The University of Nottingham

SCHOOL OF COMPUTER SCIENCE

A LEVEL 2 MODULE, AUTUMN SEMESTER 2011-2012

PLANNING AND SEARCH

Time allowed TWO hours

Candidates may complete the front cover of their answer book and sign their desk card but must NOT write anything else until the start of the examination period is announced

Answer FOUR out of SIX questions

Only silent, self contained calculators with a Single-Line Display are permitted in this examination.

Dictionaries are not allowed with one exception. Those whose first language is not English may use a standard translation dictionary to translate between that language and English provided that neither language is the subject of this examination. Subject specific translation dictionaries are not permitted.

No electronic devices capable of storing and retrieving text, including electronic dictionaries, may be used.

DO NOT turn your examination paper over until instructed to do so

1. This question is on local search.

   (a) Consider the following search problem:

   - the set of states $S = \{s_0, s_1, s_2, s_3, s_4\}$
   - successors of s_0 are $\{s_0, s_1, s_2\}$
   - successors of s_1 are $\{s_1, s_0, s_3\}$
   - successors of s_2 are $\{s_2, s_0, s_4\}$
   - successors of s_3 are $\{s_1, s_3\}$
   - successors of s_4 are $\{s_2, s_4\}$
   - objective function $Value$ is as follows: $Value(s_0) = 0$, $Value(s_1) = 2$, $Value(s_2) = 1$, $Value(s_3) = 3$, $Value(s_4) = 4$.

   Trace hill climbing search starting in state s_0 (indicate which state is considered at each iteration, and which solution is returned when the search terminates). Does it return the optimal solution? (5 marks)

   Answer. The highest-value successor of s_0 is s_1 (value 2). The highest value successor of s_1 is s_3 (value 3). s_3 does not have a higher value successor, so it will be returned as the solution. It is not optimal since s_4 has a higher value.

   (b) Explain the problem of local maxima and give an example of a local maximum. (3 marks)

   Answer. Hill climbing search gets stuck when it finds a solution which has no immediate better successors (a locally maximal solution), such as s_3 above. There may be better solutions (such as s_4 above), but to reach them the search has to pass through states which are not the best successors of the current state (such as: we chose s_1 as the better successor of s_0, but the path to the optimal solution went through s_2).

   (c) Trace k-beam search for the problem in question 1(a) with $k = 2$. Assume that the initial set of states is $\{s_0, s_2\}$. (4 marks)

   Answer. The highest value successor of s_0 is s_1, and the highest value successor of s_2 is s_4. The next set of states is $\{s_1, s_4\}$. The highest value successor of s_1 is s_3, and there is no successor of s_4 which has higher value. The next set of states is $\{s_3, s_4\}$. Now neither of the states has a better successor, so this set of states is returned.

   (d) Explain how simulated annealing search improves on hill climbing search with respect to the problem of local maxima. In particular, explain how the successor of a given state is chosen. Explain the role of the 'temperature' parameter in simulated annealing. Give the formula for the probability of a non-improving successor being chosen. (7 marks)

   Answer.

   In simulated annealing, a successor is chosen randomly. If the successor has a higher value than the current state, then successor always becomes the current state. If the successor does not have a higher value, then with probability $e^{\Delta E/T}$ (where $\Delta E < 0$ is the difference in value between the sucessor and the current state, and T is the temperature) the successor can still become the current state, so there is a chance to escape a local maximum by chosing a non-improving successor. The

temperature schedule assigns a 'temperature' T to each time step. The temperature schedule is assumed to be decreasing. The higher the temperature, the more likely a transition to a worse successor is. When the temperature is 0, the algorithm terminates and returns the current state.

(e) A genetic algorithm is to be used to evolve a binary string of length n (for some even number n) where the sum of bits is $n/2$. The initial population is a randomly generated set of binary strings of length n.

i. Give a suitable fitness function for this problem. (3 marks)

ii. Will the offspring of parents with a high fitness value generally also have a high fitness value, given your fitness function? Explain your answer. (3 marks)

Answer. A fitness function which will assign a higher value to strings whose sum is closer to $n/2$ would be for example $f(s) = n/2 - |n/2 - sum(s)|$, where $sum(s)$ is the sum of bits in the string s, and $|x|$ stands for the absolute value of x. The answer to the second question is no, because crossing two strings whose sum is very close to $n/2$ may just as likely produce an offspring with $sum(s) = n$ or $sum(s) = 0$.

2. This question is on SAT and search.

(a) Show how to encode the following search problem in propositional logic. (7 marks)

> A Computer Science department can offer the following first year modules: G51JAV, G51CPP, G51ARC, G51HAR, G51MAT. The department needs to decide which modules to make compulsory for a new degree course. The constraints on the set of compulsory modules are that there should be at least one programming module, at least one module on computer architecture, at least on module on mathematics, and there should be no more than one module on each topic. In addition, G51CPP and G51ARC are designed to be given together, so it does not make sense to include one without the other. G51JAV and G51CPP are programming modules, G51ARC, G51HAR are on computer architecture, and G51MAT is on mathematics.

Hint: use propositional symbol J for 'include G51JAV', C for G51CPP, A for G51ARC, H for G51HAR, M for G51MAT.

Answer. The constraint that there should be at least one module in each topic is expressed as

$$(J \vee C) \wedge (H \vee A) \wedge M$$

The constraint that no more than one module from each area is included is expressed as

$$\neg(J \wedge C) \wedge \neg(H \wedge A)$$

The constraint that G51CPP and G51ARC should be given together is expressed as

$$C \Leftrightarrow A$$

(b) Rewrite the constraints from part (a) in clausal form. (8 marks)

Answer. $(J \vee C) \wedge (H \vee A) \wedge M$ becomes

$$[J, C], [H, A], [M]$$

$\neg(J \wedge C) \wedge \neg(H \wedge A)$ (using de Morgan law $\neg(J \wedge C) = \neg J \vee \neg C$) becomes

$$[\neg J, \neg C], [\neg H, \neg A]$$

$C \Leftrightarrow A$ is equivalent to $(C \Rightarrow A) \wedge (A \Rightarrow C)$ which in turn is equivalent to $(\neg C \vee A) \wedge (\neg A \vee C)$, which becomes

$$[\neg C, A], [\neg A, C]$$

(c) Trace the SAT solving algorithm DPLL on the set of clauses from part (b). For each recursive call, state pure symbols and unit clauses. State the solution returned (and explain which selection of modules it corresponds to).
Assume the order of symbols: J, C, H, A, M. (10 marks)

Answer. Input to the algorithm:
clauses = $\{[J, C], [H, A], [M], [\neg J, \neg C], [\neg H, \neg A], [\neg C, A], [\neg A, C]\}$
symbols = J, C, H, A, M.

First recursive call: there are no pure symbols. There is a unit clause $[M]$. We assign $true$ to M and call the algorithm with
model $M = true$
clauses $\{[J, C], [H, A], [\neg J, \neg C], [\neg H, \neg A], [\neg C, A], [\neg A, C]\}$
symbols = J, C, H, A.

Second recursive call: there are no pure symbols and no unit clauses. We call the algorithm with
model $M = true$, $J = true$
clauses $\{[H, A], [\neg C], [\neg H, \neg A], [\neg C, A], [\neg A, C]\}$
symbols = C, H, A.

Third recursive call: there are no pure symbols. There is a unit clause $[\neg C]$. We assign $false$ to C and call the algorithm with
model $M = true$, $J = true$, $C = false$
clauses $\{[H, A], [\neg H, \neg A], [\neg A]\}$
symbols = H, A.

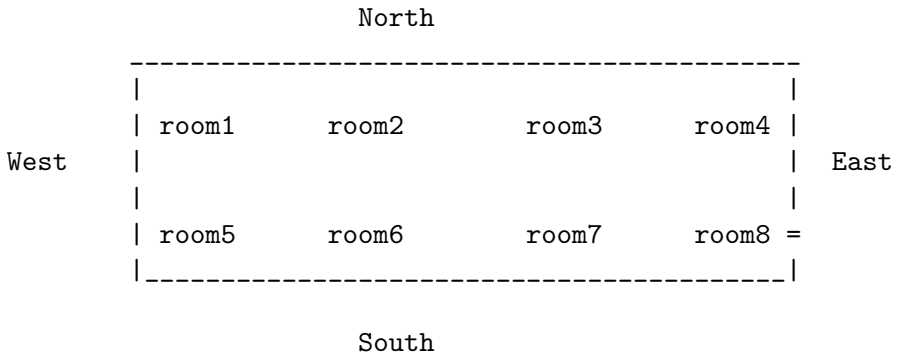
Fourth recursive call: there are no pure symbols. There is a unit clause $[\neg A]$. We assign $false$ to A and call the algorithm with
model $M = true$, $J = true$, $C = false$, $A = false$
clauses $\{[H]\}$
symbols = H.

Fifth recursive call: we assign $true$ to H and call the algorithm with the empty list of clauses and symbols. It returns true and we return the model $M = true$, $J = true$, $C = false$, $A = false$, $H = true$.

This means that the set of modules is M (G51MAT), J (G51JAV) and H (G51HAR).

3. This question is on search with non-deterministic actions and partial observability.

Consider an agent which is trying to find an exit from this 'labyrinth':



The physical states for this problem are $room_1$, ..., $room_8$ (meaning: the agent is in $room_1$, ..., in $room_8$).

The agent cannot see where it is, but it can feel the walls. Its percepts are subsets of $\{WN, WW, WE, WS, Ex\}$ which stand for 'Wall to the North', 'Wall to the West', 'Wall to the East', 'Wall to the South', and 'Exit'.

$Percept(room_1) = \{WN, WW\}$
$Percept(room_2) = Percept(room3) = \{WN\}$
$Percept(room_4) = \{WN, WE\}$
$Percept(room_5) = \{WS, WW\}$
$Percept(room_6) = Percept(room_7) = \{WS\}$
$Percept(room_8) = \{WS, WE, Ex\}$.

The actions available to the agent are moving North, West, East and South. The actions of moving North and South are deterministic and move the agent one room to the North or one room to the South if there exists such a room, otherwise leave the agent where it is. The actions of moving West and East are non-deterministic: they move the agent to some room to the West or to the East, not necessarily to the next one.

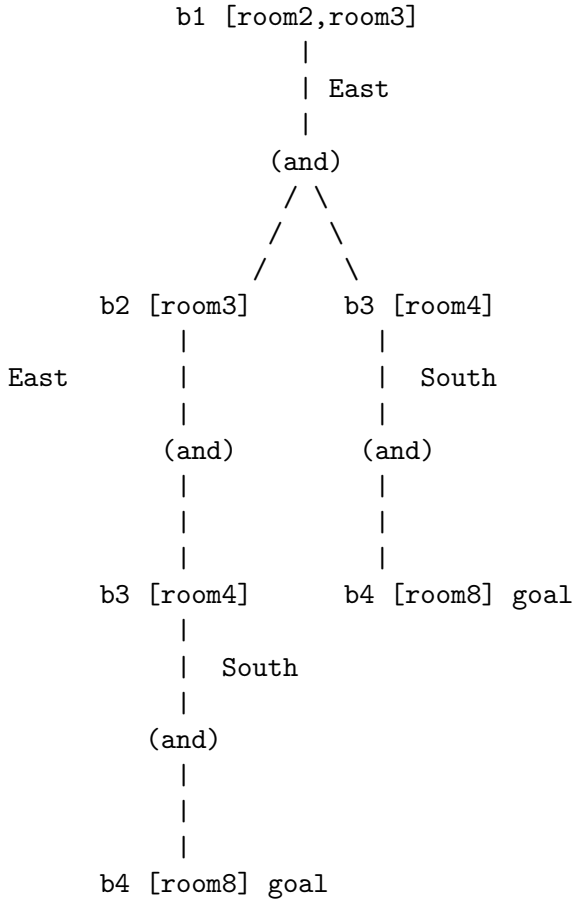
$Results(room_i, North) = \{room_i\}$ for $i \in \{1, 2, 3, 4\}$
$Results(room_i, North) = \{room_{i-4}\}$ for $i \in \{5, 6, 7, 8\}$
$Results(room_i, South) = \{room_{i+4}\}$ for $i \in \{1, 2, 3, 4\}$
$Results(room_i, South) = \{room_i\}$ for $i \in \{5, 6, 7, 8\}$
$Results(room_i, West) = \{room_i\}$ for $i \in \{1, 5\}$
$Results(room_i, West) = \{room_{i-1}$ for $i \in \{2, 6\}$
$Results(room_i, West) = \{room_{i-1}, room_{i-2}\}$ for $i \in \{3, 7\}$
$Results(room_i, West) = \{room_{i-1}, room_{i-2}, room_{i-3}\}$ for $i \in \{4, 8\}$
$Results(room_i, East) = \{room_i\}$ for $i \in \{4, 8\}$
$Results(room_i, East) = \{room_{i+1}$ for $i \in \{3, 7\}$
$Results(room_i, East) = \{room_{i+1}, room_{i+2}\}$ for $i \in \{2, 6\}$
$Results(room_i, East) = \{room_{i+1}, room_{i+2}, room_{i+3}\}$ for $i \in \{1, 5\}$

(a) The agent starts in total ignorance of where it is. Define its initial belief state b_0 (list the physical states it considers possible in b_0). (2 marks)
Answer. $b_0 = \{$room1, room2, room3, room4, room5, room6, room7, room8$\}$.

(b) The agent receives a percept $o_1 = \{WN\}$. Define its updated belief state $b_1 = Update(b_0, o_1)$. (2 marks)
Answer. $b_1 = Update(b_0, o_1) = \{s \in b_0 : Percept(s) = o_1\} = \{$room2,room3$\}$.

(c) Define $Results(b_1, East)$. (6 marks)
Answer. $\hat{b_1} = Predict(b_1, East) = \{$room3,room4$\}$ (results of executing East in room2 and room3, respectively).
Possible-Percepts$(\hat{b_1} = \{o2 = Percept(room3) = \{WN\}, o3 = Percept(room4) = \{WN, WE\}\}$.
$b_2 = Update(\hat{b_1}, o2) = \{s \in \hat{b_1} : o2 = Percept(s)\} = \{room3\}$
$b_3 = Update(\hat{b_1}, o3) = \{s \in \hat{b_1} : o3 = Percept(s)\} = \{room4\}$
$Results(b1, East) = \{b_2, b_3\} = \{\{room3\}, \{room4\}\}$

(d) Draw a solution subtree of the AND-OR search tree for this problem, starting in belief state b_1 from part (b), with the goal to be in $room_8$. (15 marks)
Answer.



Note that other solutions are possible, for example going South first and then East.

4. This question is on planning in general and on goal stack planning.

   (a) Define the following planning problem in Planning Domain Description language (specify predicates, objects, initial state, goal specification, action schemas). (10 marks)

   > There are two locations, $L1$ and $L2$, and two containers, A and B, and a fork-lift truck. If the truck is in the same location as a container and the truck is empty, then the truck can pick up the container; as a result of this action, the container is on the truck and the truck is no longer empty. If the truck is in location x, it can drive to location y; as a result of this action, the truck is in location y. If the truck has a container on it, and is in location x, then it can unload and the effect of the action is that the container is in location x and the truck is empty. In the initial state, both containers and the truck are in $L1$ and the truck is empty. The goal is to have container B in $L2$.

   Answer.

   Predicates: In, $Empty$, On

   Objects: $L1, L2, A, B, T$

   Initial state: $In(L1, A) \wedge In(L1, B) \wedge In(L1, T) \wedge Empty$

   Goal specification: $In(L2, B)$.

   Action schemas:

   $Pickup(T, x)$
   Precondition: $In(y, T), In(y, x), Empty$
   Effect: $On(T, x), \neg Empty, \neg In(y, x)$

   $Drive(T, x, y)$
   Precondition: $In(T, y)$
   Effect: $In(T, x), \neg In(T, y)$

   $Offload(T, x, y)$
   Precondition: $In(T, y), On(T, x)$
   Effect: $In(y, x), \neg On(T, x), Empty$

   (b) Trace the goal stack planning algorithm for this problem. At each step, show what the stack contains, what is the current plan, and the state of the knowledge base if it changes. (15 marks)

   Answer. (Note that different traces are possible if the order of subgoals is different.)

   Step 1. The knowledge base is: $\{In(L1, A), In(L1, B), In(L1, T), Empty\}$ The plan is empty.

   Stack is:

   ```
   In (L2,B)
   ```

   The top is an unsatisfied goal, so we push an action which would achieve it and its preconditions.

   Step 2.

   ```
   In(L2,T) /\ On(T,B)
   Offload(T,B,L2)
   In(L2,B)
   ```

The top is a compound goal, so we split it and push subgoals on the stack.
Step 3.

```
On(T,B)
In(L2,T)
In(L2,T) /\ On(T,B)
Offload(T,B,L2)
In(L2,B)
```

The top is an unsatisfied goal, so we push an action which will achieve it and its preconditions.
Step 4.

```
In(y,T) /\ In(y,B) /\ Empty
Pickup(T,B)
On(T,B)
In(L2,T)
In(L2,T) /\ On(T,B)
Offload(T,B,L2)
In(L2,B)
```

The top is a compound goal, so we split it and push subgoals on the stack.
Step 5.

```
Empty
In(y,B)
In(y,T)
In(y,T) /\ In(y,B) /\ Empty
Pickup(T,B)
On(T,B)
In(L2,T)
In(L2,T) /\ On(T,B)
Offload(T,B,L2)
In(L2,B)
```

With substitution of $L1$ for y, the top four goals on the stack are satisfied in current KB=$\{In(L1, A), In(L1, B), In(L1, T), Empty\}$. We pop the stack.
Step 6.

```
Pickup(T,B)
On(T,B)
In(L2, T)
In(L2,T) /\ On(T,B)
Offload(T,B,L2)
In(L2,B)
```

The top of the stack is an action, so we pop the stack, execute the action, add it to the plan, and update the knowledge base with the effect.
Step 7.

```
On(T,B)
In(L2, T)
In(L2,T) /\ On(T,B)
Offload(T,B,L2)
In(L2,B)
```

KB=$\{In(L1, A), In(L1, T), On(T, B)\}$.
The plan is $(Pickup(T, B))$.
The top of the stack is a satisfied goal, so we pop the stack.
Step 8.

```
In(L2, T)
In(L2,T) /\ On(T,B)
Offload(T,B,L2)
In(L2,B)
```

Top of the stack is an unsatisfied goal, so we push an action which will achieve it, and its preconditions.
Step 9.

```
In(x,T)
Drive(T,x,L2)
In(L2, T)
In(L2,T) /\ On(T,B)
Offload(T,B,L2)
In(L2,B)
```

The top of the stack is a satisfied goal with substitution of L1 for x. We pop it.
Step 10.

```
Drive(T,L1,L2)
In(L2, T)
In(L2,T) /\ On(T,B)
Offload(T,B,L2)
In(L2,B)
```

We pop and execute the action and update the knowledge base and the plan.
Step 11.

```
In(L2, T)
In(L2,T) /\ On(T,B)
Offload(T,B,L2)
In(L2,B)
```

KB=$\{In(L1, A), In(L2, T), On(T, B)\}$.
The plan is $(Pickup(T, B), Drive(T, L1, L2))$.
The two goals on top of the stack are satisfied, so we pop them.
Step 12.

```
Offload(T,B,L2)
In(L2,B)
```

Top of the stack is an action, we pop and execute it and update the knowledge base and the plan.

Step 13.

```
In(L2,B)
```

KB=$\{In(L1, A), In(L2, B), In(L2, T), Empty\}$.

The plan is $(Pickup(T, B), Drive(T, L1, L2), Offload(T, B, L2))$.

Top of the stack is a satisfied goal, we pop it.

Step 14. The stack is empty: stop.

5. This question is on partial order planning.

   (a) Explain how partial order planning works (in English or pseudocode). Your answer should mention the notions of open conditions, causal links, clobberers, and temporal constraints. (10 marks)

   Answer. Partial order planning searches within a space of partial plans. Partial plans always have a *Start* step which has the initial state description as its effect, and a *Finish* step which has the goal description as its precondition. They also have a list of temporal constraints $a_i \prec a_j$ stating that action a_i should precede action a_j. The algorithm also uses a list of causal links from outcome of one step to precondition of another. An open condition is a precondition of a step not yet causally linked. A plan is complete iff every precondition is achieved. A precondition is achieved iff it is the effect of an earlier step and no possibly intervening step undoes it. A clobberer is a potentially intervening step that destroys the condition achieved by a causal link. It should be either moved before the first step or after the second.

   Operators on partial plans:

   - add a link from an existing action to an open condition
   - add a step to fulfill an open condition
   - order one step wrt another to remove possible conflicts

   Gradually move from incomplete/vague plans to complete, correct plans. Backtrack if an open condition is unachievable or if a conflict is unresolvable.

function POP(*initial, goal, actions*) **returns** *plan*

plan ← MAKE-MINIMAL-PLAN(*initial, goal*)
loop do
  if SOLUTION?(*plan*) **then return** *plan*
  $S_{need},\ c \leftarrow$ SELECT-SUBGOAL(*plan*)
  CHOOSE-ACTION(*plan, actions*, S_{need}, c)
  RESOLVE-THREATS(*plan*)
end

---

function SELECT-SUBGOAL(*plan*) **returns** $S_{need},\ c$

pick a plan step S_{need} from STEPS(*plan*)
  with a precondition c that has not been achieved
return $S_{need},\ c$

procedure CHOOSE-ACTION(*plan, actions*, S_{need}, c)

choose a step S_{add} from *actions* or STEPS(*plan*) that has c as an effect
if there is no such step **then fail**
add the causal link $S_{add} \xrightarrow{c} S_{need}$ to LINKS(*plan*)
add the ordering constraint $S_{add} \prec S_{need}$ to ORDERINGS(*plan*)
if S_{add} is a newly added step from *actions* **then**
  add S_{add} to STEPS(*plan*)
  add $Start \prec S_{add} \prec Finish$ to ORDERINGS(*plan*)

---

procedure RESOLVE-THREATS(*plan*)

for each S_{threat} that threatens a link $S_i \xrightarrow{c} S_j$ in LINKS(*plan*) **do**
  choose either
    Demotion: Add $S_{threat} \prec S_i$ to ORDERINGS(*plan*)
    Promotion: Add $S_j \prec S_{threat}$ to ORDERINGS(*plan*)
  if not CONSISTENT(*plan*) **then fail**
end

(b) Consider the following planning problem. It involves the agent leaving the house and locking the door. The predicates are: *Locked* (meaning, the door is locked) and *Out* (meaning, the agent is outside). The actions are:

Lock:
  PRECOND: *Out*
  EFFECT: *Locked*

Leave:
  PRECOND: $\neg Out \wedge \neg Locked$
  EFFECT: *Out*

The initial state is {} (none of the predicates is true). The goal is $Out \wedge Locked$.

Solve this problem using partial order planning; trace the search from the initial empty plan to a complete solution, explaining each step. (15 marks)

Answer.

$Steps = \{Start, Finish\}$ where the effect of $Start$ is {}, and the $Finish$ step has $Out \wedge Locked$ as its precondition.
$Links = \{\ \}$
$Orderings = \{Start \prec Finish\}$
Open conditions: $Out, Locked$ (for the $Finish$ step).

Suppose we choose the first open condition as a selected subgoal.

An action which would make it true is $Leave$. We add it as a step to the partial plan, and a link from it to $Finish$:

$Steps = \{Start, Leave, Finish\}$
$Links = \{\ Leave \xrightarrow{Out} Finish\}$
$Orderings = \{Start \prec Finish, Start \prec Leave, Leave \prec Finish\}$

Open conditions: $Locked$ (for the $Finish$ step), $\neg Out$ and $\neg Locked$ (for the $Leave$ step).

There are no clobberers.

Suppose we choose the first open condition as a selected subgoal.

An action which would make it true is $Lock$. We add it as a step to the partial plan, and a link from it to $Finish$:

$Steps = \{Start, Leave, Lock, Finish\}$
$Links = \{\ Leave \xrightarrow{Out} Finish, Lock \xrightarrow{Locked} Finish\}$
$Orderings = \{Start \prec Finish, Start \prec Leave, Leave \prec Finish, Start \prec Lock, Lock \prec Finish\}$
Open conditions: $\neg Out$, $\neg Locked$ (for the $Leave$ step).

There is also clobbering: $Lock$ destroys a precondition for $Leave$. Hence we put $Leave$ before $Lock$:

$Orderings = \{Start \prec Finish, Start \prec Leave, Leave \prec Lock, Lock \prec Finish\}$

Now we can deal with the open conditions by adding a link from $Start$ to $\neg Out$ and $\neg Locked$.

The resulting complete plan is as follows:

$Steps = \{Start, Leave, Lock, Finish\}$
$Links = \{\ Leave \xrightarrow{Out} Finish, Lock \xrightarrow{Locked} Finish, Start \xrightarrow{\neg Out} Leave, Start \xrightarrow{\neg Locked} Leave\}$
$Orderings = \{Start \prec Finish, Start \prec Leave, Leave \prec Lock, Lock \prec Finish\}$

6. This question is on HTN planning.

   Consider a blocks world which consists of a Table and 3 blocks A, B, and C. The predicates are On where $On(x, y)$ means that x is on y, $Clear$, where $Clear(x)$ means that there is nothing on top of x. The actions are:

$Move(b, x, y)$:
PRECOND: $On(b, x) \wedge Clear(b) \wedge Clear(y)$
EFFECT: $On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y)$

$Movetotable(b, x)$:
PRECOND: $On(b, x) \wedge Clear(b)$
EFFECT: $On(b, Table) \wedge Clear(x) \ \wedge \neg On(b, x)$

(a) Add an abstract action $Tower$ which corresponds to building a tower of 3 blocks (A on top of B, B on top of C, C on the Table), and one refinement for it with the precondition (5 marks)

$Clear(A) \wedge Clear(B) \wedge Clear(C) \wedge On(A, Table) \wedge On(B, Table) \wedge On(C, Table)$

Answer.

$Refinement(Tower,$
PRECOND: $Clear(A) \wedge Clear(B) \wedge Clear(C) \wedge On(A, Table) \wedge$
$On(B, Table) \wedge On(C, Table)$
STEPS: $[Move(B, Table, C), Move(A, Table, B)])$

(b) Add an abstract action $Prepare(x)$ which results in the block x being clear and on the table. Write all possible refinements for it (for each possible precondition case: when x is clear, when there is one block on top of x, when there are two blocks on top of x ...) (10 marks)

Answer.

$Refinement(Prepare(x),$
PRECOND: $Clear(x) \wedge On(x, Table)$
STEPS: $[])$

$Refinement(Prepare(x),$
PRECOND: $Clear(x) \wedge On(x, y)$
STEPS: $[Movetotable(x, y)])$

$Refinement(Prepare(x),$
PRECOND: $On(x, Table) \wedge On(y, x) \wedge Clear(y)$
STEPS: $[Movetotable(y, x)])$

$Refinement(Prepare(x),$
PRECOND: $On(x, z) \wedge On(y, x) \wedge Clear(y)$
STEPS: $[Movetotable(y, x), Movetotable(x, z)])$

$Refinement(Prepare(x),$
PRECOND: $On(x, Table) \wedge On(y, x) \wedge On(z, y) \wedge Clear(z)$
STEPS: $[Movetotable(z, y), Movetotable(y, x)])$

(Note that there are only 3 blocks, so this takes care of all possibilities.)

(c) Trace hierarchical search algorithm on the following example: the planning domain is as given above plus your refinements for $Tower$ and $Prepare(x)$. In the initial state, $On(A, Table) \wedge Clear(A) \wedge On(B, Table) \wedge On(C, B) \wedge Clear(C)$. The goal is to have a tower $On(C, Table) \wedge On(B, C) \wedge On(A, B)$. Assume that the only refinement of Act is $[Prepare(A), Prepare(B), Prepare(C), Tower]$. Your answer should mention what is in the frontier at each step, what plan is expanded, what is the first abstract action in that plan, and how the content of the frontier for the next step is calculated. (10 marks)

Answer. The answer assumes variable names from Russell and Norvig's algorithm listing for hierarchical search given below. The students do not need to repeat the listing to get full marks.

```
function HIERARCHICAL-SEARCH(problem, HTN) returns a solution,
or failure
    frontier ← a FIFO queue with [Act] as the only element
    loop do
        if EMPTY?(frontier) then return failure
        plan ← POP(frontier) /* chooses the shallowest plan in frontier
*/
        hla ← the first HLA in plan, or null if none
        prefix, suffix ← the action subsequences before and after hla in
plan
        outcome ← RESULT(problem.INITIAL-STATE, prefix)
        if hla is null then /* so plan is primitive and outcome is its
result */
            if outcome satisfies problem.GOAL then return plan
        else if for each sequence in REFINEMENTS(hla, outcome,
HTN) do
            frontier ← INSERT(APPEND(prefix, sequence, suffix),frontier)
```

At first, the frontier contains only the plan $[Act]$. hla (the first abstract action in the plan) is assigned Act. Prefix and suffix are empty. $outcome$ is the initial state. We insert $[Prepare(A), Prepare(B), Prepare(C), Tower]$ in the frontier.

At the next iteration, hla is $Prepare(A)$, prefix is empty, suffix is $[Prepare(B), Prepare(C), Tower]$. $outcome$ is the initial state. The only refinement of $Prepare(A)$ available in the initial state is [], so we replace $Prepare(A)$ with an empty sequence. The frontier becomes $\{[Prepare(B), Prepare(C), Tower]\}$.

At the next iteration, hla is $Prepare(B)$, prefix is empty, suffix is $[Prepare(C), Tower]$. $outcome$ is still the initial state. The only refinement of $Prepare(B)$ applicable in the initial state is $[Movetotable(C, B)]$, so we replace $Prepare(B)$ by $Movetotable(C, B)$. The frontier becomes $\{[Movetotable(C, B), Prepare(C), Tower]\}$.

At the next iteration, hla is $Prepare(C)$, prefix is $Movetotable(C, B)$, suffix is $\{[Prepare(C), Tower]\}$. The $outcome$ is the result of executing $Movetotable(C, B)$ in the initial state, that is, now A is on the table and clear, B is on the table and clear, and C is on the table and clear. The only refinement of $Prepare(C)$ available

in this state is [], so we replace $Prepare(C)$ with an empty sequence. The frontier becomes $\{[Movetotable(C, B), Tower]\}$.

At the next iteration, *hla* is $Tower$, prefix is $Movetotable(C, B)$, suffix is empty. The *outcome* is the result of executing $Movetotable(C, B)$ in the initial state, that is, now A is on the table and clear, B is on the table and clear, and C is on the table and clear. The only refinement of $Tower$ available in this state is $[Move(B, Table, C), Move(A, Table, B)]$, so we replace $Tower$ with $[Move(B, Table, C), Move(A, Table, B)]$. The frontier becomes $\{[Movetotable(C, B), Move(B, Table, C), Move(A, Table, B)]\}$. This sequence achieves the goal.