Homework 5 – Artificial Intelligence

Task 1 - Logic

#	Inference Rule	Inferred Sentance
1-9	from knowledge base	
10	rule 2 with sentences 6 and 9	$\neg(EF \wedge SF)$
11	rule 12 with sentence 10	$\neg EF \vee \neg SF$
12	rule 11 with sentence 11	$\neg SF \vee \neg EF$
13	rule 14 with sentence 8	$\neg\neg SF$
14	rule 6 with sentences 12 and 13*	$\neg EF$
15	rule 2 with sentences 4 and 14	$\neg TF$
16	rule 6 with sentences 1 and 15	TJ

*for rule 6 to apply one need to substitute $\alpha = \neg SF$ and $\beta = \neg EF$:

$\{\alpha \vee \beta, \neg\alpha\} \vdash \beta$

$\{\neg SF \vee \neg EF, \neg\neg SF\} \vdash \neg EF$

Thus, the trainer John is the thief.

Task 2 - STRIPS Planning

2.1

The robot can only be at one place.

A exclusive OR is required to express the fact, that the robot can be either in room 1 or in room 2 but not in the same room at the same time:

$(at(robot, 1) \vee at(robot, 2)) \wedge \neg(at(robot, 1) \wedge at(robot, 2))$

If the robot holds something, its hand is not empty and vice versa.

$(\exists x) Holding(robot, x) \implies \neg Handempty$

This is equivalent to:

$Handempty \implies \neg(\exists x) Holding(robot, x)$

An object that is held by the robot is nowhere else.

If the robot is holding an item, there is no valid `at` proposition for this item.

$(\forall x) Holding(robot, x) : \neg(\exists y) at(x, y)$

2.2 Branching factors

An upper bound for the branching factor of a **forward search** is given by the maximum number of actions applicable in a state. Given a state, the robot might perform one of the following actions: - go to the other room (`GotoRoom`) -> maximum one action - pick up any key (`PickUp`) -> maximum two actions - drop a key (`Drop`) -> maximum one action - lock a door (`Lock`) -> maximum one action - unlock a door (`Unlock`) -> maximum one action

The upper bound for the branching factor is thus `6` when using forward search.

The branching factor of **backward search** depends on the concrete implementation. For example, if we add a new node for each possible action given a certain goal, the branching factor is assumed to be equal to the branching factor of the forward search. However, if we only consider the action with the least constraining precondition, the lower bound of the branching factor is one. This is, because only the node with the least constraining precondition will be inserted/considered in the next iteration. Please note, that the computation of this action requires that the goal regression for all applicable actions are computed.

In general, there are more actions relevant to a goal than there are actions applicable to a state. Thus, the branching factor is assumed to be smaller using backward search.

2.3 Additional actions

The upper bound for the branching factor for **forward search** increases by 3 because the robot has more actions available in each state.

The lower bound for the branching factor of **backward search** does not change, assuming that the implemented algorithm only considers the action with the least constraining precondition. At the same time, the computation of the goal regression becomes more complex since more actions must be considered to compute the action with the least constraining precondition.

2.4 Backward search

The following is the result of my backward search:

Initial goal: $Locked(1) \land Locked(2) \land At(key(1), 1) \land At(key(2), 1)$

Action: $Drop(key(2), 1)$

$\Longrightarrow$

Regressed goal: $At(robot, 1) \land Holding(robot, key(2)) \land Locked(1) \land Locked(2) \land At(key(1), 1)$

Action: $GoToRoom(2, 1)$

$\Longrightarrow$

Regressed goal: $At(robot, 2) \wedge Holding(robot, key(2)) \wedge Locked(1) \wedge Locked(2) \wedge At(key(1), 1)$

Action: $Lock(2)$

$\Longrightarrow$

Regressed goal: $At(robot, 2) \wedge Holding(robot, key(2)) \wedge Locked(1) \wedge At(key(1), 1)$

Action: $GoToRoom(1, 2)$

$\Longrightarrow$

Regressed goal: $At(robot, 1) \wedge Holding(robot, key(2)) \wedge Locked(1) \wedge At(key(1), 1)$

Action: $Pickup(key(2), 1)$

$\Longrightarrow$

Regressed goal: $At(robot, 1) \wedge At(key(2), 1) \wedge Handempty \wedge Locked(1) \wedge At(key(1), 1)$

Action: $Drop(key(1), 1)$

$\Longrightarrow$

Regressed goal:$At(robot, 1) \wedge Holding(robot, key(1)) \wedge At(key(2), 1) \wedge Locked(1)$

Action: $Lock(1)$

$\Longrightarrow$

Regressed goal: $At(robot, 1) \wedge Holding(robot, key(1)) \wedge At(key(2), 1)$

Action: $PickUp(key(1), 1)$

$\Longrightarrow$

Regressed goal: $At(robot, 1) \wedge At(key(1), 1) \wedge Handempty \wedge At(key(2), 1)$

This regressed goal is satisfied in the initial state.

2.5 Best-First Search

Best-First Search yields an optimal result. Since the cost for each action is one and the optimal path has length 8, the costs for the resulting path is 8.