Name:

Birth Date:

Matriculation Number:

Field of Study:

Final Exam
Künstliche Intelligenz 1

Feb 13., 2017

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	4.3	4.4	5.1	Sum
total	5	5	5	7	8	5	6	5	6	5	18	75
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 (16 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 13., 2017 .....

(Unterschrift)

1 Search

Problem 1.1 (A^* Theory)

What is the condition on the heuristic function that makes A^* optimal? Does a heuristic with this condition always exist? 5pt 3min

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

Problem 1.2 (A^* vs BFS)

Does A^* search always expands fewer nodes than BFS? Justify you answer. 5pt

Solution: No. With a bad heuristic, A^* can be forced to explore the whole space, just like BFS. 3min

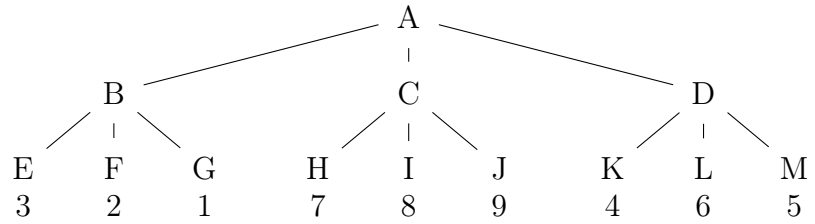
2 Adversarial Search

Problem 2.1 (Minimax Restrictions)

Name at least five criteria that a game has to satisfy in order for the minimax algorithm to be applicable. 5pt 5min

Problem 2.2 (Game Tree)

Consider the following game tree. Assume it is the maximizing player's turn to move. The values at the leaves are the static evaluation function values of the states at each of those nodes. 7pt 7min



1. Compute the minimax game value of nodes A, B, C, and D
2. Which move would be selected by Max?
3. List the nodes that the alpha-beta algorithm would prune (i.e., not visit). Assume children of a node are visited left-to-right.
4. How would the nodes in the tree need to be ordered to prune as many branches as possible?

Solution:

1. A = 7 ; B = 1 ; C = 7 ; D = 4
2. C
3. L, M
4. Level 2: left to right: C, D, B. Level 3: J,I,H,L,M,K,E,F,G. Prunes M, K, F, G

3 Constraint Satisfaction Problems & Inference

Problem 3.1 (Constraint Networks)

A constraint network is a triple $\langle V, D, C\rangle$. Explain the roles of V, D, and C. If you use the word "constraint" you have to explain it. 8pt 8min

Problem 3.2 (Arc consistency)

Define the concept of *arc consistency* using the terminology you introduced in Problem 3.1. 5pt 5min

4 Logic

Problem 4.1 (Inference and Entailment)

6pt 6min

1. Briefly explain the entailment and inference relations
2. how do we write "**A** entails **B**" and "from **A** we can infer/deduce **B**" in symbolism,
3. what is the difference between the two relations.

Solution:

1. The entailment relation $\models$ is defined via the models: We say $\mathbf{A} \models \mathbf{B}$, iff any model $\mathcal{M}$ that makes **A** true also makes **B** true. The inference relation $\vdash$ is defined by a set of purely syntactic inference rules.
2. "**A** entails **B**" is written as $\mathbf{A} \models \mathbf{B}$ and "from **A** we can infer/deduce **B**" as $\mathbf{A} \vdash \mathbf{B}$
3. Both are relations between expressions of the logic that try to model the process of argumentation from statements known to be true to new true statements. But entailment uses the notion of truth in a model, whereas the inference does this by a set of inference rules which are "known to preserve truth". In the best of all cases, the entailment and inference relations coincide. Then we have the "miracle of logics".

Problem 4.2 (First-Order Tableaux)

Prove the following formula using the tableaux calculus. 5pt 5min
$P \in \Sigma_1^p$,

$$\exists X.\,(P(X) \Rightarrow \forall Y.P(Y))$$

Problem 4.3 (First-Order Resolution)

Prove the following formula using resolution. 6pt 6min
$P \in \Sigma_1^p, R \in \Sigma_2^p, a, b \in \Sigma_0^f$

$$\exists X.\forall Y.\exists Z.\exists W.\,((\neg P(Z) \wedge \neg R(b,a)) \vee \neg R(a,b) \vee R(W,a) \vee (P(Y) \wedge R(X,b)))$$

Problem 4.4 (The Zip Function)

5pt 5min

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. 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.

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).
```

5 Planning

Problem 5.1 18pt

18min

Consider the following problem. A fused bulb hangs out of reach from the ceiling. A robot needs to repair the bulb. The room also contains a box. Pushing that box into the correct position, and climbing onto the box, will bring the bulb into reach for the robot.

The exercise is to model this problem as a STRIPS planning task. In doing so, assume the following framework. The robot is currently at position "A", the fused bulb is at position "B", and the box is at position "C". The robot and box are at the same height "Low", the fused bulb is at height "High". By climbing onto the box, the robot changes from "Low" to "High"; vice versa when climbing off the box. The actions available for the robot are "*Go*" from one place to another (only possible if the robot is at "Low"), "*Push*" an object from one place to another (only possible if the robot and object are at "Low"), "*ClimbUp*" onto or "*ClimbDown*" from an object, and "*Repair*" to fix an object. The robot needs to be at the same place and height as an object in order to repair it.

Note that the robot can only push an object or climb onto an object if both of them are at the same location. Furthermore, in case of pushing an object the robot changes to the destination location as well.

(a) Write a STRIPS formalization of the initial state and goal descriptions.

(b) Write a STRIPS formalization of the five actions. In doing so, please do make use of "object variables", i.e., write the actions up in a parameterized way. State, for each parameter, by which objects it can be instantiated.

In both (a) and (b), make use of the following predicates: (do not use any other predicates)

- $At(x, y)$: To indicate that object $x \in \{Box, Bulb, Robot\}$ is at position $y \in \{A, B, C\}$.
- $Height(x, y)$: To indicate that object $x \in \{Box, Bulb, Robot\}$ is at height $y \in \{Low, High\}$.
- $Pushable(x)$: To indicate that object $x \in \{Box, Bulb, Robot\}$ can be pushed.
- $Climbable(x)$: To indicate that the robot can climb on object $x \in \{Box, Bulb, Robot\}$.
- $Repaired(x)$: To indicate that object $x \in \{Box, Bulb, Robot\}$ is repaired.

Solution:

(a) Initial state description:

$$\begin{aligned} I \;=\; & \{At(Robot, A), At(Bulb, B), At(Box, C), \\ & Height(Robot, Low), Height(Box, Low), Height(Bulb, High), \\ & Pushable(Box), Climbable(Box)\} \end{aligned}$$

Goal description: $G = \{Repaired(Bulb)\}$

(b) Action descriptions:

- $Go(x, y) =$

$$\begin{aligned} & pre : \{At(Robot, x), Height(Robot, Low)\} \\ & add : \{At(Robot, y)\} \\ & del : \{At(Robot, x)\} \end{aligned}$$

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

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

$$\begin{aligned} & pre : \{At(Robot, y), Pushable(x), At(x, y), Height(Robot, Low), Height(x, Low)\} \\ & add : \{At(x, z), At(Robot, z)\} \\ & del : \{At(x, y), At(Robot, y)\} \end{aligned}$$

  for all $x \in \{Box, Bulb, Robot\}$, $y, z \in \{A, B, C\}$.

- $ClimbUp(x, y) =$

$$\begin{aligned} & pre : \{At(Robot, y), At(x, y), Climbable(x), Height(Robot, Low), Height(x, Low)\} \\ & add : \{Height(Robot, High)\} \\ & del : \{Height(Robot, Low)\} \end{aligned}$$

  for all $x \in \{Box, Bulb, Robot\}$, $y \in \{A, B, C\}$.

- $ClimbDown() =$

$$\begin{aligned} & pre : \{Height(Robot, High)\} \\ & add : \{Height(Robot, Low)\} \\ & del : \{Height(Robot, High)\} \end{aligned}$$

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

  $pre : \{At(Robot, y), At(x, y), Height(Robot, z), Height(x, z)\}$
  $add : \{Repaired(x)\}$
  $del : \{\}$

  for all $x \in \{Box, Bulb, Robot\}$, $y \in \{A, B, C\}$, $z \in \{High, Low\}$.
