CS4618 Artificial Intelligence I

Dr. D.G. Bridge

Answer all questions.
Silent non-programmable calculators may be used.

1.5 Hours
(80 marks: Approximately 1 minute per mark)
1. (15 marks) This question is about TLUs and neural networks. Throughout, assume that the activation function of the TLUs, \( g \), is defined as follows:

\[
g(x) \overset{\text{def}}{=} \begin{cases} 1 & \text{if } x \geq \theta \\ 0 & \text{otherwise} \end{cases}
\]

where \( \theta \) is the threshold of the TLU.

i) (5 marks) Design a TLU which has two inputs \( s_1 \) and \( s_2 \). The inputs can take values of 0 or 1. The TLU should compute \( s_1 \mid s_2 \), where \( \mid \) is the NAND operator, i.e. \( s_1 \mid s_2 \equiv \neg(s_1 \land s_2) \).

ii) (2 marks) Your answer to part (i) is to be converted into a TLU that has a threshold of 0 but it has an extra input \( s_0 \). At what value will you fix the input \( s_0 \) and what weight will you use on \( s_0 \)'s input wire to continue to compute \( s_1 \mid s_2 \)?

iii) (2 marks) The kind of conversion that you carried out in part (ii) is a useful precursor to training a neural net using a learning algorithm. Why is it useful?

iv) (6 marks) Below we have tabulated eight Boolean-valued functions:

| \( s_1 \) \( s_2 \) | \( s_1 \boxtimes s_2 \) | \( s_1 \boxdot s_2 \) | \( s_1 \oplus s_2 \) | \( s_1 \ominus s_2 \) | \( s_1 \otimes s_2 \) | \( s_1 \oslash s_2 \) | \( s_1 \ominus s_2 \)
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Suppose you intend to design eight TLUs, each having two inputs, \( s_1 \) and \( s_2 \), that can take values of only 0 or 1.

For how many of the eight functions can such TLUs be designed? Explain your answer convincingly and in detail. (There is no need to give any actual TLUs.)
2. (25 marks)

i) (2 marks) What is meant by the *branching factor* of a state space?

ii) (8 marks) Let $b$ be the branching factor of a state space, $d$ be the length of the path to the shallowest solution, and $m$ be the maximum of the lengths of the paths in the state space.

Explain convincingly and in detail why the following are true:

a. The worst-case time complexity of breadth-first search is $O(b^{d+1})$.

b. The worst-case space complexity of breadth-first search is $O(b^{d+1})$.

c. The worst-case time complexity of depth-first search is $O(b^m)$.

d. The worst-case space complexity of depth-first search is $O(bm)$.

iii) (15 marks) Consider the following state space in which the states are shown as nodes labeled A through I. A is the initial state, and H is the goal state. The numbers alongside the edges represent the costs of moving between the states. To the right of every state is the estimated cost of the path from the state to the nearest goal.

![State Space Diagram]

Show how each of the following search strategies finds a solution in this state space by writing down, in order, the names of the nodes removed from the agenda. Assume the search halts when a goal state is removed from the agenda. (In some cases, multiple answers are possible. You need give only one such answer in each case.)

a. Breadth-first;

b. Depth-first;

c. Least-cost search; and

d. Greedy search, i.e. heuristic search using $f(n) = h(n)$ as the evaluation function, where $h(n)$ is the estimated cost of the cheapest path from node $n$ to a goal; and

e. Heuristic search using $f(n) = g(n) + h(n)$ as the evaluation function, where $g(n)$ is the cost of the path to node $n$, and $h(n)$ is the estimated cost of the path from node $n$ to the nearest goal.
3. (25 marks) This question concerns a system which automatically writes computer programs for a machine that has two registers. The question uses the following ‘key’ for the ternary predicate symbol \( \text{mem} \), the unary function symbols \( \text{load} \), \( \text{add} \) and \( \text{store} \), the binary function symbol \( + \), and the constant symbols \( a \), \( b \) and \( s_0 \):

- \( \text{mem}(x, y, z) \) : in state \( z \), the contents of the two registers are \( x \) and \( y \) respectively
- \( \text{load}(z) \) : \( \text{load} \) denotes a function that returns the new state of the machine attained if the machine is in state \( z \) and the contents of the second register are loaded into the first register
- \( \text{add}(z) \) : \( \text{add} \) denotes a function that returns the new state of the machine attained if the machine is in state \( z \) and the contents of the second register are added into the first register
- \( \text{store}(z) \) : \( \text{store} \) denotes a function that returns the new state of the machine attained if the machine is in state \( z \) and the contents of the first register are stored in the second register
- \( x + y \) : \( + \) denotes a function, written in infix notation, that returns the sum of its arguments
- \( 2 \) : the number 2
- \( 5 \) : the number 5
- \( s_0 \) : the initial state

\( (x, y, z \) and subscripted versions of these will be used as variables.)

i) (5 marks) Convert the following wff of FOPL into Clausal Form Logic. Show your working.

\[
\forall x \forall z ((\text{mem}(x, x, \text{store}(z)) \lor \text{mem}(x, x, \text{load}(z))) \rightarrow \exists y (\text{mem}(x, y, z) \lor \text{mem}(y, x, z)))
\]

ii) (3 marks) Determine whether the members of the following pairs of atoms unify with each other. If they do, give their most general unifier (mgu); if they do not, give a brief explanation.

a. \( \text{mem}(x, y, \text{store}(s_0)) \) and \( \text{mem}(2, 5, \text{store}(\text{load}(z))) \)

b. \( \text{mem}(2, 5, \text{store}(s_0)) \) and \( \text{mem}(x, x, z) \)

iii) (10 marks) You are given the following clauses:

If in state \( z_1 \), the first register contains \( x_1 \) and the second contains \( y_1 \), then in the state that results from a load operation, both registers will contain \( y_1 \).

\(-\text{mem}(x_1, y_1, z_1) \lor \text{mem}(y_1, y_1, \text{load}(z_1)) \)

If in state \( z_2 \), the first register contains \( x_2 \) and the second contains \( y_2 \), then in the state that results from an add operation, the first register will contain the sum of \( x_2 \) and \( y_2 \).

\(-\text{mem}(x_2, y_2, z_2) \lor \text{mem}(x_2 + y_2, y_2, \text{add}(z_2)) \)

If in state \( z_3 \), the first register contains \( x_3 \) and the second contains \( y_3 \), then in the state that results from a store operation, both registers will contain \( x_3 \).

\(-\text{mem}(x_3, y_3, z_3) \lor \text{mem}(x_3, x_3, \text{store}(z_3)) \)

In the initial state, \( s_0 \), the first register contains 2 and the second register contains 5.

\( \text{mem}(2, 5, s_0) \)

From these clauses, use resolution refutation theorem-proving to answer the question: what program doubles the contents of the second register, i.e. in FOPL:

\[ \exists x \exists z \text{mem}(x, 5 + 5, z) \]

Show your working, presenting your proof in the form of a refutation tree.

(Hint: Among the changes you will make to the goal is the introduction of a literal \( \text{ans}(z) \).)
iv) (7 marks) Part 3iii shows that, if instructions and machine states are suitably axiomatised, resolution-refutation theorem-proving can be used for automatic program synthesis. Equally, a planning system such as STRIPS or POP can also be used for (simple) program synthesis.

Using symbols that are similar to those used in part 3iii, translate the four axioms into the STRIPS language.

(Hint: States are explicit arguments in part 3iii but will be implicit in the STRIPS language representation.)
4. (15 marks) An A.I. planner operates in a simplified Blocks World. The only operators in its repertoire move a block $x$ from the table to another block $y$:

$\text{Op}( \text{ACTION: } \text{FromTable}(x, y),$
$\text{PRECOND: } \text{onTable}(x) \land \text{clear}(x) \land \text{clear}(y),$
$\text{EFFECT: } \neg \text{onTable}(x) \land \neg \text{clear}(y) \land \text{on}(x, y))$

and move a block $x$ from block $y$ to the table:

$\text{Op}( \text{ACTION: } \text{ToTable}(x, y),$
$\text{PRECOND: } \text{on}(x, y) \land \text{clear}(x),$
$\text{EFFECT: } \neg \text{on}(x, y) \land \text{clear}(y) \land \text{onTable}(x))$

Here is an incomplete plan of the kind that could be built by the POP planner covered in lectures:

Redraw the diagram in your answer booklet and complete the plan as the POP planner would.