Problem #9.10

A popular children's riddle is "Brothers and sisters have I none, but that man's father is my father's son." Use the rules of family domain (Section 8.3.2 on page 301) to show who that man is. You may apply any of the inference methods described in this chapter. Why do you think that this riddle is difficult?

This problem in prenex normal form is:

$$\forall x \forall y \exists z \left(\neg\, Brother(me,x) \wedge \neg\, Sister(me,y) \wedge Son\big(Father(z), Father(me)\big)\right)$$

This is in conjunctive normal form.

The existential quantifier can be removed by making z into a function of x and y. Hence:

$$\forall x \forall y \left(\neg\, Brother(me,x) \wedge \neg\, Sister(me,y) \wedge Son\left(Father\big(f(x,y)\big), Father(me)\right)\right)$$

Since there are only universal quantifiers remaining, these can be dropped resulting in:

$$\neg\, Brother(me,x) \wedge \neg\, Sister(me,y) \wedge Son\left(Father\big(f(x,y)\big), Father(me)\right)$$

If a person a is the son of person b (i.e. $Son(a,b)$ is true), then the relation can be rewritten:

$$Father(a) = b$$

This is still a binary expression since if a different constant other than a was inside the $Father$ function, then the relation may not be true. This substitution allows us to rewrite the riddle expression as:

$$\neg\, Brother(me,x) \wedge \neg\, Sister(me,y) \wedge \left(Father\left(Father\big(f(x,y)\big)\right) = Father(me)\right)$$

Note the requirement of the $Father$s being $Male$ is not included for brevity.

Any person only has one $Father$; what is more, I have no siblings who could also have the same father. Hence, the outer $Father$ functions can be dropped. This simplifies the equation to:

$$\neg\, Brother(me,x) \wedge \neg\, Sister(me,y) \wedge \left(Father\big(f(x,y)\big) = me\right)$$

Using the previously described relation that transformed the Son predicate to the $Father$ function, this operation can be reversed to change the expression to:

$$\neg\, Brother(me,x) \wedge \neg\, Sister(me,y) \wedge Son(f(x,y), me)$$

Therefore, $f(x,y)$ (which was the original z) is simply the my son (by substitution) (i.e. $\{f(x,y)/son\ of\ me\}$).

This riddle is not terribly difficult. However, it obfuscates $Father(z) = me$ by wrapping the me object in what are complementary operations since me has no brothers.

Problem #9.23

From "Horses are animals," it follows that "The head of a horse is the head of an animal." Demonstrate that this inference is valid by carrying out the following steps:

a. Translate the premise and the conclusion into the language of first order logic. Use three predicates: $\boldsymbol{HeadOf(h,x)}$ (meaning "$\boldsymbol{h}$ is the head of $\boldsymbol{x}$"), $\boldsymbol{Horses(x)}$, and $\boldsymbol{Animal(x)}$.

The premise of this statement is "Horses are animals". Rewritten in first-order logic with the defined predicates, this statement is:

$$\forall x\big(Horse(x) \Rightarrow Animal(x)\big)$$

The conclusion of this statement is:

$$\forall y \forall h \exists z\big(HeadOf(h,y) \wedge Horse(y) \Rightarrow HeadOf(h,z) \wedge Animal(z)\big)$$

b. Negate the conclusion, and convert the premise and the negated conclusion into conjunctive normal form.

By definition:

$$Premise \Rightarrow Conclusion$$

To perform refutation, negate the conclusion and show that:

$$Premise \wedge \neg\ Conclusion \Leftrightarrow \{\}$$

The premise is already in prenex normal form so the quantifiers can be dropped resulting in:

$$Horse(x) \Rightarrow Animal(x)$$

This can be made into a single clause through implication elimination.

$$\overline{Horse(x)} \vee Animal(x)$$

In the conclusion, the existential quantifier can be replaced by making z a function of y and h (i.e. $f(y,h)$). Hence, the conclusion becomes:

$$\forall y \forall h\left(HeadOf(h,y) \wedge Horse(y) \Rightarrow HeadOf\big(h,f(y,h)\big) \wedge Animal\big(f(y,h)\big)\right)$$

Again, since all variables are bounded by a universal quantifier, the universal quantifier(s) can be dropped making the statement:

$$HeadOf(h,y) \wedge Horse(y) \Rightarrow HeadOf\big(h,f(y,h)\big) \wedge Animal\big(f(y,h)\big)$$

When implication elimination is applied to this equation, the result is:

$$\neg\left(HeadOf(h,y) \wedge Horse(y)\right) \vee \left(HeadOf\big(h,f(y,h)\big) \wedge Animal\big(f(y,h)\big)\right)$$

To perform resolution refutation, the conclusion is negated. This results in:

$$HeadOf(h,y) \wedge Horse(y) \wedge \left(\overline{HeadOf\big(h, f(y,h)\big)} \vee \overline{Animal\big(f(y,h)\big)}\right)$$

The conjunction of the premise and the negation of the conclusion is taken. It results in:

$$\left(\overline{Horse(x)} \vee Animal(x)\right) \wedge HeadOf(h,y) \wedge Horse(y) \wedge \left(\overline{HeadOf\big(h, f(y,h)\big)} \vee \overline{Animal\big(f(y,h)\big)}\right)$$

This is in CNF format.

c. Use resolution to show that the conclusion follows from the premise.

Unification involves applying substitutions to the clauses in an expression in order to use resolution.

Step #1: Apply substitution $\{f(y,h)/y\}$. This simplifies the expression to:

$$\left(\overline{Horse(x)} \vee Animal(x)\right) \wedge HeadOf(h,y) \wedge Horse(y) \wedge \left(\overline{HeadOf(h,y)} \vee \overline{Animal(y)}\right)$$

Step #2: The second and fourth clauses can be resolved to achieve the new clause:

$$\frac{HeadOf(h,y),\ \overline{HeadOf(h,y)} \vee \overline{Animal(y)}}{\overline{Animal(y)}}$$

Step #3: Apply substitution $\{y/x\}$. This simplifies the expression to:

$$\left(\overline{Horse(x)} \vee Animal(x)\right) \wedge HeadOf(h,x) \wedge Horse(x) \wedge \left(\overline{HeadOf(h,x)} \vee \overline{Animal(y)}\right) \wedge \overline{Animal(x)}$$

Step #4: The first and third clauses can be combined to achieve the new clause:

$$\frac{\overline{Horse(x)} \vee Animal(x),\ \ Horse(x)}{Animal(x)}$$

Step #5: The clauses from step #2 and step #5 resolve to the empty set proving this statement by resolution.

$$\frac{Animal(x),\ \overline{Animal(x)}}{\{\}}$$

Additional Problem #1

Draw the planning graph for the problem in figure 10.3 in the book. Solve the problem step-by-step using the GraphPlan algorithm.

Figure 1 shows the initial and goal states of the Block World.

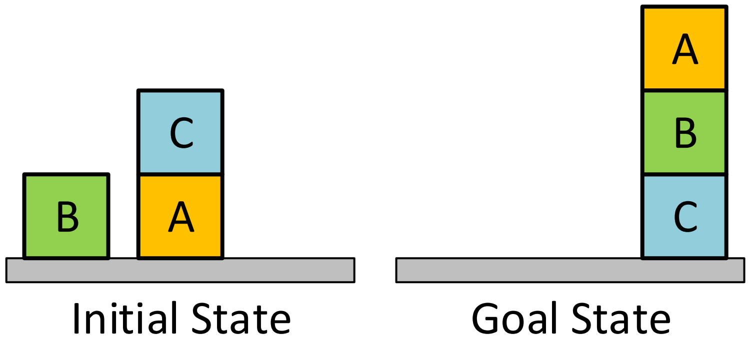

Figure 1 – Block World Initial and Goal States

The literals and actions in this world are below.

Literals:

- $Block(x)$ – A predicate for whether x is a block. **Note:** In the subsequent figures, the precondition conditions from the $Block$ literals to the actions are not shown for increased readability.
- $On(x, y)$ – A predicate for whether block x is on top of y, where y can be another block or the $Table$.
- $Clear(x)$ – A predicate for whether there is a clear space above block x where another block could be placed.

Actions:

- $Move(x, y, z)$ – Moves block x from y to z.
- $MoveToTable(x, y)$ – Moves block x from block y to the $Table$.

Additional Notes: The inequality preconditions (e.g. $x \neq y$) are not shown in the following figures also for increased readability. What is more, $Clear(Table)$ literals are not shown since according to the interpretation in the textbook, this literal is always true.

Figure 2 is the planning graph for the ground actions for the Blocks world. From the initial state, there are three possible, non-persistence actions. They are: moving block C to the table, moving block C on top of block B, and moving block B on top of block C.

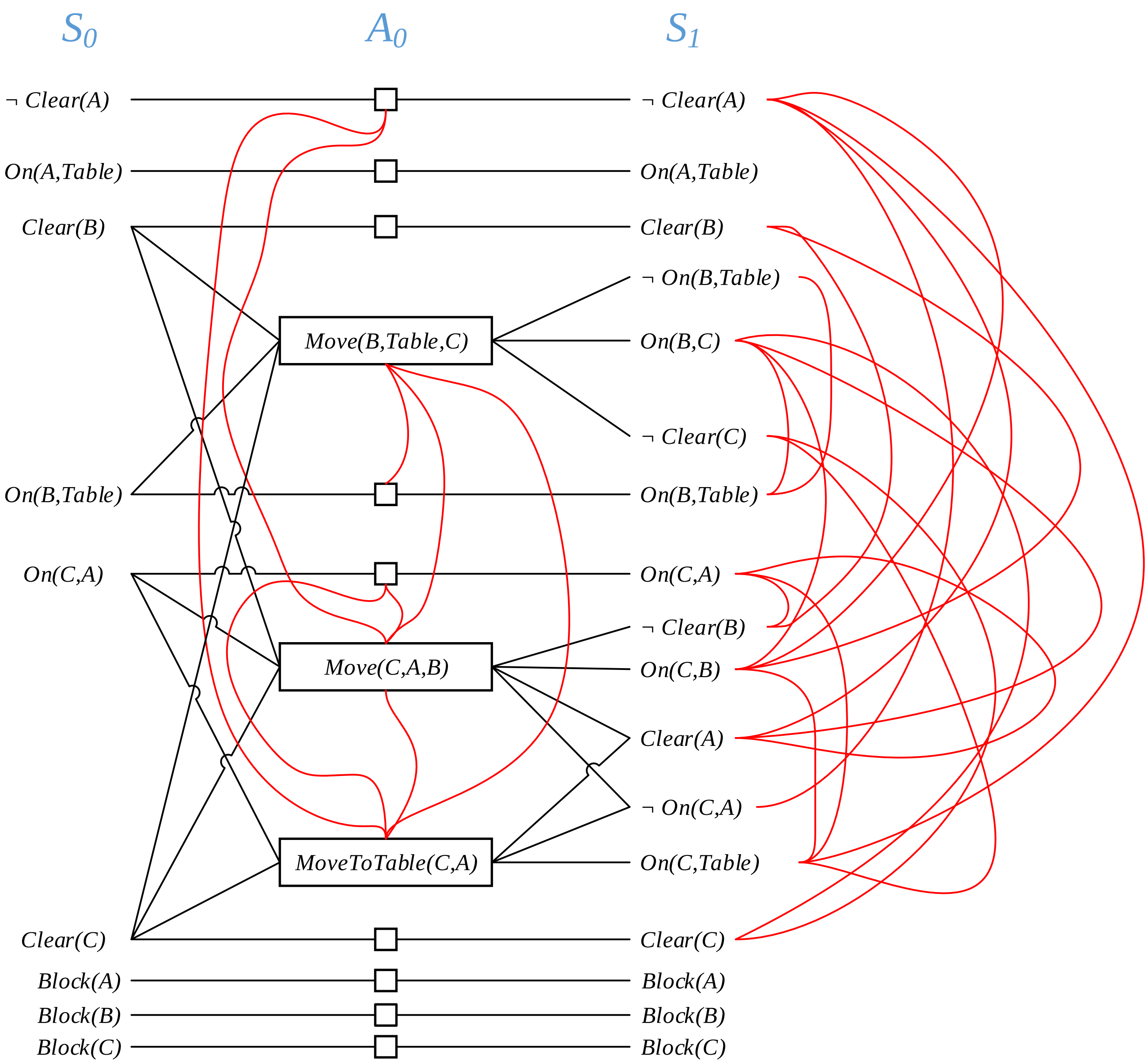

Figure 2 – Block World Ground Actions (A_0) and Mutex Relations

In line with the standard planning graph notation from class, preconditions are on the left side of the actions (actions are shown inside rectangles) while the effects are on the right side of the actions. Mutex relations are shown as red curved lines; the mutex relations not shown in this figure include: $Move(B, Table, C) \leftrightarrow Clear(C)$ and $Move(C, A, B) \leftrightarrow Clear(B)$. In subsequent levels, the existing (i.e. preceding) mutex relations decrease monotonically; actions increase monotonically, and literals increase monotonically. Therefore, any literals or actions shown between S_i and S_{i+1} will also be present between S_{i+1} and S_{i+2}; however, some (but not all) mutex relations have the potential to be dropped.

For all actions after the ground action (i.e. A_i where $i > 0$), Figure 3 shows the possible set of moves for an arbitrary block z.[1] If block z is clear, then other than the persistence actions, the two movement actions that can be performed on block z are:

[1] Block A in action A_1 is an exception to this statement because after A_0, block A cannot perform the $MoveToTable$ action; this is because regardless of what A_0 was, block A will always be on the table in state S_1.

1. $Move(z, x, y)$ – This represents the two actions where block z is moved from x (where x can be either a block other than z or the $Table$[2]) to block y.

2. $MoveToTable(z, w)$ – Action where block z can be moved from on top of another block w to the $Table$.

Note w in the action $MoveToTable(z, w)$ could be the same as x from the action $Move(z, x, y)$. However, a different symbol w is used here to denote that x is either the $Table$ or a block while w is exclusively a block. Depending on whether x and w are the same blocks, then there may be additional mutex relations between $On(z, w)$, $\neg On(z, w)$, $On(z, x)$, and $\neg On(z, x)$ which are not shown in Figure 3. In addition, as with Figure 2, the preconditions for the $Block$ literals are excluded for increased readability.

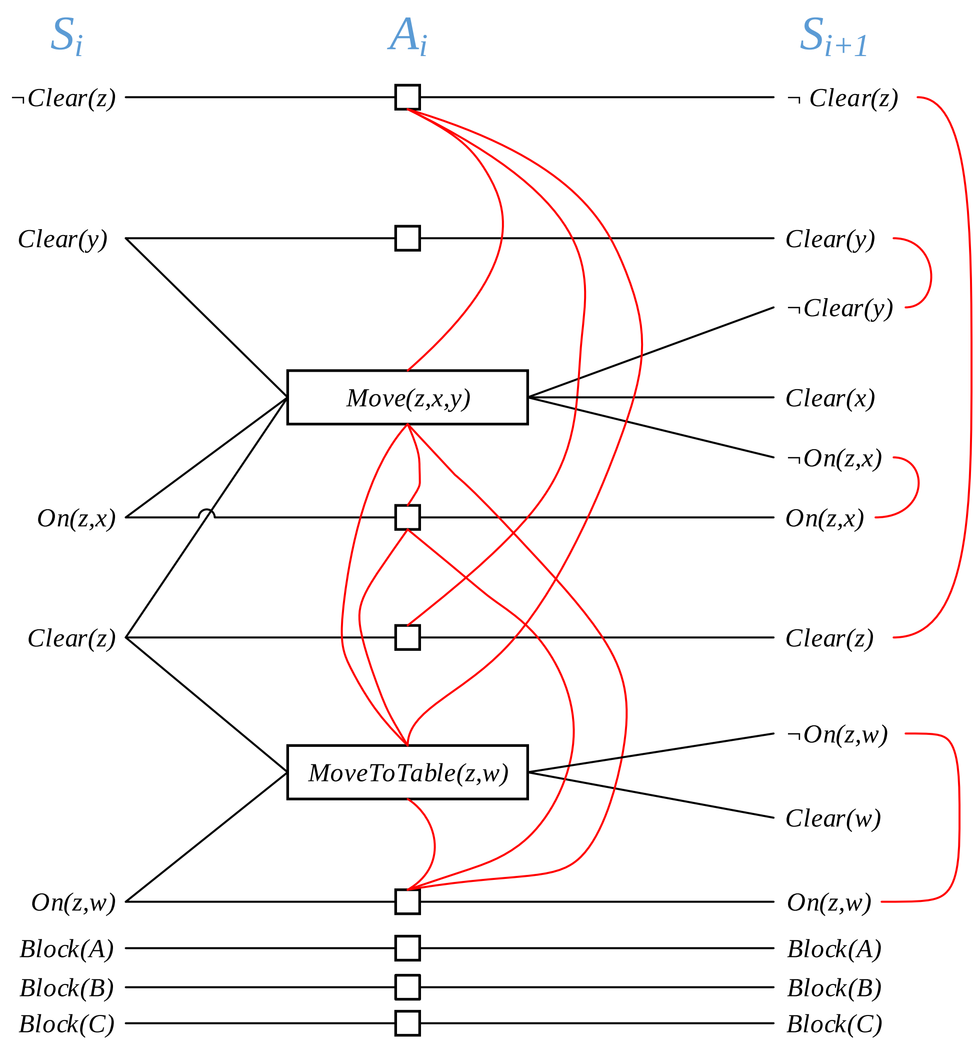

Figure 3 – Simplified Set of Generic Actions for a Block z in Action A_i where $i > 0$

Figure 3 represents the actions for block z. When the actions for the blocks other than z are included, then there will be additional mutex relations as not all actions and literals for this state become legal. For example, the action $Move(z, x, y)$ is mutually exclusive with the action $Move(x, y, z)$, $Move(y, z, x)$, etc. Similarly, a persistence action for $Clear(x)$ would be mutually exclusive with the action $Move(z, x, y)$.

[2] Note that if x is the $Table$, then the effect literal $Clear(x)$ is not applicable as the $Table$ is always clear by the problem definition.

Similar to the additional mutex relations on actions, there are additional mutex relations on literals that would necessarily be added once the blocks other than z are added for level A_i. For example, $On(z, w)$ is mutually exclusive with $On(w, z)$. What is more, $On(z, w)$ is mutually exclusive with $Clear(w)$ in the same way that $On(w, z)$ is mutually exclusive with $Clear(z)$. These additional mutex relations are not captured in the single block actions shown in Figure 3.

The arbitrary move for block z would be apply to all three blocks A, B, and C for actions A_1, A_2, and A_3 at which point the graph would have leveled-off.

Solving the Problem Using Graph Plan

Figure 4 is pseudocode for the GraphPlan algorithm.

```
function GraphPlan(problem) returns a solution or failure
    graph := INITIAL_PLANNING_GRAPH(problem)
    goals := CONJUNCTS(problem.GOAL)
    nogoods := {} # Empty hash table
    for t = 0 to ∞ do
        if goals all non-mutex to S_t of graph then
            solution := EXTRACT-SOLUTION(graph, goals, NUMLEVELS(graph), nogoods)
            if solution ≠ failure then return solution

        if graph and nogoods have both leveled off then return failure
        graph := EXPAND_GRAPH(graph, problem)
```

Figure 4 – Pseudocode for the Graph Plan Algorithm

Step #1: Building the Initial Planning Graph

The ground action in the planning graph is shown in Figure 2, and subsequent actions for blocks after A_0 are shown in Figure 3. A simplified graph with no mutex relations and the block preconditions not shown is in Figure 5. Excluding the persistence moves, it contains only the necessary moves to reach the goal.

Step #2: Express the Goal as a Conjunction of Literals.

The goal can be expressed as:

$$Goal \coloneqq On(C, Table) \wedge On(B, C) \wedge On(A, B)$$

Step #3: Check if the Goals all Non-Mutex in S_0

In S_0, none of the goal literals are met. Moreover, the graph has not yet leveled off. As such, EXPAND_GRAPH is called to add the actions from A_0, updates the effects (i.e. literals), and then updates the set of mutex relations.

Step #4: Check if the Goals all Non-Mutex in S_1

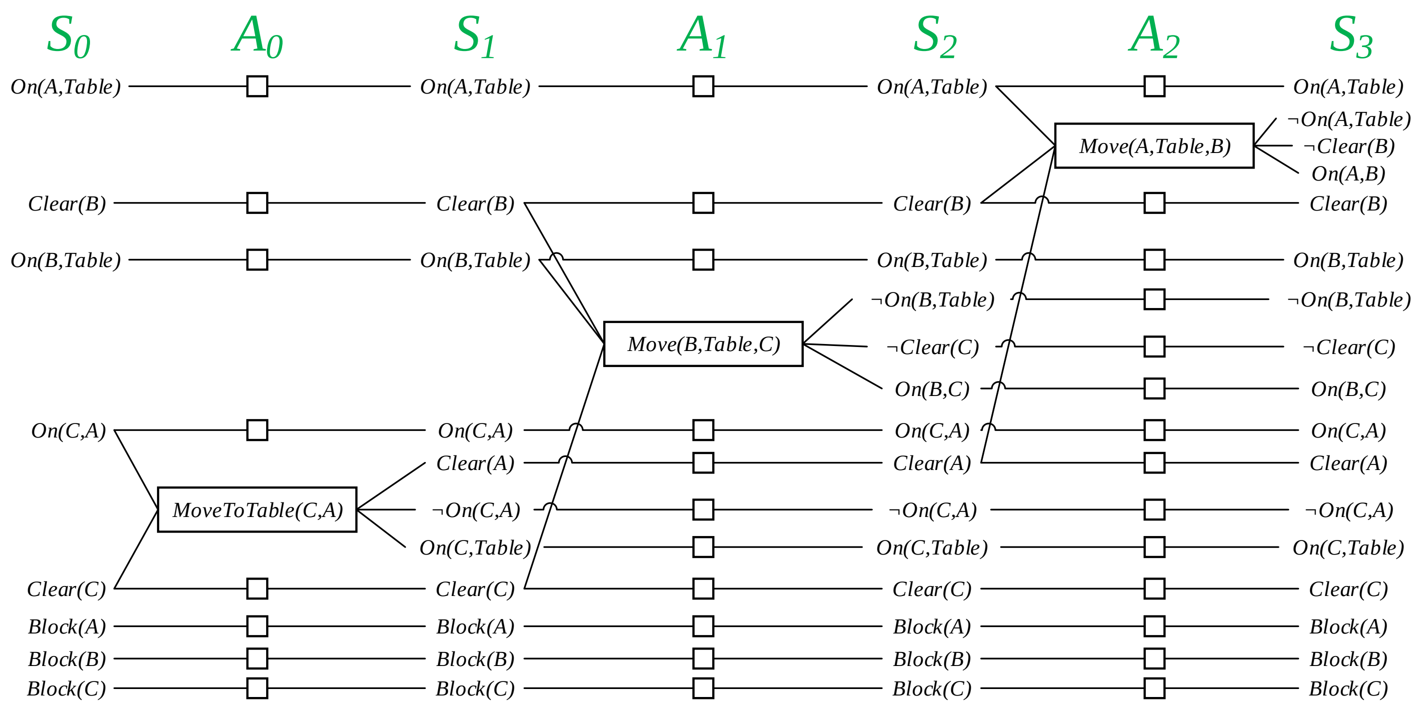

Figure 5 – Simplified Graph of the Necessary Actions to Reach the Goal for the Blocks World Problem

Additional Problem #2

Briefly explain how PDDL solves the frame problem. Given some disadvantages of formulating problems in PDDL.

As with the previous definition of a search problem, the four core items that that the Planning Domain Definition Language (PDDL) utilizes are:

1. Initial State
2. Actions available in each state
3. Result of applying an action
4. Goal Test

A state is a conjunction of fluents (i.e. facts that my change from situation to situation). The fluents are ground in that they do not rely on variables.

When performing an action, the result must explicitly define those aspects of the state that changed and those which stayed the same. The frame problem encapsulates the issue of defining what stayed the same.

By definition, classical planning focuses on those types of problems where most aspects of a state do not change when an action is performed. As such, for each action, PDDL only enumerates those aspects of the state that change. Any unmentioned aspect of the state is unchanged by the action.

First, PDDL fluents also do not explicitly include time. While preconditions refer to a time t and effects to a time $t+1$, this discretized representation of time will not be sufficient for all types of problems. Scheduling problems require information about time including how long an action takes and when it occurs. For example, with the "Air Cargo Transport" problem, actions can be ordered, but the PDDL representation has no sense of things like departure and arrival times of the aircraft. A temporal language would be better suited to this role.

Second, PDDL does not effectively capture the cost associated with an action. Instead, it generalizes action costs to a "level cost" which is the distance in levels from the initial state to the level in the planning graph where the action appears.   This oversimplification will be insufficient if the planning agent behaves more as a utility based agent than a goal based agent. For example, consider a variant of the air cargo problem where cargo must be moved from JFK to SFO with the minimum possible cost. If the only routes from JFK to SFO were through London or Kansas City, PDDL would not capture that the route through Kansas City would cost significantly less than the London itinerary.

Two additional general limitations of all planning languages are the qualification and ramification problems. The **qualification problem** highlights that there are some aspects of the environment that may cause an action to fail. What is more, these implicit and necessary preconditions for the success of an action can be innumerable and unknowable for practical purposes. For example, the Fly action in the "Air Cargo Transport" problem requires sufficient fuel in the tank, a competent pilot, good weather, no intentional sabotage, etc.; otherwise the Fly action will fail. However, the textbook's PDDL description of this action does not capture these depend.

The **ramification problem** states that when performing an action, there are many secondary effects that are not always captured. For example, when the Fly action is performed, some of the airline's gasoline reserve is consumed. Moreover, after a Fly action, in addition to the movement of a package, some airline staff as well as possibly customers are moved to a new location. However, these tertiary effects can not all be practically captured by the planning language.