Name:

Birth Date:

Matriculation Number:

Exam
Künstliche Intelligenz 1

Jul 24., 2017

	To be used for grading, do not write here													
prob.	1.1	1.2	2.1	2.2	3.1	3.2	3.3	4.1	4.2	4.3	4.4	5.1	Sum	grade
total	5	5	5	7	8	5	5	6	5	6	5	18	80	
reached														

Exam Grade: Bonus Points: Final Grade:

The "solutions" to the exam/assignment problems in this document are supplied to give students a starting point for answering questions. While we are striving for helpful "solutions", they can be incomplete and can even contain errors.

If you find "solutions" you do not understand or you find incorrect, discuss this on the course forum and/or with your TA and/notify the instructors.

In any case, grading student's answers is not a process of simply "comparing with the reference solution", therefore errors in the "solutions" are not a problem in this case.

In the course Artificial Intelligence I/II we award 5 bonus points for the first student who reports a factual error (please report spelling/formatting errors as well) in an assignment or old exam and 10 bonus points for an alternative solution (formatted in LaTeX) that is usefully different from the existing ones.

1 Search

Problem 1.1 (A^* Theory) 5 pt

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

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

Problem 1.2 (A^* vs. BFS) 5 pt

Does A^* search always expand fewer nodes than BFS? Justify you answer. 3 min

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

2 Adversarial Search

Problem 2.1 (Minimax Restrictions) 5 pt

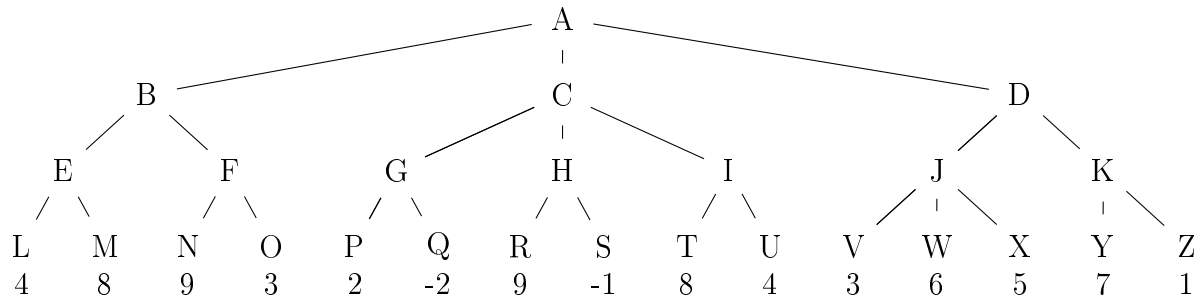
Name at least five criteria that a game has to satisfy in order for the minimax algorithm to be applicable. 5 min

Solution:Any five of:

- Two-player
- Determininstic
- Fully observable
- Players alternate
- Finitely many / discrete game states
- Zero-sum
- Game ends after finitely many rounds

Problem 2.2 (Game Tree) 7 pt

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. 7 min



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.

Solution:

1. B = 8, C = 2, D = 6, A = 8
2. B
3. O, H (and R and S), I (and T and U), K (and Y and Z)

3 Constraint Satisfaction Problems & Inference

Problem 3.1 (Constraint Networks) 8 pt

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 define and briefly explain it. 8 min

Solution:V: a set of variables

D: a set of sets D_v of values for each $v \in V$

C: a set of constraints; i.e. relations on (the domains of the) variables.

Problem 3.2 (Arc consistency) 5 pt

Define the concept of *arc consistency*. 5 min

Solution:A variable u is arc consistent relative to v, if there is either no constraint between u and v, or for every value $d \in D_u$, there is some $d' \in D_v$ such that $(d, d') \in C_{uv}$. A constraint network is arc consistent if all variables are pairwise arc consistent relative to each other.

Problem 3.3 (50 Queens) 5 pt

Formalize the *50 Queens Problem* as a constraint network. Hint: You do not have to write down all the constraints explicitly, but it has to be clear what the exact constraints are. 5 min

4 Logic

Problem 4.1 (Inference and Entailment) 6 pt

6 min

1. Define and 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) 5 pt

Prove the following formula using the first-order free variable tableaux calculus. We have $P \in \Sigma_1^p$. 5 min

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

Solution:

(1)	$\exists X.(P(X) \Rightarrow \forall Y.P(Y))^F$	
(2)	$P(V_X) \Rightarrow \forall Y.P(Y)^F$	(from 1)
(3)	$P(V_X)^T$	(from 2)
(4)	$\forall Y.P(Y)^F$	(from 2)
(5)	$P(c_Y)^F$	(from 4)
(6)	$\bot[c_Y/V_X]$	

Problem 4.3 (First-Order Resolution) 6 pt

Prove the following formula using resolution. 6 min

$P \in \Sigma_1^p, R \in \Sigma_2^p, a, b \in \Sigma_0^f$

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

Solution:We negate:

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

We skolemize:

$$(P(Z) \vee R(b,a)) \wedge R(a,b) \wedge \neg R(W,a) \wedge (\neg P(f_Y(X)) \vee \neg R(X,b))$$

This yields the clauses $\{P(Z)^T, R(b,a)^T\}, \{R(a,b)^T\}, \{R(W,a)^F\}, \{P(f_Y(X))^F, R(X,b)^F\}$. We resolve:

$$\{P(Z)^T, R(b,a)^T\} + \{R(W,a)^F\}[b/W] \Longrightarrow \{P(Z)^T\}$$
$$\{R(a,b)^T\} + \{P(f_Y(X))^F, R(X,b)^F\}[a/X] \Longrightarrow \{P(f_Y(a))^F\}$$
$$\{P(Z)^T\}[f_Y(a)/Z] + \{P(f_Y(a))^F\} \Longrightarrow \{\}$$

Problem 4.4 (The Zip Function) 5 pt

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. 5 min

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([],[],[]).
zip([H1|T1],[H2|T2],L) :- zip(T1,T2,T), L = [[H1,H2]|T).
```

5 Planning

Problem 5.1 18 pt

18 min

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: Go, $Push$, $ClimbUp$, $ClimbDown$, and $Repair$. 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:

- $Go(x, y) =$

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

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

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

  for all $x \in \{Box, Bulb, Robot\}$, $y, z \in \{A, B, C\}$.
- $ClimbUp(x, y) =$

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

  for all $x \in \{Box, Bulb, Robot\}$, $y \in \{A, B, C\}$.
- $ClimbDown() =$

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

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

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

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