supervised by iw

Henning Brandt (WS 2019): *Implementation of a model for calculating the concentration of volatile organic compounds in a multi-capillary gas chromatograph using machine learning,*

bentekk GmbH / Dräger AG

Linus Stenzel (WS 2019): *Development of an artificial intelligence with human play style in the game Canasta,*

LITE Games GmbH

Frederik Schnoege (SS 2020): *Use of natural language processing in IT support,*
Master thesis in cooperation with **Beiersdorf Shared Services GmbH**

Vincent Grohne (WS 2020): *Comparison of different machine learning models for the detection of potentially failed securities deliveries,*

Berenberg Gossler KG (Bank)

Ines Kemsies (WS 2021): *Prediction of system failures by using a recurrent neural network,*
Akquinet engineering GmbH

Shwetha Mohan Kumar (WS 2021): *Computation of delays in the public transportation of Hamburg using Deep Neural Networks,*

Master thesis in cooperation with **HBT**

Applying Neural Networks in Practice

Software Kits

SciKit Learn (<http://scikit-learn.org>)

- open-source library implemented in Python

Tensorflow (<http://tensorflow.org>)

- open-source library implemented in Python und C++
- developed at Google, used in internal software

```
1 model.add(layers.Dense(200, activation="sigmoid", input_shape=(312,)))  
2 model.add(layers.Dropout(0.1, noise_shape=None, seed=None))
```

From: Thimo Tollmien, Master thesis on „Optimisation of delay predictions in public transportation with deep learning“, FH Wedel 2018

Neural Networks

References

Ian Goodfellow, Yoshua Bengio, and Aaron Courville: *Deep Learning*. MIT Press, 2016.
<http://www.deeplearningbook.org>

Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn & TensorFlow*. O'Reilly, 2017.

Other modern ML techniques

Support Vector Machines

- applies high dimensional vector algebra
- tries to find separating hyperplanes between different sets of classification
- may be regarded as follow-up of the classical approach

Literature: (available in our library in 8.5.3)

Nello Christianini / John Shawe-Taylor:: *An Introduction to Support Vector Machines*. Cambridge Univ. Press, 2000 (2006).

Vojislav Kecman: *Learning and Soft Computing*. MIT Pr. 2001

Machine Learning (Case-Based Reasoning)

What is the crucial difference between neural networks and „classical“ CBR systems?

- Neural networks **distribute** the knowledge about the cases learnt.

Theoretical advantages of distribution:

- Arbitrariness of function class chosen does not play such an important role.
- Intransparent cases are handled by an intransparent method:
The distributed method is “self-adjusting”.

Practice shows:

- Good neural networks need fewer training cases than classical CBR systems.
- Neural networks provide better classification results.
- **Deep networks made a great boost to AI in general:** They are applied in products already.

Summary: Machine Learning (Case-Based Reasoning)

Let's get back to our sample application diagnoses for comparing different KBR methods:

Advantages and Disadvantages:

- **The method is simple.**
- + - The diagnosis of the run time component is very fast.
- + - Knowledge acquisition can easily be automatised.
- - The knowledge base can only be generated for systems where experience is given.
- - The knowledge base consumes a lot of storage (classical approach only).

Summary: Machine Learning (Case-Based Reasoning)

Advantages and Disadvantages:

- **The knowledge base does not contain any other structural knowledge than the similarity measure or the NN.**
- + - All application domains are equally suited.
- + - The same inference engine may be applied for totally different application domains.
- + - For systems *without a reasonable model*, classification results are rather good (at least for neural networks).
- - Even with a small change of the system, the knowledge base cannot be used reliably.
- - Similarity measure and neural network are arbitrary.
- - Each run time diagnosis may be wrong.
- - **The result is not justifiable (at least for neural networks): Distrust “algorithms”.**

In this context, „algorithms“ denote statistical algorithms for backpropagation plus the network design.

This is not to be confused with algorithms outside AI or algorithms for symbolic (rule-based) AI.



Applications of Artificial Intelligence

Sebastian Iwanowski
FH Wedel

Chapter 4: Knowledge-Based Systems

4.5: Concluding Comparison of the Different Reasoning Techniques

Application from practice: Technical diagnosis

Run time system:

(knowledge-based systems call this **problem solver / inference engine**)

Input:

- Setting certain control inputs
- Observing values depending on this setting

Output:

- A unique instruction how to repair which component

This is where diagnostic systems do **not** differ !

Application from practice: Technical diagnosis

Knowledge-based diagnosis:

1) Knowledge acquisition: Input into knowledge base

- symptom-based (rule-based)
 - model-based
 - case-based (machine learning)
- } as alternatives

2) Knowledge structure

- depends on knowledge acquisition

3) Knowledge processing by the problem solver

- depends on knowledge structure

This is where diagnostic systems may differ !

1. Symptom-Based Diagnosis

Input to knowledge base:

- Causing and manifest faults for the overall system
- Possible symptoms (measurements)
- Relations between faults and symptoms (rules)

Structure of knowledge base:

- Semantic network (e.g., fault networks, decision trees)

Job of inference engine:

- Navigation in semantic network

This is „classical“ expert system technology

2. Model-Based Diagnosis

Goal:

- fast knowledge acquisition
- exact and provable solution of problem solver

Input to knowledge base:

- system model: hierarchical structure of the system (+ how the components are connected)
- component models

Structure of knowledge base:

- constraint network (assembled automatically)

Job of inference engine:

- GDE approach: conflict-based candidate generation
- sophisticated acceleration techniques in order to get reasonable run time behaviour (only discussed for candidate generation, others not discussed in class)

3. Case-Based Diagnosis (Machine Learning)

Input to knowledge base (supervised approach only):

- Cases with complete symptom vector and associated faults (classified unambiguously)

Structure of knowledge base:

a) Classical AI, with similarity measure:

- Similarity measure for incomplete symptom vectors (often weighted between different types of symptoms)

b) with neural networks:

- Neural network with input layer (for symptom vector) and output layer (for faults) and (optionally) intermediate layer of nodes and edges, marked by variable weights.

Job of inference engine:

- a)**
 - For a new vector given, find the most similar symptom vector of the knowledge base.
 - Assign the same fault to the new vector as associated to the reference vector in the knowledge base (possibly with a probability value).
- b)**
 - Apply new symptom vector to the input layer of the network.
 - Read the associated fault from the output layer.

Systematic classification of inference techniques

- **heuristic:**

if <features> then <solution>

(usually the solution has got disjunctive alternatives,
in modern systems this may be combined with probabilities)

- **causal:**

- overlapping classification:

if <solution> then <features>

- structural classification:

local behavioural model => system function

(search for the best behavioural models being consistent with the
observed overall system behaviour)

Systematic classification of inference techniques

- **case-based (machine learning approach):**

Given cases with **features** and **solution**

Apply regression technique (**interpolation**)

- with similarity measure:

arbitrary regression

- in neural networks

distributed linear regression

- in data mining (unsupervised approach):

features from knowledge base => new correlations

Supplementary, apply one of the other methods (heuristic or causal)

Systematic classification of inference techniques

Classification of knowledge-based inference by depth

- heuristic *for relatively flat knowledge*
- causal *for flat and deep knowledge*
- case-based (similarity measure, neural network, data mining) *for very flat knowledge*



In principle, this may be arbitrarily combined with other dimensions of knowledge quality:

- certain vs. uncertain (consider the probability of a statement)
- exact vs. fuzzy (consider the accuracy of a statement)

Concluding comparison for applicability in practice

	symptom-based	case-based	model-based
fast run time component	++	++	o
fast knowledge acquisition	o	++	+
fits to systems of complex structure	--	++	++
fits to systems containing complex components	+	++	--
reusability of knowledge	o	--	++
fits to diagnosis of unknown faults	-	a) -- b) -	+
is readily available at product launch	o	a) -- b) -	++
provable reliability of diagnoses	+	a) o b) --	++

Interview about comparison causal knowledge vs. probabilistic knowledge:

<https://www.quantamagazine.org/to-build-truly-intelligent-machines-teach-them-cause-and-effect-20180515/>