# 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:

This is in conjunctive normal form.

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

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

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

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

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

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

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

Therefore, (which was the original ) is simply the my son (by substitution) (i.e. ).

This riddle is not terribly difficult. However, it obfuscates by wrapping the object in what are

complementary operations since 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:**

1. **Translate the premise and the conclusion into the language of first order logic. Use three predicates: (meaning “ is the head of ”), , and .**

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

The conclusion of this statement is:

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

By definition:

To perform refutation, negate the conclusion and show that:

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

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

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

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

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

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

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

This is in CNF format.

1. **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 . This simplifies the expression to:

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

**Step #3:** Apply substitution . This simplifies the expression to:

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

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

# 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.

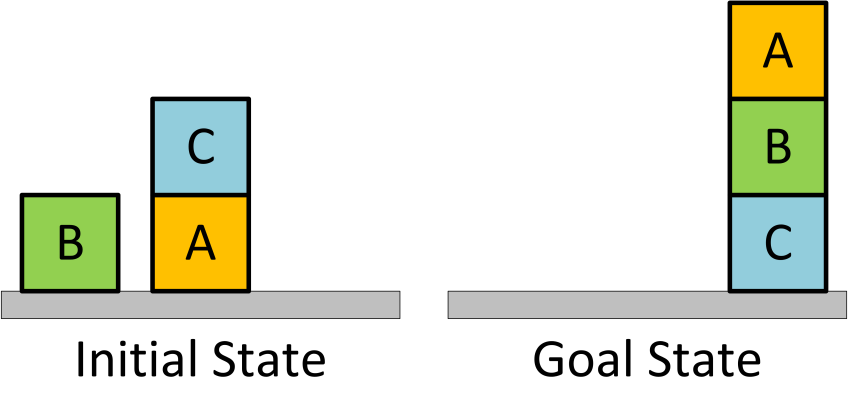


Figure – Block World Initial and Goal States

The literals and actions in this world are below.

**Literals:**

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

**Actions:**

* – Moves block from to .
* – Moves block from block to the .

**Additional Notes:** The inequality preconditions (e.g. ) are not shown in the following figures also for increased readability. What is more, 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.

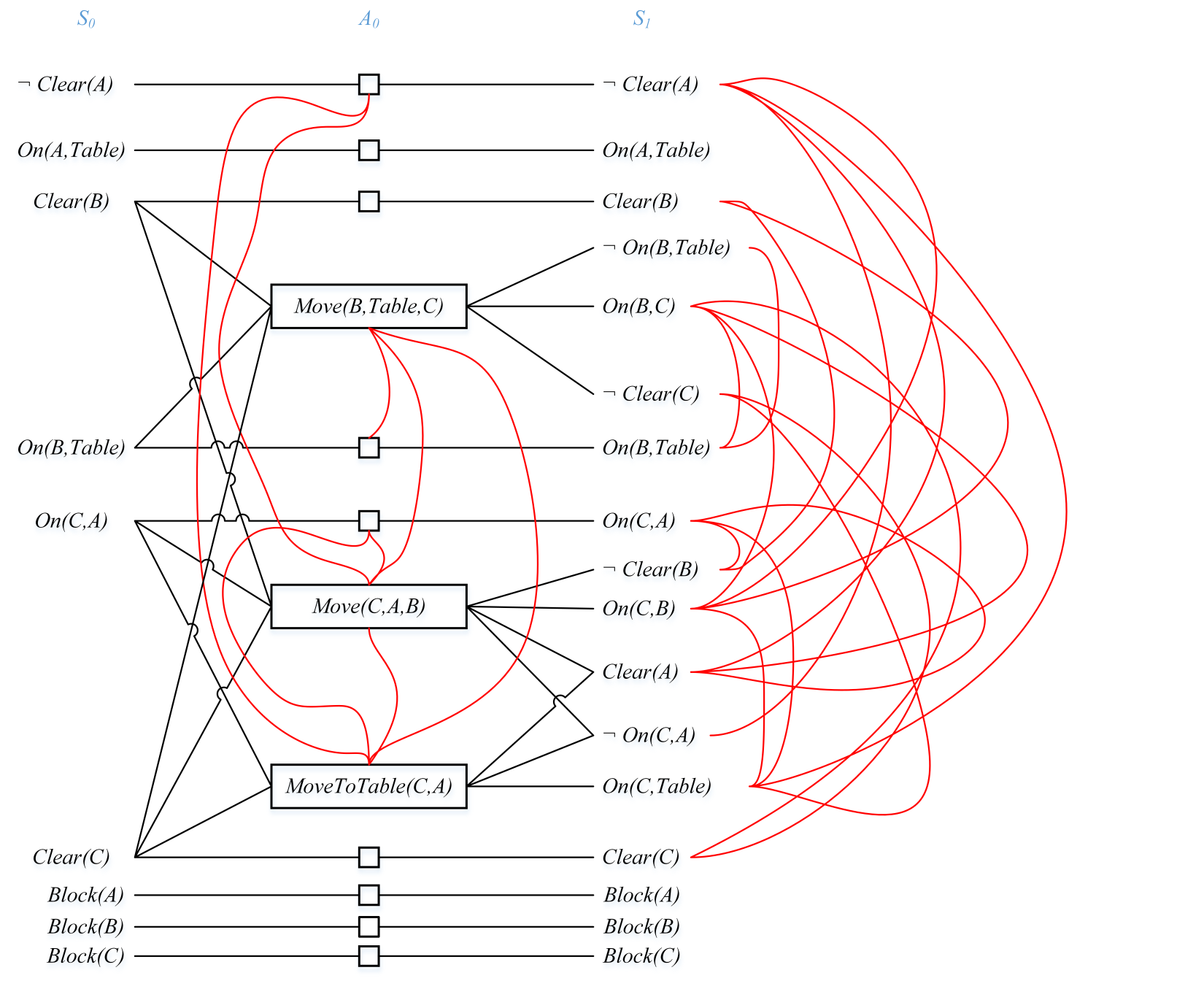


Figure – Block World Ground Actions () and Related Mutexes

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. Mutexes are shown as red curved lines. Not included *mutexes* in this figure include: ↔ and ↔ . In subsequent levels, the mutexes decrease monotonically; actions increase monotonically, and literals increase monotonically. Therefore, any literals or actions shown between and will also be present between and ; however, some (but not all) mutexes have the potential to be dropped.

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

1. – This represents the two actions where block is moved from (where can be either a block other than or the ) to block .
2. – Action where block can be moved from on top of another block to the .

Note in the action could be the same as from the action . However, a different symbol is used here to denote that is either the or a block while is exclusively a block. Depending on whether and are the same blocks, then there may be additional mutexes between , , , and which are not shown in Figure 3. In addition, as with Figure 2, the preconditions for the literals are excluded for increased readability.

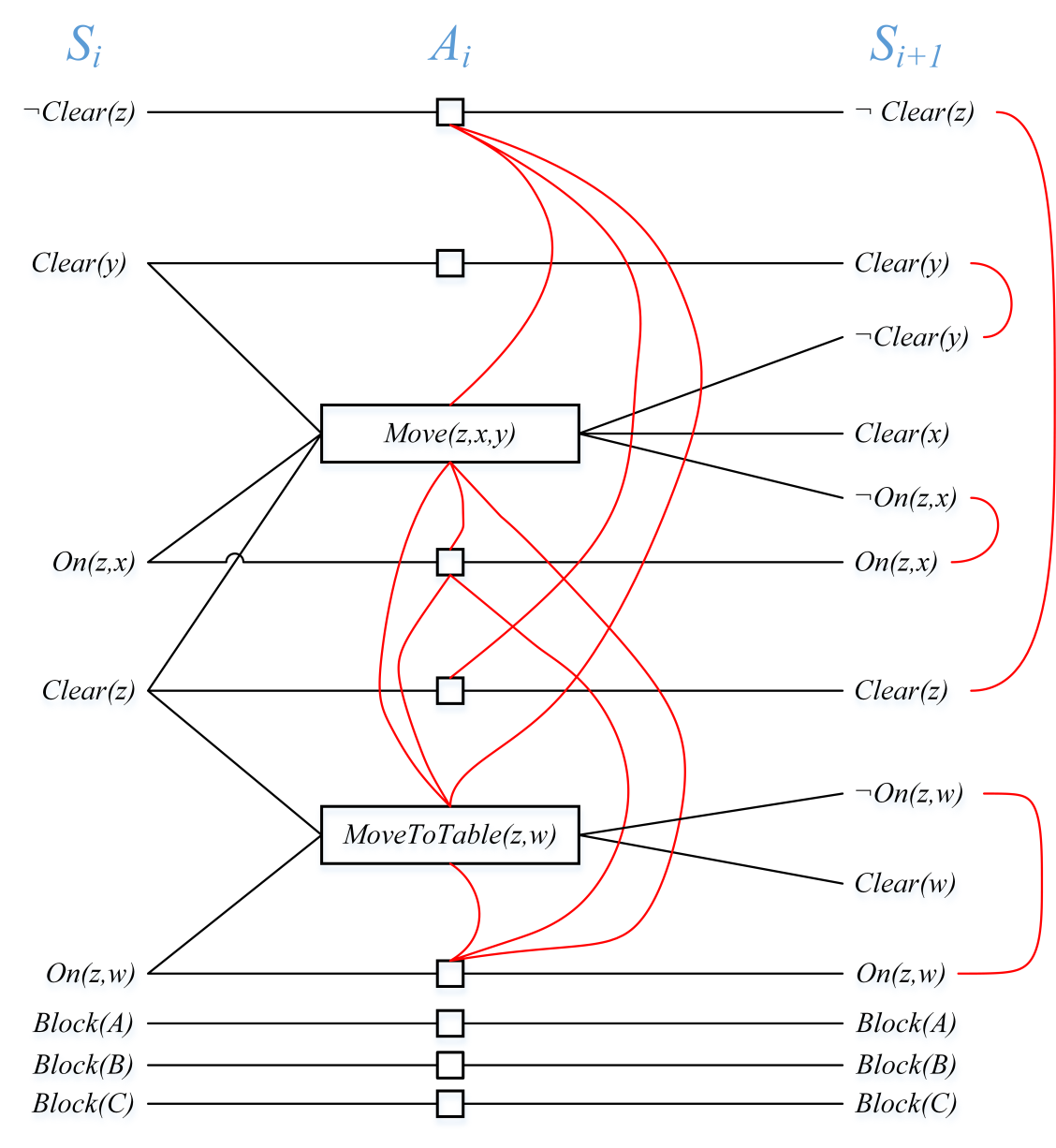


Figure – Simplified Set of Generic Actions for a Block in Action where

Figure 3 represents the actions for block . When the actions for the blocks other than are included, then there will be additional mutexes as not all actions and literals for this state become legal. For example, the action is mutually exclusive with the action , , etc. Similarly, a persistence action for would be mutually exclusive with the action .

Similar to the additional mutexes on actions, there are additional mutexes on literals that would necessarily be added once the blocks other than are added for level . For example, is mutually exclusive with . What is more, is mutually exclusive with in the same way that is mutually exclusive with . These additional mutexes are not captured in the single block actions shown in Figure 3.

The arbitrary move for block would be apply to all three blocks , , and for actions , , and 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 *St* 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 – 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 are shown in Figure 3.

**Step #2: Express the Goal as a conjunction of literals.**

The goal can be expressed as:

# 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 and effects to a time , 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 action in the “Air Cargo Transport” problem requires sufficient fuel in the tank, a competent pilot, good weather, no intentional sabotage, etc.; otherwise the 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 action is performed, some of the airline’s gasoline reserve is consumed. Moreover, after a 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.

1. Block in action is an exception to this statement because after , block cannot perform the action; this is because regardless of what was, block will always be on the table in state . [↑](#footnote-ref-1)