SAPIENZA Università di Roma – MSc. in Engineering in Computer Science
Artificial Intelligence & Machine Learning – June 14, 2024

Part 1 - Artificial Intelligence
(Time to complete the test: 2:30 hours)

Consider the following scenario. An agent (☺) moves in a map with cells A, B, C. Initially, cell A contains the agent and cells B and C have, respectively, a circle (○) and a star (⋆) drawn on the floor. The initial situation is depicted below:

A	B	C
☺	○	⋆

Assume the environment is modelled as follows.

Non-Fluents:

- $Shape(x)$, denoting that x is a shape.
- $Adjacent(x_1, x_2)$, denoting that cell x_1 and x_2 are adjacent.

Fluents:

- $AgentAt(x)$ denoting that the agent is at cell x.
- $ShapeAt(x, y)$ denoting that shape x is drawn on the ground of cell y.
- $AgentErased(x)$ denoting that, sometime in the past, the agent has erased shape x (from the ground of some cell).

Actions:

- $move(x, y)$, which allows the agent to move from cell x to y. The action can be done only if the agent is currently at cell x and cell y is adjacent to x. The effect is that the agent is in cell y (and no longer in x) and all shapes previously erased by the agent remain so.
- $erase()$, which allows the agent to erase the shape (if any) from the cell the agent is currently at. If the agent is on a cell containing a star, the action can be done only if the agent has erased all other shapes. The action can be done in all other cases. The effect is that the shape is no longer drawn on the cell and the agent has erased also this shape.
- $switchOff()$, which allows the agent to switch off. The action can be done only if there are no shapes drawn on any cell. The effect is that the agent is no longer in any cell (disappears from the map).

Initial situation:
Cell A is adjacent to B and viceversa; cell B is adjacent to C and viceversa. The agent is at cell A. There are two shapes: $circle$, at cell B, and $star$, at cell C. The agent has not erased any shape.

Exercise 1. First, formalize the above scenario as a Basic Action Theory.
Then, check by regression whether :

1. the action sequence $\varrho_1 = move(A, B); move(B, C); erase()$; is executable in S_0.
2. the action sequence $\varrho_2 = move(A, B); erase(); move(B, C); erase()$; results in a situation where cell C contains no shape.

Exercise 2. Consider the following goal: $\neg AgentAt(A) \wedge \neg AgentAt(B) \wedge \neg AgentAt(C)$. First formalize the above scenario as a PDDL domain file and a PDDL problem file. Then:

1. Draw the corresponding transition system;
2. Solve planning for achieving the above goal by using forward depth-first search (uninformed), reporting the steps of the forward search computation, and return the resulting plan.

Exercise 1. First, formalize the above scenario as a Basic Action Theory.
Then, check by regression whether :

1. the action sequence $\varrho_1 = move(A, B); move(B, C); erase();$ is executable in S_0.
2. the action sequence $\varrho_2 = move(A, B); erase(); move(B, C); erase();$ results in a situation where cell C contains no shape.

BASIC ACTION THEORY

PRECONDITION ACTIONS

$$Poss(move(x,y), s) \equiv AgentAt(x,s) \wedge Adjacent(x,y)$$
$$Poss(ERASE(), s) \equiv (\exists x.\, AgentAt(x,s) \wedge ShapeAt(STAR, x, s)) \supset (\forall y.\, Shape(y) \wedge y \neq STAR \supset AgentErased(y, s))$$
$$Poss(SwitchOff(), s) \equiv \forall x, y\ \neg ShapeAt(x, y, s)$$

SUCCESSOR STATE AXIOMS

EFFECT AXIOMS

$$a = move(x,y) \supset AgentAt(y, do(a,s)) \wedge (AgentAt(x,s) \supset \neg AgentAt(x, do(a,s)))$$

$$a = ERASE() \supset \forall x,y.\, (ShapeAt(x,y,s) \supset \neg ShapeAt(x,y,do(a,s))) \wedge (AgentAt(y,s) \supset AgentErased(x, do(a,s)))$$

$$a = SwitchOff() \supset \forall x.\, (AgentAt(x,s) \supset \neg AgentAt(x, do(a,s)))\ ]\ ?$$

NORMALIZE

$$\exists x\ a = move(x,y) \supset AgentAt(y, do(a,s))$$

$$\exists y.\, (a = move(x,y) \wedge AgentAt(x,s)) \supset \neg AgentAt(x, do(a,s))$$

$$(a = ERASE() \wedge ShapeAt(x,y,s)) \supset \neg ShapeAt(x,y,do(a,s))$$

$$\exists y.\, (a = ERASE() \wedge AgentAt(y,s)) \supset AgentErased(x, do(a,s))$$

$$(a = SwitchOff() \wedge AgentAt(x,s)) \supset \neg AgentAt(x, do(a,s))$$

EXPLANATION CLOSURE

$$AgentAt(x, do(a,s)) \equiv (\exists y\ a = move(y,x)) \vee (AgentAt(x,s) \wedge \neg((\exists y.\ a = move(x,y)) \vee (a = switchOff() \wedge AgentAt(x,s))))$$

$$\downarrow$$

$$AgentAt(x, do(a,s)) \equiv (\exists y\ a = move(y,x)) \vee (AgentAt(x,s) \wedge \neg((\exists y.\ a = move(x,y)) \vee a = switchOff()))$$

$$ShapeAt(x,y, do(a,s)) \equiv (ShapeAt(x,y,s) \wedge \neg(a = erase() \wedge ShapeAt(x,y,s)))$$

$$\downarrow$$

$$ShapeAt(x,y, do(a,s)) \equiv (ShapeAt(x,y,s) \wedge a \neq erase())$$

$$AgentErased(x, do(a,s)) \equiv (\exists y.\ a = erase() \wedge AgentAt(y,s)) \vee AgentErased(x,s)$$

INITIAL SITUATION

$Adjacent(x,y) \equiv (x=A \wedge y=B) \vee (x=B \wedge y=A) \vee (x=B \wedge y=C) \vee (x=C \wedge y=B)$
$AgentAt(x,S_0) \equiv x=A$
$Shape(x) \equiv x = circle \vee x = star$
$ShapeAt(x,y,S_0) \equiv (x = circle \wedge y=B) \vee (x=star \wedge y=C)$
$\forall x.\ \neg AgentErased(x, S_0)$

REGRESSION

1) $\varrho_1 = move(A,B); move(B,C); erase();$ ρ_1 EXECUTABLE IN S_0 ?

$S_1 = do(move(A,B), S_0)$
$S_2 = do(move(B,C), S_1)$
$S_3 = do(ERASE(), S_2)$

WE NEED TO CHECK IF:

move(A,B) IS EXECUTABLE IN S_0
move(B,C) IS EXECUTABLE IN S_1
ERASE() IS EXECUTABLE IN S_2

a. $D_0 \cup D_{una} \models R[Poss(move(A,B), S_0)]$
b. $D_0 \cup D_{una} \models R[Poss(move(B,C), S_1)]$
c. $D_0 \cup D_{una} \models R[Poss(ERASE(), S_2)]$

LET'S REGRESS EACH FORMULA:

a. $R[Poss(move(A,B), S_0)] =$
$R[AgentAt(A,S_0) \wedge Adjacent(A,B)] = AgentAt(A,S_0) \wedge Adjacent(A,B)$

TRUE → move(A,B) IS EXECUTABLE IN S_0 ✓

b. $R[Poss(move(B,C), S_1)] =$
$= R[AgentAt(B,S_1) \wedge Adjacent(B,C)] =$
$= R[AgentAt(B, do(move(A,B), S_0))] \wedge Adjacent(B,C) =$
$= R[(\exists y.\ move(A,B) = move(y,B) \vee (AgentAt(B,S_0) \wedge$
$\neg(\exists y.\ move(A,B) = move(B,y) \vee move(A,B) = SwitchOff())))]$
$\wedge Adjacent(B,C) =$
$= R[\exists y.\ move(A,B) = move(y,B)] \wedge Adjacent(B,C) =$
$= \exists y.\ move(A,B) = move(y,B) \wedge Adjacent(B,C)$

TRUE FOR y=A → move(B,C) IS EXECUTABLE IN S_0 ✓

c. $R[Poss(ERASE(), S_2)] =$
$= R[(\exists x.\ AgentAt(x,S_2) \wedge ShapeAt(STAR, x, S_2)) \supset$
$(\forall y.\ SHAPE(y) \wedge y \neq STAR \supset AgentErased(y, S_2))] =$
$= \exists x.(R[AgentAt(x,S_2)] \wedge R[ShapeAt(STAR, x, S_2)]) \supset$
$(\forall y.\ SHAPE(y) \wedge y \neq STAR \supset R[AgentErased(y, S_2)]) =$

$R[AgentAt(x,S_2)] = R[AgentAt(x, do(move(B,C), S_1))] =$
$= R[(\exists y.\ move(B,C) = move(y,x) \vee (AgentAt(x,S_1) \wedge$
$\neg(\exists y.\ move(B,C) = move(x,y) \vee move(B,C) = SwitchOff())))] =$
$= \exists y.\ move(B,C) = move(y,x) \vee (R[AgentAt(x,S_1)] \wedge \neg\exists y.\ move(B,C) = move(x,y))) =$

$R[AgentAt(x,S_1)] = R[AgentAt(x, do(move(A,B), S_0))] =$
$= R[(\exists y.\ move(A,B) = move(y,x) \vee (AgentAt(x,S_0) \wedge$
$\neg(\exists y.\ move(A,B) = move(x,y) \vee move(A,B) = SwitchOff())))] =$
$= \exists y.\ move(A,B) = move(y,x) \vee (AgentAt(x,S_0) \wedge \neg\exists y.\ move(A,B) = move(x,y)))$

$R[AgentAt(x,S_2)] = \exists y.\ move(B,C) = move(y,x) \vee (\exists y.\ move(A,B) = move(y,x)$
$\vee (AgentAt(x,S_0) \wedge \neg\exists y.\ move(A,B) = move(x,y)) \wedge \neg\exists y.\ move(B,C) = move(x,y))$

$R[ShapeAt(STAR, x, S_2)] = R[ShapeAt(STAR, x, do(move(B,C), S_1)] =$
$= R[ShapeAt(STAR, x, S_1) \wedge move(B,C) \neq ERASE()] =$
$= R[ShapeAt(STAR, x, S_1)] =$
$= R[ShapeAt(STAR, x, do(move(A,B), S_0))] =$
$= R[ShapeAt(STAR, x, S_0) \wedge move(A,B) \neq ERASE()] =$
$= R[ShapeAt(STAR, x, S_0)] = SHAPE(STAR, x, S_0)$

$R[AGENTERASED(y, S_2)] = R[AGENTERASED(y, DO(MOVE(B,C), S_1))] =$
$= R[(\exists x.\ MOVE(B,C) = ERASE() \wedge AGENTAT(x, S_1)) \vee AGENTERASED(y, S_1)] =$
$= R[AGENTERASED(y, S_1)] =$
$= R[AGENTERASED(y, DO(MOVE(A,B), S_0))] =$
$= R[(\exists x.\ MOVE(A,B) = ERASE() \wedge AGENTAT(x, S_0)) \vee AGENTERASED(y, S_0)] =$
$= R[AGENTERASED(y, S_0)] = AGENTERASED(y, S_0)$

$R[POSS(ERASE(), S_2)] =$
$= \exists x.(R[AGENTAT(x, S_2)] \wedge R[SHAPEAT(STAR, x, S_2)]) \supset$
$(\forall y.\ SHAPE(y) \wedge y \neq STAR \supset R[AGENTERASED(y, S_2)]) =$
$= (\exists x\ (\exists y.\ move(B,C) = move(y,x)) \vee (\exists y.\ move(A,B) = move(y,x)) \vee$
$(AGENTAT(x, S_0) \wedge \neg \exists y.\ move(A,B) = move(x,y)) \wedge \neg \exists y.\ move(B,C) = move(x,y))$
$\wedge SHAPE(STAR, x, S_0)) \supset$
$(\forall y.\ SHAPE(y) \wedge y \neq STAR \supset AGENTERASED(y, S_0)) =$

SINCE $AGENTERASED(y, S_0)$ IS FALSE :

$ERASE()$ FALSE $\longrightarrow$ NOT EXECUTABLE IN S_0 ✗

SO P_1 IS NOT EXECUTABLE IN S_0 ✗

2) $\varrho_2 = move(A,B); erase(); move(B,C); erase();$ CELL C CONTAINS NO SHAPE

$S_1 = DO(MOVE(A,B), S_0)$
$S_2 = DO(ERASE(), S_1)$
$S_3 = DO(MOVE(B,C), S_2)$
$S_4 = DO(ERASE(), S_3)$

WE NEED TO CHECK IF: $D_0 \cup D_{una} \models R[\neg \exists x.\ SHAPE(x) \wedge SHAPEAT(x, C, S_4)]$

$R[\neg \exists x.\ SHAPE(x) \wedge SHAPEAT(x, C, S_4)] = \neg \exists x.\ SHAPE(x) \wedge R[SHAPEAT(x, C, S_4)]$

$R[SHAPEAT(x, C, S_4)] = R[SHAPEAT(x, C, DO(ERASE(), S_3))] =$
$= R[SHAPEAT(x, C, S_3) \wedge ERASE() \neq ERASE()] =$ FALSE

$= \neg \exists x.\ SHAPE(x) \wedge FALSE = \neg \exists x.\ FALSE =$ TRUE

P_2 RESULTS IN A SITUATION WHERE CELL C CONTAINS NO SHAPE

Exercise 2. Consider the following goal: $\neg AgentAt(A) \wedge \neg AgentAt(B) \wedge \neg AgentAt(C)$. First formalize the above scenario as a PDDL domain file and a PDDL problem file. Then:

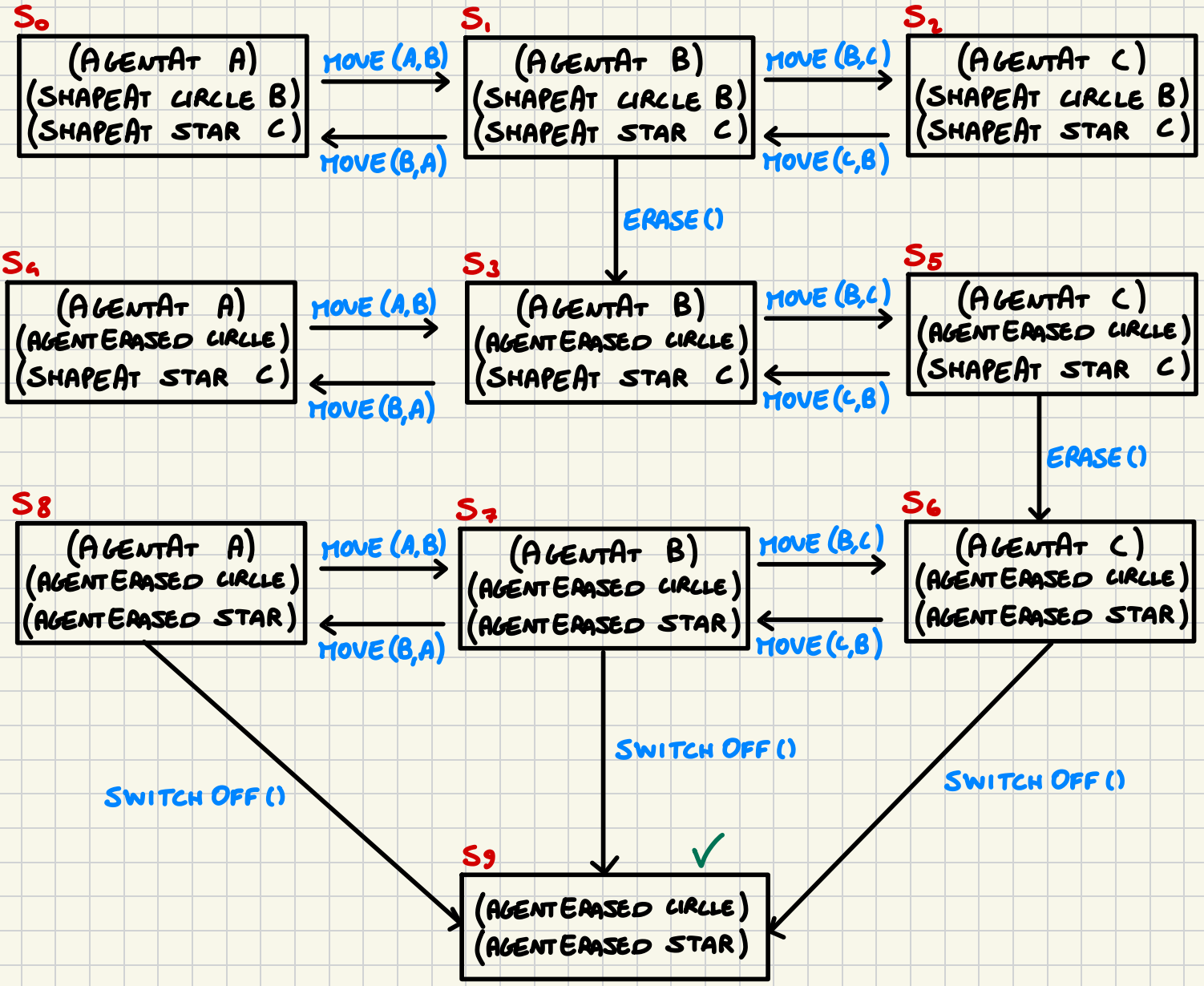
1. Draw the corresponding transition system;
2. Solve planning for achieving the above goal by using forward depth-first search (uninformed), reporting the steps of the forward search computation, and return the resulting plan.

PDDL

```
(DEFINE (DOMAIN SHAPE_DOMAIN)
  (:REQUIREMENTS :ADL)
  (:TYPES CELL SHAPE)
  (:CONSTANTS CIRCLE STAR - SHAPE  A B C - CELL)
  (:PREDICATES
    (SHAPE ?x - SHAPE)
    (ADJACENT ?x ?y - CELL)
    (AGENTAT ?x - CELL)
    (SHAPEAT ?x - SHAPE ?y - CELL)
    (AGENTERASED ?x - SHAPE)
  )
  (:ACTION MOVE
    :PARAMETERS (?x ?y - CELL)
    :PRECONDITION (AND (AGENTAT ?x)(ADJACENT ?x ?y))
    :EFFECT (AND (AGENTAT ?x)(NOT(AGENTAT ?y)))
  ); END OF MOVE
  (:ACTION ERASE
    :PARAMETERS ()
    :PRECONDITION (OR
                    (NOT(EXISTS (x? - CELL)(AND (AGENTAT ?x)(SHAPEAT STAR ?x))))
                    (FORALL (?y - SHAPE)
                      (OR (= ?y STAR)(AGENTERASED ?y))
                    )
                  )
    :EFFECT (FORALL (?x - SHAPE ?y - CELL)
              (AND
                (OR (NOT (SHAPEAT ?x ?y)(SHAPEAT ?x ?y)))
                (OR (NOT (AGENTAT ?y))(AGENTERASED ?x))
              )
            )
  ); END OF ERASE
  (:ACTION SWITCHOFF
    :PARAMETERS ()
    :PRECONDITION (FORALL (?x - SHAPE ?y - CELL)
                    (NOT (SHAPEAT ?x ?y))
                  )
    :EFFECT (FORALL (?x - CELL)
              (NOT (AGENTAT ?x))
            )
  ); END OF SWITCHOFF
); END OF DEFINE
```

```
(DEFINE (PROBLEM SHAPE_PROBLEM)(:DOMAIN SHAPE.DOMAIN)
  (:OBJECT )
  (:INIT (ADJACENT A B)(ADJACENT B A)(ADJACENT B C)(ADJACENT C B)
         (AGENTAT A)
         (SHAPEAT CIRCLE B)(SHAPEAT STAR C)
  )
  (:GOAL (FORALL (?x - CELL)
           (NOT (AGENTAT ?x))
         )
  )
); END OF DEFINE
```

1) goal: $\neg AgentAt(A) \wedge \neg AgentAt(B) \wedge \neg AgentAt(C)$



2) goal: $\neg AgentAt(A) \wedge \neg AgentAt(B) \wedge \neg AgentAt(C)$

0. $\mathcal{I} = [(S_0, \text{EMPTY})]$
   $m = \{S_0\}$

1. (STATE, PLAN) = $\mathcal{I}$.POP() (S_0, EMPTY)
   m ADD (S_1)
   $\mathcal{I}$.PUSH $(S_1, \text{MOVE}(A,B))$

   $\mathcal{I} = [(S_1, \text{MOVE}(A,B))]$ $m: \{S_0, S_1\}$

2. (STATE, PLAN) = $\mathcal{I}$.POP() $(S_1, \text{MOVE}(A,B))$
   m ADD (S_2)
   $\mathcal{I}$.PUSH $(S_2, \text{MOVE}(A,B)\ \text{MOVE}(B,C))$
   m ADD (S_3)
   $\mathcal{I}$.PUSH $(S_3, \text{MOVE}(A,B)\ \text{ERASE}())$

   $\mathcal{I} = [(S_3, \text{MOVE}(A,B)\ \text{ERASE}()),$
   $(S_2, \text{MOVE}(A,B)\ \text{MOVE}(B,C))]$ $m: \{S_0, S_1, S_2, S_3\}$

3. (STATE, PLAN) = $\mathcal{I}$.POP() $(S_3, \text{MOVE}(A,B)\ \text{ERASE}())$
   m ADD (S_4)
   $\mathcal{I}$.PUSH $(S_4, \text{MOVE}(A,B)\ \text{ERASE}()\ \text{MOVE}(B,A))$
   m ADD (S_5)
   $\mathcal{I}$.PUSH $(S_5, \text{MOVE}(A,B)\ \text{ERASE}()\ \text{MOVE}(B,C))$

   $\mathcal{I} = [(S_5, \text{MOVE}(A,B)\ \text{ERASE}()\ \text{MOVE}(B,C)),$
   $(S_4, \text{MOVE}(A,B)\ \text{ERASE}()\ \text{MOVE}(B,A)),$
   $(S_2, \text{MOVE}(A,B)\ \text{MOVE}(B,C))]$ $m: \{S_0, S_1, S_2, S_3, S_4, S_5\}$

4. (STATE, PLAN) = $\mathcal{I}$.POP() $(S_5, \text{MOVE}(A,B)\ \text{ERASE}()\ \text{MOVE}(B,C))$
   m ADD (S_6)
   $\mathcal{I}$.PUSH $(S_6, \text{MOVE}(A,B)\ \text{ERASE}()\ \text{MOVE}(B,C)\ \text{ERASE}())$

   $\mathcal{I} = [(S_6, \text{MOVE}(A,B)\ \text{ERASE}()\ \text{MOVE}(B,C)\ \text{ERASE}()),$
   $(S_4, \text{MOVE}(A,B)\ \text{ERASE}()\ \text{MOVE}(B,A)),$
   $(S_2, \text{MOVE}(A,B)\ \text{MOVE}(B,C))]$ $m: \{S_0, S_1, S_2, S_3, S_4, S_5, S_6\}$

5. (STATE, PLAN) = $\mathcal{I}$.POP() $(S_6, \text{MOVE}(A,B)\ \text{ERASE}()\ \text{MOVE}(B,C)\ \text{ERASE}())$
   m ADD (S_7)
   $\mathcal{I}$.PUSH $(S_7, \text{MOVE}(A,B)\ \text{ERASE}()\ \text{MOVE}(B,C)\ \text{ERASE}()\ \text{MOVE}(C,B))$
   m ADD (S_9)
   $\mathcal{I}$.PUSH $(S_9, \text{MOVE}(A,B)\ \text{ERASE}()\ \text{MOVE}(B,C)\ \text{ERASE}()\ \text{SWITCHOFF}())$

   $\mathcal{I} = [(S_9, \text{MOVE}(A,B)\ \text{ERASE}()\ \text{MOVE}(B,C)\ \text{ERASE}()\ \text{SWITCHOFF}()),$
   $(S_7, \text{MOVE}(A,B)\ \text{ERASE}()\ \text{MOVE}(B,C)\ \text{ERASE}()\ \text{MOVE}(C,B)),$
   $(S_4, \text{MOVE}(A,B)\ \text{ERASE}()\ \text{MOVE}(B,A)),$
   $(S_2, \text{MOVE}(A,B)\ \text{MOVE}(B,C))]$ $m: \{S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_9\}$

6. (STATE, PLAN) = $\mathcal{I}$.POP() $(S_9, \text{MOVE}(A,B)\ \text{ERASE}()\ \text{MOVE}(B,C)\ \text{ERASE}()\ \text{SWITCHOFF}())$

Exercise 3.

1. Given the propositional formula: $\phi = ((a \supset (b \wedge c)) \supset d)$:
   (a) Using Tseitin's transformation, transform ϕ into an equi-satisfiable CNF formula.
   (b) Using the DPLL algorithm, check whether ϕ is satisfiable.
2. Using the tableaux method, check whether $\phi \models (a \supset d) \supset (b \wedge c)$. If this is not the case, show a counter-model (i.e., a model satisfying ϕ but not $(a \supset d) \supset (b \wedge c)$), obtained from the tableau.

1) a. $\phi = (a \supset (b \wedge c)) \supset d$ $\quad x_1 = b \wedge c$ $\quad x_2 = a \supset x_1$ $\quad x_3 = x_2 \supset d$

$CNF(x_1) = (\neg x_1 \vee (b \wedge c)) \wedge (x_1 \vee \neg(b \wedge c)) = (\neg x_1 \vee b) \wedge (\neg x_1 \vee c) \wedge (x_1 \vee \neg b \vee \neg c)$

$CNF(x_2) = (\neg x_2 \vee (\neg a \vee x_1)) \wedge (x_2 \vee (a \wedge \neg x_1)) = (\neg x_2 \vee \neg a \vee x_1) \wedge (x_2 \vee a) \wedge (x_2 \vee \neg x_1)$

$CNF(x_3) = (\neg x_3 \vee (\neg x_2 \vee d)) \wedge (x_3 \vee (x_2 \wedge \neg d)) = (\neg x_3 \vee \neg x_2 \vee d) \wedge (x_3 \vee x_2) \wedge (x_3 \vee \neg d)$

$\phi' = x_3 \wedge (\neg x_3 \vee \neg x_2 \vee d) \wedge (x_3 \vee x_2) \wedge (x_3 \vee \neg d) \wedge (\neg x_2 \vee \neg a \vee x_1) \wedge (x_2 \vee a) \wedge (x_2 \vee \neg x_1) \wedge (\neg x_1 \vee b) \wedge (\neg x_1 \vee c) \wedge (x_1 \vee \neg b \vee \neg c)$

b. $\phi' = \{\{x_3\}, \{\neg x_3, \neg x_2, d\}, \{x_3, x_2\}, \{x_3, \neg d\}, \{\neg x_2, \neg a, x_1\}, \{x_2, a\}, \{x_2, \neg x_1\}, \{\neg x_1, b\}, \{\neg x_1, c\}, \{x_1, \neg b, \neg c\}\}$

UP) $x_3 = T \rightarrow \phi' = \{\{\neg x_2, d\}, \{\neg x_2, \neg a, x_1\}, \{x_2, a\}, \{x_2, \neg x_1\}, \{\neg x_1, b\}, \{\neg x_1, c\}, \{x_1, \neg b, \neg c\}\}$

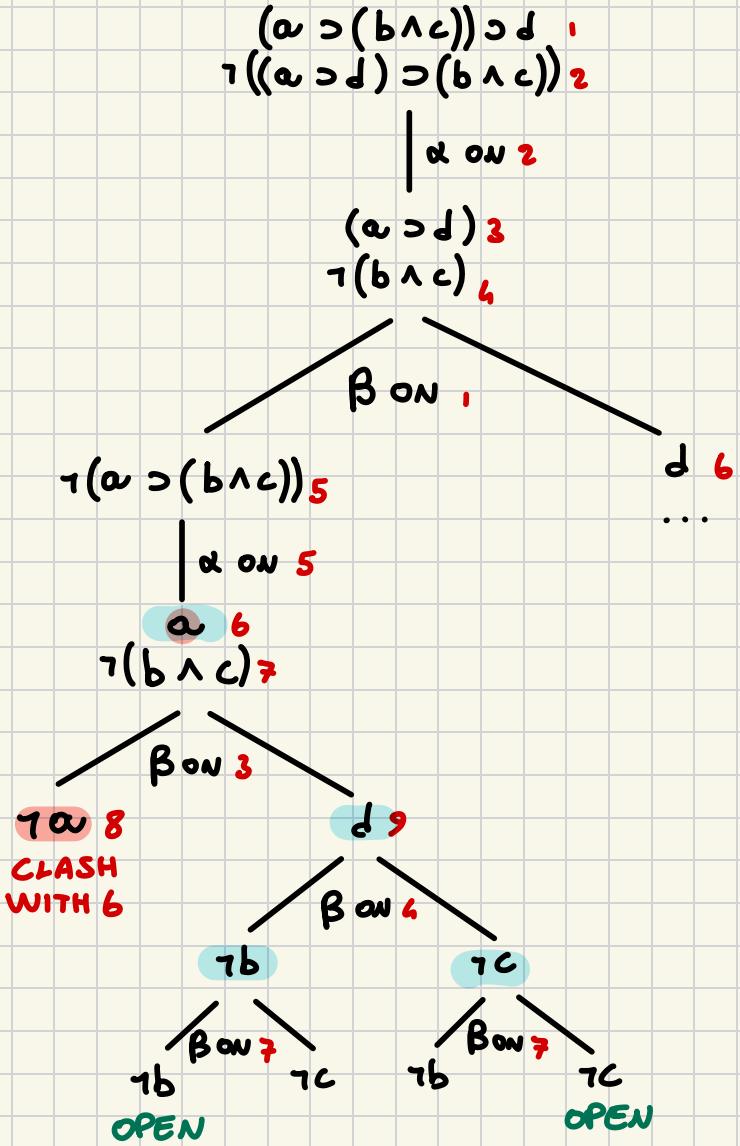
SR) $x_2 = F \rightarrow \phi' = \{\{a\}, \{\neg x_1\}, \{\neg x_1, b\}, \{\neg x_1, c\}, \{x_1, \neg b, \neg c\}\}$

UP) $x_1 = F \rightarrow \phi' = \{\{a\}, \{\neg b, \neg c\}\}$

UP) $a = T \rightarrow \phi' = \{\{\neg b, \neg c\}\}$

SR) $b = F \rightarrow \phi' = \{\}$ $\quad I = \{\neg b, a, \neg x_1, \neg x_2, x_3\}$

2) $(a \supset (b \wedge c)) \supset d \not\vdash (a \supset d) \supset (b \wedge c)$



SINCE THERE IS AN OPEN BRANCH $\phi \not\vdash (a \supset d) \supset (b \wedge c)$

WE NEED A MODEL I THAT SATISFY ϕ BUT NO $(a \supset d) \supset (b \wedge c)$

A MODEL CAN BE $I = \{a, d\} \rightarrow \neg b, \neg c$

↳ $\phi = (a \supset (b \wedge c)) \supset d =$
$= (T \supset (F \wedge F)) \supset T =$
$= (\neg T \vee F) \supset T =$
$= \neg F \vee T = T$

$(a \supset d) \supset (b \wedge c) =$
$= (\neg T \vee T) \supset (F \wedge F) =$
$= \neg T \vee F = F$

Exercise 3.

1. Given the propositional formula: $\phi = ((a \supset (b \wedge c)) \supset d)$:
   (a) Using Tseitin's transformation, transform ϕ into an equi-satisfiable CNF formula.
   (b) Using the DPLL algorithm, check whether ϕ is satisfiable.
2. Using the tableaux method, check whether $\phi \models (a \supset d) \supset (b \wedge c)$. If this is not the case, show a counter-model (i.e., a model satisfying ϕ but not $(a \supset d) \supset (b \wedge c)$), obtained from the tableau.

Solution

Precondition Axioms:

- $Poss(move(x, y), s) \equiv AgentAt(x, s) \wedge Adjacent(x, y)$
- $Poss(erase(), s) \equiv (\exists x.AgentAt(x, s) \wedge ShapeAt(star, x, s)) \supset (\forall y.Shape(y) \wedge y \neq star \supset AgentErased(y, s))$
- $Poss(switchOff(), s) \equiv \forall xy.\neg ShapeAt(x, y, s)$

Successor State Axioms:

1. Start with *Effect Axioms*:
   - $a = move(x, y) \supset (AgentAt(y, do(a, s)) \wedge \neg AgentAt(x, do(a, s)))$
   - $a = erase() \supset (AgentAt(y, s) \wedge ShapeAt(x, y, s)) \supset (\neg ShapeAt(x, y, do(a, s)) \wedge AgentErased(x, do(a, s)))$
   - $a = switchOff() \supset \neg AgentAt(x, do(a, s))$
2. Normalize Effect Axioms:
   - $(\exists x.a = move(x, y)) \supset AgentAt(y, do(a, s))$
   - $(\exists y.a = move(x, y)) \supset \neg AgentAt(x, do(a, s))$
   - $(AgentAt(y, s) \wedge ShapeAt(x, y, s) \wedge a = erase()) \supset \neg ShapeAt(x, y, do(a, s))$
   - $(\exists y.AgentAt(y, s) \wedge ShapeAt(x, y, s) \wedge a = erase()) \supset AgentErased(x, do(a, s))$
   - $a = switchOff() \supset \neg AgentAt(x, do(a, s))$
3. Apply *Explanation Closure* to obtain Successor-State Axioms:
   - $AgentAt(x, do(a, s)) \equiv (\exists y.a = move(y, x)) \vee (AgentAt(x, s) \wedge \neg((\exists y.a = move(x, y)) \vee a = switchOff()))$
   - $ShapeAt(x, y, do(a, s)) \equiv ShapeAt(x, y, s) \wedge \neg(AgentAt(x, s) \wedge ShapeAt(x, y, s) \wedge a = erase())$,
     which simplifies as:
     $ShapeAt(x, y, do(a, s)) \equiv ShapeAt(x, y, s) \wedge \neg(AgentAt(y, s) \wedge a = erase())$
   - $AgentErased(x, do(a, s)) \equiv (\exists y.AgentAt(y, s) \wedge ShapeAt(x, y, s) \wedge a = erase()) \vee AgentErased(x, s)$

Initial Situation:

$\mathcal{D}_{S_0}$ is the set including the following formulas:

- $Shape(x) \equiv x = circle \vee x = star$
- $Adjacent(x_1, x_2) \equiv (x_1 = A \wedge x_2 = B) \vee (x_1 = B \wedge x_2 = A) \vee (x_1 = B \wedge x_2 = C) \vee (x_1 = C \wedge x_2 = B)$
- $AgentAt(x, S_0) \equiv x = A$
- $ShapeAt(x, y, S_0) \equiv (x = circle \wedge y = B) \vee (x = star \wedge y = C)$
- $\forall x.\neg AgentErased(x, S_0)$

Regression:

For $\varrho_1 = move(A, B); move(B, C); erase()$, let:
$S_1 = do(move(A, B), S_0)$
$S_2 = do(move(B, C), S_1)$

To check whether ϱ_1 is executable in S_0, we need to check whether:
$move(A, B)$ is executable in S_0
$move(B, C)$ is executable in S_1
$erase()$ is executable in S_2

These are equivalent to checking whether:

- $\mathcal{D}_0 \cup \mathcal{D}_{una} \models \mathcal{R}[Poss(move(A, B), S_0)]$;
- $\mathcal{D}_0 \cup \mathcal{D}_{una} \models \mathcal{R}[Poss(move(B, C), S_1)]$;
- $\mathcal{D}_0 \cup \mathcal{D}_{una} \models \mathcal{R}[Poss(erase(), S_2)]$.

Let's regress the formulas.

- $\mathcal{R}[Poss(move(A, B), S_0)] = \mathcal{R}[AgentAt(A, S_0) \wedge Adjacent(A, B)] = AgentAt(A, S_0) \wedge Adjacent(A, B)$, which is $true$
- $\mathcal{R}[Poss(move(B, C), S_1)] = \mathcal{R}[AgentAt(B, S_1) \wedge Adjacent(B, C)] = \mathcal{R}[AgentAt(B, S_1)] \wedge Adjacent(B, C)$
  We have:
  $\mathcal{R}[AgentAt(B, S_1)] = \mathcal{R}[AgentAt(B, do(move(A, B), S_0))] =$
  $\mathcal{R}[\exists y.move(A, B) = move(y, B) \vee (AgentAt(B, S_0) \wedge \neg(\exists y.move(A, B) = move(B, y) \vee move(A, B) = switchOff()))] =$
  $\mathcal{R}[\exists y.move(A, B) = move(y, B) \vee (false \wedge \neg((\exists y.move(A, B) = move(B, y)) \vee move(A, B) = switchOff()))] =$
  $\mathcal{R}[\exists y.move(A, B) = move(y, B)] = \exists y.move(A, B) = move(y, B)$
  We thus have:
  $\mathcal{R}[Poss(move(B, C), S_1)] = \exists y.move(A, B) = move(y, B) \wedge Adjacent(B, C)$, which is $true$, for $y = A$
- $\mathcal{R}[Poss(erase(), S_2)] =$
  $\mathcal{R}[(\exists x.AgentAt(x, S_2) \wedge ShapeAt(star, x, S_2)) \supset (\forall y.Shape(y) \wedge y \neq star \supset AgentErased(y, S_2))] =$
  $(\exists x.\mathcal{R}[AgentAt(x, S_2)] \wedge \mathcal{R}[ShapeAt(star, x, S_2)]) \supset (\forall y.Shape(y) \wedge y \neq star \supset \mathcal{R}[AgentErased(y, S_2)])$

  We have:
  $\mathcal{R}[AgentAt(x, S_2)] = \mathcal{R}[AgentAt(x, do(move(B, C), S_1))] =$
  $\mathcal{R}[\exists y.move(B, C) = move(y, x) \vee (AgentAt(x, S_1) \wedge \neg(\exists y.move(B, C) = move(x, y) \vee move(B, C) = switchOff()))] =$
  $\mathcal{R}[\exists y.move(B, C) = move(y, x) \vee (AgentAt(x, S_1) \wedge \neg(\exists y.move(B, C) = move(x, y) \vee false))] =$
  $\mathcal{R}[\exists y.move(B, C) = move(y, x) \vee (AgentAt(x, S_1) \wedge \neg\exists y.move(B, C) = move(x, y))] =$
  $\exists y.move(B, C) = move(y, x) \vee (\mathcal{R}[AgentAt(x, S_1)] \wedge \neg\exists y.move(B, C) = move(x, y))$

  We have:
  $\mathcal{R}[AgentAt(x, S_1)] = \mathcal{R}[AgentAt(x, do(move(A, B), S_0)] =$ (see above) $=$
  $move(A, B) = move(A, x) \vee (\mathcal{R}[AgentAt(x, S_0)] \wedge \neg\exists y.move(A, B) = move(x, y)) =$
  $\exists y.move(A, B) = move(y, x) \vee (AgentAt(x, S_0) \wedge \neg\exists y.move(A, B) = move(x, y))$

  Thus: $\mathcal{R}[AgentAt(x, S_2)] =$
  $\exists y.move(B, C) = move(y, x) \vee ((\exists y.move(A, B) = move(y, x) \vee (AgentAt(x, S_0) \wedge \neg\exists y.move(A, B) = move(x, y))) \wedge$
  $\neg\exists y.move(B, C) = move(x, y))$, which is true only for $x = C$

  For $\mathcal{R}[ShapeAt(star, x, S_2)]$, we have:
  $\mathcal{R}[ShapeAt(star, x, do(move(B, C), S_1))] = \mathcal{R}[ShapeAt(star, x, S_1) \wedge \neg(AgentAt(x, S_1) \wedge move(B, C) = erase())] =$
  $\mathcal{R}[ShapeAt(star, x, S_1) \wedge \neg(AgentAt(x, S_1) \wedge false)] =$
  $\mathcal{R}[ShapeAt(star, x, S_1) \wedge true] =$
  $\mathcal{R}[ShapeAt(star, x, S_1)] =$
  $\mathcal{R}[ShapeAt(star, x, S_0) \wedge \neg(AgentAt(x, S_0) \wedge move(A, B) = erase())] = \mathcal{R}[ShapeAt(star, x, S_0)] =$
  $ShapeAt(star, x, S_0)$, which is true only for $x = C$

  Now, consider the righthand-side formula

  $$\forall y.Shape(y) \wedge y \neq star \supset \mathcal{R}[AgentErased(y, S_2)].$$

  The only interesting case is when $Shape(y) \wedge y \neq star$ holds. In the other cases, the entire formula is trivially $true$.

We have:
$\mathcal{R}[AgentErased(y, S_2)] = \mathcal{R}[AgentErased(y, do(move(B, C), S_1))] =$
$\mathcal{R}[(\exists x.AgentAt(x, S_1) \wedge ShapeAt(y, x, S_1) \wedge move(B, C) = erase()) \vee AgentErased(y, S_1)] =$
$\mathcal{R}[(\exists x.AgentAt(x, S_1) \wedge ShapeAt(y, x, S_1) \wedge false) \vee AgentErased(y, S_1)] =$
$\mathcal{R}[false \vee AgentErased(y, S_1)] =$
$\mathcal{R}[AgentErased(y, S_1)] =$
$\mathcal{R}[AgentErased(y, do(move(A, B), S_0))] =$
$\mathcal{R}[(\exists x.AgentAt(x, S_0) \wedge ShapeAt(y, x, S_0) \wedge move(A, B) = erase()) \vee AgentErased(y, S_0)] =$
$\mathcal{R}[false \vee AgentErased(y, S_0)] = AgentErased(y, S_0)$, which is $false$ for every y, thus making the entire righthand-side formula $false$, as this must be evaluated on every shape y, in particular, when y is $circle$ (thus $y \neq star$).

Finally, considering:
$\mathcal{R}[Poss(erase(), S_2)] =$
$(\exists x.\mathcal{R}[AgentAt(x, S_2)] \wedge \mathcal{R}[ShapeAt(star, x, S_2)]) \supset (\forall y.Shape(y) \wedge y \neq star \supset \mathcal{R}[AgentErased(y, S_2)])$, we have that the lefthand-side formula is $true$ (for $x = C$), while the righthand one is $false$ (when $y = circle$), thus making $false$ the entire formula $\mathcal{R}[Poss(erase(), S_2)]$.

Since $erase()$ is not executable in S_2, we conclude that ϱ_1 is not executable in S_0.

For $\varrho_2 = move(A, B); erase(); move(B, C); erase()$, let:
$S_1 = do(move(A, B), S_0)$
$S_2 = do(erase(), S_1)$
$S_3 = do(move(B, C), S_2)$
$S_4 = do(erase(), S_3)$

To check whether the action sequence ϱ_2 results in a situation where cell C contains no shape, need to check whether

$$\mathcal{D}_0 \cup \mathcal{D}_{una} \models \mathcal{R}[\neg\exists x.Shape(x) \wedge ShapeAt(x, C, S_4)]$$

Let's regress the formula:
$\mathcal{R}[\neg\exists x.Shape(x) \wedge ShapeAt(x, C, S_4)] =$
$\neg\exists x.Shape(x) \wedge \mathcal{R}[ShapeAt(x, C, S_4)]$

We have:
$\mathcal{R}[ShapeAt(x, C, S_4)] =$
$\mathcal{R}[ShapeAt(x, C, do(erase(), S_3)] =$
$\mathcal{R}[ShapeAt(x, C, S_3) \wedge \neg(AgentAt(C, S_3) \wedge erase() = erase())] =$
$\mathcal{R}[ShapeAt(x, C, S_3) \wedge \neg(AgentAt(C, S_3) \wedge true)] =$
$\mathcal{R}[ShapeAt(x, C, S_3) \wedge \neg AgentAt(C, S_3)] =$
$\mathcal{R}[ShapeAt(x, C, S_3)] \wedge \neg\mathcal{R}[AgentAt(C, S_3)]$

We have:
$\mathcal{R}[AgentAt(C, S_3)] =$
$\mathcal{R}[\exists y.move(B, C) = move(y, x) \vee (AgentAt(C, S_2) \wedge \neg((\exists y.move(B, C) = move(x, y)) \vee move(B, C) = switchOff()))] =$
$\mathcal{R}[true \vee (AgentAt(C, S_2) \wedge \neg((\exists y.move(B, C) = move(x, y)) \vee move(B, C) = switchOff()))] =$
$\mathcal{R}[true] = true$

Thus:
$\mathcal{R}[ShapeAt(x, C, S_4)] =$
$\mathcal{R}[ShapeAt(x, C, S_3)] \wedge \neg\mathcal{R}[AgentAt(C, S_3)] =$
$\mathcal{R}[ShapeAt(x, C, S_3)] \wedge \neg true =$
$\mathcal{R}[ShapeAt(x, C, S_3)] \wedge false = false$

Therefore:
$\mathcal{R}[\neg\exists x.Shape(x) \wedge ShapeAt(x, C, S_4)] =$
$\neg\exists x.Shape(x) \wedge \mathcal{R}[ShapeAt(x, C, S_4)] =$
$\neg\exists x.Shape(x) \wedge false =$
$\neg\exists x.false =$
$\neg false = true$

Thus, ϱ_2 results in a situation where cell C contains no shape.

PDDL:
Domain file:

```
(define (domain shapes_domain)
```

```
  (:requirements :adl)
  (:types cell shape)
  (:constants star - shape)
  (:predicates
    (adjacent ?x1 ?x2 - cell)
    (agentAt ?x - cell)
    (shapeAt ?x - shape ?y - cell)
    (agentErased ?x - shape)
  )

  (:action move
    :parameters (?x ?y - cell)
    :precondition (and (agentAt ?x) (adjacent ?x ?y))
    :effect (and (agentAt ?y) (not (agentAt ?y)))
  ); end of move

  (:action erase
    :parameters ()
    :precondition
        (or
            (not (exists (?x - cell) (and (agentAt ?x) (shapeAt star ?x))))
            (forall (?y - shape) (or (= ?y star) (agentErased ?y)))
        )
    :effect
        (forall (?x - cell ?y - shape)
            (when
                (and (agentAt ?x) (shapeAt ?y ?x))
                (and (not (shapeAt ?y ?x)) (agentErased ?y))
            )
        )
  ); end of erase

  (:action switchOff
    :parameters ()
    :precondition (forall (?x - cell ?y - shape) (not (shapeAt ?y ?x)))
    :effect (forall (?x-cell) (not (agentAt ?x)))
  ); end of switchOff

); end of define domain
```

Problem file:

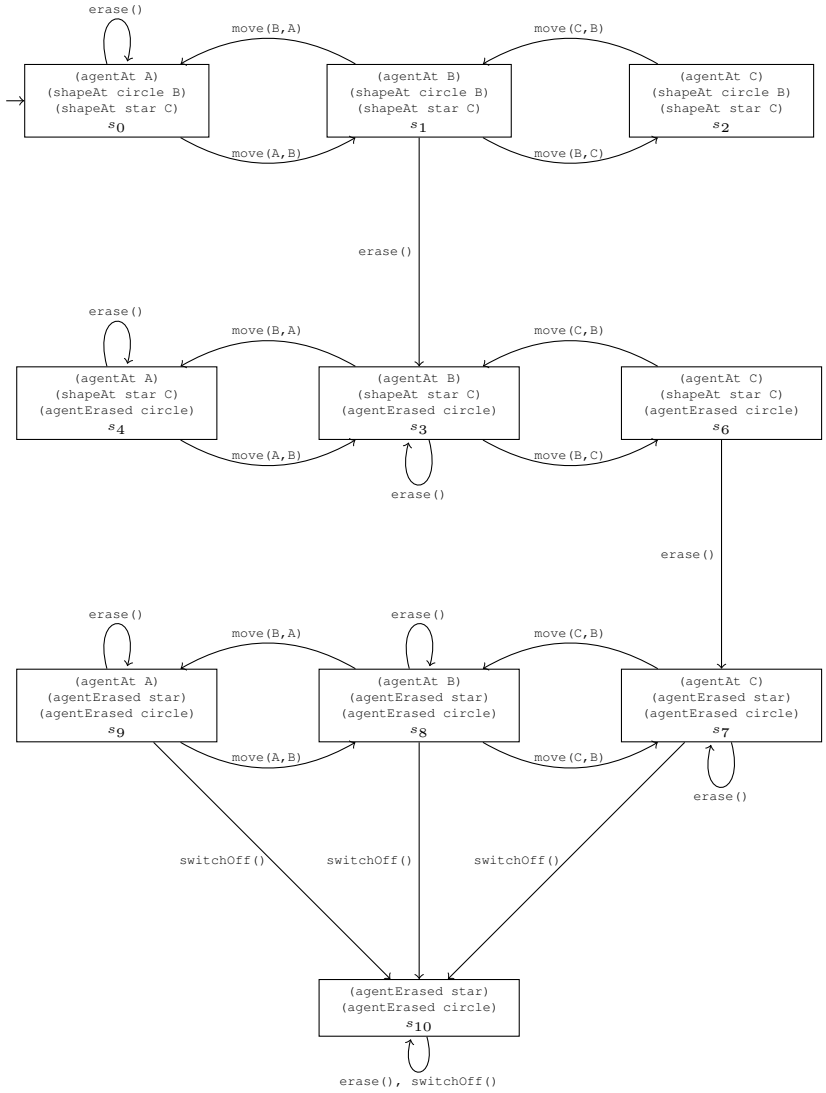
```
(define (problem grid_problem) (:domain shapes_domain)

  (:objects
    A B C - cell
    circle - shape; star already defined as constant
  )

  (:init
    (adjacent A B) (adjacent B A) (adjacent B C) (adjacent C B)
    (agentAt A)
    (shapeAt circle B) (shapeAt star C)
  )

  (:goal (and (not (agentAt A)) (not (agentAt B)) (not (agentAt C))))
) ; end of define problem
```

Transition System:



DFS Algorithm (variant with marking check before pushing):

```
DFS(domain d, state init, formula goal){
  // d:  input domain;
  // init:  initial state;
  // goal:  goal formula;
  Stack t = [(init, empty)]; // Open set (stack)
  Set m = {init}; // Marked states
  while (!t.empty()) {
    (state,plan) = t.pop(); // plan is action sequence followed to reach s
    if (s ⊨ goal) return plan; // if s satifies goal state, return plan
    forall (actions a executable in s)
      s'=d.delta(s,a) // s' is successor of s under a (d.delta is transition function of d)
      if (s' ∉ m) {// if s' not marked
        m.add(s'); // mark s'
        t.push((s',plan·a)); // add s' with plan extended by a, to open set t}
  }
  return noplan; // no plan found
}
```

Current node, open set and marked states at end of every iteration of call
`DFS(shapes_domain,`s_0`, (and (not (agentAt A)) (not (agentAt B)) (not (agentAt C))):`

0. `t = [(`s_0`, empty)]`
   `m =` $\{s_0\}$
1. `(state,plan)=t.pop() // (`s_0`, empty)`
   `m.add(`s_1`)`
   `t.push((`s_1`, move(A,B)))`

   State of `t` and `m` at end of iteration:
   `t=[(`s_1`, move(A,B))]`
   `m =` $\{s_0, s_1\}$
2. `(state,plan)=t.pop() // (`s_1`, move(A,B))`
   `m.add(`s_2`)`
   `t.push((`s_2`, move(A,B) move(B,C)))`
   `m.add(`s_3`)`
   `t.push((`s_3`, move(A,B) erase()))`

   State of `t` and `m` at end of iteration:
   `t = [(`s_3`, move(A,B) erase()),`
   `(`s_2`, move(A,B) move(B,C))]`
   `m =` $\{s_0, s_1, s_2, s_3\}$
3. `(state,plan)=t.pop() // (`s_3`, move(A,B) erase())`
   `m.add(`s_4`)`
   `t.push((`s_4`, move(A,B) erase() move(B,A)))`
   `m.add(`s_6`)`
   `t.push((`s_6`, move(A,B) erase() move(B,C)))`

   State of `t` and `m` at end of iteration:
   `t = [(`s_6`, move(A,B) erase() move(B,C)),`
   `(`s_4`, move(A,B) erase() move(B,A)),`
   `(`s_2`, move(A,B) move(B,C))]`
   `m =` $\{s_0, s_1, s_2, s_3, s_4, s_6\}$
4. `(state,plan)=t.pop()=(`s_6`, move(A,B) erase() move(B,C))`
   `m.add(`s_7`)`
   `t.push((`s_7`, move(A,B) erase() move(B,C)) erase())`

   State of `t` and `m` at end of iteration:
   `t = [(`s_7`, move(A,B) erase() move(B,C)) erase()),`
   `(`s_4`, move(A,B) erase() move(B,A)),`
   `(`s_2`, move(A,B) move(B,C))]`
   `m =` $\{s_0, s_1, s_2, s_3, s_4, s_6, s_7\}$
5. `(state,plan)=t.pop()=(`s_7`, move(A,B) erase() move(B,C)) erase())`
   `m.add(`s_8`)`
   `t.push((`s_8`, move(A,B) erase() move(B,C) erase() move(C,B)))`
   `m.add(`s_{10}`)`
   `t.push((`s_{10}`, move(A,B) erase() move(B,C) erase() switchOff()))`

   State of `t` and `m` at end of iteration:
   `t = [(`s_{10}`, move(A,B) erase() move(B,C) erase() switchOff()),`
   `(`s_8`, move(A,B) erase() move(B,C) erase() move(C,B)),`
   `(`s_4`, move(A,B) erase() move(B,A)),`
   `(`s_2`, move(A,B) move(B,C))]`
   `m =` $\{s_0, s_1, s_2, s_3, s_4, s_6, s_7, s_8, s_{10}\}$
6. `(state,plan)=t.pop()=(`s_{10}`, move(A,B) erase() move(B,C) erase() switchOff())`
   `return plan`

Returned plan: `plan=move(A,B) erase() move(B,C) erase() switchOff())`

DPLL:

Let's transform $\phi = (a \supset (b \wedge c)) \supset d$ into an equi-satisfiable CNF formula, using Tseitin's transformation.

First, name every non-literal subformula with a fresh proposition:

- $x_1 \equiv (b \wedge c)$
- $x_2 \equiv (a \supset x_1)$
- $x_3 \equiv (x_2 \supset d)$

Then, compute CNF form for every equivalence:

- $CNF(x_1 \equiv (b \wedge c)) = (\neg x_1 \vee (b \wedge c)) \wedge (x_1 \vee \neg(b \wedge c)) = (\neg x_1 \vee b) \wedge (\neg x_1 \vee c) \wedge (x_1 \vee \neg b \vee \neg c)$
- $CNF(x_2 \equiv (a \supset x_1)) = (\neg x_2 \vee (\neg a \vee x_1)) \wedge (x_2 \vee (a \wedge \neg x_1)) = (\neg x_2 \vee \neg a \vee x_1) \wedge (x_2 \vee a) \wedge (x_2 \vee \neg x_1)$
- $CNF(x_3 \equiv (x_2 \supset d)) = (\neg x_3 \vee \neg x_2 \vee d) \wedge (x_3 \vee x_2) \wedge (x_3 \vee \neg d)$

The corresponding equi-satisfiable formula is:

$\phi' = x_3 \wedge (\neg x_3 \vee \neg x_2 \vee d) \wedge (x_3 \vee x_2) \wedge (x_3 \vee \neg d) \wedge (\neg x_2 \vee \neg a \vee x_1) \wedge (x_2 \vee a) \wedge (x_2 \vee \neg x_1) \wedge (\neg x_1 \vee b) \wedge (\neg x_1 \vee c) \wedge (x_1 \vee \neg b \vee \neg c)$

We can now apply DPLL to ϕ':

$DPLL(\{\{x_3\}, \{\neg x_3, \neg x_2, d\}, \{x_3, x_2\}, \{x_3, \neg d\}, \{\neg x_2, \neg a \vee x_1\}, \{x_2, a\}, \{x_2, \neg x_1\}, \{\neg x_1, b\}, \{\neg x_1, c\}, \{x_1, \neg b, \neg c\}\}, \{\}) =$
$UnitPropagation(\{\{x_3\}, \{\neg x_3, \neg x_2, d\}, \{x_3, x_2\}, \{x_3, \neg d\}, \{\neg x_2, \neg a, x_1\}, \{x_2, a\}, \{x_2, \neg x_1\}, \{\neg x_1, b\}, \{\neg x_1, c\}, \{x_1, \neg b, \neg c\}\}, \{\}) =$
$(\{\{\neg x_2, d\}, \{\neg x_2, \neg a, x_1\}, \{x_2, a\}, \{x_2, \neg x_1\}, \{\neg x_1, b\}, \{\neg x_1, c\}, \{x_1, \neg b, \neg c\}\}, \{x_3\})$

$DPLL(\{\{\neg x_2, d\}, \{\neg x_2, \neg a, x_1\}, \{x_2, a\}, \{x_2, \neg x_1\}, \{\neg x_1, b\}, \{\neg x_1, c\}, \{x_1, \neg b, \neg c\}\}, \{x_3\})$:
Split on x_2 (can split on any other literal):
$UnitPropagation(\{\{\neg x_2, d\}, \{\neg x_2, \neg a, x_1\}, \{x_2, a\}, \{x_2, \neg x_1\}, \{\neg x_1, b\}, \{\neg x_1, c\}, \{x_1, \neg b, \neg c\}, \{x_2\}\}, \{x_3\}) =$
$(\{\{d\}, \{\neg a, x_1\}, \{\neg x_1, b\}, \{\neg x_1, c\}, \{x_1, \neg b, \neg c\}\}, \{x_2, x_3\})$

$DPLL(\{\{d\}, \{\neg a, x_1\}, \{\neg x_1, b\}, \{\neg x_1, c\}, \{x_1, \neg b, \neg c\}\}, \{x_2, x_3\})$:
$UnitPropagation(\{\{d\}, \{\neg a, x_1\}, \{\neg x_1, b\}, \{\neg x_1, c\}, \{x_1, \neg b, \neg c\}\}, \{x_2, x_3\}) =$
$(\{\{\neg a, x_1\}, \{\neg x_1, b\}, \{\neg x_1, c\}, \{x_1, \neg b, \neg c\}\}, \{x_2, x_3, d\})$

$DPLL(\{\{\neg a, x_1\}, \{\neg x_1, b\}, \{\neg x_1, c\}, \{x_1, \neg b, \neg c\}\}, \{x_2, x_3, d\})$:
Split on $\neg x_1$:
$UnitPropagation(\{\{\neg a, x_1\}, \{\neg x_1, b\}, \{\neg x_1, c\}, \{x_1, \neg b, \neg c\}, \{\neg x_1\}\}, \{x_2, x_3, d\}) =$
$(\{\{\neg a\}, \{\neg b, \neg c\}\}, \{x_2, x_3, d, \neg x_1\})$

$DPLL(\{\{\neg a\}, \{\neg b, \neg c\}\}, \{x_2, x_3, d, \neg x_1\})$:
$UnitPropagation(\{\{\neg a\}, \{\neg b, \neg c\}\}, \{x_2, x_3, d, \neg x_1\}) =$
$(\{\{\neg b, \neg c\}\}, \{x_2, x_3, d, \neg x_1, \neg a\})$

$DPLL(\{\{\neg b, \neg c\}\}, \{x_2, x_3, d, \neg x_1, \neg a\})$:
Split on $\neg b$:
$UnitPropagation(\{\{\neg b, \neg c\}, \{\neg b\}\}, \{x_2, x_3, d, \neg x_1, \neg a\}) =$
$(\{\}, \{x_2, x_3, d, \neg a, \neg b\}) = (\{\}, \{x_2, x_3, d, \neg x_1, \neg a, \neg b\})$

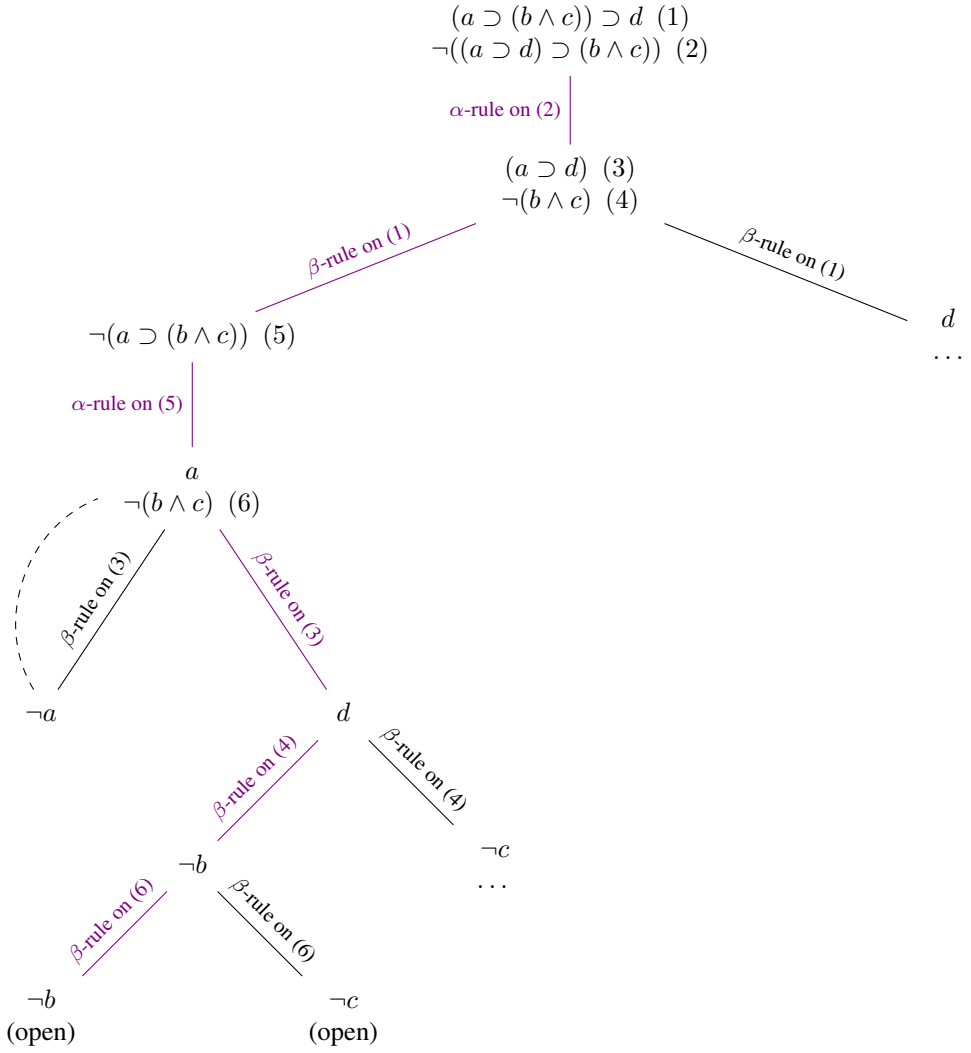
$DPLL(\{\}, \{x_2, x_3, d, \neg x_1, \neg a, \neg b\}) = (\{\}, \{x_2, x_3, d, \neg x_1, \neg a, \neg b\})$

Thus, ϕ is satisfiable and a model is $I = \{d\}$.

Tableau:

Checking whether $(a \supset (b \wedge c)) \supset d \models (a \supset d) \supset (b \wedge c)$, is equivalent to checking whether the KB $\mathcal{K} = \{(a \supset (b \wedge c)) \supset d, \neg((a \supset d) \supset (b \wedge c))\}$ is unsatisfiable.

Let's construct the tableau for $\mathcal{K}$:



Since the tableau contains an open branch (there is no need to complete the construction), $\phi \not\models (a \supset d) \supset (b \wedge c)$.

To obtain a counter-model, we take all the literals occurring in some arbitrary open path. By taking, e.g., the coloured path, we obtain $I = \{a, d\}$, which is such that $I \models \phi$ but $I \not\models (a \supset d) \supset (b \wedge c)$.

Observe that since (4) and (6) match, there is no need to expand both on the same path, as they produce the same effect. We have reported their expansion for completeness only.