Disclaimer

The following slides contain mainly examples for the chapters 2-11 of the lecture „Artificial Intelligence 1“. They do not repeat the stuff in the lecture videos resp. slides.

Often they start with a task and present the solution in subsequent videos. You probably profit the most from the slides, if you try to solve the task before looking at the solution.

The language of the slides is a mix of German and English.

Frank Puppe

Chapter 2: Intelligent Agents

KI@LS6: Bildverarbeitung

- Transkription von Texten
  - Layouterkennung von eingescannten Seiten
    - Drucktexte: Überschriften, Text in Spalten, Bilder, Kopf- und Fußzeile, Fußnoten
    - Handschriftliche Notizen
    - Tabellen
    - Rechnungen
  - OCR von Texten (einschl. Nachkorrektur)
  - Notenerkennung (mittelalterliche Musik)
  - Extraktion von Evaluationsergebnissen aus wissenschaftlichen Publikationen
- Medizinische Bild- und Videoerkennung
  - Polypenerkennung in Koloskopie-Videos in Echtzeit
  - Medizinische Bildinterpretation zur semiautomatischen Dokumentation
  - Semiautomatische Annotation (Bilder und Befundberichte)
  - Tutorsystem zur radiologischen Befundung

KI@LS6: NLP (Textverarbeitung)

- Textverarbeitung (NLP)
  - Information Extraction
    - Befundberichte, Arztbriefe
    - medizinische Leistungsanforderungen
    - Juristische Lösungsskizzen
    - Gerichtsurteile
  - Chatbots
    - Ethische Diskussionen zur KI
    - Juristische Falleingabe
  - Analyse literarischer Texte
    - Personen-Erkennung und Ko-Referenz-Resolution
    - Figurennetzwerke
    - Handlungsanalyse

KI@LS6: Weitere Themen

- Simulation und Kalibrierung von Heizungsanlagen
- Automatische Korrektur von Übungsaufgaben in formalen Sprachen
- Kolloquiumsplanung für Schulen
- KI-Bots für Spiele wie Civilization II
- Anamnese-Fragebogen für Patienten
- Optimierung von Richtanlagen (Parameter für Maschinen)
- Workflow-Optimierung für bildgebende Verfahren im Krankenhaus
- Herleitung radiologischer Kennzahlen aus Krankenhausinformationssystem (Datawarehouse)

Rough Performance Measure & Environment for Projects

			Performance Measure	Environment					
				observable	# Agents	Deterministic	Sequential	Static	Discrete
Bildverarbeitung				fully	single	no?	episodic	yes	no
	Layouterkennung von Dokumenten		erkannte Regionen / Zeilen / Tabellenelemente / Artefakte						
	OCR von Textregionen in Dokumenten		erkannte Zeichen; Konfidenz				sequential		
	Notenerkennung		erkannte Noten mit Attributen (Tonhöhe, Verbundenheit, Silben)						
	Publikationsextraktion		Evaluationsergebnisse (Zahlen, Aufgaben, Daten, Methoden/Systeme)				sequential	no	
	Regionen-Erkennung in Endoskopie-Videos		Erkannte Regionen (Echtzeit)						
	Regionen-Erkennung in Endoskopie-Bildern		Erkannte Regionen						
Sprachverarbeitung				fully	single	no?	yes?	yes	yes
	Informations-Extraktion		Extrahierte Informationen						
		in Befundberichten							
		Leistungsanforderungen							
		Lösungsskizzen							
		Gerichtsurteile							
	Literarische Textanalyse								
		Personen und Ko-Referenz	Erkannte Personen						
		Figurennetzwerke	Personen mit Eigenschaften und Relationen						
		Szenenerkennung und Handlungsanalyse	Plausibilität der Szenen und ihrer erkannten Handlungen						
	Chat-Bots			partially	multi	no	yes	no	yes
		Ethische Diskussions-Bots	Abdeckung Argumente, "Natürlichkeit"						
		Fallberatungs-Bots	Verständnis des Falles, Lösungsvorschläge						
Sonstiges									
	Simulation von Heizungsanlagen		Übereinstimmung mit realer Anlage / Qualität von Vorhersagen	partially	single	no?	yes	no	no
	Aufgabenkorrektur für formale Sprachen		Finden aller Fehler	fully	single	yes	yes	semi	yes
	Kolloquiumsplanung für Schulen		Plan, der Constraints einhält und ggf. optimiert	fully	single	yes	yes	semi	yes
	KI-Bots für Civilization II		gute Agenten-Performance im Spiel	partially	multi	no	yes	dynamic	yes
	Anamnese-Fragebogen		Nutzer sollen Fragebogen schnell und korrekt ausfüllen						
		für Dokumentation							
		für Beratung	Zusätzlich: Herleitung von Lösungen						
	Optimierung von Richtanlagen		Verbesserte Maschinen-Einstellungen	partially	single	no	yes/no	semi	no
	Workflow-Optimierung für Untersuchungen		Insgesamt schneller Workflow mit wenig Kommunikations-Overhead	partially	multi	no	yes	dynamic	no
	Radiologische Dashboard		Kennzahlenberechnung						

Chapter 3: Solving Problem by Searching

Eimer Problem

- Sie haben zwei Eimer, in den einen Eimer passen 5 Liter Wasser, in den anderen 3 Liter Wasser.
- Sie wollen 4 Liter in einem Eimer haben. Ihre einzigen Operationen sind das Ausschütten, das Auffüllen oder das Umschütten eines Eimers in den anderen (Sie haben beliebig viel Wasser zur Verfügung).
  1. Zeichnen Sie den vollständigen Suchbaum bis zur ersten Lösung.

     Hinweis: Kennzeichnen Sie Zustände, die bereits irgendwo im Suchbaum aufgetreten sind, und expandieren Sie diese nicht weiter.
  2. Gibt es mehr als eine Lösung?
  3. Geben Sie an, welche Suchstrategien für das Problem am besten geeignet sind?

Suchbaum für Eimerrätsel

Tiefe	Suchbaum für Eimer-Problem: *Inhalt_Großer_Eimer, Inhalt_Kleiner_Eimer*																						
0	**0,0**																						
1	**5,0**												**0,3**										

Suchbaum für Eimerrätsel

Tiefe	Suchbaum für Eimer-Problem: *Inhalt_Großer_Eimer, Inhalt_Kleiner_Eimer*														
0	**0,0**														
1	**5,0**												**0,3**		
2	0,0	5,3		**2,3**									5,3	0,0	**3,0**

Suchbaum für Eimerrätsel

Tiefe	Suchbaum für Eimer-Problem: *Inhalt_Großer_Eimer, Inhalt_Kleiner_Eimer*																						
0	**0,0**																						
1	**5,0**												**0,3**										
2	0,0	5,3		**2,3**									5,3	0,0	**3,0**								
3		5,0	0,3	5,0	3,0	**2,0**									5,0	0,0	**3,3**						

Suchbaum für Eimerrätsel

Tiefe	Suchbaum für Eimer-Problem: *Inhalt_Großer_Eimer, Inhalt_Kleiner_Eimer*																					
0	**0,0**																					
1	**5,0**											**0,3**										
2	0,0	5,3		**2,3**								5,3	0,0	**3,0**								
3		5,0	0,3	5,0	3,0	**2,0**								5,0	0,0	**3,3**						
4						0,0	5,0	2,3	**0,2**							5,3	0,3	3,0	**5,1**			

Suchbaum für Eimerrätsel

Tiefe	Suchbaum für Eimer-Problem: *Inhalt_Großer_Eimer, Inhalt_Kleiner_Eimer*																						
0	**0,0**																						
1	**5,0**											**0,3**											
2	0,0	5,3		**2,3**								5,3	0,0	**3,0**									
3		5,0	0,3	5,0	3,0	**2,0**								5,0	0,0	**3,3**							
4						0,0	5,0	2,3	**0,2**							5,3	0,3	3,0	**5,1**				
5									0,3	0,0	**5,2**								3,3	5,0	**0,1**		

Suchbaum für Eimerrätsel

Tiefe	Suchbaum für Eimer-Problem: *Inhalt_Großer_Eimer, Inhalt_Kleiner_Eimer*																					
0	**0,0**																					
1	**5,0**											**0,3**										
2	0,0	5,3		**2,3**								5,3	0,0	**3,0**								
3		5,0	0,3	5,0	3,0	**2,0**								5,0	0,0	**3,3**						
4						0,0	5,0	2,3	**0,2**							5,3	0,3	3,0	**5,1**			
5									0,3	0,0	**5,2**								3,3	5,0	**0,1**	
6											4,3	…									**1,0**	…

Suchbaum für Eimerrätsel

Tiefe	Suchbaum für Eimer-Problem: *Inhalt_Großer_Eimer, Inhalt_Kleiner_Eimer*																						
0	**0,0**																						
1	**5,0**											**0,3**											
2	0,0	5,3		**2,3**									5,3	0,0	**3,0**								
3		5,0	0,3	5,0	3,0	**2,0**									5,0	0,0	**3,3**						
4						0,0	5,0	2,3	**0,2**								5,3	0,3	3,0	**5,1**			
5									0,3	0,0	**5,2**									3,3	5,0	**0,1**	
6											4,3	...										**1,0**	...
7																						**1,3**	...

Suchbaum für Eimerrätsel

Tiefe	Suchbaum für Eimer-Problem: *Inhalt_Großer_Eimer, Inhalt_Kleiner_Eimer*																						
0	**0,0**																						
1	**5,0**												**0,3**										
2	0,0	5,3		**2,3**									5,3	0,0	**3,0**								
3		5,0	0,3	5,0	3,0	**2,0**									5,0	0,0	**3,3**						
4						0,0	5,0	2,3	**0,2**								5,3	0,3	3,0	**5,1**			
5									0,3	0,0	**5,2**									3,3	5,0	**0,1**	
6											4,3	...										**1,0**	...
7																						**1,3**	...
8																						**4,0**	...

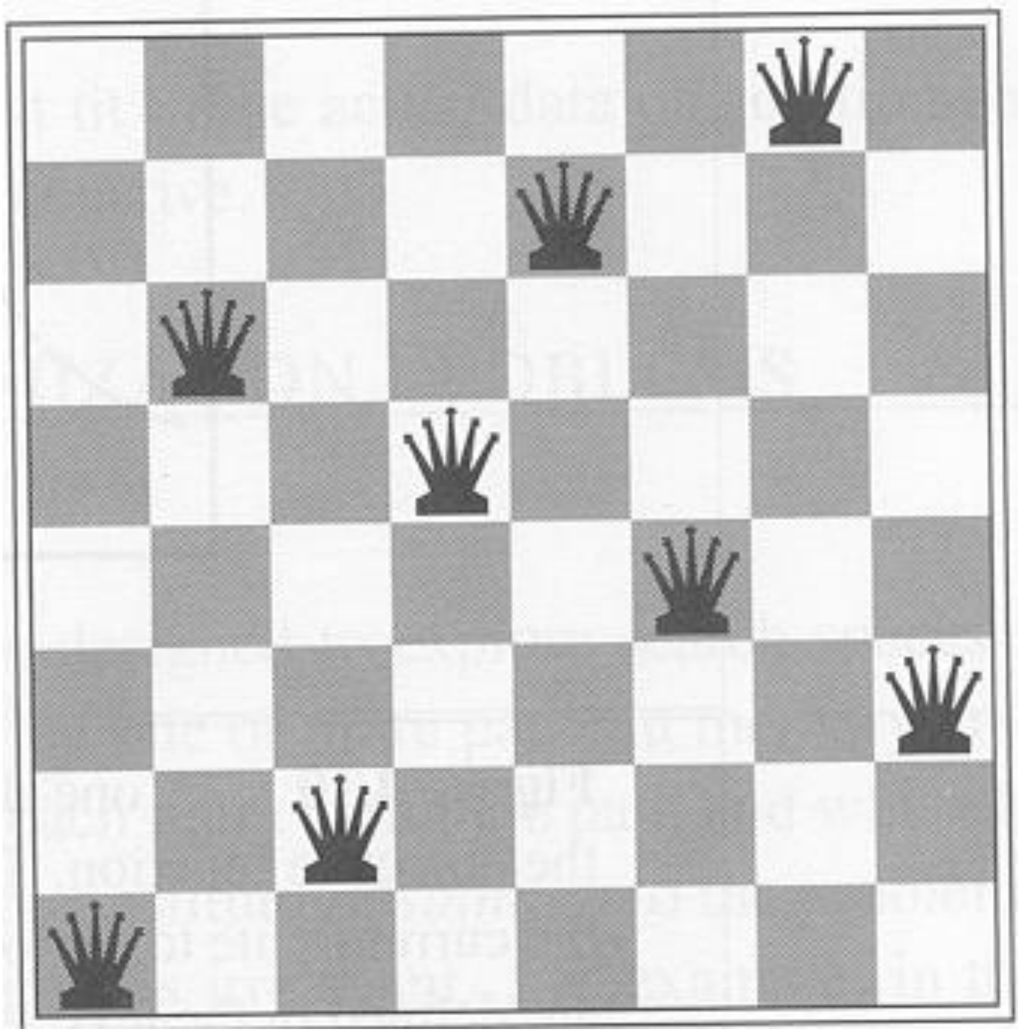
Lösungen

- Die erste Lösung findet sich auf Ebene 6.
- Die zweite Lösung findet sich auf Ebene 8.
- Es eignen sich Breitensuche und Iterative Tiefensuche
  - Tiefensuche eignet sich nicht, da sie in Endlosschleifen führt
  - A* eignet sich nicht, da es keine brauchbare optimistische Heuristik gibt.
  - Bei Breitensuche kann man sich alle bekannten Zustände merken, um sie nicht zu wiederholen $\rightarrow$ beträchtlicher Effizienzgewinn

Chapter 4: Search in Complex Environments

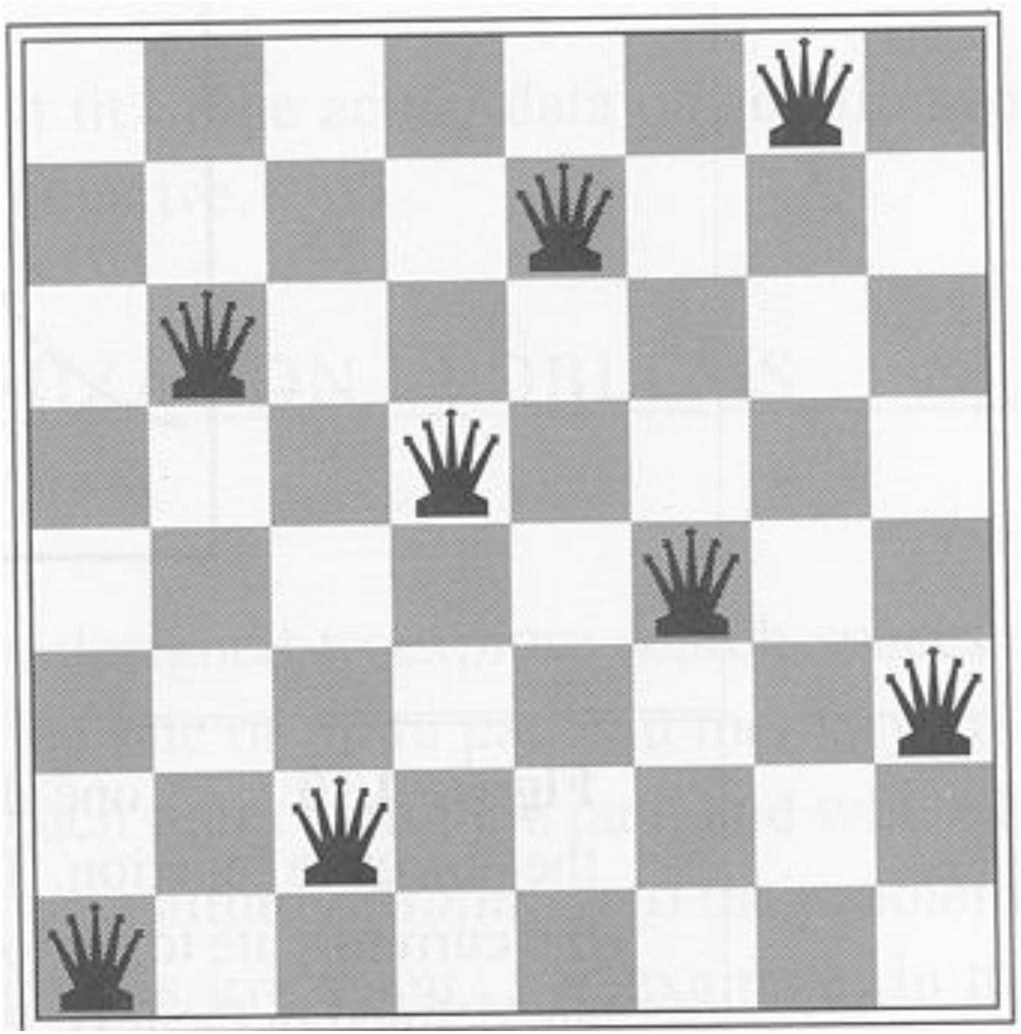
Improvements to Hill-Climbing

- **Sideways move**: Limited; to pass shoulders but to avoid infinite loops in e.g. flat maxima
- **Stochastic hill-climbing**: Random selection of all improvements
  - **First-choice hill-climbing**: Generate sucessor-nodes and take the first improvement
- **Random-restart hill-climbing**: Repeat hill-climbing with randomly generated initial states
- **Simulated Annealing:** Allow worsening with a low probability
- **Local beam search:** Simultaneous search an several paths



Improvements to Hill-Climbing

- **Sideways move**: Limited; to pass shoulders but to avoid infinite loops in e.g. flat maxima
- **Simulated Annealing:** Allow worsening with a low probability



Improvements to Hill-Climbing

- **Sideways move**: Limited; to pass shoulders but to avoid infinite loops in e.g. flat maxima
  - not applicable
- **Simulated Annealing:** Allow worsening with a low probability
  - might choose any move, e.g. G1 -> G2
  - *but some are more probable than others*

function SIMULATED-ANNEALING(*problem*, *schedule*)
  current ← *problem*.INITIAL
  for $t = 1$ **to** ∞ **do**
    T ← *schedule*(t)
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    ΔE ← VALUE(*current*) – VALUE(*next*)
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{-\Delta E/T}$

	A	B	C	D	E	F	G	H
1	3	3	3	3	2	3	x	3
2	3	3	4	2	x	4	2	4
3	2	x	3	3	4	4	2	3
4	3	2	4	x	4	3	3	2
5	3	3	4	3	4	x	2	3
6	3	3	3	2	4	3	2	x
7	3	3	x	2	2	3	3	3
8	x	3	3	2	2	3	2	3

One of many possible search paths for Simulated Annealing

1

3	3	3	3	2	3	x	3
3	3	4	2	x	4	2	4
2	x	3	3	4	4	2	3
3	2	4	x	4	3	3	2
3	3	4	4	4	x	2	3
3	3	3	2	4	3	2	x
3	3	x	2	2	3	3	3
x	3	3	2	2	3	2	3

2

			4	1		1	
			4	x		x	
	x		5	4		2	
			x	5		3	
			6	4	x	2	
			4	4		2	x
		x	4	2		3	
x			4	2		2	

1

2				x		2	
2						x	
1	x					3	
2			x			2	
2					x	2	
1						2	x
2		x				3	
x						2	

1

2				x			2
2						x	2
1	x						3
2			x				2
2					x		1
x							x
2		x					2
1	3	3	2	1	2	2	1

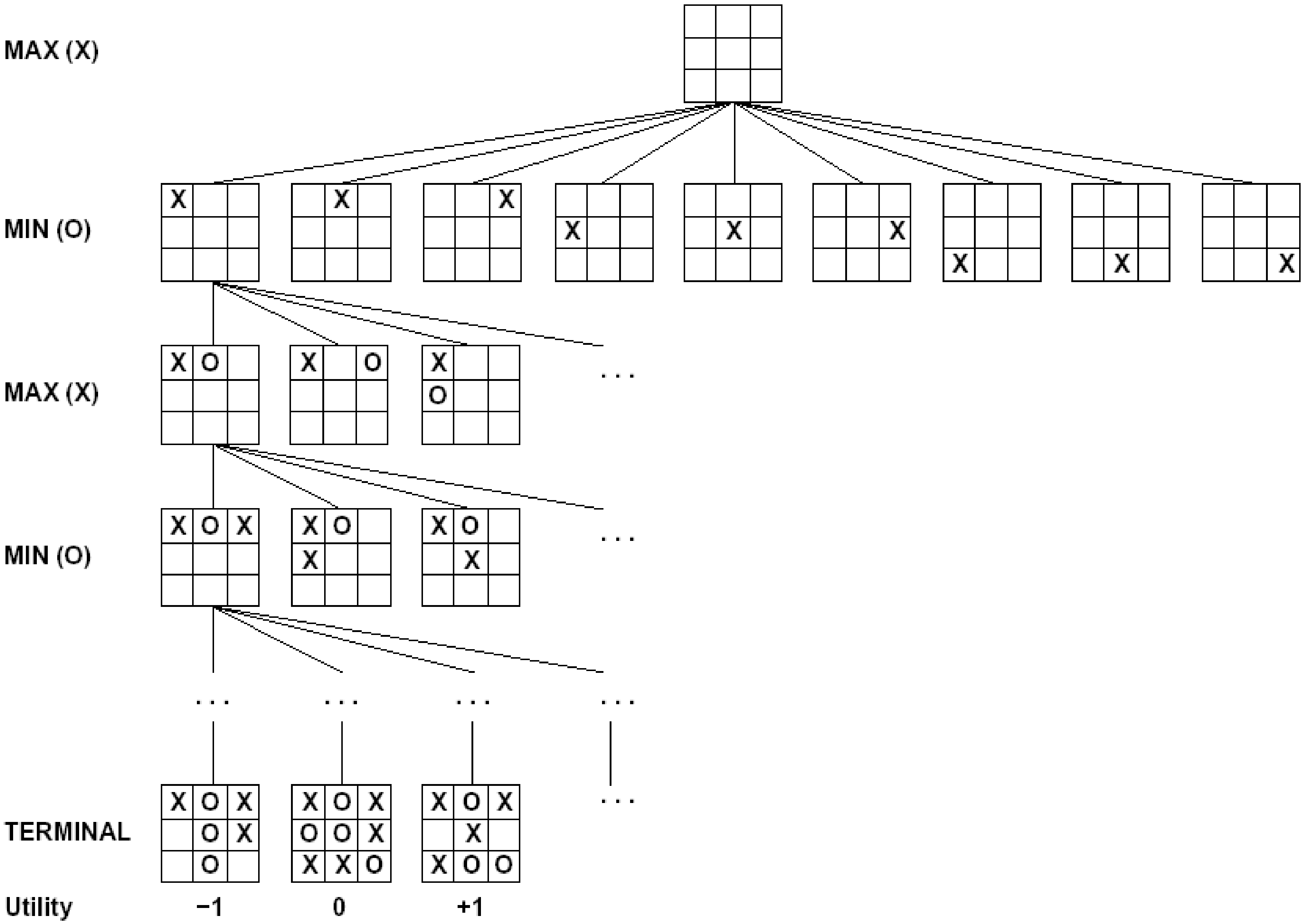
1

				x	3		
					3	x	
	x				3		
			x		2		
					x		x
x					2		
		x			2		
					0		

0

				x			
						x	
	x						
			x				
							x
x							
		x					
					x		

- Evaluation-Function
- Alpha-Beta-Pruning
- Monte-Carlo Tree Search



Evaluation Function

(Number of own chances to win) – (number of opposite chances)

	x	

8 – 4 = 4

x		

8 – 5 = 3

	x	

8 – 6 = 2

o		
	x	

5 – 4 = 1

	o	
	x	

6 – 4 = 2

x	o	
	x	

6 – 2 = 4

	o	
x	x	

6 – 3 = 3

Improvement (idea similar to Quiescense Search):

If: Immediate-Win then 100

elseif: Double-Double then 10

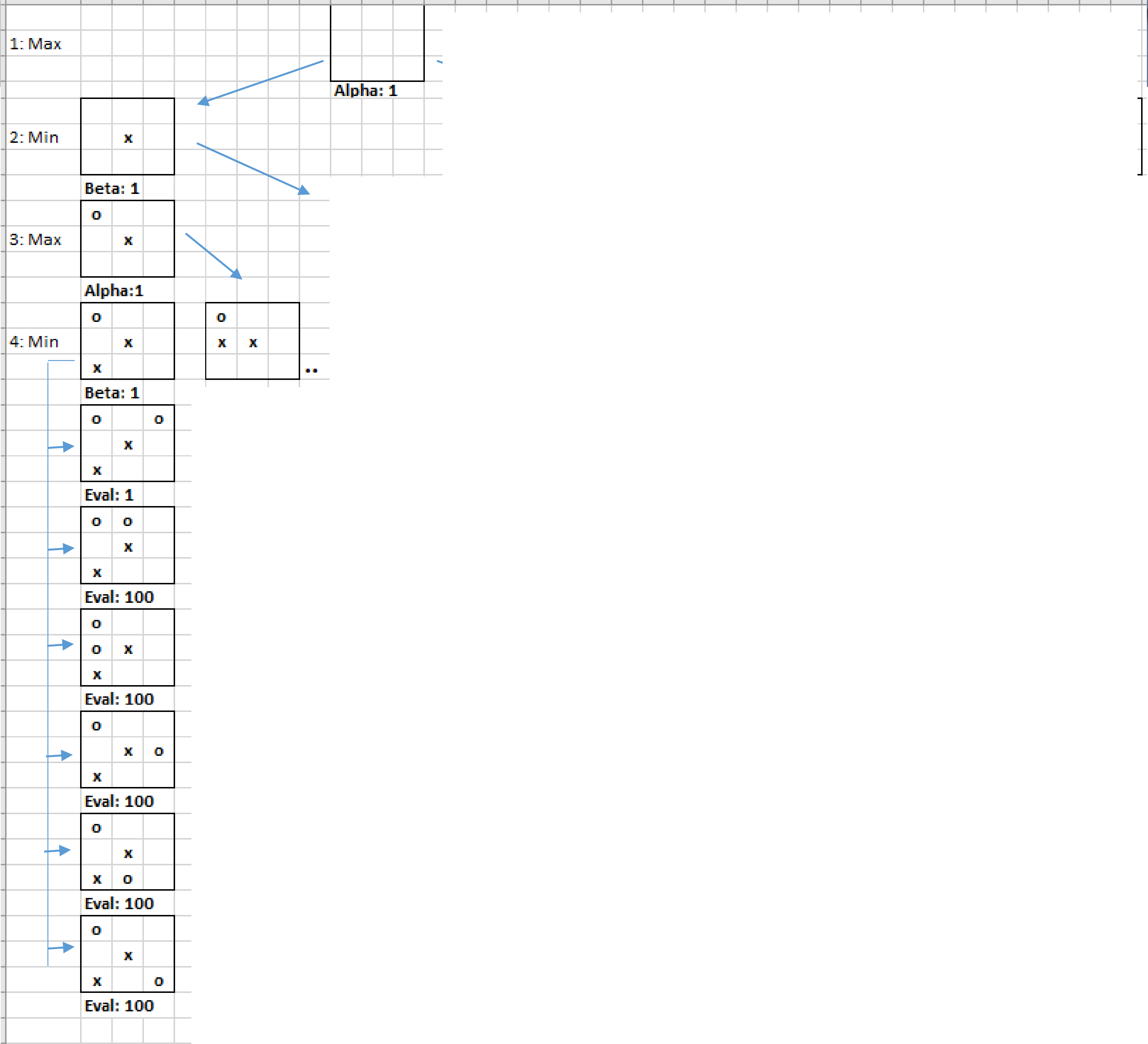
x	o	
x	x	
		o

else: (Number of own chances to win) – (number of opposite chances)

Example

- … for Alpha-Cut (light orange)
- … for Beta-Cut (full orange)

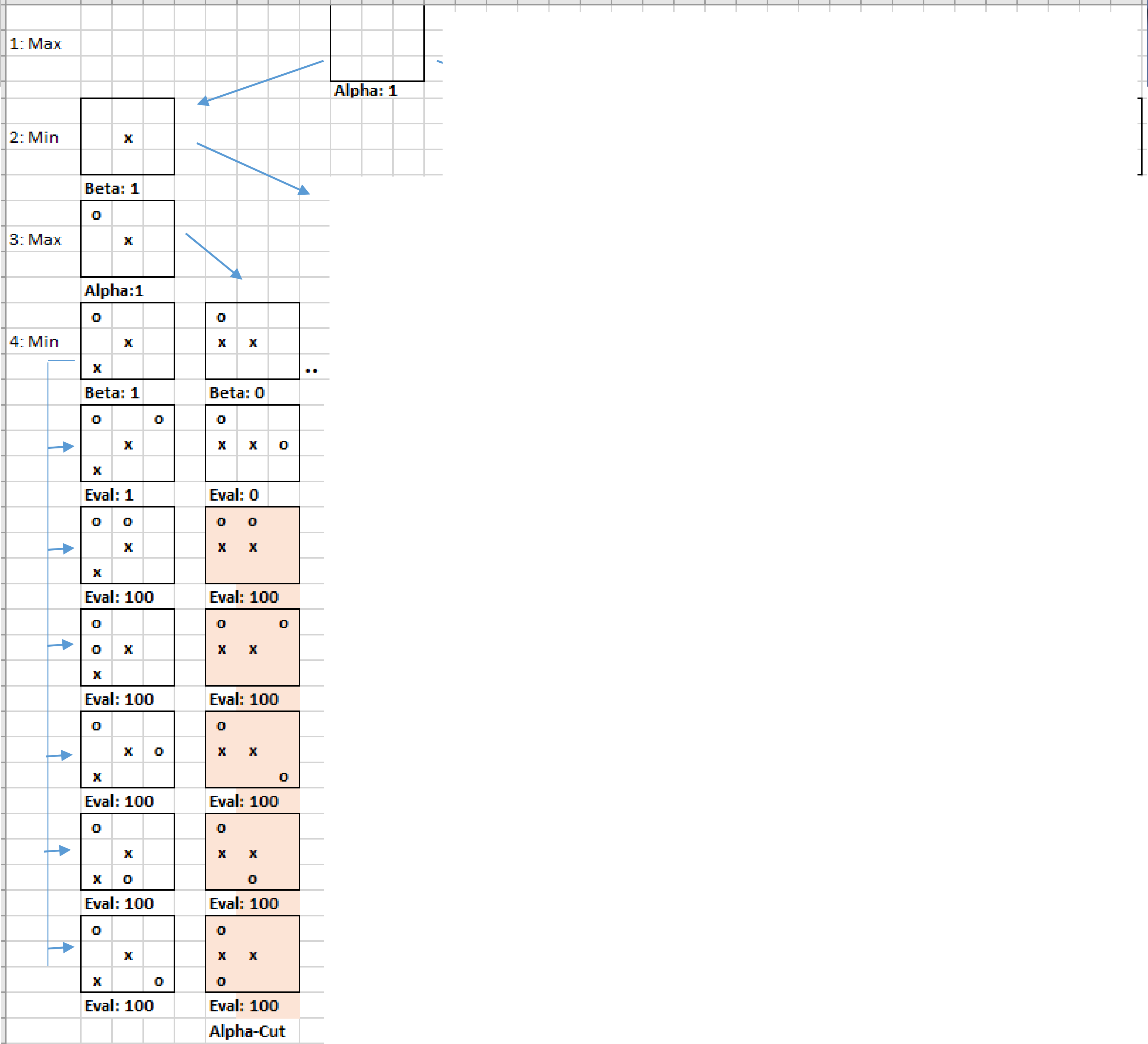
- With search depth of 4



Example

- … for Alpha-Cut (light orange)
- … for Beta-Cut (full orange)

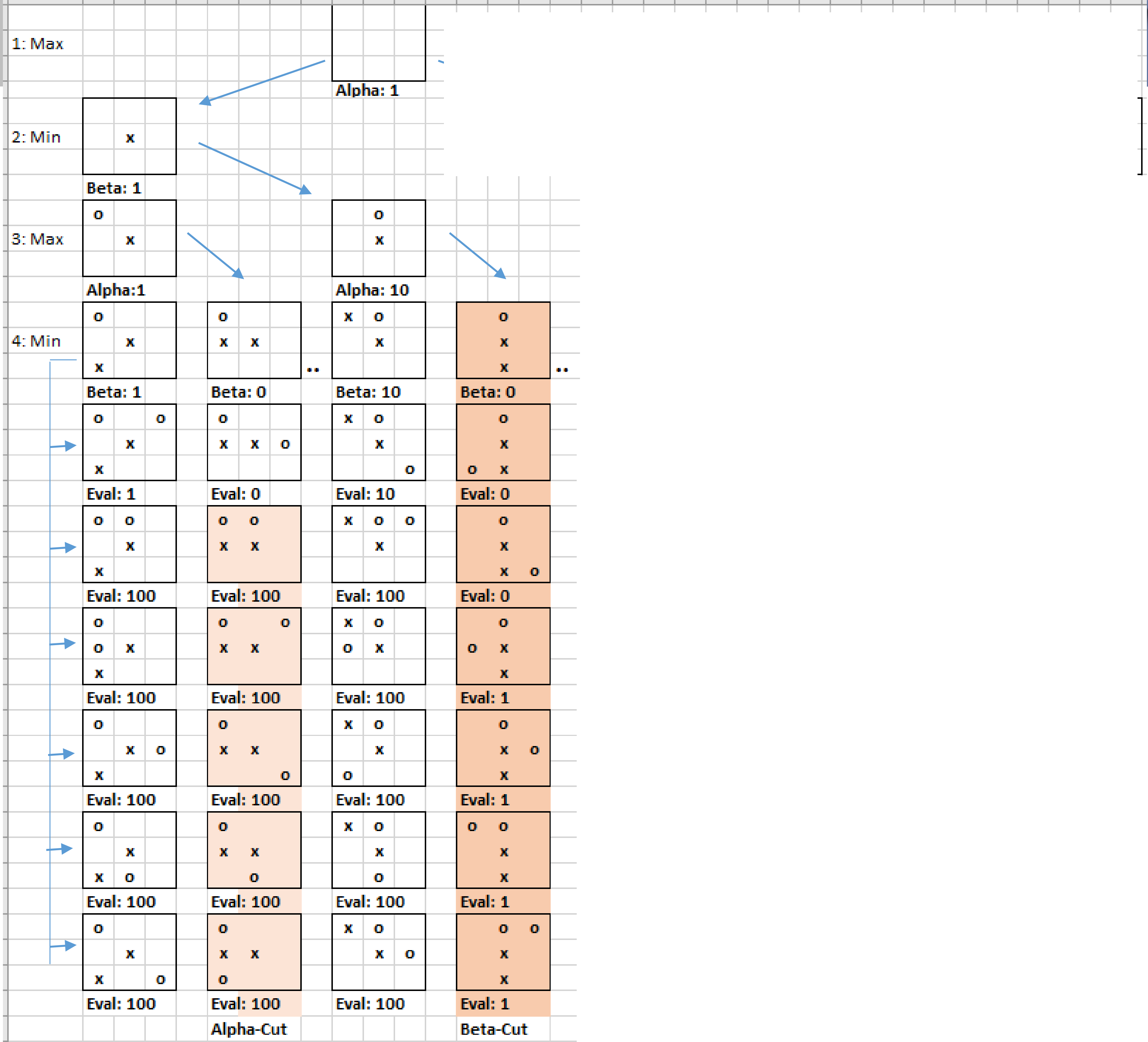
- With search depth of 4



Example

- … for Alpha-Cut (light orange)
- … for Beta-Cut (full orange)

- With search depth of 4

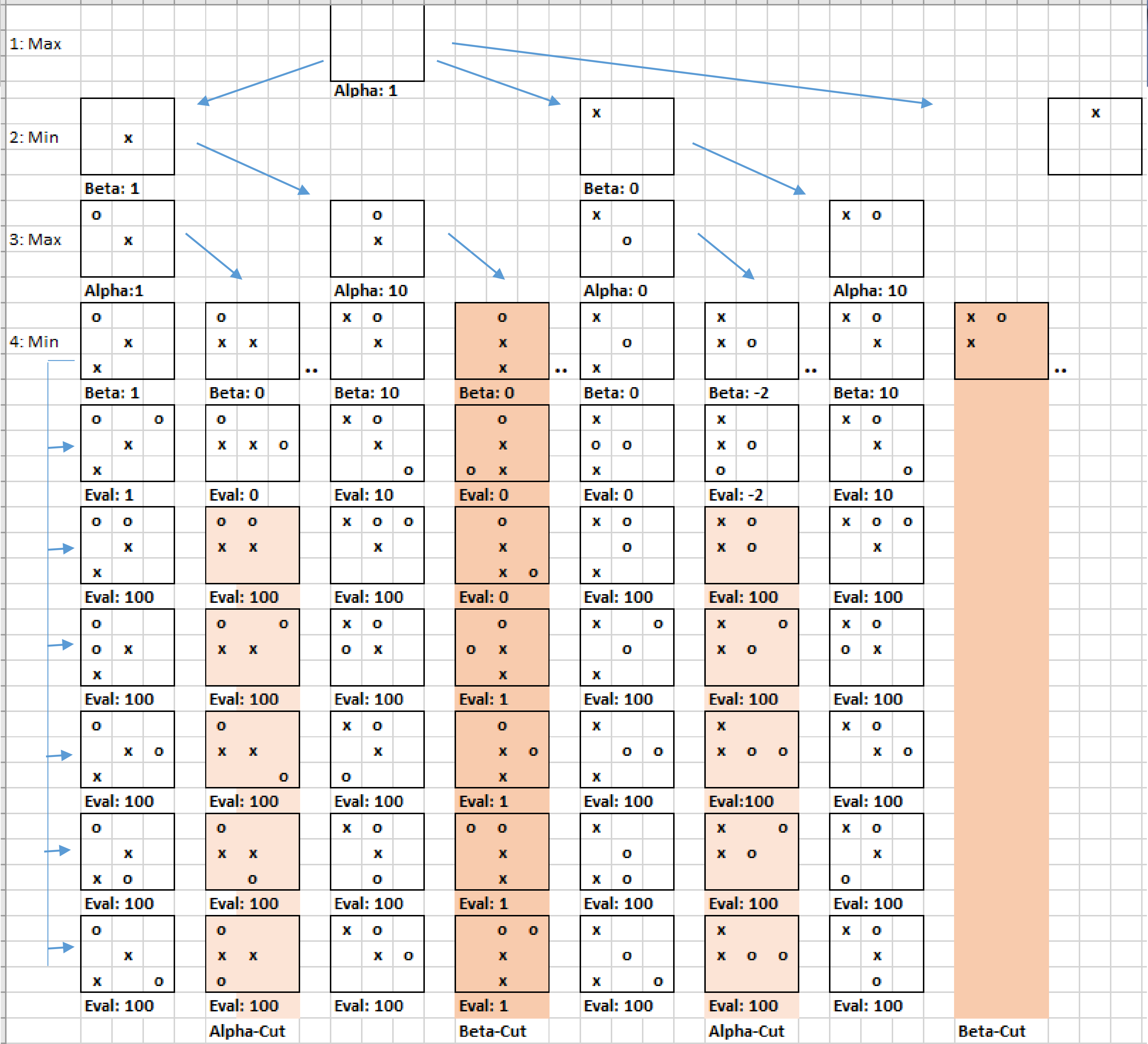


Julius-Maximilians-
UNIVERSITÄT WÜRZBURG

Example

- … for Alpha-Cut (light orange)
- … for Beta-Cut (full orange)

- With search depth of 4



Monte Carlo Tree Search

```
function MONTE-CARLO-TREE-SEARCH(state) returns an action
  tree ← NODE(state)
  while IS-TIME-REMAINING() do
    leaf ← SELECT(tree)
    child ← EXPAND(leaf)
    result ← SIMULATE(child)
    BACK-PROPAGATE(result, child)
  return the move in ACTIONS(state) whose node has highest number of playouts
```

Selection strategy:

$$UCB1(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log N(\text{PARENT}(n))}{N(n)}}$$

U(n) = Utility of node n
N(n) = Number of visits
C = Constant, e.g √2

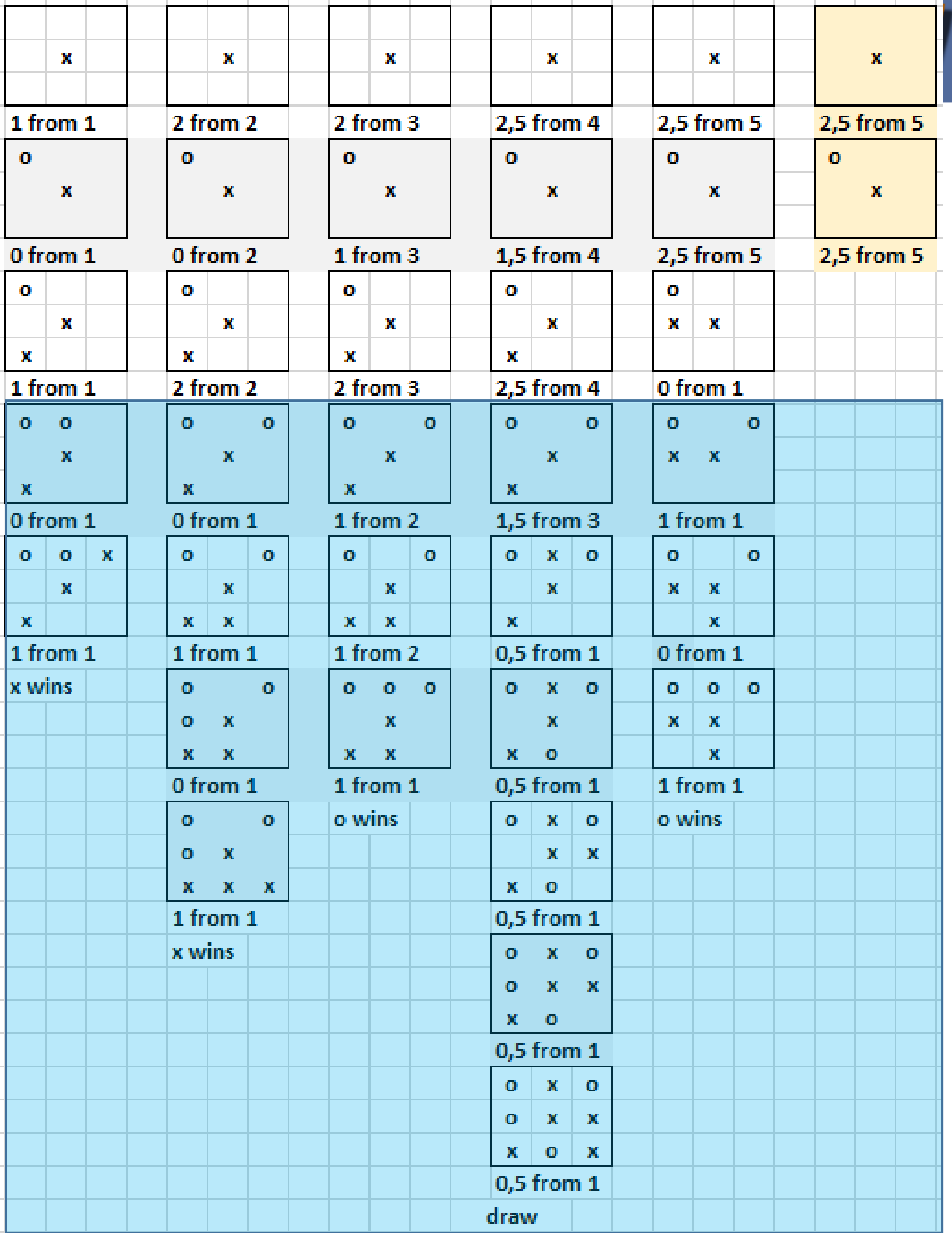
Monte Carlo Tree Search

function MONTE-CARLO-TREE-SEARCH(*state*) **returns** *an action*
  tree ← NODE(*state*)
  while IS-TIME-REMAINING() **do**
    leaf ← SELECT(*tree*)
    child ← EXPAND(*leaf*)
    result ← SIMULATE(*child*)
    BACK-PROPAGATE(*result*, *child*)
  return the move in ACTIONS(*state*) whose node has highest number of playou

Selection strategy:

$$UCB1(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log N(\mathrm{PARENT}(n))}{N(n)}}$$

U(n) = Utility of node n
N(n) = Number of visits
C = Constant, e.g √2



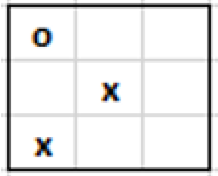
Monte Carlo Tree Search

function MONTE-CARLO-TREE-SEARCH(*state*) **returns** *an action*
  tree ← NODE(*state*)
  while IS-TIME-REMAINING() **do**
    leaf ← SELECT(*tree*)
    child ← EXPAND(*leaf*)
    result ← SIMULATE(*child*)
    BACK-PROPAGATE(*result*, *child*)
  return the move in ACTIONS(*state*) whose node has highest number of playouts

Selection strategy:

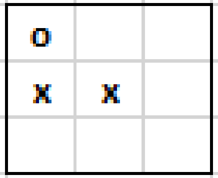
$$UCB1(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log N(\text{PARENT}(n))}{N(n)}}$$

U(n) = Utility of node n
N(n) = Number of visits
C = Constant, e.g √2
U(n) = # Wins



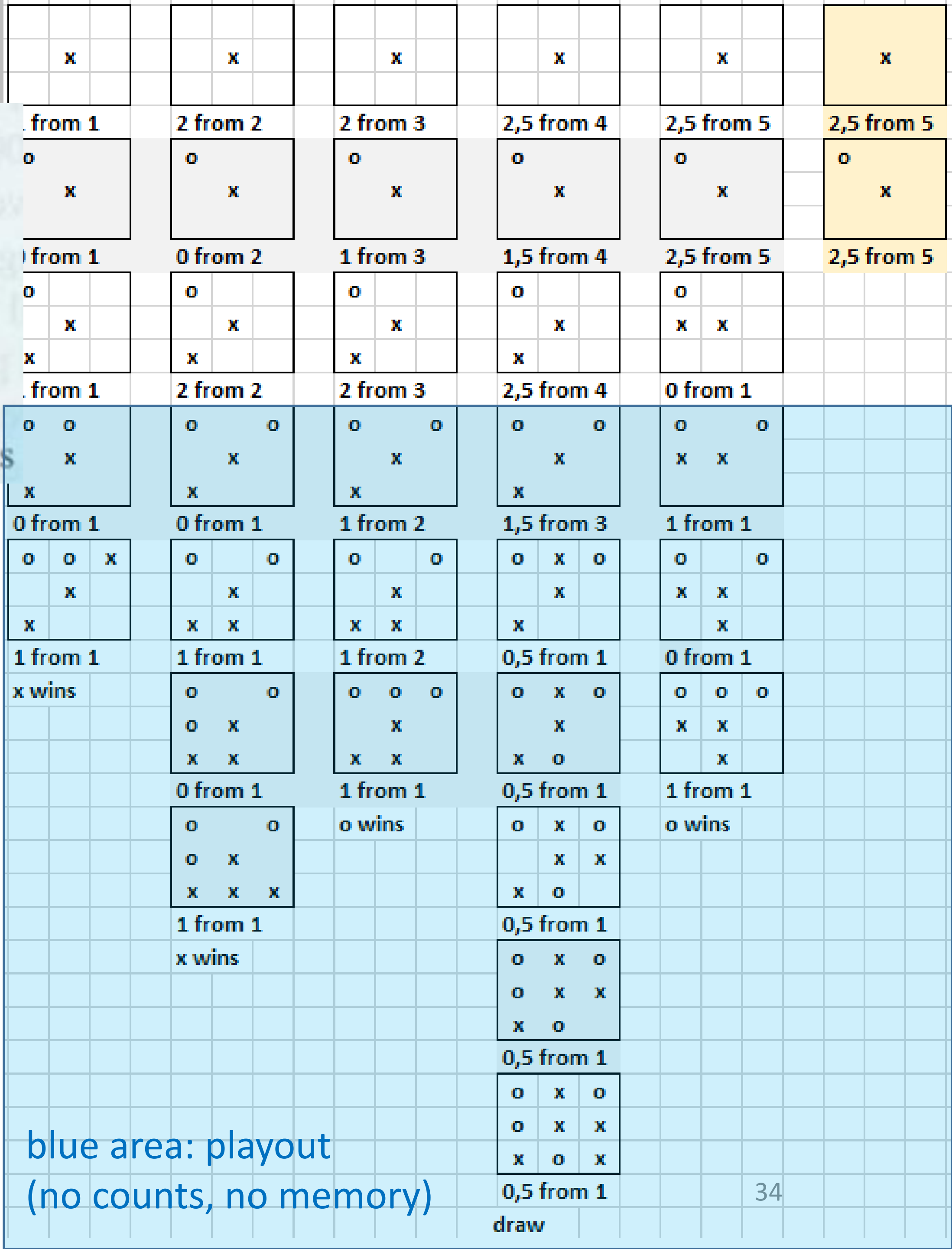
2,5 from 4

5/8+√2*√(log5/4) ≈
0,625 + 1,4*1,6/4 ≈
0,625 + 0,56 ≈ **1,2**



0 from 1

0 +√2*√(log5/1) ≈
0 + 1,4*1,6/1 ≈
0 + 2,24 ≈ **2,2**



Constraint-Propagation

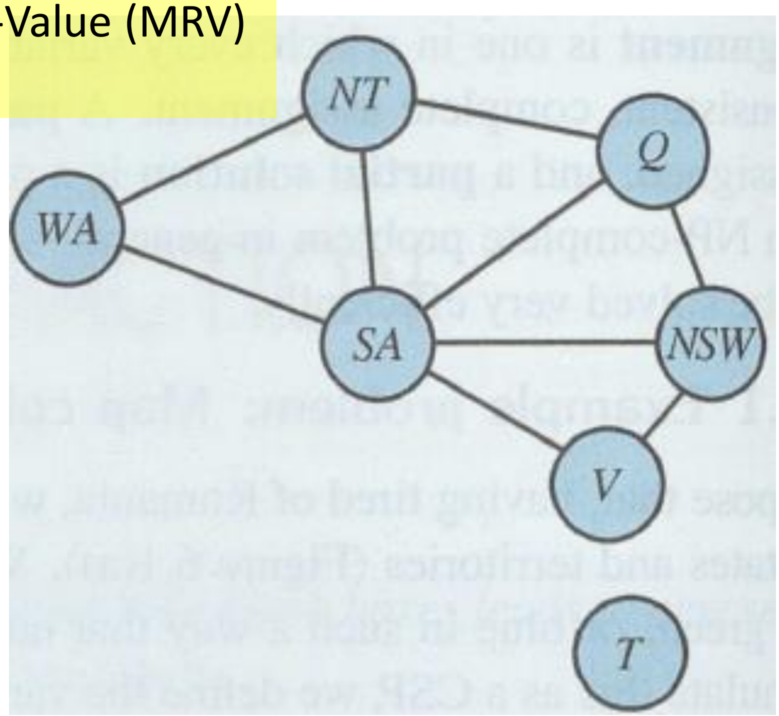

```
function BACKTRACKING-SEARCH(csp) returns a solution or failure
  return BACKTRACK(csp, { })

function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
     if value is consistent with assignment then
        add {var = value} to assignment
        inferences ← INFERENCE(csp, var, assignment)
        if inferences ≠ failure then
           add inferences to csp
           result ← BACKTRACK(csp, assignment)
           if result ≠ failure then return result
           remove inferences from csp
        remove {var = value} from assignment
  return failure
```

Minimimum-Remaining-Value (MRV)
Degree-Heuristic

Order-Domain-Values

Forward Checking
Maintaining Arc Consistency (MAC)

Order of Variables?



Constraint-Propagation

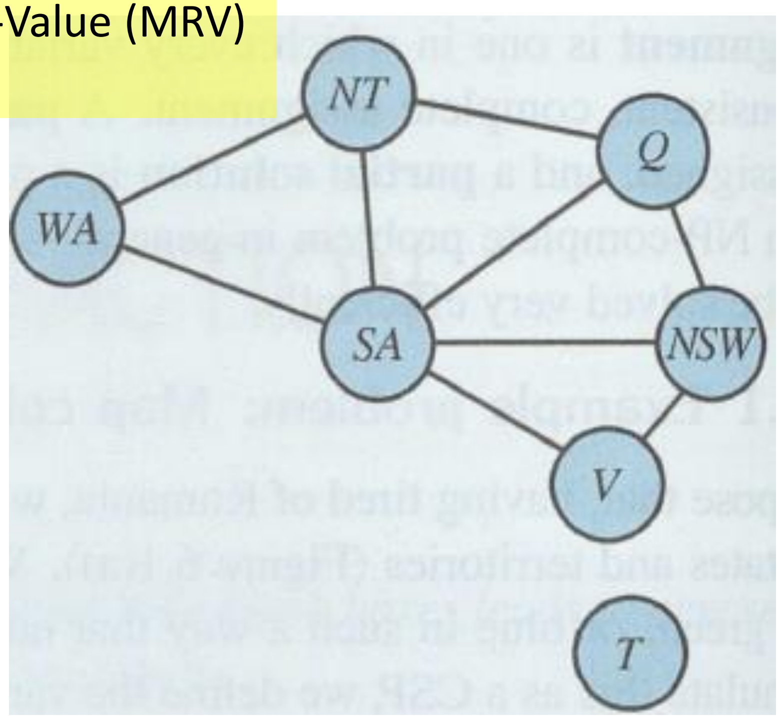

```
function BACKTRACKING-SEARCH(csp) returns a solution or failure
  return BACKTRACK(csp, { })

function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
     if value is consistent with assignment then
        add {var = value} to assignment
        inferences ← INFERENCE(csp, var, assignment)
        if inferences ≠ failure then
           add inferences to csp
           result ← BACKTRACK(csp, assignment)
           if result ≠ failure then return result
           remove inferences from csp
        remove {var = value} from assignment
  return failure
```

Minimimum-Remaining-Value (MRV)
Degree-Heuristic

Order-Domain-Values

Forward Checking
Maintaining Arc Consistency (MAC)

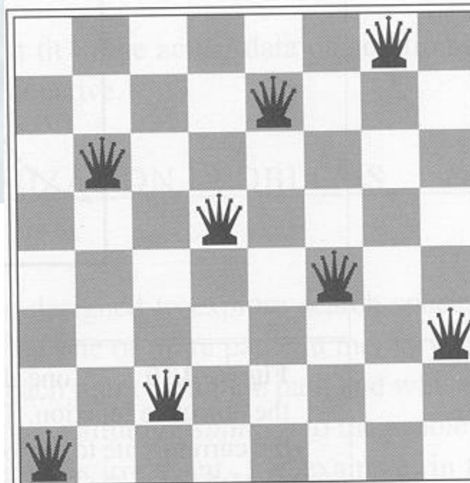
Order of Variables?



- Degree: SA
- MRV & Degree: {NT, Q, NSW} e.g. NT
- Forward Checking: WA, Q, NSW, V

Constraint-Propagation: Min-Conflict

```
function MIN-CONFLICTS(csp, max_steps) returns a solution or failure
  inputs: csp, a constraint satisfaction problem
          max_steps, the number of steps allowed before giving up

  current ← an initial complete assignment for csp
  for i = 1 to max_steps do
     if current is a solution for csp then return current
     var ← a randomly chosen conflicted variable from csp.VARIABLES
     value ← the value v for var that minimizes CONFLICTS(csp, var, v, current)
     set var = value in current
  return failure
```

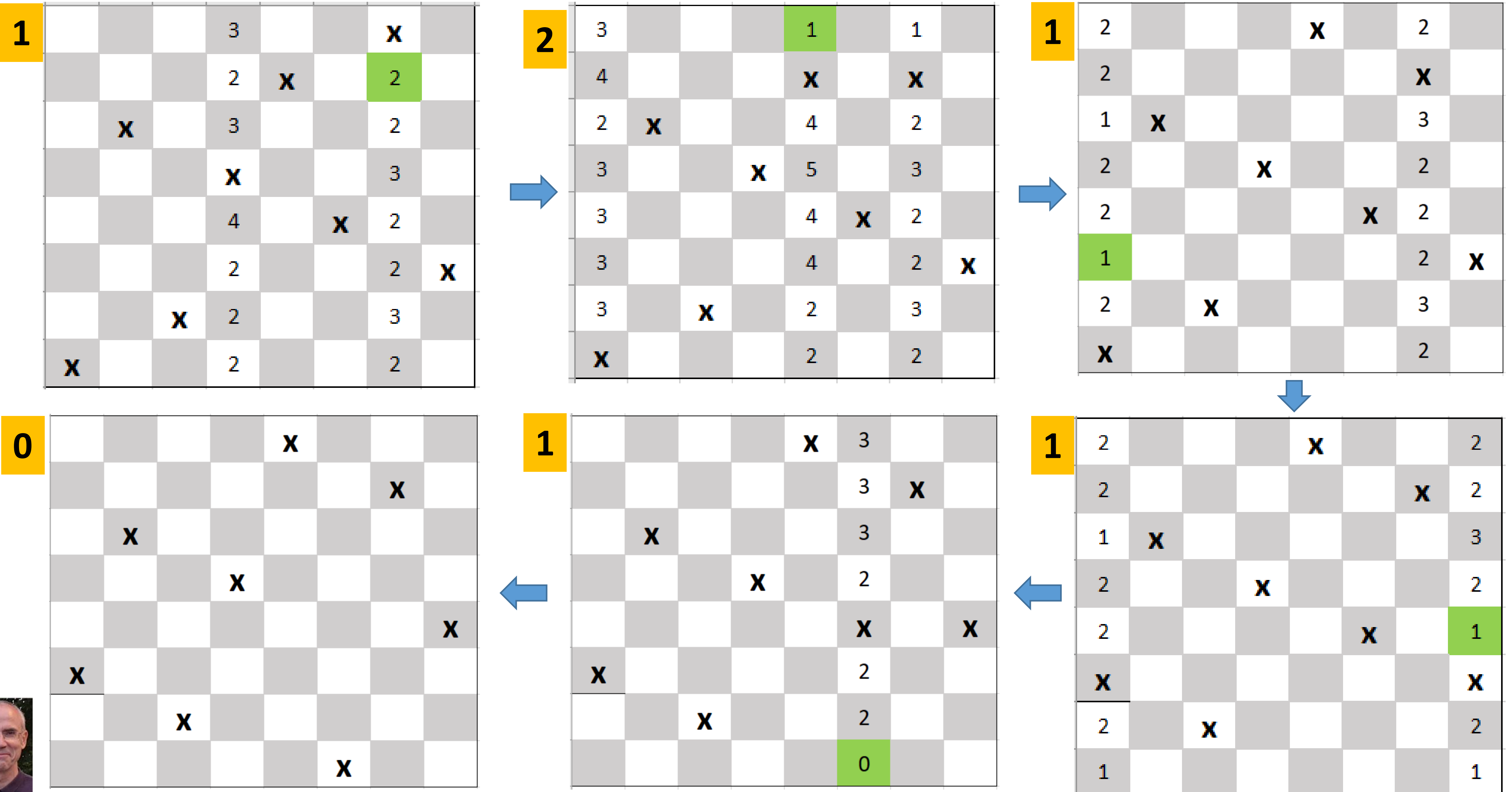


Min-Conflict: Example (one search path)

1

			3			x	
			2	x		2	
	x		3			2	
			x			3	
			4		x	2	
			2			2	x
		x	2			3	
x			2			2	

Min-Conflict: Example (one search path)



Evaluation function and Forward Checking

Improvement (Quiescense Search):

If: Immediate-Win then 100

elseif: Double-Double then 10

else: (Number of own chances to win) – (number of opposite chances)

x	o	
x	x	
		o

Constraint Analogy:

(Forward Checking)

(Sort of Arc or Path Consistency)

Chapter 7: Logical Agents

Forward Chaining Algorithm PL-FC-Entails?

- Not yet processed symbols are noted in a variable „queue“ and processed only once
- Rules have a counter for unsatisfied symbols in its premise, which is continuously decremented; if the counter = 0, the rules „fires“, i.e. its conclusion is added to queue

```
function PL-FC-ENTAILS?(KB, q) returns true or false
  inputs: KB, the knowledge base, a set of propositional definite clauses
          q, the query, a proposition symbol
  count ← a table, where count[c] is initially the number of symbols in clause c's premise
  inferred ← a table, where inferred[s] is initially false for all symbols
  queue ← a queue of symbols, initially symbols known to be true in KB

  while queue is not empty do
    p ← POP(queue)
    if p = q then return true
    if inferred[p] = false then
      inferred[p] ← true
      for each clause c in KB where p is in c.PREMISE do
        decrement count[c]
        if count[c] = 0 then add c.CONCLUSION to queue
  return false
```

Forward Chaining Example

Rules	Premises							Conclusions					
	A	B	C	D	E	F	G	C	D	E	F	G	H
R1	X	X						X					
R2	X	X							X				
R3	X		X			X				X			
R4		X			X							X	
R5		X	X									X	
R6	X				X								X
R7	X		X										X
R8		X		X				X					
R9						X							X
R10							X						X

Goal: Proof of H; Given: A and B

Forward Chaining

Depth First

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
	2	2	3	2	2	2	2	2	1	1
A										
B										

Breadth First

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
	2	2	3	2	2	2	2	2	1	1

Forward Chaining Example

Rules	Premises							Conclusions					
	A	B	C	D	E	F	G	C	D	E	F	G	H
R1	X	X						X					
R2	X	X							X				
R3	X		X			X				X			
R4		X			X							X	
R5		X	X									X	
R6	X				X								X
R7	X		X										X
R8		X		X				X					
R9						X							X
R10							X						X

Goal: Proof of H; Given: A and B

Forward Chaining

Depth First

		R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
		2	2	3	2	2	2	2	2	1	1
A		1	1	2			1	1			
B		0	0		1	1			1		
C	R1			1		0		0			
G	R5										0
H	R10										

Breadth First

		R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
		2	2	3	2	2	2	2	2	1	1
A											
B											

UNIVERSITÄT WÜRZBURG

Forward Chaining Example

Rules	Premises							Conclusions					
	A	B	C	D	E	F	G	C	D	E	F	G	H
R1	X	X						X					
R2	X	X							X				
R3	X		X			X				X			
R4		X			X							X	
R5		X	X									X	
R6	X				X								X
R7	X		X										X
R8		X		X				X					
R9						X							X
R10							X						X

Goal: Proof of H; Given: A and B

Forward Chaining

Depth First

		R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
		2	2	3	2	2	2	2	2	1	1
A		1	1	2			1	1			
B		0	0		1	1			1		
C	R1			1		0		0			
G	R5										0
H	R10										

Breadth First

		R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
		2	2	3	2	2	2	2	2	1	1
A		1	1	2			1	1			
B		0	0		1	1			1		
C	R1			1		0		0			
D	R2								0		
G	R5										0
H	R7										

Backward Chaining as AND-OR-Search

```
function AND-OR-SEARCH(problem) returns a conditional plan, or failure
  return OR-SEARCH(problem, problem.INITIAL, [])

function OR-SEARCH(problem, state, path) returns a conditional plan, or failure
  if problem.IS-GOAL(state) then return the empty plan
  if IS-CYCLE(path) then return failure
  for each action in problem.ACTIONS(state) do
     plan ← AND-SEARCH(problem, RESULTS(state, action), [state] + path])
     if plan ≠ failure then return [action] + plan]
  return failure

function AND-SEARCH(problem, states, path) returns a conditional plan, or failure
  for each s_i in states do
     plan_i ← OR-SEARCH(problem, s_i, path)
     if plan_i = failure then return failure
  return true
```

Backward Chaining Example

Rules	Premises							Conclusions					
	A	B	C	D	E	F	G	C	D	E	F	G	H
R1	X	X						X					
R2	X	X							X				
R3	X		X			X				X			
R4		X			X							X	
R5		X	X									X	
R6	X				X								X
R7	X		X										X
R8		X		X				X					
R9						X							X
R10							X						X

Backward Chaining Example

Backward Chaining (AND-OR-Search)

H : **OR:** R6, R7, R9, R10

R6: **AND**: A, E

A: o.k.

Rules	Premises							Conclusions					
	A	B	C	D	E	F	G	C	D	E	F	G	H
R1	X	X						X					
R2	X	X							X				
R3	X		X			X				X			
R4		X			X							X	
R5		X	X									X	
R6	X				X								X
R7	X		X										X
R8		X		X				X					
R9						X							X
R10							X						X

Backward Chaining Example

Rules	Premises							Conclusions					
	A	B	C	D	E	F	G	C	D	E	F	G	H
R1	X	X						X					
R2	X	X							X				
R3	X		X			X				X			
R4		X			X							X	
R5		X	X									X	
R6	X				X								X
R7	X		X										X
R8		X		X				X					
R9						X							X
R10							X						X

Backward Chaining (AND-OR-Search)

- H : **OR:** R6, R7, R9, R10
  - R6: **AND**: A, E
    - A: o.k.
    - E: **OR**: R3
      - R3: **AND**: A,C,F
        - A: o.k.
        - C: **OR**: R1, R8
          - R1: A, B
            - A: o.k.
            - B: o.k.
        - C: o.k.
        - F: n.o.k.

Backward Chaining Example

Rules	Premises							Conclusions					
	A	B	C	D	E	F	G	C	D	E	F	G	H
R1	X	X						X					
R2	X	X							X				
R3	X		X			X				X			
R4		X			X							X	
R5		X	X									X	
R6	X				X								X
R7	X		X										X
R8		X		X				X					
R9						X							X
R10							X						X

Backward Chaining (AND-OR-Search)

- H : **OR:** R6, R7, R9, R10
  - R6: **AND**: A, E
    - A: o.k.
    - E: **OR**: R3
      - R3: **AND**: A,C,F
        - A: o.k.
        - C: **OR**: R1, R8
          - R1: A, B
            - A: o.k.
            - B: o.k.
        - C: o.k.
        - F: n.o.k.
  - R7: **AND**: A, C
    - A: o.k.
    - C: **OR**: R1, R8
      - R1: A, B
        - A: o.k.
        - B: o.k.
    - C: o.k.
- H o.k.

Model Checking: DPLL Example

$(\neg B \lor \neg C \lor \neg D) \land (\neg A \lor B \lor \neg C) \land (\neg B \lor C \lor E) \land (B \lor C \lor \neg E) \land$
$(B \lor \neg C \lor E) \land (A \lor C \lor \neg E) \land (A \lor B \lor C) \land (\neg B \lor C \lor \neg E)$

DPLL:

- **Pure Symbols: $\neg D$**
- Set D= False
- $(\neg A \lor B \lor \neg C) \land (\neg B \lor \neg C \lor E) \land (B \lor \neg C \lor E) \land (B \lor C \lor \neg E) \land (A \lor C \lor \neg E) \land (A \lor B \lor C) \land (\neg B \lor C \lor \neg E)$
- **A = True OR A = False;**
  - Set A = True
  - $(B \lor \neg C) \land (\neg B \lor \neg C \lor E) \land (B \lor C \lor E) \land (B \lor C \lor \neg E) \land (\neg B \lor C \lor \neg E)$
  - **B = True OR B = False;**
    - Set B = False
      - $(\neg C) \land (C \lor E) \land (C \lor \neg E)$
      - **Unit Clause ($\neg$C)**
      - C = False
      - $(E) \land (\neg E)$: Unit Clause E = True -> **Return False**
    - Set B = True
      - $(\neg C \lor E) \land (C \lor \neg E)$
      - **C = True OR C = False;**
        - Set C = True
          - (E)
          - **Unit Clause = E** = True -> Return True
- Return True (D= False, A=True, B=True, C=True, E=True)

```
function DPLL-SATISFIABLE?(s) returns true or false
  inputs: s, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of s
  symbols ← a list of the proposition symbols in s
  return DPLL(clauses, symbols, { })

function DPLL(clauses, symbols, model) returns true or false

  if every clause in clauses is true in model then return true
  if some clause in clauses is false in model then return false
  P, value ← FIND-PURE-SYMBOL(symbols, clauses, model)
  if P is non-null then return DPLL(clauses, symbols – P, model ∪ {P=value})
  P, value ← FIND-UNIT-CLAUSE(clauses, model)
  if P is non-null then return DPLL(clauses, symbols – P, model ∪ {P=value})
  P ← FIRST(symbols); rest ← REST(symbols)
  return DPLL(clauses, rest, model ∪ {P=true}) or
         DPLL(clauses, rest, model ∪ {P=false}))
```

Model Checking: Walksat Example

$(\neg B \vee \neg C \vee \neg D) \wedge (\neg A \vee B \vee \neg C) \wedge (\neg B \vee C \vee E) \wedge (B \vee C \vee \neg E) \wedge (B \vee \neg C \vee E) \wedge (A \vee C \vee \neg E) \wedge (A \vee B \vee C) \wedge (\neg B \vee C \vee \neg E)$

Random Assignment: (A=True, B=True, C=True, D=True, E=True)

False $\wedge$ True $\wedge$ True $\wedge$ True $\wedge$ True $\wedge$ True $\wedge$ True $\wedge$ True

(i=1) Flip B = False (randomly selected)

True $\wedge$ **False** $\wedge$ True $\wedge$ True $\wedge$ True $\wedge$ True $\wedge$ True $\wedge$ True

(i=2) Flip A= False (maximizes satisfied clauses; better than C = False or B = True)

True $\wedge$ True $\wedge$ True $\wedge$ True $\wedge$ True $\wedge$ True $\wedge$ True $\wedge$ True

function WALKSAT(*clauses*, *p*, *max_flips*) **returns** a satisfying model or *failure*
  inputs: *clauses*, a set of clauses in propositional logic
    p, the probability of choosing to do a "random walk" move, typically around 0.5
    max_flips, number of value flips allowed before giving up

  model ← a random assignment of *true*/*false* to the symbols in *clauses*
  for each *i* = 1 **to** *max_flips* **do**
    if *model* satisfies *clauses* **then return** *model*
    clause ← a randomly selected clause from *clauses* that is false in *model*
    if RANDOM(0, 1) ≤ *p* **then**
      flip the value in *model* of a randomly selected symbol from *clause*
    else flip whichever symbol in *clause* maximizes the number of satisfied clauses
  return *failure*

Kinship Domain: Facts

parent (Christopher, Arthur)
parent (Christopher, Victoria)
parent (Andrew, James)
parent (Andrew, Jennifer)
parent (James, Colin)
parent (James, Charlotte)
parent (Penelope, Arthur)
parent (Penelope, Victoria)
parent (Christine, James)
parent (Christine, Jennifer)
parent (Victoria, Colin)
parent (Victoria, Charlotte)

spouse (Christopher, Penelope)
spouse (Andrew, Christine)
spouse (Arthur, Margaret)
spouse (James, Victoria)
spouse (Charles, Jennifer)

male (Christopher)
male (Andrew)
male (Arthur)
male (James)
male (Charles)
male (Colin)

female (Penelope)
female (Christine)
female (Margaret)
female (Victoria)
female (Jennifer)
female (Charlotte)

Implicit relations:

- father (x,y)
- mother (x,y)
- husband (x,y)
- wife (x,y)
- son (x,y)
- daughter (x,y)
- sibling (x,y)
- uncle (x,y)
- aunt (x,y)
- nephew (x,y)
- niece (x,y)
- grandparent (x,y)
- grandchild (x,y)

Kinship Domain: Inferences

father (Christopher, Arthur)
father (Christopher, Victoria)
father (Andrew, James)
father (Andrew, Jennifer)
father (James, Colin)
father (James, Charlotte)

mother (Penelope, Arthur)
mother (Penelope, Victoria)
mother (Christine, James)
mother (Christine, Jennifer)
mother (Victoria, Colin)
mother (Victoria, Charlotte)

son (Arthur, Christopher)
son (James, Andrew)
son (Colin, James)
son (Arthur, Penelope)
son (James, Christine)
son (Colin, Victoria)

daugther (Victoria, Christopher)
daugther (Jennifer, Andrew)
daugther (Charlotte, James)
daugther (Victoria, Penelope)
daugther (Jennifer, Christine)
daugther (Charlotte, Victoria)

husband (Christopher, Penelope)
husband (Andrew, Christine)
husband (Arthur, Margaret)
husband (James, Victoria)
husband (Charles, Jennifer)

wife (Penelope, Christopher)
wife (Christine, Andrew)
wife (Margaret, Arthur)
wife (Victoria, James)
wife (Jennifer, Charles)

sibling (Arthur, Victoria)
sibling (James, Jennifer)
sibling (Colin, Charlotte)

uncle (Arthur, Colin)
uncle (Charles, Colin)
uncle (Arthur, Charlotte)
uncle (Charles, Charlotte)

aunt (Jennifer, Colin)
aunt (Margaret, Colin)
aunt (Jennifer, Charlotte)
aunt (Margaret, Charlotte)

nephew (Colin, Arthur)
nephew (Colin, Jennifer)
nephew (Colin, Margaret)
nephew (Colin, Charles)

niece (Charlotte, Arthur)
niece (Charlotte, Jennifer)
niece (Charlotte, Margaret)
niece (Charlotte, Charles)

grandparent (Christophe,r Colin)
grandparent (Christopher, Charlotte)
grandparent (Andrew, Colin)
grandparent (Andrew, Charlotte)
grandparent (Penelope, Colin)
grandparent (Penelope, Charlotte)
grandparent (Christine, Colin)
grandparent (Christine, Charlotte)
grandchild (Colin, Christopher)
grandchild (Charlotte, Christopher)
grandchild (Colin, Andrew)
grandchild (Charlotte, Andrew)
grandchild (Colin, Penelope)
grandchild (Charlotte, Penelope)
grandchild (Colin, Christine)
grandchild (Charlotte, Christine)

Task

- **Relations given:**
- parent (x,y)
- spouse (x,y)
- male (x)
- female (x)

Relations to be inferred:

- father (x,y)
- mother (x,y)
- husband (x,y)
- wife (x,y)
- son (x,y)
- daughter (x,y)
- sibling (x,y)
- uncle (x,y)
- aunt (x,y)
- nephew (x,y)
- niece (x,y)
- grandparent (x,y)
- grandchild (x,y)

Example inference:

$\forall$ x,y male (x) $\wedge$ parent (x, y) $\Rightarrow$ father (x, y)

Kinship Domain: Rules

$\forall$ x,y male (x) $\wedge$ parent (x, y) $\Rightarrow$ father (x, y)

$\forall$ x,y female (x) $\wedge$ parent (x, y) $\Rightarrow$ mother (x, y)

$\forall$ x,y male (y) $\wedge$ parent (x, y) $\Rightarrow$ son (y, x)

$\forall$ x,y female (y) $\wedge$ parent (x, y) $\Rightarrow$ daugther (y, x)

$\forall$ x,y male (x) $\wedge$ spouse (x, y) $\Rightarrow$ husband (x,y)

$\forall$ x,y female (x) $\wedge$ spouse (x, y) $\Rightarrow$ wife (y,x)

$\forall$ x,y,z parent (x, z) $\wedge$ parent (y,z) $\Rightarrow$ sibling (x,y)

$\forall$ x,y,z parent (x, z) $\wedge$ sibling (y,z) $\wedge$ male (x) $\Rightarrow$ uncle (x,y)

$\forall$ x,y,z parent (x, z) $\wedge$ sibling (y,z) $\wedge$ female (x) $\Rightarrow$ aunt (x,y)

$\forall$ x,y,z parent (x, z) $\wedge$ sibling (y,z) $\wedge$ male (y) $\Rightarrow$ nephew (y,x)

$\forall$ x,y,z parent (x, z) $\wedge$ sibling (y,z) $\wedge$ female (y) $\Rightarrow$ niece (y,x)

$\forall$ x,y,z parent (x, y) $\wedge$ parent (y,z) $\Rightarrow$ grandparent (x,z)

$\forall$ x,y,z parent (x, y) $\wedge$ parent (y,z) $\Rightarrow$ grandchild (z,x)

Example for CNF- Conversion

Right: Textual representation
Below: FOL representation
Below right: CNF representation

Everyone who loves all animals is loved by someone.
Anyone who kills an animal is loved by no one.
Jack loves all animals.
Either Jack or Curiosity killed the cat, who is named Tuna.
Did Curiosity kill the cat?

A. $\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$
B. $\forall x\ [\exists z\ Animal(z) \land Kills(x,z)] \Rightarrow [\forall y\ \neg Loves(y,x)]$
C. $\forall x\ Animal(x) \Rightarrow Loves(Jack,x)$
D. $Kills(Jack,Tuna) \lor Kills(Curiosity,Tuna)$
E. $Cat(Tuna)$
F. $\forall x\ Cat(x) \Rightarrow Animal(x)$
¬G. $\neg Kills(Curiosity,Tuna)$

A1. $Animal(F(x)) \lor Loves(G(x),x)$
A2. $\neg Loves(x,F(x)) \lor Loves(G(x),x)$
B. $\neg Loves(y,x) \lor \neg Animal(z) \lor \neg Kills(x,z)$
C. $\neg Animal(x) \lor Loves(Jack,x)$
D. $Kills(Jack,Tuna) \lor Kills(Curiosity,Tuna)$
E. $Cat(Tuna)$
F. $\neg Cat(x) \lor Animal(x)$
¬G. $\neg Kills(Curiosity,Tuna)$

Example for CNF-Conversion

A. $\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$
B. $\forall x\ [\exists z\ Animal(z) \wedge Kills(x,z)] \Rightarrow [\forall y\ \neg Loves(y,x)]$
C. $\forall x\ Animal(x) \Rightarrow Loves(Jack,x)$
D. $Kills(Jack,Tuna) \vee Kills(Curiosity,Tuna)$
E. $Cat(Tuna)$
F. $\forall x\ Cat(x) \Rightarrow Animal(x)$
¬G. $\neg Kills(Curiosity,Tuna)$

A1 and A2: s. next slide
B. ¬Animal (z) ∨ ¬Kills (x,z) ∨ ¬Loves (y,x)
C. ¬Animal (x) ∨ Loves (Jack,x)]
D. Kills (Jack, Tuna) ∨ Kills (Curiosity, Tuna)
E. Cat (Tuna)
F. ¬Cat(x) ∨ Animal (x)
¬G. ¬Kills (Curiosity, Tuna)

∀ x [∃ z Animal (z) ∧ Kills (x,z)] ⇒ [∀ y ¬Loves (y,x)] — *Eliminate implication*
∀ x [¬ [∃ z Animal (z) ∧ Kills (x,z)] ∨ [∀ y ¬Loves (y,x)] — *Move ¬ inwards*
∀ x [¬ ∃ z Animal (z) ∨ ¬Kills (x,z)] ∨ [∀ y ¬Loves (y,x)] — *Move ¬ inwards*
∀ x [∀ z ¬Animal (z) ∨ ¬Kills (x,z)] ∨ [∀ y ¬Loves (y,x)] — *Drop universal quantifiers*
¬Animal (z) ∨ ¬Kills (x,z) ∨ ¬Loves (y,x)

Example 2 for Conversion in CNF

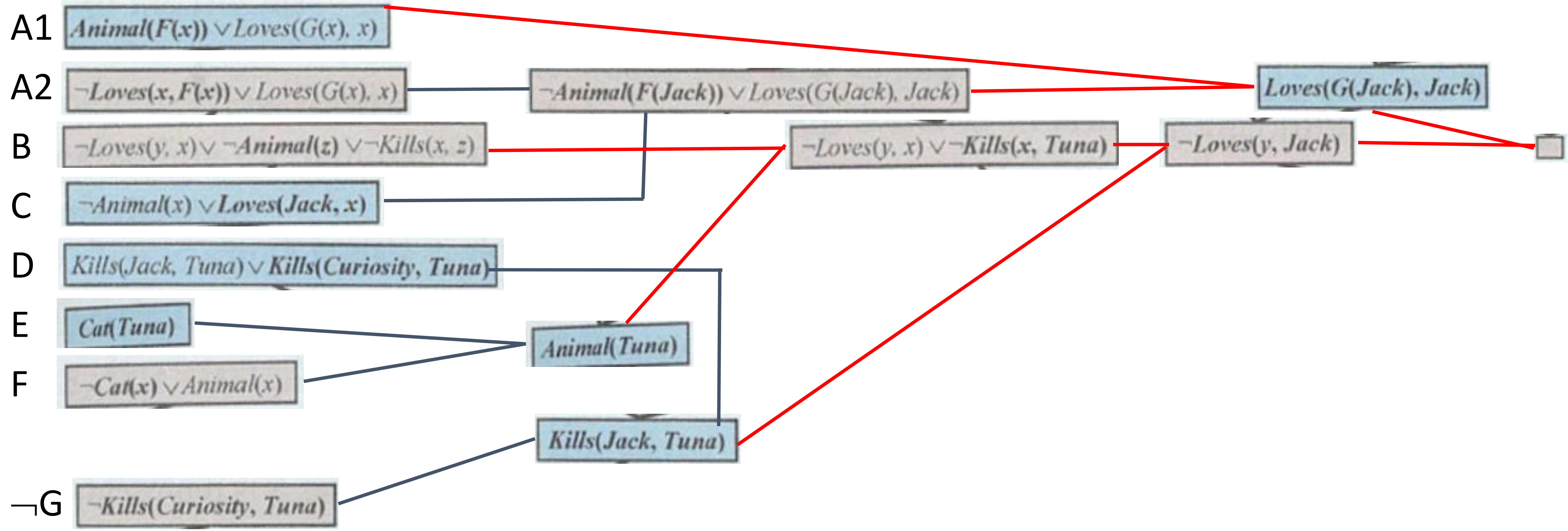
$\forall$ x [$\forall$ y Animal y) $\Rightarrow$ Loves (x,y)] $\Rightarrow$ [$\exists$ y Loves (y,x)]	*Eliminate implication*
$\forall$ x $\neg$[$\forall$ y Animal y) $\Rightarrow$ Loves (x,y)] $\vee$ [$\exists$ y Loves (y,x)]	*Eliminate implication*
$\forall$ x $\neg$[$\forall$ y $\neg$ Animal y) $\vee$ Loves (x,y)] $\vee$ [$\exists$ y Loves (y,x)]	*Move $\neg$ inwards*
$\forall$ x [$\exists$ y $\neg$($\neg$ Animal y) $\vee$ Loves (x,y))] $\vee$ [$\exists$ y Loves (y,x)]	*Move $\neg$ inwards*
$\forall$ x [$\exists$ y $\neg\neg$ Animal y) $\wedge$ $\neg$Loves (x,y))] $\vee$ [$\exists$ y Loves (y,x)]	*Move $\neg$ inwards*
$\forall$ x [$\exists$ y Animal (y) $\wedge$ $\neg$Loves (x,y))] $\vee$ [$\exists$ y Loves (y,x)]	*Standardize variables*
$\forall$ x [$\exists$ y Animal(y) $\wedge$ $\neg$Loves (x,y))] $\vee$ [$\exists$ **z** Loves (**z**,x)]	*Skolemize (2 x)*
$\forall$ x [Animal (**F(x)**) $\wedge$ $\neg$Loves (x,**F(x)**))] $\vee$ [Loves (**G(x)**,x)]	*Drop universal quantifier*
[Animal (F(x)) $\wedge$ $\neg$Loves (x,F(x)))] $\vee$ [Loves (G(x),x)]	*Distribute $\vee$ oder $\wedge$*

[Animal (F(x)) $\vee$ Loves (G(x),x)] $\wedge$ [$\neg$Loves (x,F(x))] $\vee$ Loves (G(x),x)]

A1 [Animal (F(x)) $\vee$ Loves (G(x),x)]

A2 [$\neg$Loves (x,F(x))] $\vee$ Loves (G(x),x)]

Resolution


A1 Animal(F(x)) ∨ Loves(G(x), x)
A2 ¬Loves(x, F(x)) ∨ Loves(G(x), x)
B ¬Loves(y, x) ∨ ¬Animal(z) ∨ ¬Kills(x, z)
C ¬Animal(x) ∨ Loves(Jack, x)
D Kills(Jack, Tuna) ∨ Kills(Curiosity, Tuna)
E Cat(Tuna)
F ¬Cat(x) ∨ Animal(x)
¬G ¬Kills(Curiosity, Tuna)
¬Animal(F(Jack)) ∨ Loves(G(Jack), Jack)
Loves(G(Jack), Jack)
¬Loves(y, x) ∨ ¬Kills(x, Tuna)
¬Loves(y, Jack)
Animal(Tuna)
Kills(Jack, Tuna)

Comparison of $\wedge$ and $\Rightarrow$

A	B	$A \Rightarrow B$	$A \wedge B$
T	T	T	T
T	F	F	F
F	T	T	F
F	F	T	F

General difference:
$A \wedge B$ two facts (A and B) but
$A \Rightarrow B$ only one fact.

To be discussed:
Implication within Implication

Everyone who loves all animals is loved by someone.

If valid: „Wenn y ein Tier ist, wird es von x geliebt", then there is somebody (z), who loves x.

$\forall$ x [$\forall$ y Animal (y) $\Rightarrow$ Loves (x,y)] $\Rightarrow$ [$\exists$ z Loves (z,x)] versus

$\forall$ x [$\forall$ y Animal (y) $\wedge$ Loves (x,y)] $\Rightarrow$ [$\exists$ z Loves (z,x)]

If all things (all y) are animals and x loves y, then there is somebody (z), who loves x.

Example 2 for Conversion in CNF with $\wedge$ instead of $\Rightarrow$

$\forall x\ [\forall y\ Animal\ y) \wedge Loves\ (x,y)] \Rightarrow [\exists y\ Loves\ (y,x)]$	*Eliminate implication*
$\forall x\ \neg[\forall y\ Animal\ y) \wedge Loves\ (x,y)] \vee [\exists y\ Loves\ (y,x)]$	*Move $\neg$ inwards*
$\forall x\ [\exists y\ \neg Animal\ (y) \vee \neg Loves\ (x,y)] \vee [\exists y\ Loves\ (y,x)]$	*Standardize variables*
$\forall x\ [\exists y\ \neg Animal\ (y) \vee \neg Loves\ (x,y)] \vee [\exists \mathbf{z}\ Loves\ (\mathbf{z},x)]$	*Skolemize (2 x)*
$\forall x\ [\neg Animal\ (\mathbf{F(x)}) \vee \neg Loves\ (x,\mathbf{F(x)})] \vee [Loves\ (\mathbf{G(x)},x)]$	*Drop universal quantifier*
$[\neg Animal\ (F(x)) \vee \neg Loves\ (x,F(x))] \vee [Loves\ (G(x),x)]$	

Adapted Resolution (with ∧ instead of ⇒)

A [¬Animal (F(y)) ∨ ¬Loves (y,F(y))] ∨ [Loves (G(y),y)];

[¬Animal (F(Jack)) ∨ [Loves (G(Jack),Jack)]

B ¬Loves(y, x) ∨ ¬Animal(z) ∨ ¬Kills(x, z)

[¬Animal(F(Jack)) ∨ ¬Kills (Jack, z)]

¬Animal(F(Jack))

??

C ¬Animal(x) ∨ Loves(Jack, x)

D Kills(Jack, Tuna) ∨ Kills(Curiosity, Tuna)

E Cat(Tuna)

Animal(Tuna)

F ¬Cat(x) ∨ Animal(x)

Kills(Jack, Tuna)

¬G ¬Kills(Curiosity, Tuna)

Resolutions:

[¬Animal (F(y)) ∨ ¬Loves (y,F(y))] ∨ [Loves (G(y),y)]; ¬Animal(x) ∨ Loves(Jack, x) = Jack; x=F(Jack)

[¬Animal (F(Jack)) ∨ ¬Animal (F(Jack)) ∨ [Loves (G(Jack),Jack)]

[¬Animal (F(Jack)) ∨ [Loves (G(Jack),Jack)] ; ¬Loves(y, x) ∨ ¬Animal(z) ∨ ¬Kills(x, z) = G(Jack), x = Jack

[¬Animal (F(Jack)) ∨ ¬Animal (z) ∨ ¬Kills (Jack, z)]

[¬Animal(F(Jack)) ∨ ¬Kills (Jack, z)]

Chapter 10: Knowledge Representation

Ontologies

“An ontology is an explicit, formal specification of a shared conceptualization” (Gruber 1993)

- Conceptualization: abstract model (terms and relations)
- Explizit: Semantic of all terms is defined
- Formal: Computer interpretable
- Shared: Consense for Ontology in a community

Types of Ontologies

- **Catalogue, glossar, taxonomy:** simple controlled vocabularies
- **Classification, thesaurus, (taxonomy):** includes hierarchical relations, synonyms
- **Semantic net, (ontology), knowledge graph:** includes other types of relations in addition to hierarchical relations

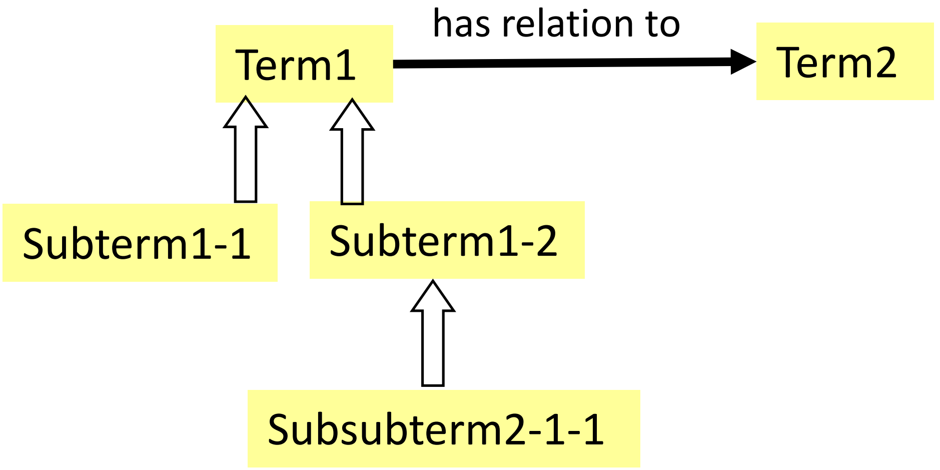
Catalogue, glossar:

1. Term1
2. Term2
3. Term3
4. Term4
5. …

Classification, thesaurus:

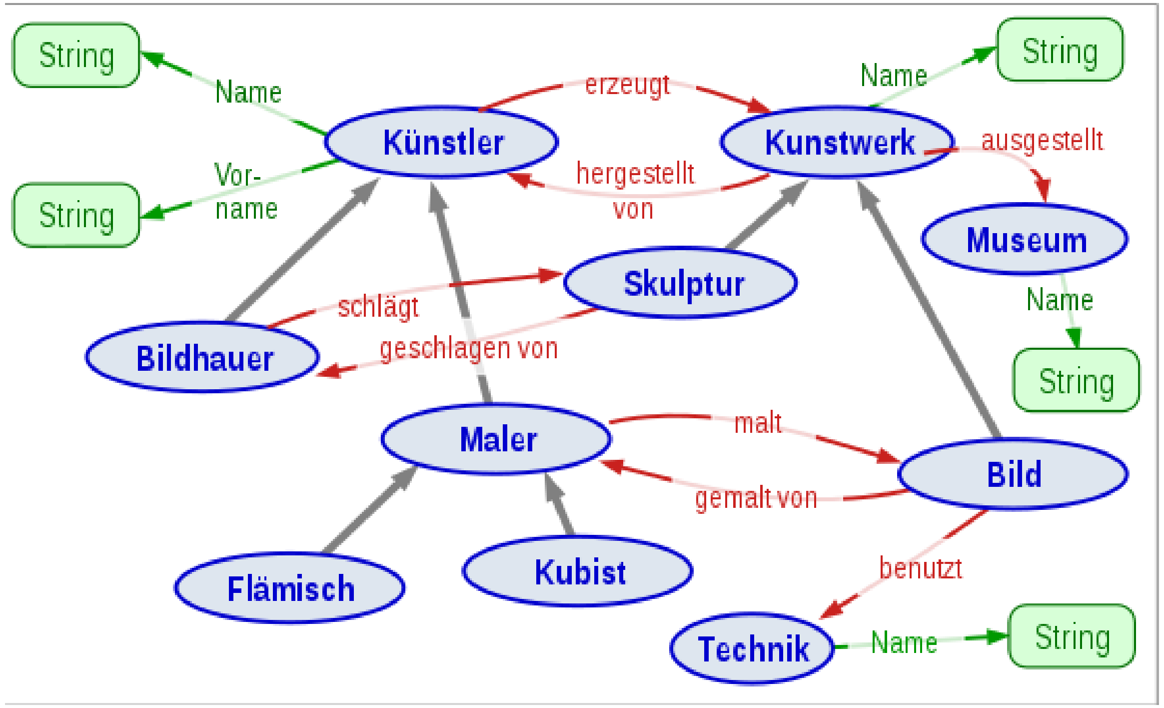
1. Term1
1.1 Subterm1.1
1.2 Subterm1.2
1.2.1 Subsubterm1.2.1
2. Term2
…

Semantic net, (ontology), knowledge graph:

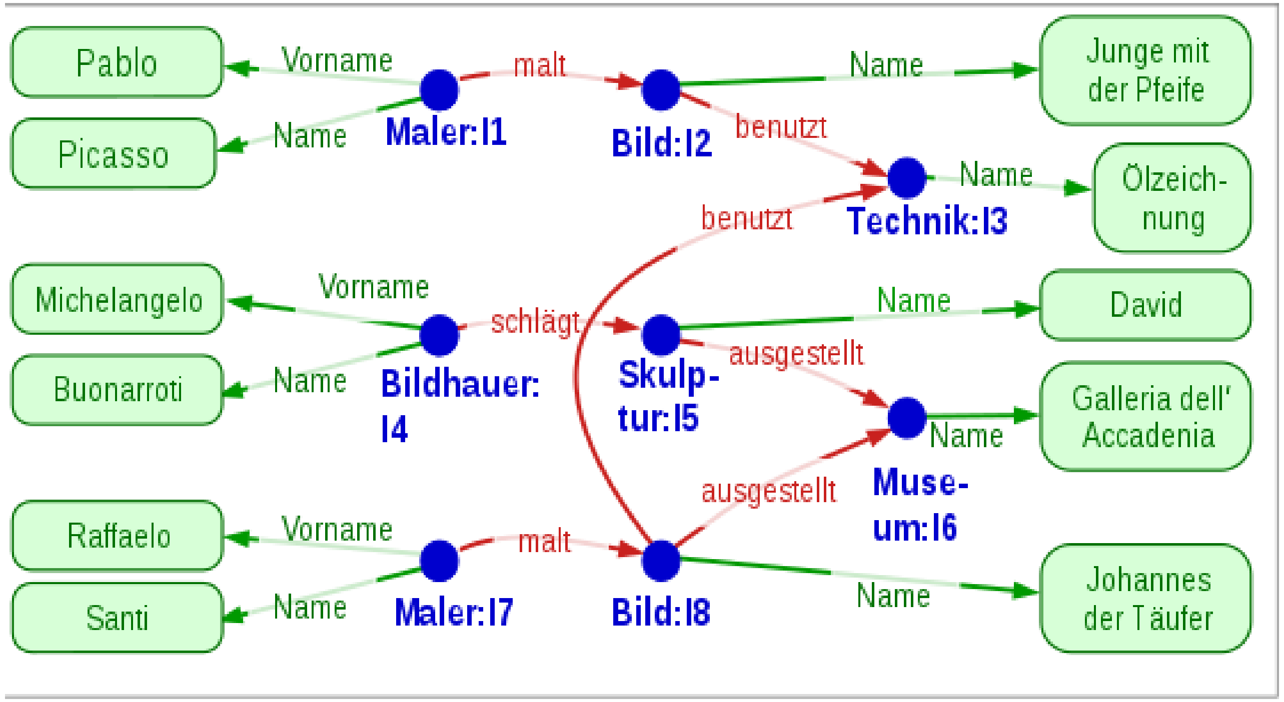


Example for Semantic Net (Knowledge graph)

Concepts of Semantic Net



Instances of Semantic Net



from: https://de.wikipedia.org/wiki/Ontologie_(Informatik)

Example for an Ontology (Knowledge Graph): Dbpedia

- Contains structured information from Wikipedia (e.g. infoboxes, categorization information, images, geo-coordinates and links to external Web pages)
- This structured information is extracted and put in a uniform dataset which can be queried.
- Founder: People at the Free University of Berlin and Leipzig University with OpenLink Software
- Maintained by people at the University of Mannheim and Leipzig University.
- First publicly available dataset was published in 2007.
- 2014: The ontology covers 685 classes which form a subsumption hierarchy and are described by 2795 different properties with 4 233 000 instances
- June 2021: it contains over a trillion entities.

Instances per class

Class	Instances
Resource (overall)	4,233,000
Place	735,000
Person	1,450,000
Work	411,000
Species	251,000
Organisation	241,000

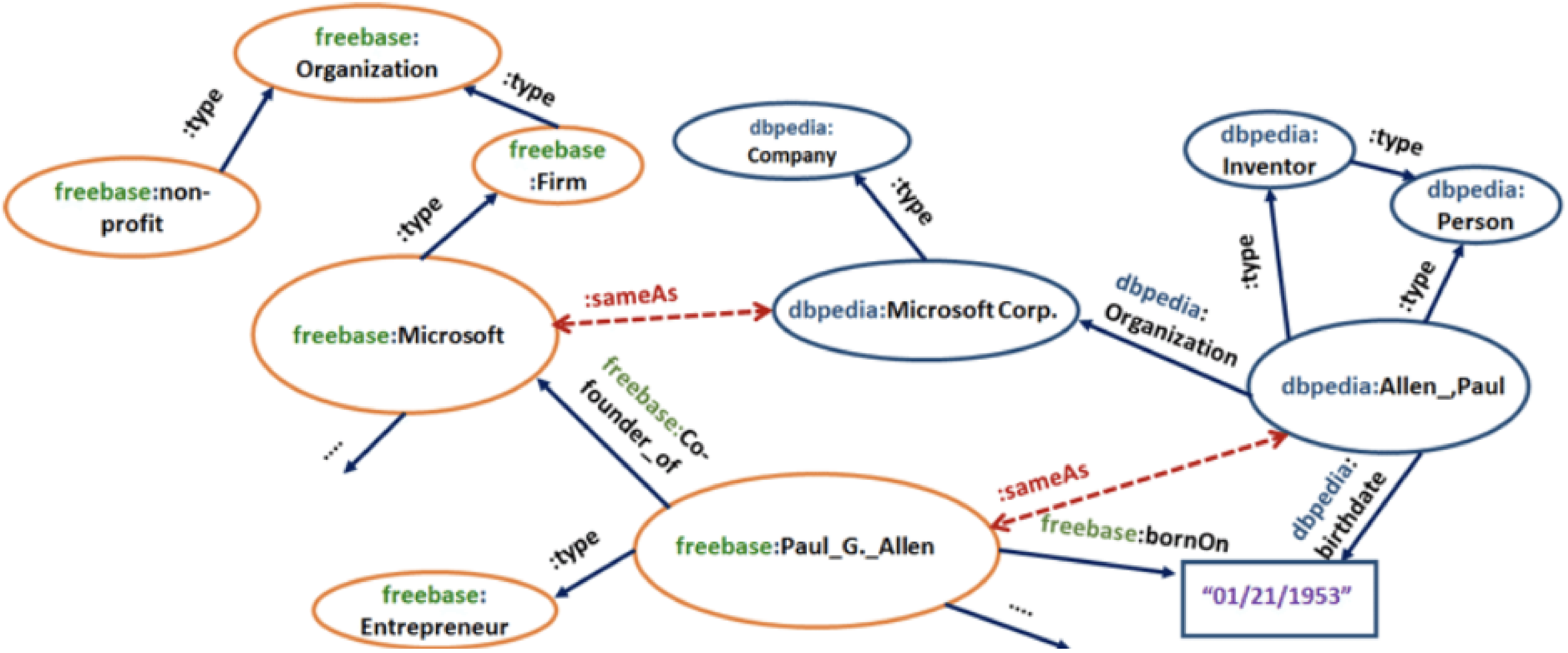
Dbpedia

The DBpedia Ontology is provided for download in four parts (in many different languages):

- DBpedia Ontology T-BOX (Schema)
- DBpedia Ontology A-Box RDF type statements (Instances with their classes)
- DBpedia Ontology A-Box RDF object relations (Instances with properties and values)
- DBpedia Ontology A-Box RDF literal facts (Instances with numeric data and text data)
- DBpedia Ontology A-Box RDF specific properties (Instances with numeric properties and values in normalized units)

- http://mappings.dbpedia.org/server/ontology/classes/

Part of Dbpedia and freebase



Example Query for Dbpedia in SPARQL (1)

(2011-dbpedia-train-3)

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX yago: <http://dbpedia.org/class/yago/>
PREFIX prop: <http://dbpedia.org/property/>
SELECT ?uri ?string
WHERE
{
        ?uri rdf:type yago:FemaleHeadsOfGovernment.
        ?uri prop:office ?office .
        FILTER regex(?office, 'Chancellor of Germany').
        OPTIONAL {?uri rdfs:label ?string . FILTER (lang(?string) = 'en') }
}
```

Female heads of government that have an office that match the expression /Chancellor of Germany/. Show also, if available, these female heads of government's English labels.

Example Query for Dbpedia in SPARQL (2)

(2011-dbpedia-train-9)

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX onto: <http://dbpedia.org/ontology/>
SELECT ?uri ?string
WHERE
{
        ?subject rdf:type onto:Software .
        ?subject rdfs:label 'World of Warcraft'@en .
        ?subject onto:developer ?uri .
        OPTIONAL {?uri rdfs:label ?string . FILTER (lang(?string) = 'en') }
}
```

Developers of software that have the English label "World of Warcraft". Show also, if available, these developers' English labels.

Example Query for Dbpedia in SPARQL (3)

(2011-dbpedia-train-14)

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX onto: <http://dbpedia.org/ontology/>
SELECT ?uri ?string
WHERE
{
        ?orga rdf:type onto:Organisation .
        ?orga onto:keyPerson ?uri .
        ?orga rdfs:label 'Aldi'@en .
        OPTIONAL {?uri rdfs:label ?string . FILTER (lang(?string) = 'en') }
        FILTER (lang(?string) = 'en')
}
```

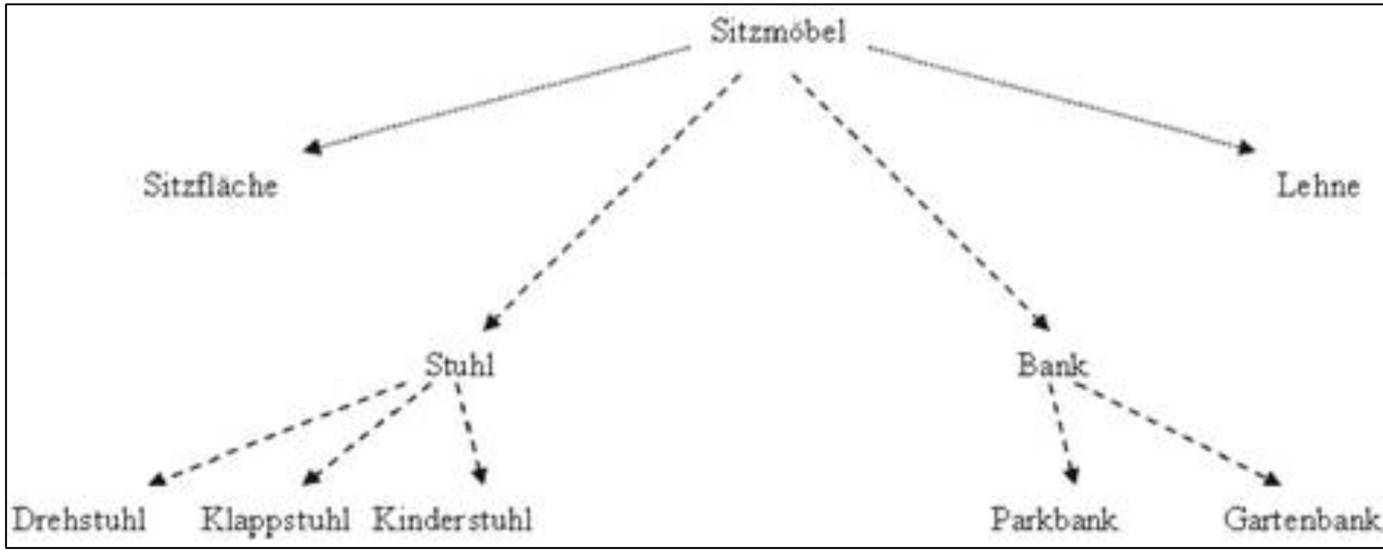
Things that are organisation key people of things that have the English label "Aldi". Show also, if available, these things' English labels.

GermaNet

- Semantic network for the German language (similar to WordNet for English)
- Relates nouns, verbs, and adjectives semantically
- Synset: Grouping lexical units that express the same concept
- Defining semantic relations between these synsets (e.g. subclass and part-of)
- Distinct concepts for ambiguous words

Examples from GermaNet

- Entität
  - Objekt
    - Ding, Gebilde, Gegenstand, Sache
      - Artefakt, Werk
        - Einrichtungsgegenstand, Möbel, Möbelstück
          - Sitzmöbel
            - Stuhl
              - Drehstuhl



Partof and subclass relations in GermaNet.

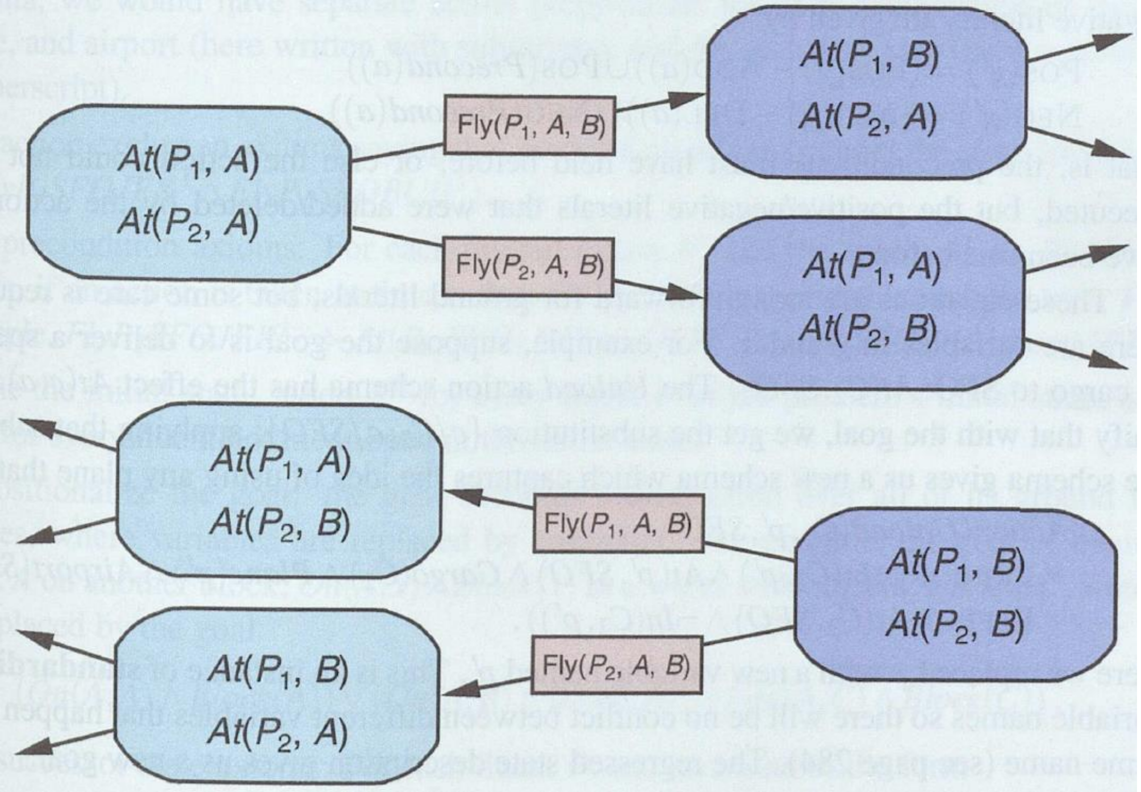
- Entität
  - Objekt
    - natürliches Objekt
      - Kreatur, Wesen, Wesenheit
        - Lebewesen, Organismus
          - höheres Lebewesen
            - Individuum, Mensch, Person, Persönlichkeit
              - Beschaffener, beschaffener Mensch
                - Charakterbeschaffener
                  - negativer Charakter
                    - Bösewicht, Ganove, Ganovin, Gauner, Gaunerin, Missetäter, Missetäterin, Übeltäter, Übeltäterin

Classical Planning

Actions of Cargo Planning Example:

$Action(Load(c,\ p,\ a),$
$\quad \text{PRECOND: } At(c,a) \wedge At(p,a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
$\quad \text{EFFECT: } \neg At(c,a) \wedge In(c,p))$
$Action(Unload(c,\ p,\ a),$
$\quad \text{PRECOND: } In(c,p) \wedge At(p,a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
$\quad \text{EFFECT: } At(c,a) \wedge \neg In(c,p))$
$Action(Fly(p,\ from,\ to),$
$\quad \text{PRECOND: } At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
$\quad \text{EFFECT: } \neg At(p, from) \wedge At(p, to))$

Forward and Backward state-space search:



Example Problem Description of Cargo Planning

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

Problem: Given: 1000 Cargos with orders from A to B, 10 planes, 10 airports. Goal: a plan …

- minimizing flight distances of planes
- minimizing time for completion

Approaches

- Forward state-space-search with optimistic heuristic for A*:
  - Sum up all distances of the Cargos not at their goal
  - Sum up all flight durations of the Cargos not at their goal and divide by 10 (for parallelism)

- Greedy strategy for initial plan:
  - For each plane: Take best open cargo (sum of cargo-start nearest to plane and cargo-goal nearest to another open cargo)
- Plan improvement (local search strategy on solutions):
  - Two planes exchange one cargo
  - Two planes exchange two cargos
  - Three planes exchange one cargo
  - Three planes exchange two cargos

Hierarchical Planning

- There are huge savings compared to non-hierarchical planning
  - Non-hierarchical: Final plan with d actions and b allowable actions in each state: $O(b^d)$
  - Hierarchical: If each non-primitive action has r possible refinements, each into k actions at the next level: $O(r^{(d-1)/(k-1)})$
- Give an application scenario for the formula!

Simple Application Scenario for Hierarchical Planning

- **Assumption:** Final plan with d actions, the non-primitive actions (HLAs) each have r possible refinement, into k actions at the next level $\Rightarrow$ $O(r^{(d-1)/(k-1)})$ decomposition trees
- d primitive actions in a tree with branching factor k $\Rightarrow$ $\log_k d$ levels beneath root
  - *e.g. d= 4; k=2 $\Rightarrow$ 2 levels beneath the root: 2,4,*
- number of nodes in the last refinement level: $k^{(\log_k d) - 1}$
  - *e.g. $2^{2-1} = 2$*
- total number of refinement nodes: $1 + k + k^2 + \ldots + k^{(\log_k d) - 1}$
  - *e.g. 1 + 2 = 3*
- $1 + k + k2 + \ldots + k^{(\log_k d) - 1} = (d-1) / (k-1)$
  - *e.g. (4-1) / (2-1) = 3*
- Each refinement node has r possible refinements: $r^{(d-1) / (k-1)}$
  - e.g. with r = 2: $2^{3/1} = 8$ refinements in total

	Start							
Choice (OR)	HLA1				HLA2			
Expansion (AND)	HLA11		HLA12		HLA21		HLA22	
Choice (OR)	HLA111	HLA112	HLA121	HLA122	HLA211	HLA212	HLA221	HLA222
Expansion (AND)	B111-1	B112-1	B121-1	B122-1	B211-1	B212-1	B221-1	B222-1
	B111-2	B112-2	B121-2	B122-2	B211-2	B212-2	B221-2	B222-2

Example for Simple Application Scenario

Given 2 vacuum cleaners V1 or V2 each with 2 brushes B1 or B2 and 2 rooms X and Y each with 2 squares L and R to be cleaned
Sequence of squares is given: XL, XR, YL, YR, but vacuum cleaner and brushes are choices
Non-hierarchical search tree: for each square, there are 4 choices: 4*4*4*4=256 sequences: $O(b^d) = O(4^4) = 256$
Hierarchical search tree with HLA (High Level Actions): (1) Decide cleaner, (2) Decide brush for each room: $O(r^{(d-1)/k-1)})$
with r =2; d= 4 and k=2: $2^{(4-1)/(2-1)} = 2^3 = 8$ sequences of basic actions (B)

	Start								
Choice (OR)	HLA-V1				HLA-V2				vacuum cleaner V1 or V2
Expansion (AND)	HLA-V1-X		HLA-V1-Y		HLA-V2-X		HLA-V1-Y		Room X and Room Y
Choice (OR)	HLA-V1-X-B1	HLA-V1-X-B2	HLA-V1-Y-B1	HLA-V1-Y-B2	HLA-V2-X-B1	HLA-V2-X-B2	HLA-V1-Y-B1	HLA-V2-Y-B2	Brush B1 or B2 of vacuum cleaner
Expansion (AND)	B-V1B1-XL	B-V1B2-XL	B-V1B1-YL	B-V1B2-YL	B-V2B1-XL	B-V2B2-XL	B-V2B1-YL	B-V2B2-YL	Square left = L
	B-V1B1-XR	B-V1B2-XR	B-V1B1-YR	B-V1B2-YR	B-V2B1-XR	B-V2B2-XR	B-V2B1-YR	B-V2B2-YR	Square right = R
	Choose V	**Choose B for X**	**Choose B for Y**	**Resulting Basic Action Sequences**					
Possible Sequences:	HLA-V1	HLA-V1-X-B1	HLA-V1-Y-B1	B-V1B1-XL	B-V1B1-XR	B-V1B1-YL	B-V1B1-YR		
		HLA-V1-X-B2	HLA-V1-Y-B1	B-V1B2-XL	B-V1B2-XR	B-V1B1-YL	B-V1B1-YR		
		HLA-V1-X-B1	HLA-V1-Y-B2	B-V1B1-XL	B-V1B1-XR	B-V1B2-YL	B-V1B2-YR		
		HLA-V1-X-B2	HLA-V1-Y-B2	B-V1B2-XL	B-V1B2-XR	B-V1B2-YL	B-V1B2-YR		
	HLA-V2	HLA-V2-X-B1	HLA-V2-Y-B1	B-V2B1-XL	B-V2B1-XR	B-V2B1-YL	B-V2B1-YR		
		HLA-V2-X-B2	HLA-V2-Y-B1	B-V2B2-XL	B-V2B2-XR	B-V2B1-YL	B-V2B1-YR		
		HLA-V2-Y-B1	HLA-V2-Y-B2	B-V2B1-XL	B-V2B1-XR	B-V2B2-YL	B-V2B2-YR		
		HLA-V2-Y-B2	HLA-V2-Y-B2	B-V2B2-XL	B-V2B2-XR	B-V2B2-YL	B-V2B2-YR		
				one-of	one-of	one-of	one-of		
Possible Sequences:				B-V1B1-XL	B-V1B1-XR	B-V1B1-YL	B-V1B1-YR		
without HLAs				B-V1B2-XL	B-V1B2-XR	B-V1B2-YL	B-V1B2-YR		
				B-V2B1-XL	B-V2B1-XR	B-V2B1-YL	B-V2B1-YR		
				B-V2B2-XL	B-V2B2-XR	B-V2B2-YL	B-V2B2-YR		

Extended Example

Extended situation:
Given 2 vacuum cleaners each with 2 two brushes and **4 rooms each with 4 squares** to be cleaned.

(1) What is the complexity of non-hierarchical search?

(2) What is the complexity of hierarchical search?

(2.1) As above, plan first the type of cleaner for all rooms, then the type of brushes for each room.
(2.2) Plan the type of cleaner and the type of brushers for all rooms identical.

Solution to Example

- (Non-)Hierarchical search tree **with 4 rooms each with 4 squares**?
  - Non-Hierarchical: $O(b^d)$) with d=16 and b=4: $O(4^{16}) = O(2^{32}) \approx 4\,000\,000\,000$;
  - Hierarchical: $O(r^{(d-1)/k-1})$) with r=2, d=16 and k=4: $2^{(16-1)/(4-1)} = 2^5 = 32$

- Hierarchical planning with cleaner and brush for all squares identically?
  - Hierarchical: $O(r^{(d-1)/k-1})$) with r=4, d=16 and k=16: $= 4^{(16-1)/(16-1)} = 4^1 = 4$

Human Hierarchical Planning

- Even higher saving can be achieved, if we can select the correct cleaner and room in the rules for the HLA. Then nearly no search at all is necessary.

Applications of Hierarchical Planning

- Build a house: standard example
- Plan a trip to multiple distinations
- Deliver things like food, parcels etc.
- Design a program

- Clean all rooms in a building with several floors
- Design a strategy for correction of the tasks of many examinations

Applications of Hierarchical Planning

- Plan a trip to multiple distinations:
  - decide places for accomodation and make reservations for places
  - decide main attractions and make reservations
  - decide trips to and between destinations
- Deliver things like food, parcels etc.:
  - divide orders in regions
  - divide orders in time intervalls
  - make a plan for each region and time intervall
- Design a program:
  - definition of data structures
  - definition of main modules with interfaces (beyond data structures)
  - definition of submodules within modules (maybe several times)
  - finally implementation of submodules

Applications of Hierarchical Planning

- Clean all rooms in a building with several floors
  - Sort the floors (e.g. from top to bottom or from bottom to top or according to features like carpets)
  - Sort the rooms within a floor (e.g. to minimum distance for TSP)

- Design a strategy for examination correction
  - HLA:
    - Sort the exams alphabetically, randomly or competence-oriented
      - Take one Exam after the other and correct all its tasks
      - Correct one or several Task(s) of all exams and than the next task(s)