1. STATE SPACE SEARCH

(x4)1.1 Using pseudocode or a programming language of your choice, define the general algorithm for tree search.

```
struct node = [currentState: a state, i.e., a data structure representing an element of States;
pastAction: an action, i.e., a symbol identifying an element of Actions;
parentNode: a node, i.e., a reference to an instance of this structure;
totalCost: a number];
```

```
function isEmpty(list: a list of elements of type T)
returns (a Boolean) {
        if (list has no element)
            true
        else false;
}
```

```
function insertFirst(x: an element of type T; list: a list of elements of type T)
returns (a list of elements of type T) {
        add x to list as the first element;
        return list;
}
```

```
function insertLast(x: an element of type T; list: a list of elements of type T)
returns (a list of elements of type T) {
        add x to list as the last element;
        return list;
}
```

```
function insertList(list1, list2: two lists of elements of type T)
returns (a list of elements of type T) {
        add all the nodes of list1 to list2 according to a policy to be specified;
        return list2;
}
```

```
function removeFirst(list: a list of elements of type T)
returns (an element of type T) {
        vars x: an element of type T;
        x = the first element of list;
        remove the first element of list;
        return x;
}
```

```
function treeSearch(initialState: a state; Actions: a list of actions; GoalStates: a set of states)
returns (a list of actions or 'failure') {
        vars {
            n: a node;
            frontier: a list of nodes (initially empty);
```

```
        }
        n = new node(initialState,null,null,0);
        insertFirst(n,frontier);
        loop {
            if isEmpty(frontier) return 'failure';
            n = removeFirst(frontier);
            if (n.currentState in GoalStates) return buildSolution(n);
            insertList(expandNode(n,Actions),frontier);
        }
}
```

```
function buildSolution(n: a node) returns a list of actions {
        vars { a: an action;
            solution: a list of actions (initially empty);
        }
        while (n != null) {
            insertFirst(n.pastAction,solution);
            n = n.parentNode;
        }
        return solution;
}
```

```
function expandNode(n: a node; Actions: a list of actions) returns a list of nodes {
        vars { newNode: a node;
            a: an action;
            successorList: a list of nodes (initially empty);
        }
        for (a in Actions) {
            if applicable(a,n.currentState) {
                newNode = new node(result(a,n.currentState), a, n, n.totalCost+cost(a,n.state));
                insertLast(newNode,successorList);
            }
        }
        return successorList;
}
```

Then explain how the algorithm must be adapted to generate the following search strategies: depth first, breadth first, uniform cost, A*.

Depth First: genero l'albero espandendo per primo il nodo generato più di recente. Per far ciò implemento la funzione insertList con una politica LIFO, espando il nodo che è all'inizio della frontiera e inserisco il successivo nodo all'inizio della frontiera, la frontiera è implementata come uno STACK.

Breadth First: genero l'albero espandendo tutti i nodi di un livello, prima di passare al livello dopo. Per far ciò implemento la funzione insertList con una politica FIFO, espando il nodo che è all'inizio della frontiera e inserisco il successivo nodo alla fine della frontiera, la frontiera è implementata come una QUEQUE (CODA).

Uniform Cost: genero l'albero come nella Breadth First, ma non in base al livello del nodo, ma al costo totale per raggiungere quel nodo. La frontiera è implementata come una PRIORITY QUEQUE (CODA CON PRIORITA') ordinata in ordine crescente in base al valore di costoTotale.

A*: è una variante della strategia Uniform Cost, dove i nodi nella frontiera sono ordinati con una funzione (funzione euristica) che da una stima del costo totale della soluzione ottima dalla radice all'obiettivo passando dal nodo n.

(x2)1.15 Explain what it means for a search strategy to be optimal. Then, again with respect to Tree Search, specify the conditions under which the A* strategy is optimal.
Una strategia si dice ottima se se solo se ogni volta che trova una soluzione a un problema, questa soluzione è ottima, cioè non è possibile trovare un'altra soluzione che abbia costo minore di questa.
In A* una strategia è ottima se:

- Tutti i nodi dell'albero di ricerca hanno un numero finito di successori
- Il costo delle azioni ha un limite inferiore positivo (lower bound)
- La funzione euristica usata è ammissibile, cioè la stima del costo totale dato dalla funzione euristica deve essere un limite inferiore del costo totale della soluzione ottima attuale.

!!!!!!!!(x2)1.2 Using pseudocode or a programming language of your choice, define the A* algorithm for <u>graph</u> search. Specify under what conditions the first solution found by this algorithm is optimal.

(x1)1.5 Rigorously define the following concepts: (i) state space, (ii) problem, (iii) solution.

- Lo spazio degli stati è una tripletta S = (S, A, e) dove S è un insieme di stati non vuoto, A è un insieme di azioni e la funzione di esecuzione e è una funzione parziale che assegna uno stato risultante e un costo a un insieme di coppie stato-azione.
- Un problema P è una tripletta P = (S, s_0, G) dove S = (S, A, e) è uno spazio degli stati, $s_0 \in S$ è lo stato iniziale e $G \subseteq S$ è l'insieme degli stati obiettivo.
- Dato un problema P = (S, s_0, G), una soluzione è una sequenza finita di azioni (a_1,…,a_n) che per $1 \leq k \leq n$, a_k può essere eseguito nello stato s_{k-1} e il risultato nello stato s_k con $s_n \in G$.

(x1)1.6 Define what a search strategy is, and classify the most important search strategies according to relevant aspects.
Una strategia di ricerca è un metodo per la costruzione di una soluzione, dato un problema.

	Uninformed	Informed
Parallel	breadth-first uniform cost	A*
Sequential	depth-first backtracking iterative deepening	greedy best-first

(x1)1.7 Briefly describe (in words) the main ideas underlying the Iterative Deepening strategy, and comment about the effectiveness of such a strategy.
La strategia ID è basata sulla ripetizione iterativa di una ricerca in profondità limitata. Il limite è inizialmente 0 e viene incrementato di 1 a ogni iterazione. La complessità della strategia ID è lineare in memoria e esponenziale nel tempo.

!!!!!!!!(x3)1.8 Explain the difference between tree search and graph search, clarifying:

- **why and when we need graph search**
- **what improvements are required by the tree search algorithm to implement graph search (you are not asked to describe the whole algorithm, only the relevant modifications)**

Now consider the 8-puzzle: does it require graph search? if so, why?

Negli alberi di ricerca non controllo se lo stesso stato è rappresentato da più nodi, invece nei grafi di ricerca effettuo questo controllo.

(x1)1.18 Explain the difference between tree search and graph search, and clarify the advantages of the latter over the former. Then explain how the difference between tree search and graph search is implemented in the basic search algorithm. (You are not required to report the whole search algorithm, only to explain the main difference between the tree-search and the graph-search version.)

Negli alberi di ricerca non controllo se lo stesso stato è rappresentato da più nodi, invece nei grafi di ricerca effettuo questo controllo.

Questo implica che i grafi di ricerca occuperanno più memoria, poiché bisogna salvare anche tutti gli stati incontrati e espansi, ma eviteranno i cicli infiniti, in quanto è impossibile ripassare da un nodo che rappresenta uno stato già espanso.

```
function treeSearch(initialState: a state; Actions: a list of actions; GoalStates: a set of states)
returns (a list of actions or 'failure') {
        vars {
            n: a node;
            frontier: a list of nodes (initially empty);
        }
        n = new node(initialState,null,null,0);
        insertFirst(n,frontier);
        loop {
            if isEmpty(frontier) return 'failure';
            n = removeFirst(frontier);
            if (n.currentState in GoalStates) return buildSolution(n);
            insertList(expandNode(n,Actions),frontier);
        }
}
```

```
function expandNode(n: a node; Actions: a list of actions) returns a list of nodes {
        vars { newNode: a node;
            a: an action;
            successorList: a list of nodes (initially empty);
        }
        for (a in Actions) {
            if applicable(a,n.currentState) {
                newNode = new node(result(a,n.currentState), a, n, n.totalCost+cost(a,n.state));
                insertLast(newNode,successorList);
            }
       }
        return successorList;
}
```

```
function graphSearch(initialState: a state; Actions: a list of actions; GoalStates: a set of states)
returns (a list of actions or 'failure') {
        vars {
            n: a node;
            frontier: a list of nodes (initially empty);
            explored: a set of states (initially empty);
        }
        n = new node(initialState,null,null,0);
        insertFirst(n,frontier);
        loop {
            if isEmpty(frontier) return 'failure';
            n = removeFirst(frontier);
            addToSet(n.state,explored);
```

```
            if (n.currentState in GoalStates) return buildSolution(n);
            insertList(graphExpandNode(n,Actions,explored),frontier);
        }
}
```

```
function graphExpandNode(n: a node; Actions: a list of actions; explored: a set of states)
returns a list of nodes {
        vars { newNode: a node;
            a: an action;
            successorList: a list of nodes (initially empty);
        }
        for (a in Actions){
            if applicable(a,n.currentState) {
                newNode = new node(result(a,n.currentState),a,n,n.totalCost+cost(a,n.state));
                if !(newNode.state in explored)
                    insertLast(newNode,successorList);
            }
        }
        return successorList;
}
```

(x2)1.9 Suggest a state space representation for the problem stated below: define a data structure for representing the states and specify the allowable actions (you are not required to find a solution).

(x1) Boys and girls.

To go from A to B, three boys and three girls wants to use a motor bike (which is initially at A). The motor bike can carry a maximum of two people, therefore several trips will be necessary. Since boys are notoriously untrustable, in no case should the number of boys present at A or B be larger than the number of girls present at the same place.

(x1) An old Russian puzzle. While travelling with a wolf, a goat and a (big) cabbage, a farmer has to cross a river from side A to side B. Initially, a small boat is available at side A. The boat can carry at most one object plus the farmer, who obviously is the only one able to row. The farmer has to find a way of bringing the wolf, the goat and the cabbage from A to B, with the following constraints: at no stage of the solution should the wolf be left with the goat in absence of the farmer (because the wolf would eat the goat); analogously, at no stage of the solution should the goat be left with the cabbage in absence of the farmer (because the goat would eat the cabbage).

(x2)1.10 Compare the Breadth First and Depth First search strategies. In particular:

- **say how the general tree-search algorithm can be adapted to implement the two strategies (you are not required to describe the full algorithm, only the part that implements the strategies);**
- Depth First: genero l'albero espandendo per primo il nodo generato più di recente. Per far ciò implemento la funzione insertList con una politica LIFO, espando il nodo che è all'inizio della frontiera e inserisco il successivo nodo all'inizio della frontiera, la frontiera è implementata come uno STACK.
- Breadth First: genero l'albero espandendo tutti i nodi di un livello, prima di passare al livello dopo. Per far ciò implemento la funzione insertList con una politica FIFO, espando il nodo che è all'inizio della frontiera e inserisco il successivo nodo alla fine della frontiera, la frontiera è implementata come una QUEQUE (CODA).
- **compare the two strategies from the point of view of completeness, optimality and complexity (in both memory and time);**

	Completezza	Ottimalità	Compl. Temporale	Compl. Spaziale
Breadth First	Sì	Sì (se costi uguali)	$O(b^{d+2})$ c.u: $O(b^{d+1})$	$O(b^{d+2})$ c.u: $O(b^{d+1})$

Depth First No (sì con grafo) No (sì, costi unitari) $O(b^{m+1})$ O(bm)

- **describe a meaningful example of a class of problems that can be safely solved by Depth First strategies.**

(x1)1.11 Explain the difference between uninformed and informed search at a conceptual level.
Nelle strategie non informate (uninformed) conosciamo solo la descrizione del problema, mentre in quelle informate (informed) abbiamo accesso anche ad altre informazioni (ad esempio una funzione euristica)
Then compare the Uniform Cost and A* strategies, highlighting the differences in the corresponding procedures (you do not need to describe the whole procedures, only the differences).
Costo Uniforme (Uniform cost): costi uguali, frontiera come coda di priorità in ordine crescente di costo totale.
A*: costi non uguali, frontiera come coda di priorità, in ordine crescente di costi, ma il costo è dato da una funzione euristica data una stima del costo di una soluzione ottima completa passante per il nodo in questione.
Finally clarify whether the solutions to state space problems built by informed search are better than the solutions build by uninformed search.
A* genera un numero minore di nodi rispetto a UC, poichè abbassa il Branching factor medio, quindi occupa una minore memoria e impiega meno tempo a trovare la soluzione. Dipende tutto da quanto è accurata la funzione euristica, comunque.

(x1)1.12 Considering the A* informed strategy, specify and compare the optimality conditions in the case of tree search and graph search.
Per prima cosa si devono rispettare le condizioni di ottimalità di UC:

- ogni nodo ha un numero finito di successori
- esiste un limite inferiore nei costi che è positivo e maggiore di zero.

Inoltre, nel caso degli alberi di ricerca la funzione euristica deve essere ammissibile, cioè deve dare un valore inferiore al vero valore di costo per il cammino dal nodo n al nodo goal
Nel caso dei grafi di ricerca dobbiamo anche assicurarci che la funzione euristica sia consistente (monotona).
Then explain in details:

- **what changes are needed to turn the algorithm into a general algorithm for graph search;**
- **how the algorithm is adapted to obtain breadth-first search, depth-first search, and uniform-cost search.**
-

(x1)1.17 Suggest a state space representation for the problem stated below: define a data structure for representing the states, specify the allowable actions, represent the initial state and the goal, and describe a solution. Finally say what search strategy you would choose.

> **There are a lake (of unlimited size), two cans (of 4 and 7 litres, respectively), and a container (of 12 litres). You are allowed to fill a can with water from the lake or from the container, and to pour the content of a can into the container or into the other can or even back into the lake.**
> **The goal is to fill the container with exactly 6 litres of water from the river. The cans and the container are initially empty.**

2. CFP

(x5)2.1 Define the concept of a Constraint Satisfaction Problem (CFP); in particular, explain the meaning of the following terms: variable, domain, value, assignment, constraint, consistent assignment, solution.
I CSP sono una strategia basata su una euristica indipendente dal dominio del problema, ciò consente di definire l'euristica a priori senza conoscere il problema.

- un set finito $X = \{x_1,\ldots, x_n\}$ di variabili
- Il dominio è l'insieme D_i di possibili valori attribuibili alla variabile x_i
- Un valore è un possibile assegnamento della variabile x
- Un assegnamento è un insieme funzionale di coppie ordinate $<x_i,v_i>$ chiamate vincoli
- Un vincolo è un possibile e ammesso assegnamento di un valore a una variabile
- Un assegnamento consistente è un assegnamento che soddisfa tutti i vincoli
- Una soluzione è un assegnamento completo e consistente che estende un assegnamento iniziale

(x4)2.2 Define a map-colouring problem on a map of your choice (with at least 5 countries) as a CFP, specifying all variables, domains, and constraints. Then choose a search strategy for the solution of the problem and justify your choice. Specify and concisely define the domain-independent heuristics that you would adopt to speed up the search.

(x1)2.3 Define a simplified version of the SuDoku puzzle with four rows, four columns, and four 2×2 boxes, in which the digits 1, 2, 3, and 4 have to be arranged so that no digit is repeated in any row, column, or box. Note that in a typical SuDoku puzzle, the initial assignment is not empty, for example:

1 2	
3	4

Define such a puzzle as a CSP, clearly specifying all variables, domains, and constraints. Then choose a search strategy for the solution of the problem and justify your choice. Finally specify and concisely describe the domain-independent heuristics that you would adopt to speed up the search process. (Note that you are *not* required to describe the solution of a concrete example).

(x1)2.4 Precisely and completely define 2-Sudoku (a version of Sudoku on a 4×4 array, using figures 1 to 4) as a CSP, specifying all variables, domains, and constraints. Then choose a search strategy for the solution of the problem and justify your choice. Finally identify and describe the domain-independent heuristics that you would adopt to speed up the search process.

(x1)2.5 Explain why the Depth First search strategy on a CSP is both complete and optimal.
Se usiamo un albero di ricerca per risolvere un CSP ci accorgiamo che tale albero ha della proprietà particolari:

- Se ci sono n variabili e lo stato iniziale contiene k coppie variabile-valore, allora tutti i nodi che rappresentano una soluzione si troveranno a una profondità n-k
- Inoltre ogni nodo a profondità n-k rappresenterà una soluzione poiché ogni nodo dell'albero rappresenta una estensione consistente dello stato iniziale ed è anche un assegnamento completo.

Per queste ragioni la strategia di ricerca DF è completa: l'albero di ricerca è finito (completezza) e tutte le soluzioni sono allo stesso livello (ottimalità)
Then clarify why CSP search trees tend to have high branching factors (which makes the use of heuristics very important).
Ogni nodo degli alberi di ricerca CSP ha un numero di successori uguali al numero delle variabili non ancora assegnate moltiplicato per il numero di valori assegnabili; ciò provoca un alto branching factor.

3. PLANNING

(x1)3.1 Explain the approach of the *Situation Calculus* to the representation of states and actions.
Come tutti i metodi di pianificazione (planning method) Situation Calculus usa una rappresentazione simbolica degli stati e una rappresentazione dichiarativa delle azioni, cioè basata su un linguaggio logico.
Then explain what are the *qualification problem*, the *frame problem*, and the *ramification problem*.

- Frame Problem: in ogni problema siamo obbligati a specificare non solo gli effetti di ogni azione, ma anche tutti i fatti che rimangono invariati, poiché un sistema logico non è in grado di dedurre automaticamente cosa non cambia. Se il mondo descritto è complesso, quindi c'è il rischio di avere teoremi molto pesanti con moltissimi assiomi.
- Qualification Problem: è il problema di stabilire le condizioni sufficienti per gli effetti di una determinata azione, ciò si scontra con lo specificare prima tutti gli eventi imprevisti.

- Ramification Problem: è il problema di tenere traccia di tutti gli effetti collaterali di ogni azione.

(x1)3.2 Using the approach of the Situation Calculus, specify the *predicates* and the *action axioms* for a version of the Blocks World with two types of blocks, cubes and pyramids, with the obvious further constraint that nothing can be put on top of a pyramid. Then using an example show how a specific initial state and a specific goal of your choice can be specified using the predicates.

(x3)3.4 Explain the approach of the STRIPS system to the representation of states, goals and actions.
In STRIPS gli stati sono un insieme finito di fatti, ogni fatto è un letterale positivo di base (formula atomica con un simbolo predicativo e un numero di costanti individuali come argomenti), l'obiettivo (goal) è un qualsiasi letterale positivo di base e le azioni sono specificate attraverso degli schemi di azioni, ogni schema è formato da:
- Il nome è i parametri dell'azione
- Le precondizioni necessarie per l'esecuzione dell'azione
- Gli effetti dell'esecuzione dell'azione

Sia precondizioni che effetti sono rappresentate come congiunzione di letterali sia positivi che negativi.
Then explain what the *frame problem* is, and why STRIPS does not suffer from it.
Il frame problem è il problema di specificare tutti i fatti che in un modello non cambiano a seguito di un'azione.
STRIPS non soffre di questo problema dal momento che adotta CWA (Closed World Assumption): tutti i fatti che non sono esplicitamente affermati come veri sono assunti come falsi.
"tutto ciò che non cambia esplicitamente viene considerato inalterato"

(x3)3.5 Using the STRIPS approach, specify the *predicates* and the *action schemes* for a version of:
(x2) the Blocks World with two types of blocks, cubes and pyramids, with the obvious further constraint that no block can be put on top of a pyramid. Then using an example of your choice show how a specific initial state and a specific goal can be specified.
(x1) the Blocks World with a floor, and blocks of three different colors (red, green, and blue), such that: (i) a block cannot be put on a block of the same color; (ii) red blocks cannot be put on the floor. To specify the preconditions of an action you are allowed to use negation and equality (or inequality), but you cannot use disjunction.

(x2)3.8 Explain what the *qualification problem* is, and say whether STRIPS suffers from this problem.
The qualification problem is the problem of specifying sufficient conditions for the success of an action. STRIPS suffers from this problem exactly like every other formal approach to the representation of actions, because in the real world the success of an action cannot be guaranteed.

(x3)3.9 Using the STRIPS approach, specify all the *predicates* and all the *action schemes* for the following problem:
– to turn the light on (off), the light must be off (on) and the agent must press the button;
– to press some object, the agent must be near that object;
– to be near some object, the agent must walk from its current location to the location of that object.

2.3 For the sake of simplicity, we assume that there is only one agent, one light, and one button.
Predicates: LightOn, LightOff, At(object,location)
Action schemes:
PressButton1
preconditions: At(Button,x), At(Agent,x), LightOff
negative effects: LightOff
positive effects: LightOn
PressButton2
preconditions: At(Button,x), At(Agent,x), LightOn
negative effects: LightOn
positive effects: LightOff
Move(x,y)
preconditions: At(Agent,x)
negative effects: At(Agent,x)
positive effects: At(Agent,y)
A sample problem and the corresponding plan (not required):
Initial state: At(Agent,1), At(Button,2), LightOff
Goal state: LightOn
Plan: (Move(1,2), PressButton2)

(x1)3.10 Specify a STRIPS representation for the planning problem stated below: define a set of predicates and a set of actions, specify the action schemes, and represent the goal and a possible initial state (you are not required to find a plan).
Making tea. To make a cup of tea, you need to put a tea-bag into a cup of boiling water. To have a tea bag you may take one from the packet (assume the packet is never empty). You can have a cup of boiling water by putting a cup of fresh water into the microwave oven for two minutes. To have a cup of fresh water you need to fill an empty cup using the water tap.

(x3)3.11 Define the frame problem, then say whether STRIPS planners suffer from it, and explain why.

(x3)3.12 Specify a STRIPS representation for the planning problem stated below (please read it carefully):

- **define a set of predicates and constants with their intuitive manings**
- **specify a set of action schemes**
- **represent the initial state and the goal (you are not required to find a plan).**

(x2) There are two closed safeboxes, one of which contains a cigar. The key of this safebox is in the other safebox, and the key of the latter safebox is on the table.
You want to have the cigar.
To have an object, you need to take it. You can always take an object from the table. To take an object from a safebox, the safebox must be open. To open a closed safebox, you must have the key of that safebox.

(x1) On a floor there are 2 red, 2 blue, and 2 green blocks.
Blocks can be stacked, but a red block cannot be placed on the floor, and no block can be placed on a green block if such a green block is on the floor.

(x1) The “Monkey and Banana” problem.
In a room there is a box, a monkey (the agent), and a bunch of bananas hanging from the ceiling. The monkey can grasp a banana from the bunch only if it stands on the box when the box is right below the bunch of bananas, the box and the monkey are at different places in the room, and the box is not under the bunch of bananas.
The monkey can pull the box to move it, can climb the box, and can grasp a banana (in the previously specified conditions).

The goal is that the monkey has a banana.

(x1) In a room there is a box containing a coin. The agent's goal is to get the coin.

The agent can: walk from one location in the room to another (you can represent different locations in the room as L1, L2, etc.); open a box; pick up an object from an open box.

Initially, the box is closed and the agent is far from the box.

(x1)3.13 Concisely describe the three main difficulties of planning: the qualification problem, the frame problem, and the ramification problem.

Il Frame Problem riguarda il fatto di dover specificare tutti i fatti che non cambiano a seguito di un azione, ciò comporta un aumento considerevole della complessità dei teoremi; il Qualification Problem riguarda il trovare le precondizioni necessarie di un azione, infine il Ramification Problem riguarda il trovare tutti gli effetti collaterali di un'azione.

(x1)3.14 Suggest a STRIPS representation for the problem stated below:

Consider a room whose map is represented as a 10 × 10 grid (cells are numbered from 1 to 100). In the room there are an agent (at a cell), a shelf (at some other cell) and a table (still at some other cell). On the shelf there are a number of books. The goal for the agent is to have a specific book on the table.

To take a book from the shelf the agent has to be at the same position of the shelf, and to place the book on the table the agent has to be at the same position of the table.

Represent the relevant actions so that they are parametric with respect to: (i) the positions of the agent, of the shelf, and of the table; (ii) the number of books that are initially on the shelf; (iii) the specific book that the agent wants to have on the table.

Then represent a specific initial state (specifying the positions of agent, shelf, and table, and listing which books are on the shelf) and a specific goal (specifying which book has to be brought to the table).

Finally describe a plan solving the specified problem (you are not required to describe the planning process).

(x1)3.16 Explain the concept of Sussman's anomaly. Then completely specify in STRIPS a blocks world (with action schemes and initial state), and specify a goal that is an example of Sussman's anomaly.

L'anomalia di Sussman si presenta quando si ha un goal formato dalla congiunzione di due o più predicati, quando si cerca di soddisfare il secondo (o seguenti) di deve tornare indietro a uno stato in cui il primo non era soddisfatto e da qui soddisfarli entrambi.

(x1)Consider the planning problem stated below:

A robot wants to put a book on the table. Initially, the book is on the chair, far from the table, and the robot is at the window, far from both the book and the table. The robot can: walk from any place to any other place; pick up things from any place (if its hand is free); put down things at any place.

Now:

define a set of predicates (and possibly constants), briefly describing their intuitive meanings;

define a sufficient set of action schemes;

represent the initial state and the goal;

write a plan that achieves the goal;
describe the states obtained after the execution of each action of the plan.
(You do not have to explain how an automatic planner would build the plan).

4. LOGIC (theory)

(x1)4.1 Provide a definition of the following concepts:
a) inference rule;
b) consistency;
c) completeness.