CPSC 322 Assignment 3

Jose Abraham Torres Juarez - 79507828
Gustavo Martin - 62580121

We are claiming 2 late days for this Assignment.

1 Question 1 [35 points]: Heuristics in STRIPS: a video game example

(a) **4 points** Define the STRIPS features (variables) and their domains for this problem.

agent_has_charge {T/F}

agent_loc1 {T/F}
agent_loc2 {T/F}
agent_loc3 {T/F}
agent_loc4 {T/F}
agent_loc5 {T/F}
agent_loc6 {T/F}
agent_loc7 {T/F}
agent_loc8 {T/F}
agent_loc9 {T/F}
agent_loc10 {T/F}

one_has_charge {T/F}
four_has_charge {T/F}

three_has_enemy {T/F}
three_not_has_enemy {T/F}

nine_has_enemy {T/F}
nine_not_has_enemy {T/F}

(b) **8 points** Give the STRIPS representation of the actions for location 4 ("moveRight4", "moveLeft4", pickUp4, and fire4).

moveRight4:

- Preconditions: agent_loc_4 = T
- Effects: agent_loc_4 = F, agent_loc_5 = T

moveLeft4:

- Preconditions: agent_loc_4 = T, three_not_has_enemy = T
- Effects: agent_loc_4 = F, agent_loc_3 = T

pickUp4:

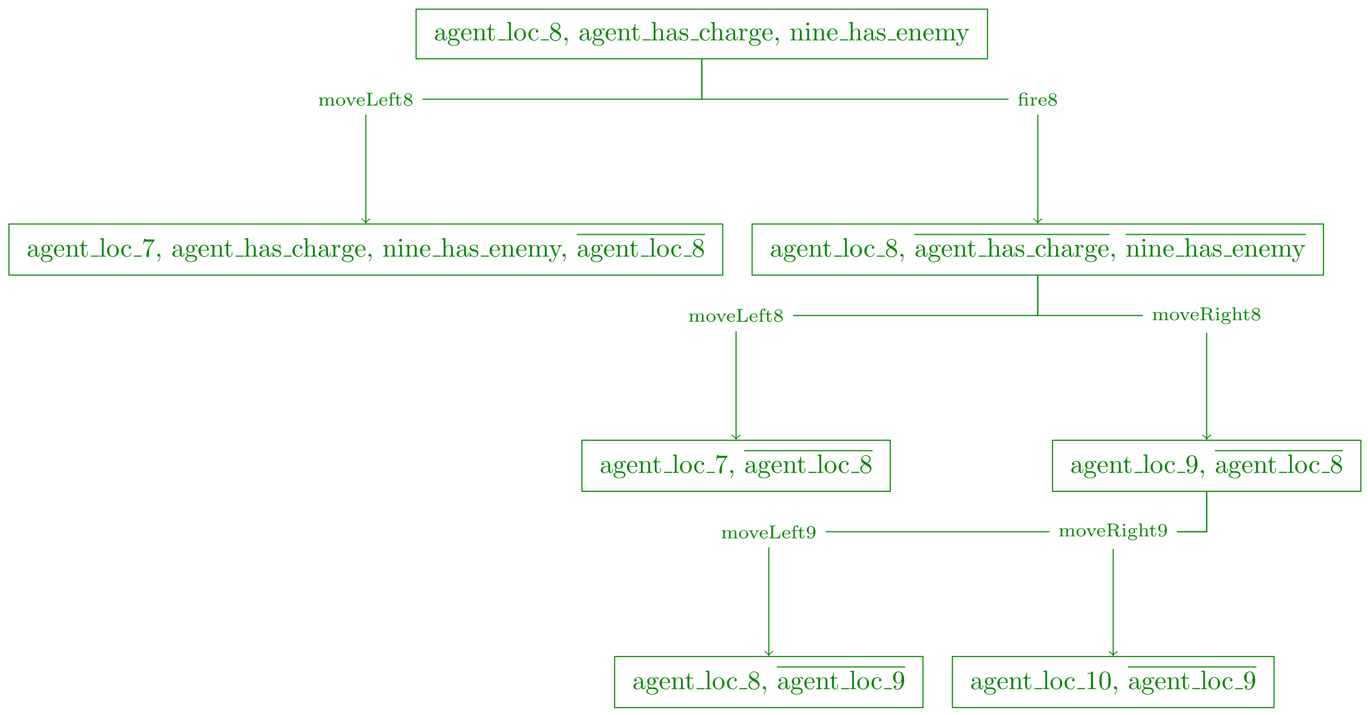
- Preconditions: agent_loc_4 = T, four_has_charge = T
- Effects: four_has_charge = F, agent_has_charge = T

fire4:

- Preconditions: agent_loc_4 = T, agent_has_charge = T

- Effects: agent_has_charge = F, three_has_enemy = F, three_not_has_enemy = T,

(c) **6 points** Draw a part of the search space that includes the optimal plan



agent_loc_8, agent_has_charge, nine_has_enemy
moveLeft8
fire8
agent_loc_7, agent_has_charge, nine_has_enemy, $\overline{\text{agent\_loc\_8}}$
agent_loc_8, $\overline{\text{agent\_has\_charge}}$, $\overline{\text{nine\_has\_enemy}}$
moveLeft8
moveRight8
agent_loc_7, $\overline{\text{agent\_loc\_8}}$
agent_loc_9, $\overline{\text{agent\_loc\_8}}$
moveLeft9
moveRight9
agent_loc_8, $\overline{\text{agent\_loc\_9}}$
agent_loc_10, $\overline{\text{agent\_loc\_9}}$

(d) **3 points** What is a good admissible heuristic for this planning goal?
The number of locations plus the number of enemies, between the agent's location and location 10. This will give us the minimum number of moves and fire actions that the agent has to make to reach the goal. Furthermore we can add 1 to the heuristic, if there is at least one enemy between them and the agent doesn't have the weapon charged, so the agent has to do the pickup action to load the weapon and then fire to the enemy that is between him and the goal.

(e) **2 points** What is the value of this heuristic for the state that has the agent in location 9, cells 3 and 9 free of monsters, the weapon charged and charges available in location 1 and 4? Represent this state using your features from (a), and justify your answer.

agent_loc_9, agent_has_charge, $\overline{\text{three\_has\_enemy}}$, $\overline{\text{nine\_has\_enemy}}$, one_has_charge, four_has_charge

In this state the distance between the agent and the goal is 1 and the number of enemies between them is 0. So the heuristic in this state would be just 1.

(f) **2 points** What is the value of this heuristic for the state that has the agent in location 8, monsters in both cell 3 and 9, the weapon charged and charges available in location 1 and 4? Represent this state using your features from (a), and justify your answer.

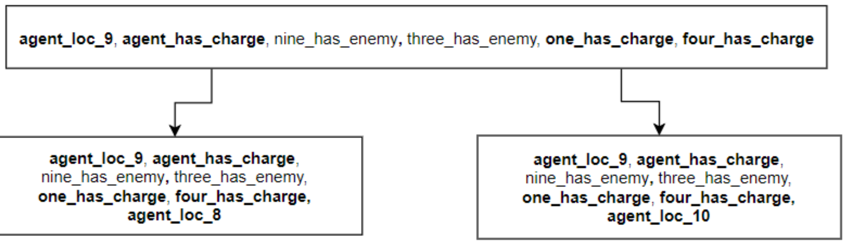
agent_loc_8, agent_has_charge, three_has_enemy, nine_has_enemy, one_has_charge, four_has_charge

In this state the distance between the agent and the goal is 2 and the number of enemies between them is 1. So the heuristic in this state would be 3.

(g) **2 points** Why is this heuristic not very useful?
Although this heuristic is admissible, it is a very relaxed version of the problem. In other words the heuristic is too strong. Also, the calculation of this heuristic depends of knowing completely the state of the world. If we are uncertain if there are enemies or not, the heuristic might not be very informative.

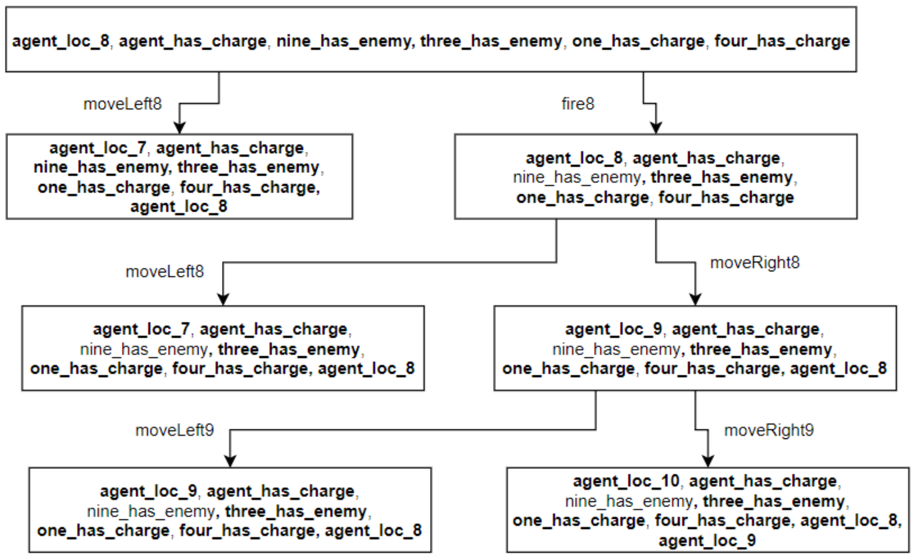
(h) **3 points** What is the value of the ignore-delete-list heuristic for the state in (f)? Also give theoptimal plan for the relaxed problem (starting at the state in (f)). Justify your answer

The heuristic in this state would be three. The agent can fire from location 8 to destroy the enemy on cell 9 and keep it's weapon charged, this is because we ignore effects that set values to false. Then, the agent moves from location 8 to location 9 and finally to location 10.



agent_loc_9, agent_has_charge, nine_has_enemy, three_has_enemy, one_has_charge, four_has_charge
agent_loc_9, agent_has_charge, nine_has_enemy, three_has_enemy, one_has_charge, four_has_charge, agent_loc_8
agent_loc_9, agent_has_charge, nine_has_enemy, three_has_enemy, one_has_charge, four_has_charge, agent_loc_10

(i) **3 points** What is the value of the ignore-delete-list heuristic for the state in (e)? Also give the optimal plan for the relaxed problem (starting at the state in (e)). Justify your answer
The heuristic in this state would be one. The agent just has to move to location 10 from location 9 to reach the goal state.



agent_loc_8, agent_has_charge, nine_has_enemy, three_has_enemy, one_has_charge, four_has_charge
moveLeft8
fire8
agent_loc_7, agent_has_charge, nine_has_enemy, three_has_enemy, one_has_charge, four_has_charge, agent_loc_8
agent_loc_8, agent_has_charge, nine_has_enemy, three_has_enemy, one_has_charge, four_has_charge
moveLeft8
moveRight8
agent_loc_7, agent_has_charge, nine_has_enemy, three_has_enemy, one_has_charge, four_has_charge, agent_loc_8
agent_loc_9, agent_has_charge, nine_has_enemy, three_has_enemy, one_has_charge, four_has_charge, agent_loc_8
moveLeft9
moveRight9
agent_loc_9, agent_has_charge, nine_has_enemy, three_has_enemy, one_has_charge, four_has_charge, agent_loc_8
agent_loc_10, agent_has_charge, nine_has_enemy, three_has_enemy, one_has_charge, four_has_charge, agent_loc_8, agent_loc_9

(j) **2 points** Is this heuristic more useful than ignoring preconditions? Why or why not?
Yes, this heuristic provides us a more realistic simplified version of the problem. It is not as over-simplified as ignoring preconditions so it gives us a better estimate while being fast enough for us to calculate.

2 Question 2 [38 points]: STRIPS representation

The DrinkUp Bartendering school just opened, so it only has two students (Jane and Laura), one small cocktail shaker that can only make one cocktail at a time, and one cocktail glass. Both Jane and Laura need to practice making cocktails that require having a base of shaken Vodka. Jane will use this to make a blue cocktail with Blue Curacao (Blue for short), Laura to make a pink one with Berry Liqueur (Berry for short). The cocktail glass needs to be clean when making a new cocktail. They will only make more shaken vodka base when there is no more left.

Figure 1 shows the STRIPS representation for this problem.

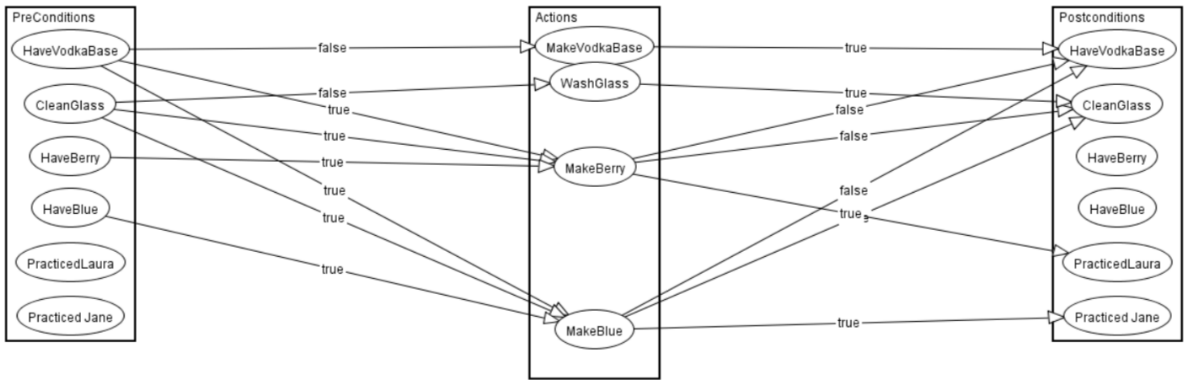


Figure 1 - STRIPS representation for the cocktail making problem

Load the applet at `http //aispace.org/strips_to_csp/`, and load the file cocktail.xml containing this STRIPS problem.

(a) **6 points** List the variables/features that define the states in the problem, as well as their domains.

- HaveVodkaBase {T,F}
- PracticedLaura {T,F}
- HaveBerry {T,F}
- HaveBlue {T,F}
- PracticedJane {T,F}
- CleanGlass {T,F}

(b) **8 points** List the actions in this problem, with their preconditions and effects.

MakeBlue:

- Preconditions: { HaveVodkaBase = T, CleanGlass = T, HaveBlue = T }
- Effects: { HaveVodkaBase = F, PracticedJane = T, CleanGlass = F }

MakeBerry:

- Preconditions: { HaveVodkaBase = T, CleanGlass = T, HaveBerry = T }
- Effects: { HaveVodkaBase = F, PracticedLaura = T, CleanGlass = F }

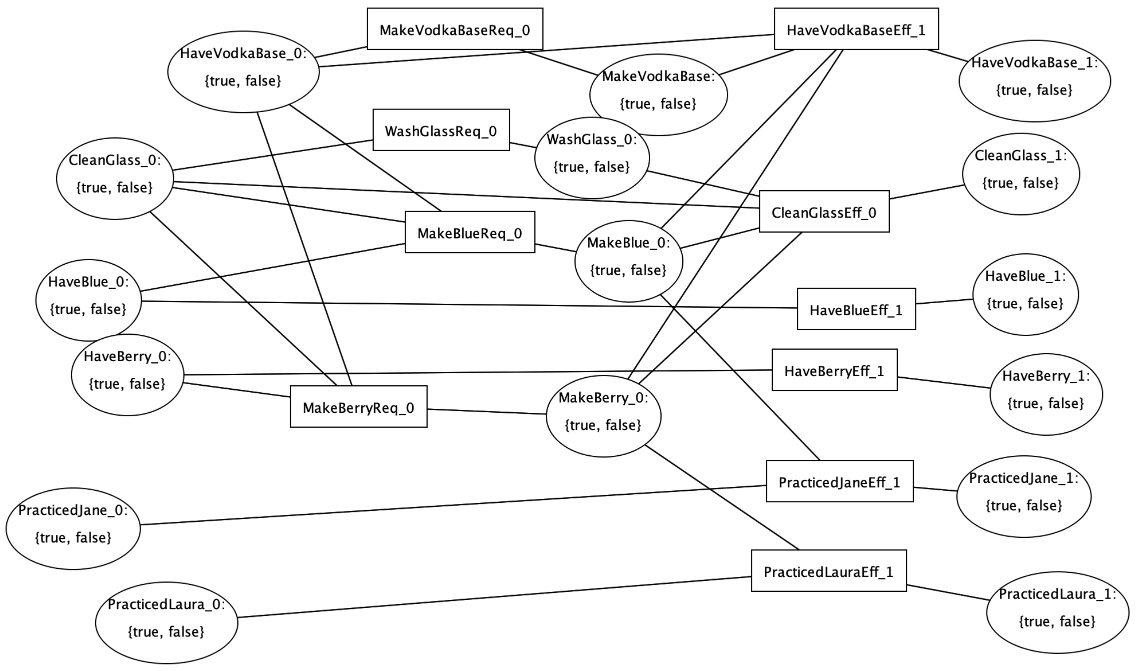
WashGlass:

- Preconditions: { CleanGlass = F }
- Effects: { CleanGlass = T }

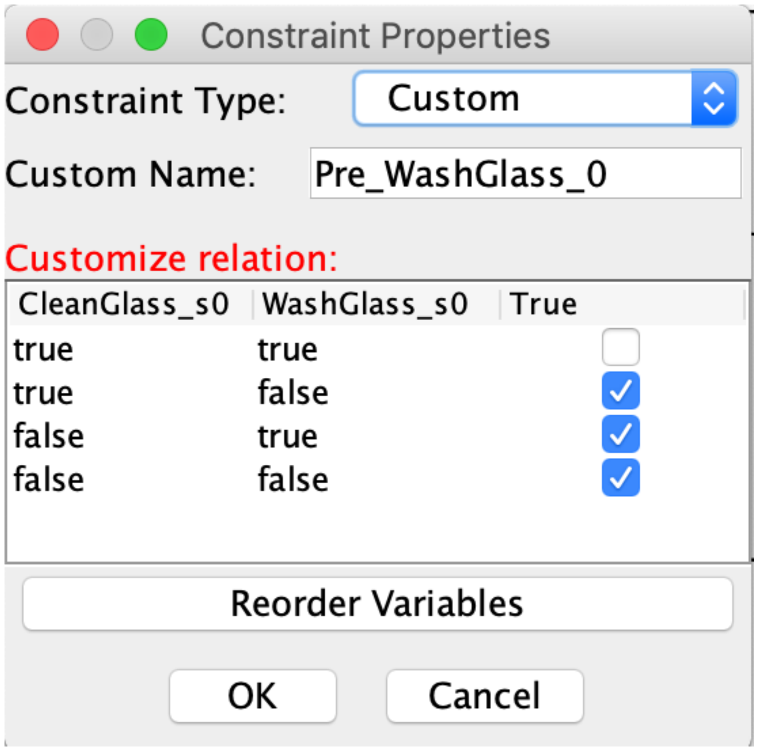
MakeVodkaBase:

- Preconditions: { HaveVodkaBase = F }
- Effects: { HaveVodkaBase = T }

(c) **1 point** Convert this STRIPS problem to a CSP planning problem with a horizon of 1. We suggest that you first try to set up the constraints network by yourself, but you can check your solution in the applet by clicking "Solve" and then "Solve with Arc Consistency". Show a screenshot of the network thus obtained. (Note: if the network appears cramped, you can enlarge the window and then select View-¿Autoscale to spread it out; you may still need to move around some of the nodes to make it easier to read.)


MakeVodkaBaseReq_0
HaveVodkaBaseEff_1
HaveVodkaBase_0: {true, false}
MakeVodkaBase: {true, false}
HaveVodkaBase_1: {true, false}
WashGlassReq_0
WashGlass_0: {true, false}
CleanGlass_0: {true, false}
CleanGlass_1: {true, false}
CleanGlassEff_0
MakeBlueReq_0
MakeBlue_0: {true, false}
HaveBlue_0: {true, false}
HaveBlueEff_1
HaveBlue_1: {true, false}
HaveBerry_0: {true, false}
HaveBerryEff_1
HaveBerry_1: {true, false}
MakeBerryReq_0
MakeBerry_0: {true, false}
PracticedJaneEff_1
PracticedJane_1: {true, false}
PracticedJane_0: {true, false}
PracticedLauraEff_1
PracticedLaura_0: {true, false}
PracticedLaura_1: {true, false}

(d) **6 points** Show a screenshot of the truth table representing the constraint Pre_WashGlass_0 and explain what the different rows mean.



Constraint Properties

Constraint Type: Custom

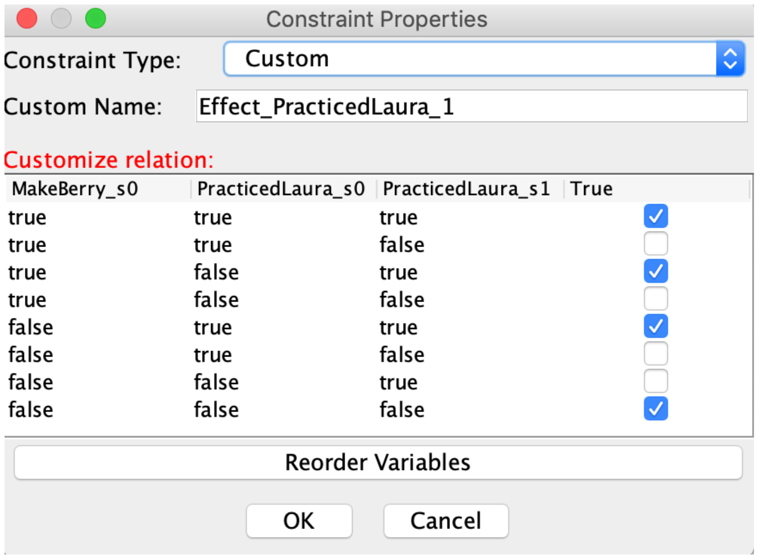
Custom Name: Pre_WashGlass_0

Customize relation:

CleanGlass_s0	WashGlass_s0	True
true	true	☐
true	false	✓
false	true	✓
false	false	✓

The first row, with CleanGlass_s0 = T and WashGlass_s0 = T has a value of false in the constraint, this is because you can't wash a glass that is already clean. You can only wash it when is not clean, or not wash it at all this is represented by the next rows being set to true.

(e) **6 points** Show a screenshot of the truth table representing the constraint Effect_PracticedLaura_1 and explain what the different rows mean (for this part, you only need to report and discuss the rows checked as "true" in the applet)



Constraint Properties

Constraint Type: Custom

Custom Name: Effect_PracticedLaura_1

Customize relation:

MakeBerry_s0	PracticedLaura_s0	PracticedLaura_s1	True
true	true	true	✓
true	true	false	☐
true	false	true	✓
true	false	false	☐
false	true	true	✓
false	true	false	☐
false	false	true	☐
false	false	false	✓

In the first case we have everything set to True, and this is correct because PracticedLaura_s0 was true so it doesn't matter that MakeBerry_s0 was True, PracticedLaura_s1 has to be True.

The second case is similar to the first one, except that PracticedLaura_s0 is False, but this doesn't matter because MakeBerry_s0 = True forces PracticedLaura_s1 to be True.

The third case that is true, occurs when PracticedLaura_s0 was True, then PracticedLaura_s0 is also True even if MakeBerry_s0 is false.

The last case is when Laura doesn't practice at all, PracticedLaura_s0 and MakeBerry_s0 are False, so PracticedLaura_s1 has no more option than remain False.

(f) **5 points** Suppose neither Jane nor Laura have practiced making their cocktails. Also suppose that we are in a state with no vodka base, a dirty glass, but we have both Blue Curacao and Berry Liqueur. At what minimum horizon do we need to unroll the CSP if the goal is $PracticedLaura = T$ and $PracticedJane = T$? Give a plan to reach the goal, listing the action(s) taken at each step.

Remember that to solve this problem as a CSP you need to define the features you want to hold in your start and goal states; in the AIspace STRIPS to CSP applet, you can do that via "create → set start state" and "create → set goal state". (You also set the horizon here.) After setting the start and goal state, when you click "solve → solve with arc consistency", the resulting CSP will have the desired constraints for start and goal states.
The mininum horizon to reach the solution is 2. We can take the following actions:

1. MakeVodkaBase, WashGlass
2. MakeBerry, MakeBlue

The problem allow us to do this, because the actions MakeBerry and MakeBlue do not have constraints on each other. So even it is imposible to do two different cocktails with the same clean glass and vodka base, the problem has not the constraints to manage this so we are able to reach the goal with just two steps.

(g) **2 points** Explain why the plan you found in part (f) is inconsistent with the problem description given at the beginning of this question. (Note: you may find a bug in one of the constraint tables in this problem. That is not relevant to this question, and you may ignore it.)
It is imposible to do two different cocktails with the same clean glass and vodka base. Furthermore the problem description says that the DrinkUp Bartendering school has only one small cocktail shaker that can only make one cocktail at a time.

(h) **2 points** Why does this issue happen with the CSP approach of solving this planning problem, and what can we do to the CSP to fix it?
The problem is that the MakeBerry and MakeBlue must have a constraint between them, so is not allowed to do one cocktail if the other one is being done. This can be fixed by adding an extra constraint between MakeBerry and MakeBlue on every state, this constraint should only allow { MakeBerry=T,MakeBlue=F } or { MakeBerry=F,MakeBlue=T } or { MakeBerry=F,MakeBlue=F }

3 Question 3 [23 points]: Propositional Definite Clauses - Proof Procedures

Consider the following knowledge base KB.

1.	$j \leftarrow q \wedge z$
2.	k
3.	$m \leftarrow w \wedge q \wedge p$
4.	$w \leftarrow z$
5.	$w \leftarrow u \wedge x$
6.	$p \leftarrow x \wedge s$
7.	$z \leftarrow s$
8.	$z \leftarrow p$
9.	$w \leftarrow k \wedge j$
10.	$q \leftarrow s$
11.	s
12.	$q \leftarrow j \wedge w$
13.	$u \leftarrow s$

(a) **5 points** Using the bottom-up proof procedure, list all of the atoms v such that KB $\vdash$ v. Show your work, using the numbers provided above to indicate which clauses are involved in each step of your work.
Step 1: { k } clause involved: 2
Step 2: { k, s } clause involved: 11
Step 3: { k, s, u } clause involved: 13
Step 4: { k, s, u, q } clause involved: 10
Step 5: { k, s, u, q, z } clause involved: 7
Step 6: { k, s, u, q, z, j } clause involved: 1
Step 7: { k, s, u, q, z, j, w } clause involved: 9

(b) **3 points** Is it true that for all these atoms v, KB $\models$ v? Why or why not?
Yes, because the set of atoms v is true in every model of KB

(c) **5 points**

i. **3 points** Is it possible to pick an atom that is not a logical consequence of this KB, and give a model of that KB in which this atom is true? If yes, give the model. If no, explain why.
Yes, atom m is not a logical consequence, and it can be true using the following representation:

j	k	m	w	p	z	q	s	u	x
T	T	T	T	T	T	T	T	T	T

ii. **2 points** Provide an interpretation of the KB that is not a model (justify your answer).
Any KB that is not a model, must have at least one clause that is false. In the interpretation provided below, every atom is false, this means that some of the clauses aren't true. An example of this is clause 11.

j	k	m	w	p	z	q	s	u	x
F	F	F	F	F	F	F	F	F	F

(d) **10 points** Using the top-down resolution proof method, answer if the following queries are logical consequences of KB. If the answer is "yes" show the successful derivation; and if the answer is "no", show one of the failing derivations.

i. **5 points** ? $m \wedge j$.

It isn't a logical consequence:

$\text{yes} \leftarrow m \wedge j.$
$\text{yes} \leftarrow w \wedge q \wedge p \wedge j.$
$\boxed{\text{yes} \leftarrow u \wedge x \wedge q \wedge p \wedge j.}$

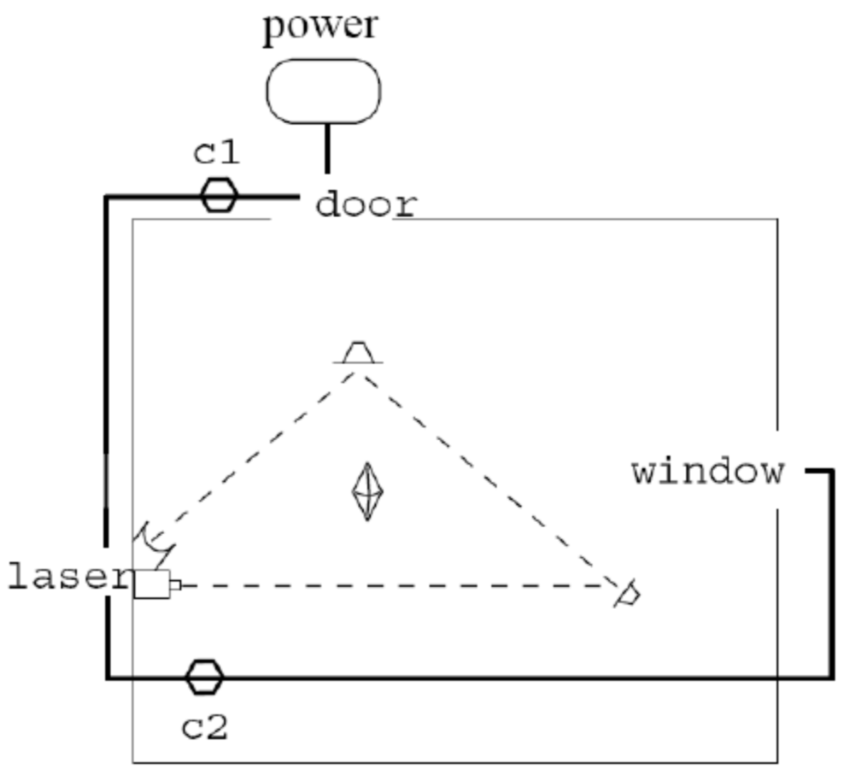
ii. **5 points** ? $j \wedge w.$
It is a logical consequence:

$\text{yes} \leftarrow j \wedge w.$
$\text{yes} \leftarrow q \wedge z \wedge w$
$\text{yes} \leftarrow s \wedge z \wedge w$
$\text{yes} \leftarrow z \wedge w$
$\text{yes} \leftarrow z \wedge z$
$\text{yes} \leftarrow z$
$\text{yes} \leftarrow s$
$\text{yes} \leftarrow$

4 4 [30 points]: Predicate Logic

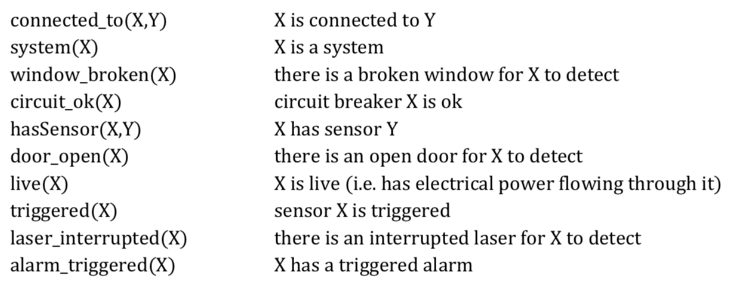
A priceless diamond, the Orange Ocelot, will be displayed at the Vancouver Jewel Museum. You have been hired to write the software for the security system that will be used to monitor the room holding the diamond for display. If some dastardly jewel thief attempts to steal the diamond, the system alarm must trigger. The system, museum, shown in the figure below, has three sensors:

1. a "laser" motion sensor, laser
2. a broken window sensor, window
3. a door lock sensor, door



The three sensors are wired in one circuit around the room as shown. The door sensor is connected directly to the power. The laser and the window are each preceded by circuit breakers, c1 and c2, which stop the power at that point if the fuse has blown. If they are "ok" the power flows through it to the next sensor.

You will use the Definite Clause Deduction applet available at `http://www.aispace.org` to implement the system. This will involve writing a short set of axioms to define the problem and running some queries. First, take a look at the sample knowledge base called "An Electrical System" and understand how it works. You can use the "Move Subtree" button to get a better view of the answer to queries. Your program will use some of the same predicates used here to represent the flow of electricity through a set of switches and devices. You should use only (and all of) the following predicates for your system:



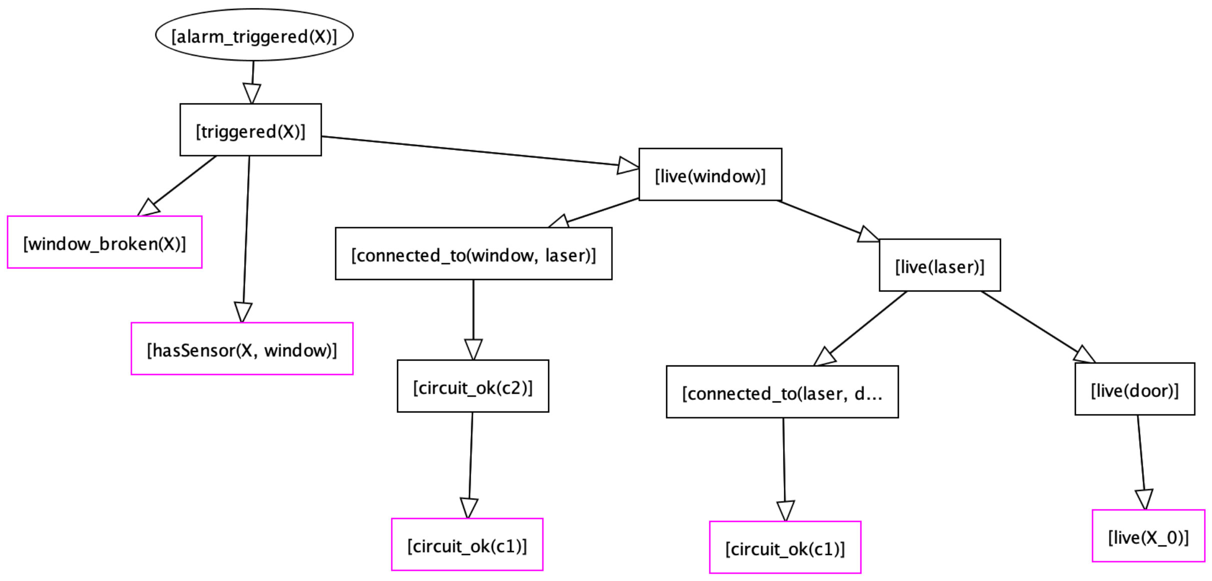
connected_to(X,Y)	X is connected to Y
system(X)	X is a system
window_broken(X)	there is a broken window for X to detect
circuit_ok(X)	circuit breaker X is ok
hasSensor(X,Y)	X has sensor Y
door_open(X)	there is an open door for X to detect
live(X)	X is live (i.e. has electrical power flowing through it)
triggered(X)	sensor X is triggered
laser_interrupted(X)	there is an interrupted laser for X to detect
alarm_triggered(X)	X has a triggered alarm

Test out your system by stating different facts such as door_open(door) and performing queries such as live(laser) or triggered(door). Remember that uppercase letters denote variables and lowercase denote particular values or instances. Make sure disabling a circuit breaker causes any dependent sensors to fail to trigger an alarm.

(a) **15 points** Implement the system using the Definite Clause applet, and paste the contents of the .pl file you created into your submission.

```
system(X).
hasSensor(X,laser).
hasSensor(X,window).
hasSensor(X,door).
live(X).
circuit_ok(c1).
circuit_ok(c2) <- circuit_ok(c1).
connected_to(laser,door) <- circuit_ok(c1).
connected_to(window,laser) <- circuit_ok(c2).
live(door) <- live(X).
live(laser) <- connected_to(laser,door) & live(door).
live(window) <- connected_to(window,laser) & live(laser).
triggered(X) <- door_open(X) & hasSensor(X,door) & live(door).
triggered(X) <- laser_interrupted(X) & hasSensor(X,laser) & live(laser).
triggered(X) <- window_broken(X) & hasSensor(X,window) & live(window).
alarm_triggered(X) <- triggered(X).
```

(b) **15 points** Provide a screen capture of the resulting proof deduction tree for the query alarm_triggered(X) when the power is on, all the circuits are ok and the window is broken. You can get this by clicking on "View Proof Deduction" and then selecting the true node from the tree.


[alarm_triggered(X)]
[triggered(X)]
[window_broken(X)]
[hasSensor(X, window)]
[live(window)]
[connected_to(window, laser)]
[circuit_ok(c2)]
[circuit_ok(c1)]
[live(laser)]
[connected_to(laser, d...
[circuit_ok(c1)]
[live(door)]
[live(X_0)]