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

(a)

Loc_i = {T/F} for i = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, and Boolean variables Loc3 free, Loc9 free, Weapon charged, Charge1Available, Charge4Available

(b)

- *move right4: preconditions {Loc4=T }, effects: { Loc4=F, Loc5=T}*
- *move left4: preconditions {Loc4=T, loc3 free = T }, effects: { Loc4=F, Loc3= T}*
- *pickup4: preconditions {Loc4 = T, charge4 available=T}, effects: {weapon charged =T, charge4 available = F}*
- *fire4: preconditions {Loc4 = T ,weapon charged = T}, effects: {loc3 free = T; weapon charged = F}*

(c)

Optimal path: fire, right, right. Figure 1 shows the corresponding search graph.

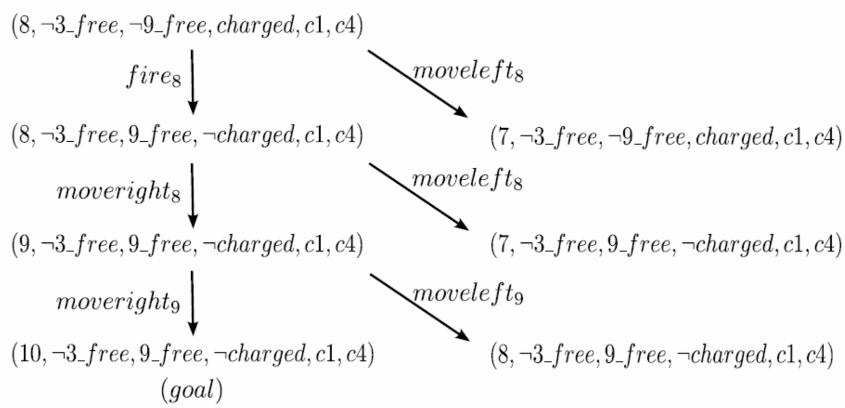


Figure 2: Search graph containing the optimal path starting at state (8, ¬3_free, ¬9_free, charged, c1, c4)

(d)

The following answers are possible:
• The agent's distance to the goal
• Add to the above the number of monsters standing in between the agent and the goal (location 10).
• Add to the above the minimal number of extra steps required to recharge the weapon for each monster between the agent's position and the goal location.

(e,f,g)

*State in e**: (**¬Loc_1, ¬Loc_2, ¬Loc_3, ¬Loc_4, ¬Loc_5, ¬Loc_6, ¬Loc_7, ¬Loc_8, Loc_9, ¬Loc_10, Loc3 free, Loc9 free, Weapon charged, Charge1Available, Charge4Available)*
*State in f: **(**¬Loc_1, ¬Loc_2, ¬Loc_3, ¬Loc_4, ¬Loc_5, ¬Loc_6, ¬Loc_7, Loc_8, ¬Loc_9, ¬Loc_10, ¬Loc3 free, ¬Loc9 free, Weapon charged, Charge1Available, Charge4Available)*

This heuristic is 1 everywhere except in goal states: if the precondition of move right9 is empty, we can apply this regardless of the state we're in, and it will make the location 10. This does not help us distinguish good states from bad states.

(h,i,j)

Relaxed plan for n=(8, ¬3 free, ¬9 free, charged, c1, c4): fire8, move right8, move right9.
h(n) = 3.
Relaxed plan for n'=(9, 3 free, 9 free, charged, c1, c4): move right 9. h(n) = 1.
The heuristic is more useful since it provides more guidance towards useful states

Question 2 [38 points]: STRIPS representation

(a)

PracticedJane ∈ {true, false}
PracticedLaura ∈ {true, false}
HaveVodkaBase ∈ {true, false}
CleanGlass ∈ {true, false}
HaveBlue ∈ {true, false}
HaveBerry ∈ {true, false}

(b)

MakeVodkaBase
preconditions: haveVodkaBase =false
effects: haveVodkaBase =true

washGlass
preconditions: cleanGlass=false
effects: cleanGlass=true

makeBlue
preconditions: cleanGlass=true, haveVodkaBase=true, haveBlue=true
effects: cleanGlass=false, haveVodkaBase=false, PracticedJane = true

makeBerry
preconditions: cleanGlass=true, haveVodkaBase=true, haveBerry=true
effects: cleanGlass=false, haveVodkaBase=false, PracticedLaura = true

(c)

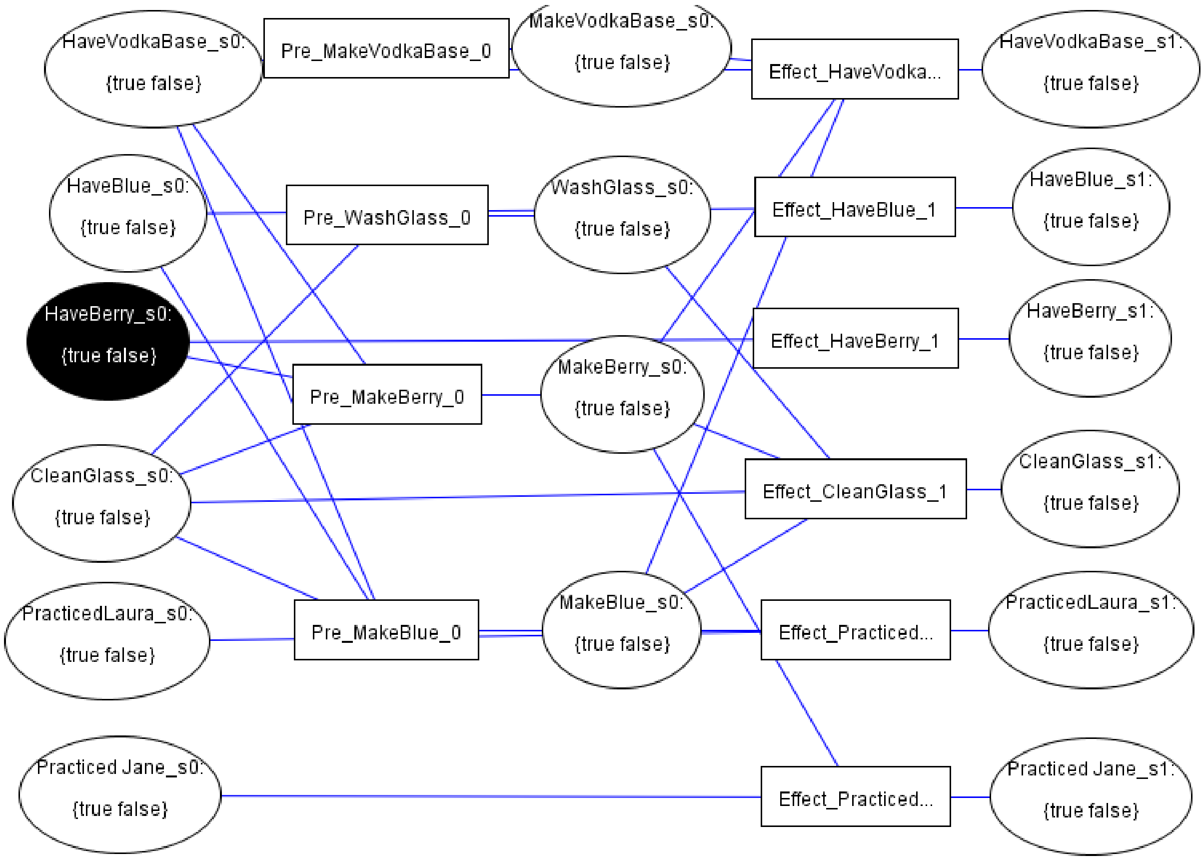
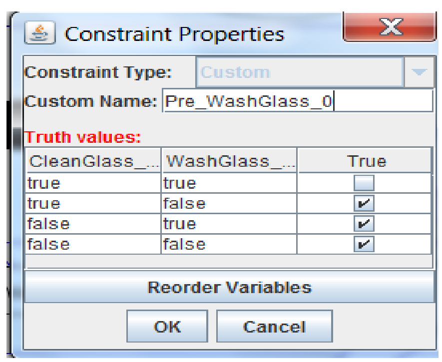


Figure 1 - CSP representation for the cocktail problem

(d)



Rows 1 and 2: When CleanGlass_s0 is true, the action WashGlass_s0 cannot be performed (it is defined so that we only wash the glass when it is not clean).
Row 3 and 4: If CleanGlass_s0 is false, we can perform WashGlass_s0, but we can also chose not to wash the glass,

(e)

Custom Name: Effect_PracticedLaura_1

Truth values:

MakeBlue_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	☑

Row 1 and 3: When makeBlue_s0 is true then PracticedLaura_s1 is true, regardless of whether Laura had already practiced in s0 (
Row 1 and 5: When PracticedLaura _s0 is true, PracticedLaura _s1 is also true, even if MakeBlue did not happen at time s0
Row 8: When PracticedLaura _s0 is false and MakeBlue _s0 is false then PracticedLaura _s1 should be still false (Laura had not practiced before, and she did not make the Blue cocktail at S0, so she still has no practice in S1.

(f)

The required horizon is 2. At stage 1 the Vodka base is made and the glass is washed. At stage 2 both Laura and Jane can make their cocktails.

(g)

The problem description says that the shaker can only make base for one cocktail, and we only have one cocktail glass, so Jane and Laura should not be able to make their cocktails together.

(h)

The CSP approach allows the planner to perform more than one action at a time unless there are constraints that explicitly prevent it. We could add for instance a mutex constraint between the two actions.

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

(a)

The order that terms are added to the set does not matter, as long as the final set is correct.

{k} clause 2
{k,s} clause 11
{k,s,q} clause 10
{k,s,q,z} clause 7
{k,s,q,z,u} clause 13
{k,s,q,z,u,w} clause 4
{k,s,q,z,u,w,j} clause 1

(b)

yes, because bottom up is sound

(c)

i. p, m, x are not logical consequences. Any interpretation that assigns T to one of them and to all of k,s,q,z,u,w,j is a model that satisfies the request above

ii. any interpretation with k=F, for instance. k is a logical consequence of KB, and so it must be true in all models of KB. If it is false, then the corresponding interpretation cannot be a model.

(d)

i. $m \wedge j$.

yes $\leftarrow$ $m \wedge j$.
yes $\leftarrow$ $w \wedge q \wedge p \wedge j$. resolving against 3
yes $\leftarrow$ $z \wedge q \wedge p \wedge j$. resolving against 4
yes $\leftarrow$ $s \wedge q \wedge p \wedge j$. resolving against 7
yes $\leftarrow$ $q \wedge p \wedge j$. resolving against 11
yes $\leftarrow$ $s \wedge p \wedge j$. resolving against 10
yes $\leftarrow$ $p \wedge j$. resolving against 11
yes $\leftarrow$ $x \wedge s \wedge j$. resolving against 6
no options for resolution, fail

iii. ? $j \wedge w$.

? $j \wedge w$.
yes $\leftarrow$ $j \wedge w$.

yes ← q ∧ z ∧ w. resolving against 1
yes ← q ∧ z ∧ z. resolving against 4
yes ← s ∧ z. resolving against 10
yes ← s ∧ s. resolving against 7
yes ← . resolving against 11

Thus j ∧ w is a logical consequence of KB.

4 [30 points]: Predicate Logic

a)

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

b)