Universität des Saarlandes
Fachrichtung 6.2 – Informatik
J. Siekmann, S. Autexier, K. Fischer, E. Melis, C. Zinn
M. Schiller, E. Schulz, R.Vollmann

9th Theoretical Assignment in Artificial Intelligence (WS 2006/2007) **Solutions**

Exercise 9.1 Consider the blocksworld with the operators **pickup, putdown, stack** and **unstack** defined in the lecture[1]. Assume that there are four blocks *A,B,C* and *D*, and a table. The start state is: *ON(A,B), ONTABLE(B), ONTABLE(D),ONTABLE(C),CLEAR(A), CLEAR(D),CLEAR(C),HANDEMPTY.*
The goal state is: *ON(C,A), ON(A,D).*

1. Find a plan that transforms the start state to the goal state according to the STRIPS algorithm presented in the lecture[2]. Hint: do not expand the whole search space – give one branch that leads to a plan! Indicate each step of the algorithm!
2. Explain what the Sussman-anomaly is. Can this effect occur in the above example?

Solution:

1. *We initialize* GOALS *with* ON(C,A), ON(A,D) *and we initialize* S *with* ON(A,B), ONTABLE(B), ONTABLE(D),ONTABLE(C),CLEAR(A), CLEAR(D),CLEAR(C),HANDEMPTY.
   - *pick: g = on(A,D)*
   - *pick: o = stack*
   - $\sigma = \{x \to A, y \to D\}$
   - *STRIPS ({holding(A), clear(D)})*
     – *pick: g = holding(A)*
     – *pick: o = unstack*
     – $\sigma = \{x \to A, y \to B\}$
     – *STRIPS ({handempty, clear(A), on(A,B)}) returns immediately, because these goals are all in S.*
     – *o' = unstack(A,B)*
     – *S = {holding(A), clear(B), ontable(B), ontable(D), ontable(C), clear(D), clear(C)}*
     – *clear(D) is already in S, return*
   - *o' = stack(A,D)*
   - *S {ontable(B), ontable(D), ontable(C), clear(C), handempty, on(A,D), clear(A)}*
   - *pick: g = on(C,A)*
   - *pick: o = stack*
   - $\sigma = \{x \to C, y \to A\}$
   - *STRIPS({holding(C), clear(A)})*
     – *pick: g = holding(C)*
     – *pick: o = pickup*
     – $\sigma = \{x \to C\}$

[1]Shown on slide 11, Chapter 4.
[2]Shown on slide 35, Chapter 4.

  - *STRIPS ({ontable(C), clear(C), handempty}) returns immediately, because these goals are all in S.*
  - *o' = pickup(C)*
  - *S = {ontable(B), ontable(D), on(A,D), clear(A), holding(C)}*
- *o' = stack(C,A)*
- *S = {ontable(B), ontable(D), on(A,D), handempty, on(C,A), clear(C)}*

So the final plan is:

- *unstack(A,B)*
- *stack(A,D)*
- *pickup(C)*
- *stack(C,A)*

2. *The classic STRIPS planning algorithm tries to find operators that make the literals in the goal true – in a non-deterministic way (it chooses the order of the literals that are considered, and it chooses among the choice of operators, if several are applicable). In our example, the algorithm could choose to satisfy $on(C,A)$ first, by picking up C and putting it on A, which achieves $on(C,A)$. However, if the algorithm then tries to make $on(A,D)$ true, it would have to take C off from A again (thus making $on(C,A)$ false again).*

   This is indeed an example of the Sussman anomaly, where in order to fulfill several goals, it becomes necessary to undo one goal that has been achieved in order to achieve another goal. This can prevent the planner from finding the plan with the minimal length, even if there exists one (which is demonstrated by the above example).

Exercise 9.2 Assume that Pat got drunk last night and has to go the supermarket to buy groceries. Unfortunately, Pat got involved in an onion eating contest and a mud-wrestling match last night. Pat is currently sleeping and is hungry. One last thing that can be said about Pat is that he is lazy and does not like to do anything more than he has to. How should Pat get ready to go to the supermarket?

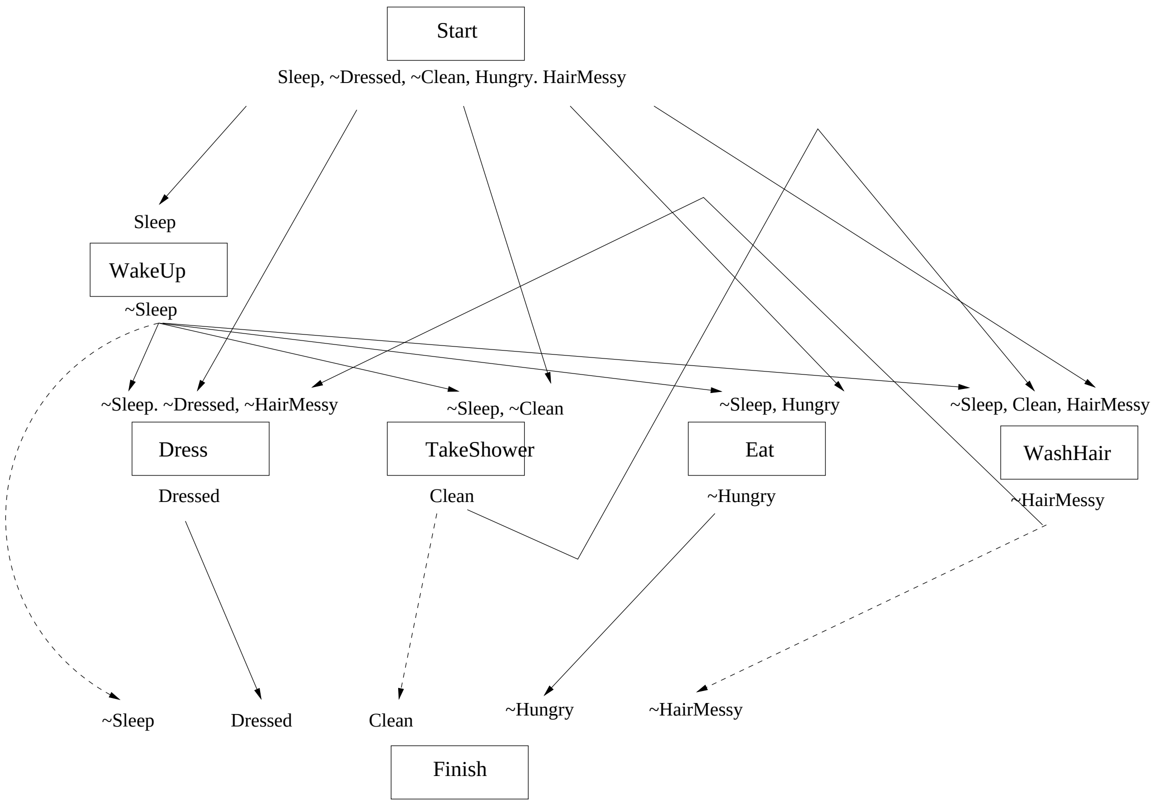
1. Define the problem by specifying the environment and operators. The environment should comprehend the states `HairMessy`, `Dressed`, `Clean`, `Hungry`, `Sleeping`.

Solution:

$$
\begin{array}{l}
Op(\textsc{Action}{:}WakeUp,\\
\quad \textsc{Precond}{:}Sleeping\\
\quad \textsc{Effect}{:}\neg Sleeping)\\[1em]
Op(\textsc{Action}{:}Eat,\\
\quad \textsc{Precond}{:}Hungry \wedge \neg Sleeping\\
\quad \textsc{Effect}{:}\neg Hungry)\\[1em]
Op(\textsc{Action}{:}TakeShower,\\
\quad \textsc{Precond}{:}\neg Clean \wedge \neg Sleeping\\
\quad \textsc{Effect}{:}Clean)\\[1em]
Op(\textsc{Action}{:}WashHair,\\
\quad \textsc{Precond}{:}HairMessy \wedge Clean \wedge \neg Sleeping\\
\quad \textsc{Effect}{:}\neg HairMessy)\\[1em]
Op(\textsc{Action}{:}Dress,\\
\quad \textsc{Precond}{:}\neg Dressed \wedge \neg Sleeping \wedge \neg HairMessy\\
\quad \textsc{Effect}{:}Dressed)
\end{array}
$$

2. Create a Minimal Partial Order Plan. How many Total Order Plans can be obtained from it?

Solution:



Exercise 9.3 Consider (again) a blockworlds in which there is a *table*, and the blocks A, B, and C. Moreover, there are predicates $Clear(x)$ where x is a block and $On(x, y)$, where x is a block and y is a block or the table. Furthermore, consider the following operators:

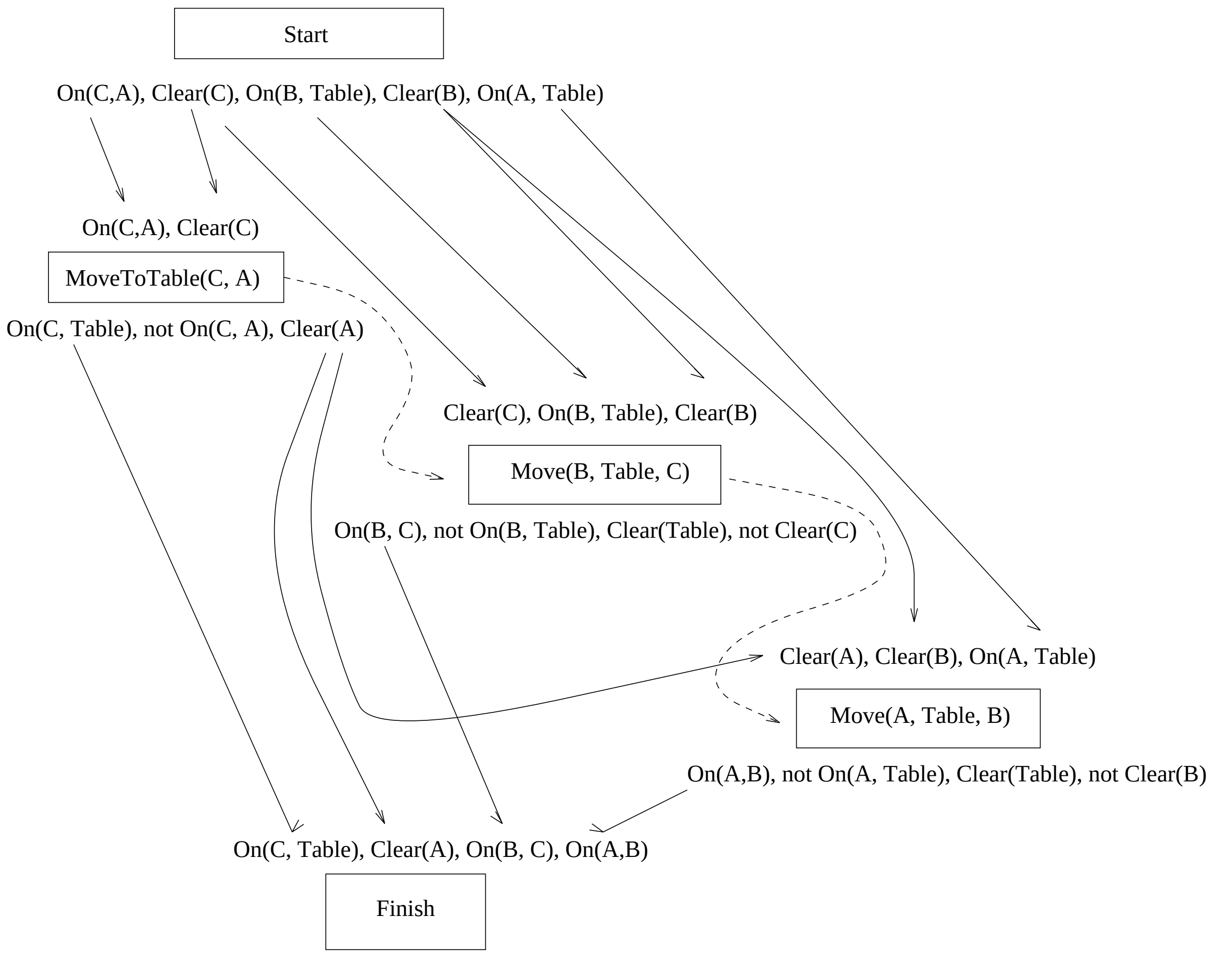
$Op(\textsc{Action}{:}Start,$
  $\textsc{Effect}{:}On(C, A) \wedge On(A, Table) \wedge Clear(C) \wedge On(B, Table) \wedge Clear(B))$
$Op(\textsc{Action}{:}Finish,$
  $\textsc{Precond}{:}On(A, B) \wedge On(B, C) \wedge On(C, Table) \wedge Clear(A))$
$Op(\textsc{Action}{:}Move(b, x, y),$
  $\textsc{Precond}{:}On(b, x) \wedge Clear(b) \wedge Clear(y)$
  $\textsc{Effect}{:}On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y))$
$Op(\textsc{Action}{:}MoveToTable(b, x),$
  $\textsc{Precond}{:}On(b, x) \wedge Clear(b)$
  $\textsc{Effect}{:}On(b, Table) \wedge Clear(x) \wedge \neg On(b, x))$

Explain in detail how a *partial order planner (POP)* generates a plan for the Sussman anomaly. Give the different steps during the planning process as well as a diagram that shows the final plan with causal links including preconditions and ordering constraints.

Solution:

S_{need}	c	S_{add}	Bindings
Finish	*On(A, B)*	*Move(x1,x2,x3)*	$x1/A, x3/B$
Move(A,x2,B)	*On(A,x2)*	*Start*	$x2/Table$
Move(A, Table, B)	*Clear(B)*	*Start*	
Finish	*On(C, Table)*	*MoveToTable(y1,y2)*	$y1/C$
MoveToTable(C, y2)	*On(C, y2)*	*Start*	$y2/A$
MoveToTable(C,A)	*Clear(C)*	*Start*	
Finish	*Clear(A)*	*MoveToTable(C,A)*	
Move(A, Table, B)	*Clear(A)*	*MoveToTable(C,A)*	
Finish	*On(B,C)*	*Move(z1,z2,z3)*	$z1/B, z3/C$
Move(B, z2, C)	*On(B, z2)*	*Start*	$z2/Table$
Move(B, Table, C)	*Clear(B)*	*Start*	
Move(B, Table, C)	*Clear(C)*	*Start*	

Threats: The `not Clear(C)` *effect of the action* `Move(B, Table, C)` *threatens the effect* `Clear(C)` *of the start action in* `MoveToTable(C,A)`. *To solve this problem,* `Move(B, Table, C)` *is shifted behind* `MoveToTable(C,A)`. *The effect of* `not Clear(B)` *of* `Move(A, Table, B)` *threatens the effect of* `Start` *which is used in the action* `Move(B, Table, C)`. *Therefore,* `Move(A, Table, B)` *is shifted behind* `Move(B, Table, C)`.


Start
On(C,A), Clear(C), On(B, Table), Clear(B), On(A, Table)
On(C,A), Clear(C)
MoveToTable(C, A)
On(C, Table), not On(C, A), Clear(A)
Clear(C), On(B, Table), Clear(B)
Move(B, Table, C)
On(B, C), not On(B, Table), Clear(Table), not Clear(C)
Clear(A), Clear(B), On(A, Table)
Move(A, Table, B)
On(A,B), not On(A, Table), Clear(Table), not Clear(B)
On(C, Table), Clear(A), On(B, C), On(A,B)
Finish