# Assignment 5

Benjamin Sorenson

December 10, 2015

**Question 1:** (a)

$$\begin{aligned}&Action(GoAndPick(x, y, o)\\&\quad Precond: At(Robot, x) \land EmptyHand(Robot) \land At(o, y)\\&\quad Effect: \neg At(robot, x) \land At(Robot, y) \land \neg EmptyHand(robot) \land \neg At(o, y) \land Holding(Robot, o))\end{aligned}$$

(b)

$$\begin{aligned}&Action(PickAndGo(x, y, o)\\&\quad Precond: At(Robot, x) \land EmptyHand(robot) \land At(o, x)\\&\quad Effect: \neg At(Robot, x) \land \neg EmptyHand(Robot) \land \neg At(o, x) \land Holding(Robot, o) \land At(Robot, y))\end{aligned}$$

(c) In general, action schemes can be combined by conjoining their preconditions and effects. This can be useful useful when a conceptually single action may take several smaller actions to complete, but this may not be a good idea when the smaller component actions have more universal complexity. In this case, we may end up defining a lot of overly complex actions.

**Question 2:** The action *Go* is unmodified, but *Pick* can be replaced with the following actions;

$$\begin{aligned}&Action(PickLeft(o, x)\\&\quad Precond: EmptyLeftHand(Robot) \land At(Robot, x) \land At(o, x)\\&\quad Effect: \neg EmptyLeftHand(Robot) \land \neg At(o, x) \land Holding(Robot, o))\end{aligned}$$

$$\begin{aligned}&Action(PickRight(o, x)\\&\quad Precond: EmptyRightHand(Robot) \land At(Robot, x) \land At(o, x)\\&\quad Effect: \neg EmptyRightHand(Robot) \land \neg At(o, x) \land Holding(Robot, o))\end{aligned}$$

**Question 3:** (a)

$$
\begin{aligned}
&Action(MoveTray(x, y)\\
&\qquad Precond: At(Robot, x) \land At(Tray, y) \land EmptyHand(Robot)\\
&\qquad Effect: \neg At(Robot, x) \land \neg At(Tray, x) \land At(Robot, y) \land At(Tray, x))
\end{aligned}
$$

$$
\begin{aligned}
&Action(RemoveFromTray(o, x)\\
&\qquad Precond: On(o, Tray) \land At(Tray, x) \land At(Robot, x) \land EmptyHand(Robot)\\
&\qquad Effect: \neg On(o, Tray))
\end{aligned}
$$

(b)

$$
\begin{aligned}
&Action(RemoveGlassFromTray\\
&\qquad Precond: GlassOnTray \land RobotInLivingRoom \land TrayInLivingRoom \land RobotHandEmpty\\
&\qquad Effect: \neg GlassOnTray))
\end{aligned}
$$

$$
\begin{aligned}
&Action(MoveTrayToKitchen\\
&\qquad Precond: TrayInLivingRoom \land RobotInLivingRoom \land RobotHandEmpty\\
&\qquad Effect: TrayInKitchen \land \neg TrayInLivingRoom \land RobotInKitchen \land \neg RobotInLivingRoom)
\end{aligned}
$$

**Question 4:** The predicate calculus allows variables, and proposition clculus does not. In progpositional clculus we need a different fluent for each possible combination of variables which may become impractical as the number of variables becomes large. While the predicate calculus allows variables, and thus has more expressive power than the propositional calculus, the propositional calculus is easier to analyze because each proposition is decidable.

**Question 5:** (a)

$$
\begin{aligned}
&Action(CallElevatorFloor1\\
&\qquad Precond: OnFloor1 \land AtElevatorFloor2\\
&\qquad Effect: AtElevatorFloor1 \land \neg AtElevatorFloor2)
\end{aligned}
$$

$$
\begin{aligned}
&Action(RideElevatorFLoor1\text{-}Floor2\\
&\qquad Precond: OnFloor1 \land AtElevatorFloor1\\
&\qquad Effect: OnFloor2 \land \neg OnFloor1 \land AtElevatorFloor2 \land \neg AtElevatorFloor1)
\end{aligned}
$$

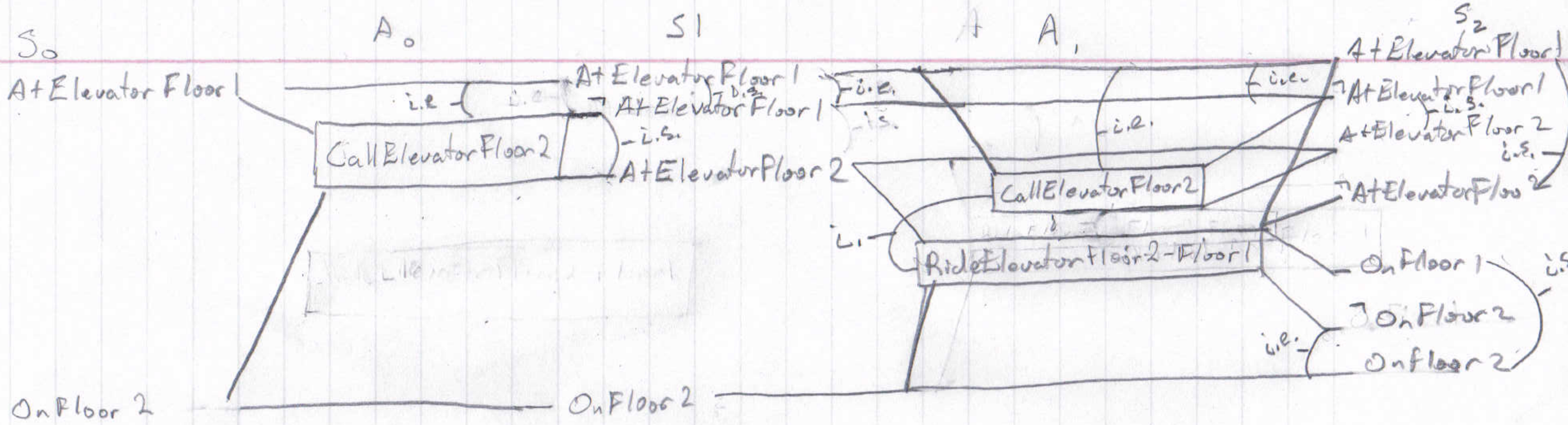
(b) See last page ...

(c) The problem is solved at the second level. The plan is

$$CallElevatorFloor2 \rightarrow RideElevatorFloor2\text{-}Floor1$$

**Question 6:** (a) If a literal does not appear in the final level of the planning graph it means that it is not a possible result of any action—if it were, it would persist to the final level.

(b) It is not necessary to add to the predecessor state the literals that are negative effects of the action because if the literal is a negative effect of the action, it could not have been negative in the predecessor state.



$S_0$
$A_0$
S1
$A_1$
$S_2$
AtElevatorFloor1
CallElevatorFloor2
OnFloor 2
i.e.
i.s.
AtElevatorFloor1
¬AtElevatorFloor1
AtElevatorFloor2
OnFloor2
CallElevatorFloor2
RideElevatorFloor2-Floor1
AtElevatorFloor1
¬AtElevatorFloor1
AtElevatorFloor2
¬AtElevatorFloor2
OnFloor1
¬OnFloor2
OnFloor2