Name:

Birth Date:

Matriculation Number:

Field of Study:

# Exam
# Künstliche Intelligenz 1

Feb 12., 2018

| To be used for grading, do not write here | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| prob. | 1.1 | 1.2 | 2.1 | 2.2 | 3.1 | 3.2 | 4.1 | 4.2 | 5.1 | 5.2 | 6.1 | Sum |
| total | 4 | 12 | 5 | 5 | 3 | 10 | 4 | 10 | 5 | 10 | 14 | 82 |
| reached | | | | | | | | | | | | |

Exam Grade: Bonus Points: Final Grade:

Organizational Information

**Please read the following directions carefully and acknowledge them with your signature.**

- Bitte legen Sie Ihren Studentenausweis und einen Lichtbildausweis zur Personenkontrolle bereit!
- Die angegebene Punkteverteilung gilt unter Vorbehalt.
- Es sind keine Hilfsmittel erlaubt.
- Die Lösung einer Aufgabe muss auf den vorgesehenen freien Raum auf dem Aufgabenblatt geschrieben werden; die Rückseite des Blatts kann mitverwendet werden. Wenn der Platz nicht ausreicht, können bei der Aufsicht zusätzliche Blätter angefordert werden.
- Wenn Sie die Prüfung aus gesundheitlichen Gründen abbrechen müssen, so muss Ihre Prüfungsunfähigkeit durch eine Untersuchung in der Universitätsklinik nachgewiesen werden. Melden Sie sich in jedem Fall bei der Aufsicht und lassen Sie sich das entsprechende Formular aushändigen.
- Die Bearbeitungszeit beträgt 90 Minuten.
- Überprüfen Sie Ihr Exemplar der Klausur auf Vollständigkeit (**??** Seiten inklusive Deckblatt und Hinweise) und einwandfreies Druckbild! Vergessen Sie nicht, auf dem Deckblatt die Angaben zur Person einzutragen!

# Erklärung

Durch meine Unterschrift bestätige ich den Empfang der vollständigen Klausurunterlagen und die Kenntnisnahme der obigen Informationen.

Erlangen, Feb 12., 2018 .....

(Unterschrift)

# 1 Prolog

**Problem 1.1 (The Zip Function)**

The zip function takes two lists with lengths that differ at most by 1, and outputs a list of lists containing one element from the first list and the element with the same index from the other list, possibly followed by a one-element list with the left-over argument. 4pt 4min

Create a `ProLog` predicate with 3 arguments: the first two would be the two lists you want to zip, and the third one would be the result. For instance:

```
?− zip([1,2,3],[4,5,6],L).
L = [[1, 4], [2, 5], [3, 6]].
```

```
?− zip([1,2],[3,4,5],L).
L = [[1, 3], [2, 4], [5]].
```

Feel free to implement any helper functions.

**Hint:** Remember that you can pattern match a list L as [HEAD|TAIL].

**Solution**:

```
zip([L],[],[[L]]).
zip([],[L],[[L]]).
zip([A],[B],[[A,B]]).
zip([H1|T1],[H2|T2],L) :− zip(T1,T2,T), append([[H1,H2]],T,L).
```

**Problem 1.2 (DFS in Prolog)**

We want to implement DFS in prolog. We'll use the following data structures for search trees: 12pt 12min

```
subtrees([]).
subtrees([(Cost,T)|Rest]) :- number(Cost),istree(T), subtrees(Rest).
istree(tree(_,Children)) :- subtrees(Children).
```

Write a method dfs such that `dfs(G,T,X,Y)` on a tree T returns the path to the goal G in X and the cost of the path in Y

**Solution**:

```
dfs(GoalValue,tree(GoalValue,_),GoalValue,0).
dfs(GoalValue,tree(Value,[(Cost,T)|Rest]),Path,FinalCost) :- T = tree(IV,_), write(IV ),
     dfs(GoalValue, T,P,C),string_concat(Value,P,Path),FinalCost is C+Cost; % go down one depth
     dfs(GoalValue,tree(Value,Rest),Path,FinalCost). % next child
```

# 2 Search

**Problem 2.1 (Astar vs. Greedy)**
Shortly explain the principle of operation of the A* search. How does it differ from the Greedy search? 5pt 5min

**Solution**: A* will expand the nodes in the fringe in an ascending order of the function f(node)=h(node)+g(node), where h(node) is the heuristic of the node and g(node) is the (current) distance from the initial node to this node. Greedy will expand only taking the heuristic into account.

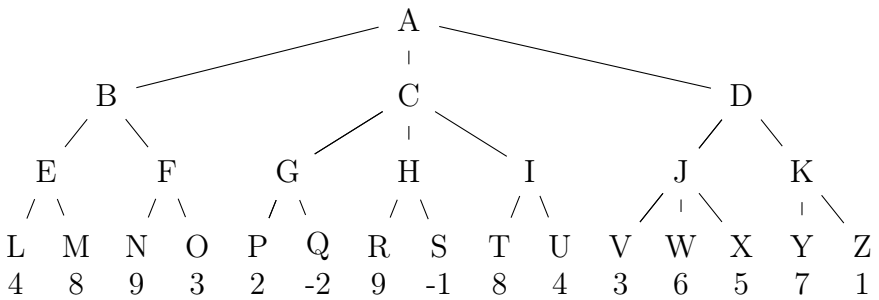
**Problem 2.2 ($A^*$ Theory)**
What is the condition on the heuristic function that makes $A^*$ optimal? Does a heuristic with this condition always exist? 5pt 5min

**Solution**: Admissible heuristic - always underestimates the real cost to the goal. This always exists: $h(x) = 0$.

# 3 Adversarial Search

**Problem 3.1 (Minimax Restrictions)**
Name at least five criteria that a game has to satisfy in order for the minimax algorithm to be applicable. 3pt 3min

**Problem 3.2 (Minimax Algorithm and Alphabeta Pruning)**
Consider the following (complete!) search tree: 10pt 10min




1. What is the minimax value at node B?
2. Which subtrees in the tree can be pruned during alpha-beta search? What is the criterion for pruning a subtree?

# 4 Constraint Satisfaction Problems & Inference

**Problem 4.1 (Constraint Networks)**
* * A constraint network is a triple $\langle V, D, C \rangle$. Explain the roles of $V$, $D$, and $C$. 4pt
If you use the word "constraint" you have to define and briefly explain it. 4min

**Problem 4.2 (Constraint Networks)**
Describe the following problems as constraint networks 10pt 10min

1. Sudoku
2. The 8 queens problem

---

**Solution**: Here are some possible answers:

**Sudoku**

- Variables: One variable $v_{i,j}$ for each coordinate $(i, j)$ in the sudoku field.
- Domains: $\{1, ..., 9\}$ for each variable.
- Constraints: For each pair $v_{i,j}, v_{k,\ell}$ a binary constraint that is invalid iff
  - $i = k$ and $v_{i,j} = v_{k,\ell}$ (same row)
  - $j = \ell$ and $v_{i,j} = v_{k,\ell}$ (same column)
  - $v_{i,j}$ and $v_{k,\ell}$ are in the same block and $v_{i,j} = v_{k,\ell}$.

**8-queens-problem**

- Variables: $\{Q_1, ..., Q_8\}$ (where $Q_i$ represents the queen in row $i$).
- Domains: $\{1, ..., 8\}$ (representing the columns).
- Constraints: For each pair $Q_i, Q_j$ (with $i < j$) a binary constraint that is satisfied iff $Q_i \neq Q_j$ (no two queens in the same column) and $Q_j \neq Q_i + (j - i)$ and $Q_j \neq Q_i - (j - i)$ (not diagonal).

---

# 5 Logic

**Problem 5.1 (Propositional Resolution)**
Prove the following formula using the resolution calculus: 5pt

$$\neg(\neg P \Rightarrow Q) \vee (Q \vee R) \vee \neg(P \Rightarrow R)$$ 5min

**Problem 5.2 (First-Order Tableau)**
Prove the following formula using the first-order free variable tableaux calculus. We have $P \in \Sigma_2^p$, $f, g \in \Sigma_1^f$ and $a, b \in \Sigma_0^f$. 10pt 10min

$$\exists X\,((\exists Y \neg P(Y, f(b))) \vee \neg(P(b, f(X)) \Rightarrow \neg P(g(a), f(X))))$$

---

**Solution**:

$$(1) \;\big|\; \dots \;\big\|$$

---

# 6 Planning

**Problem 6.1** Cheeta is an intelligent and lazy monkey. Your task is to help him grab a banana. 12pt 12min

Initially, the monkey and a boat are "Low" on the ground at position "A" and the banana hangs "High" at position "B". There is a river between "A" and "B", and Cheeta can only travel between these positions by boat. Cheeta can climb up the tree where the banana hangs, meaning that it can climb from height "Low" to "High". Cheeta can grab the banana when they are both at the same position and at the same height.

The following STRIPS model is used. The facts are:

- $At(x, y)$: $x \in \{Boat, Banana, Cheeta\}$ is at position $y \in \{A, B, Boat\}$.
- $Height(x, y)$: $x \in \{Boat, Banana, Cheeta\}$ is at height $y \in \{Low, High\}$.
- $Climbable(x)$: Cheeta can climb at position $x \in \{A, B\}$.
- $Grabbed()$: Cheeta has grabbed the banana.

The goal for Cheeta is to grab the banana using the following available actions:

- $Drive(x, y)$ to get from $x$ to $y$
- $GetIn(x)$ to get in the boat (at location $x$)
- $GetOut(x)$ to get out of the boat (at location $x$)
- $Climb(x, y, z)$ to climb at position $x$ from height $y$ to height $z$
- $Grab(x, y)$ to grab the banana (at position $x$ and height $y$)

(a) Properly define the actions $Drive(x, y)$ and $GetIn(x)$

(b) Give the initial state and a solution to this planning problem

---

**Solution**:

(a)
- $Drive(x, y) =$

$$pre : \{At(Cheeta, Boat), At(Boat, x), Height(Boat, Low)\}$$
$$add : \{At(Boat, y)\}$$
$$del : \{At(Boat, x)\}$$

  for all $x, y \in \{A, B\}$.
- $GetIn(x) =$

$$pre : \{At(Cheeta, x), At(Boat, x), Height(Cheeta, Low), Height(Boat, Low)\}$$
$$add : \{At(Cheeta, Boat)\}$$
$$del : \{At(Cheeta, x)\}$$

- $GetOut(x) =$

  $pre : \{At(Cheeta, Boat), At(Boat, x), Height(Cheeta, Low), Height(Boat, Low)\}$
  $add : \{At(Cheeta, x)\}$
  $del : \{At(Cheeta, Boat)\}$

  for all $x \in \{A, B\}$.

- $ClimbUp(x, y, z) =$

  $pre : \{At(Cheeta, x), Climbable(x), Height(Cheeta, y), Next(y, z)\}$
  $add : \{Height(Cheeta, z)\}$
  $del : \{Height(Cheeta, y)\}$

  for all $x \in \{A, B\}$, $y, z \in \{Low, Middle, High\}$.

- $ClimbDown(x, y, z) =$

  $pre : \{At(Cheeta, x), Climbable(x), Height(Cheeta, y), Next(z, y)\}$
  $add : \{Height(Cheeta, z)\}$
  $del : \{Height(Cheeta, y)\}$

  for all $x \in \{A, B\}$, $y, z \in \{Low, Middle, High\}$.

- $UseLiana() =$

  $pre : \{Height(Cheeta, High)\}$
  $add : \{Height(Cheeta, Low)\}$
  $del : \{Height(Cheeta, High)\}$

- $Grab(x, y) =$

  $pre : \{At(Cheeta, x), At(Banana, x), Height(Cheeta, y), Height(Banana, y)\}$
  $add : \{Grabbed()\}$
  $del : \{At(Banana, x), Height(Banana, y)\}$

  for all $x \in \{A, B\}$, $y \in \{Low, Middle, High\}$.

(b)

---