Core Question – Principles of Autonomy and Decision Making:

Please note there are **3** core questions. Please answer any **2 of 3**, but **NOT** all three.

Core Question 1: Propositional Logic Modelling.

Encode the following statement in propositional logic. Then reduce to conjunctive normal form:

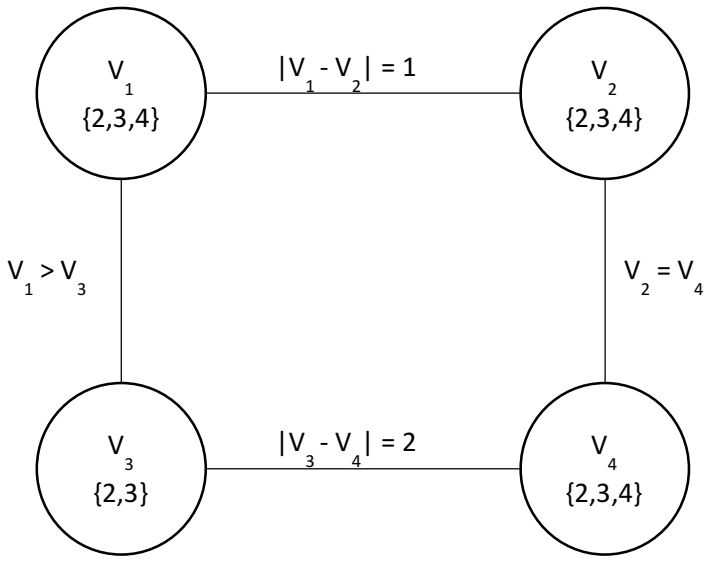
An autonomous car is shared between two users for on-demand use. On a given day, the car may be used by traveler A (denoted by proposition UA), and/or by traveler B (denoted by UB). If, and only if, both use the car (UA ∧ UB), they either make two individual uses (denoted by TI) or they carpool (denoted by CP), but not both. The car needs to be refueled (denoted by CR) after two uses that day, but should not be refueled otherwise. Please note there are exactly 5 propositions: UA, UB, TI, CP and CR.

Please note there are **3** core questions. Please answer any **2 of 3**, but **NOT** all three.

Core Question 2: Constraint Satisfaction

Satisfaction: Describe how **back-track search with forward checking** can be used to solve the following Constraint Satisfaction Problem, and show the search tree for the first step of the algorithm (and only the first step). The CSP is depicted graphically below. The circles represent variables with their initial domains enclosed, and lines between variables represent binary constraints, as labeled. Please show the progression of the algorithm using a search tree. Number the tree nodes in the order visited. Label each node with the variable domain elements that remain after pruning.

DO NOT use dynamic variable ordering. Instead, assign variable V_1, then V_2, then V_3, then V_4, and select domain elements in the order 2, then 3, then 4.



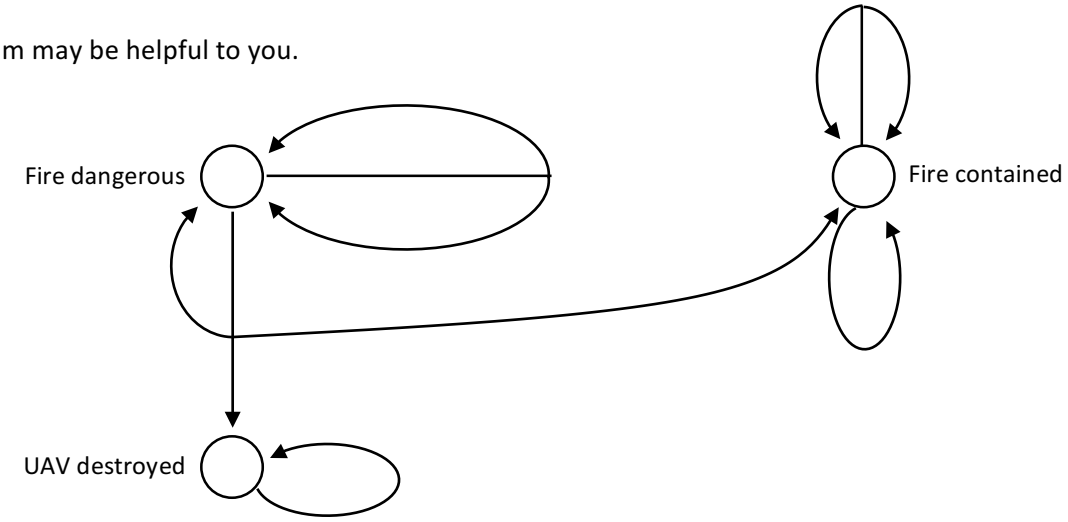
Please note there are **3** core questions. Please answer any **2 of 3**, but **NOT** all three.

Core Question 3: Markov Decision Processes

Given the following scenario description, model the system as an MDP and describe how the optimal policy can be computed. We do **not** want you to solve for the optimal policy.

> *An unmanned aerial vehicle assists during forest fires by fulfilling two functions: a)* ***search*** *for civilians in areas of risk, and b)* ***drop*** *retardant on the forest fire. The fire is classified as* ***'contained'*** *or '****dangerous****,'* *and changes as a result of the drop action. The UAV performs an action after every time step, with a fixed duration between time steps. The mission is long-term, and is* ***approximated as infinite*** *in duration. Future reward is discounted, with each successive action being only 0.8 times as valuable as an action at the preceding time step.*
>
> *The action of dropping retardant on a dangerous fire will cause the UAV to be* ***destroyed*** *with 5% probability (in which case the mission ends and no reward is gained). There is no chance of failure when either the fire is contained or the UAV is searching. When the UAV is not destroyed, dropping retardant on a dangerous fire has a 10% probability of containing that fire. In addition, when the fire is dangerous, if dropping retardant succeeds at containing a dangerous fire, it nets a reward of 20; otherwise, the fire remains dangerous and the reward is 10. Dropping retardant on a contained fire will always result in a contained fire, with a reward of 5.*
>
> *The action of searching for civilians will find civilians 10% of the time and will fail to find civilians 90% of the time. No damage is inflicted to the AUV by searching. Finding civilians is worth a reward of 100 when the fire is dangerous, and a reward of 20 when the fire is contained. Failure to find civilians always nets 0 reward.*

This diagram may be helpful to you.

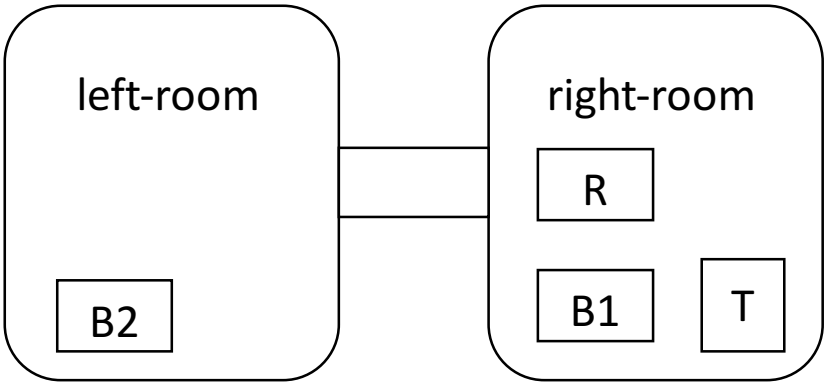


Cognitive Robotics Option: Planning and Heuristics:

A local robotics company has made a breakthrough in their robotics research: they've taught their robot how to move and load packages on a truck by drop-kicking them! (Let's hope they have good insurance for those packages...).

They've hired you to develop an activity planner for their truck loading, drop-kicking robot. Their facility consists of two rooms: the loading dock (right-room) and the warehouse (left-room). The robot can move between the rooms, can kick packages between rooms (but only from left to the right!), and can load the truck. Trucks can only carry two packages at a time. These actions are described in PDDL on the last page of this question, along with the objects and predicates you will use.

a) You begin by evaluating different methods for computing a heuristic of the number of actions required in order to reach the goal state. You decide to first evaluate the fast forward heuristic (that is, the relaxed plan heuristic). Given the following initial and goal states, use the fast forward heuristic to determine a lower bound on the number of actions required to reach the goal state. Support your answer by drawing the relaxed plan graph.

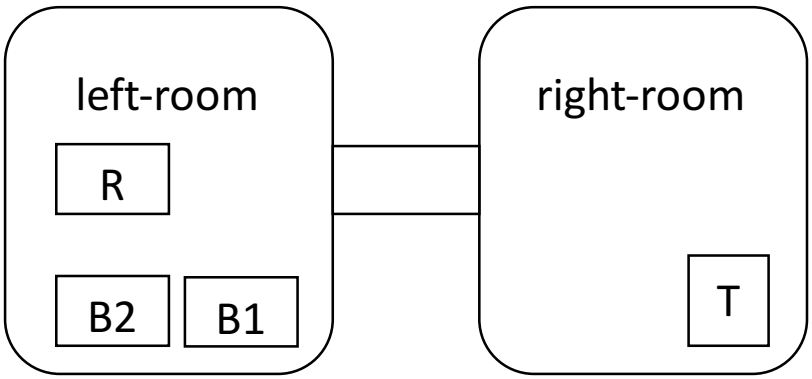



```
(:init (left-of left-room right-room)
       (right-of right-room left-room)
       (in robot right-room)
       (in truck right-room)
       (in box1 right-room)
       (in box2 left-room)
       (empty truck))

(:goal (full truck))
```

b) You decide that the heuristic provided by fast forward is too loose in this scenario. However, you realize it may not be the heuristic at fault, but instead the model you were given! Identify a way to modify the actions and predicates such that the fast forward heuristic would provide a better bound for the initial and goal states given in part a. You do not need to write out the new model, just explain how you would change it at a high level.

c) Next, you start looking at exact solutions to the planning problem. You decide to use the plan-graph algorithm, which allows compatible actions to be executed in the same time step. Given the following initial state, for each pair of actions below (i & ii), say whether or not the action pair can be executed in the same time step. If not, specify why. Continue to use the original action and predicate model.




```
(:init (left-of left-room right-room)
       (right-of right-room left-room)
       (in robot left-room)
       (in truck right-room)
       (in box1 left-room
       (in box2 left-room)
       (empty truck))
```

i) (kick-box-right robot box1 left-room right-room)
   (kick-box-right robot box2 left-room right-room)

ii) (kick-box-right robot box1 left-room right-room)
    (move-right robot left-room right-room)

Actions, objects and predicates for Drop-Kick Robot are given on the next page:

```
(:action move-left
         :parameters (?r - robot
                      ?from ?to - room)
         :precondition (and (left-of ?to ?from)
                            (in ?r ?from))
         :effect (and (in ?r ?to)
                      (not (in ?r ?from))))

(:action move-right
         :parameters (?r - robot
                      ?from ?to - room)
         :precondition (and (right-of ?to ?from)
                            (in ?r ?from))
         :effect (and (in ?r ?to)
                      (not (in ?r ?from))))

(:action load-halfway
         :parameters (?r - robot
                      ?truck - truck
                      ?box - box
                      ?room - room)
         :precondition (and (in ?r ?room)
                            (in ?box ?room)
                            (in ?truck ?room)
                            (empty ?truck))
         :effect (and (half-full ?truck)
                      (not (in ?box ?room))
                      (not (empty ?truck))))

(:action load-fully
         :parameters (?r - robot
                      ?truck - truck
                      ?box - box
                      ?room - room)
         :precondition (and (in ?r ?room)
                            (in ?box ?room)
                            (in ?truck ?room)
                            (half-full ?truck))
         :effect (and (full ?truck)
                      (not (in ?box ?room))
                      (not (half-full ?truck))))

(:action kick-box-right
         :parameters (?r - robot
                      ?box - box
                      ?from ?to - room)
         :precondition (and (in ?r ?from)
                            (in ?box ?room)
                            (right-of ?to ?from))
         :effect (and (in ?box ?to)
                      (not (in ?box ?from))))

(:objects robot - robot
          box1 box2 - box
          left-room right-room - room
          truck - truck)

(:predicates (in ?object - object ?room - room)
             (left-of ?room1 ?room2 - rooms)
             (right-of ?room1 ?room2 - rooms)
             (empty ?truck - truck)
             (half-full ?truck - truck)
             (full ?truck - truck))
```