Luke Salamone, Simon Benigeri, Renpin Luo, William Ansehl
KRR Final Report - Zoo Planning

Final Report

What Does Your Project Do?

Our project represents a straightforward implementation of a knowledge storage system which when combined with a reasoning engine allows for the creation of a high-level plan for an automated zoo management system. It contains a PDDL domain file, and five PDDL problem files.

PDDL, or Planning Domain Definition Language, allows for the definition of knowledge and means to reason with that knowledge in furtherance of a goal. PDDL files define the legal actions in any given state, but do not define a concrete plan for accomplishing the goal. Instead, a planning engine generates a plan through a recursive search of the state space entailed by the initial conditions of each problem.

The domain file defines three main components. First, the domain file contains the types our planning system uses, including the superclasses associated with those types. For example, our domain file defines a “carnivore” as a type, while carnivore might be thought of as a subclass of “animal”. Second, the domain file contains all of the predicates used by the reasoning engine, which maintain stateful information set and unset during the execution of a plan. Third, the domain file defines all of the possible actions which might be taken at any given time.

The project also contains problem files, each with varying levels of complexity. The problem files define an initial state and a goal state. The initial state might be thought of as a small knowledge base, with the ontology composed of the types defined in the domain file, and facts defined in the initial state. The goal may be composed of any number of predicates which must be accomplished for the planner to be successful.

Generally the goal of each of the problems we have submitted is that the animals are placed in their cages and fed. Each of the problems involves at least one robot which is to carry out these actions. Due to constraints which will be discussed later on, the task of feeding the zoo animals may take dozens of steps in some cases.

Why Should We Care?

Planning is a very practical application of KRR. It is an abstract way to represent knowledge and reasoning in the microtheory of a specific domain. A robot feeding animals in a zoo is simply a fun example. However, this use-case can be extended for other similar scenarios. Rather than a robot care-taker in a zoo, we could have chosen to represent a robot servicing a factory and the machines within, or a robot packing and preparing shipments in a warehouse.

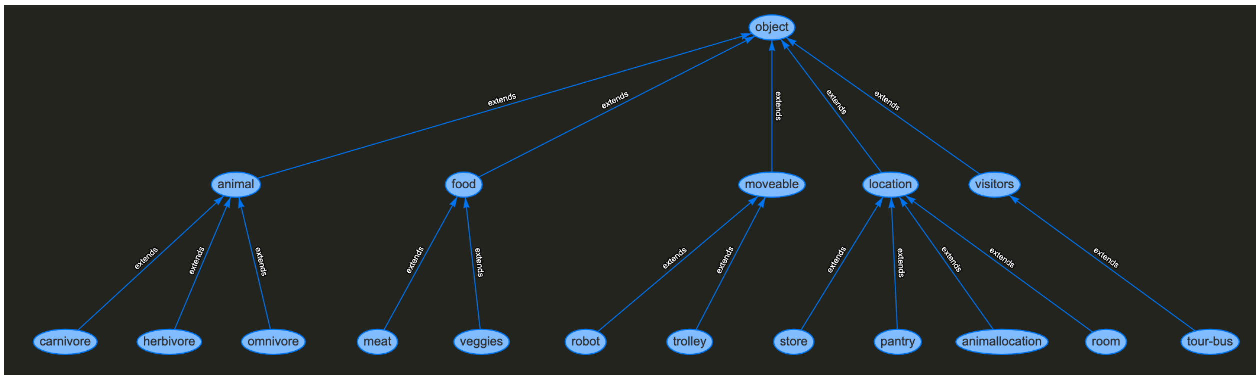
Luke Salamone, Simon Benigeri, Renpin Luo, William Ansehl
KRR Final Report - Zoo Planning

How Did You Represent Knowledge?

We represent knowledge using objects, predicates, and actions. The objects are typed. They include different types of animals, food, a robot, a trolley, visitors, and locations that serve different purposes in the plan.

```
(:types
  carnivore herbivore omnivore - animal
  meat veggies - food
  robot trolley - moveable
  store pantry animallocation room - location
  tour-bus - visitors
)
```

They can also be represented in a hierarchy:



We define the following predicates. Each predicate is preceded by its description in a comment, denoted ;; comment.

```
(:predicates
  ;; determines whether an animal needs to be fed
  (hungry ?x - animal)
  ;; an instance of robot object is at a location
  (robot-at ?x - robot ?y - location)
  ;; an instance of an animal object is at a location
  (animal-at ?x - animal ?y - location)
  ;; an instance of a food object is at a location
  (food-at ?x - food ?y - location)
  ;; an instance of a visitors object is at a location
  (visitors-at ?x - visitors ?y - location)
```

```
   ;; indicates that a location object does not have visitors there
   (no-visitors ?x - location)
   ;; indicates two objects of type location are connected
   (connected ?x - location ?y - location)
   ;; indicates that a robot object is pushing a trolley object
   (pushing-trolley ?x - robot ?y - trolley)
   ;; indicates that a trolley object is holding a food object
   (trolley-holding ?x - trolley ?y - food)
   ;; indicates that a trolley object is located at a location object
   (trolley-at ?x - trolley ?y - location)
   ;; indicates that a location object does not have a trolley object
there
   (no-trolley ?x - location)
   ;; indicated that a location object does not have a robot object
there
   (no-robot ?x - location)
   ;; indicated that a location object does not have an animal there
   (no-animal ?x - animallocation)
   ;; indicates that a robot object is not pushing a trolley
   (free ?x - robot)
   ;; indicates that a robot object is holding an animal object
   (holding-animal ?x - robot ?y - animal)
   ;; indicates that a location object can be visited by a visitor
object
   ;; store, pantry do not allow visitors
   ;; animallocation room allow visitors
   (visitors-allowed  ?x - location)
 )
```

Planning can only be performed by defining actions. An action is structured as follows:

```
(:action action_name
     :parameters ()
     :precondition (and )
     :effect (and )
 )
```

Let's look at an example of the action feed-carnivore. Its parameters are a carnivore object, an animallocation object, a meat object, a robot object, and a trolley object. If a carnivore is hungry and a robot is pushing a trolley, and that trolley is holding meat, and the carnivore, robot, and trolley are all at an animallocation, then this action is triggered. The action does pretty much what the name suggests. It consists of feeding a carnivore. So the effects of the action are that the carnivore is no longer hungry and that the meat is no longer being held on the trolley.

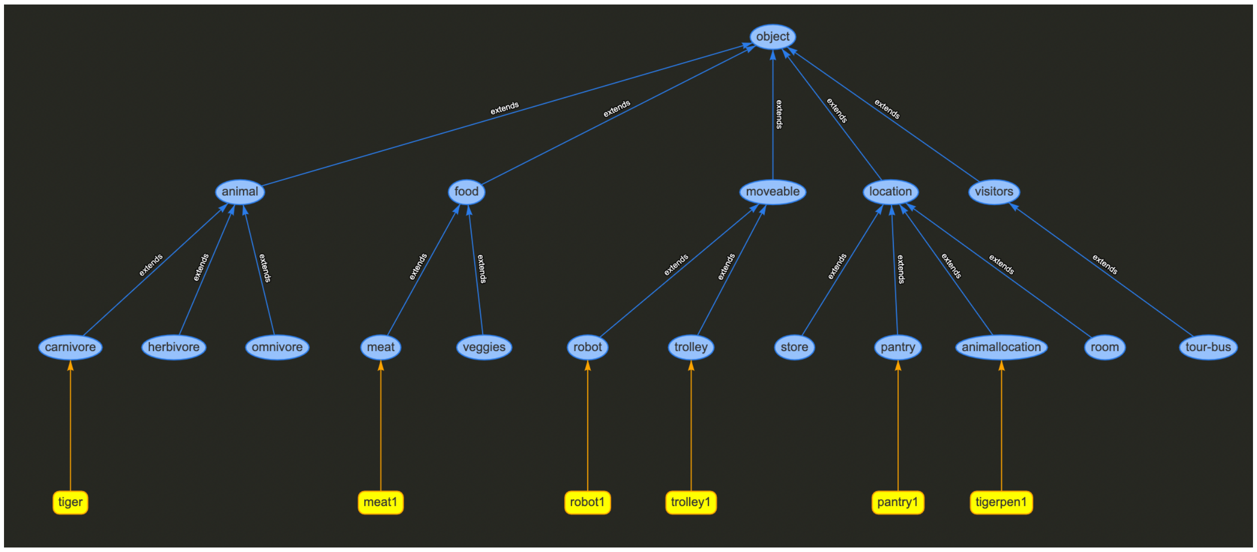
```
(:action feed-carnivore
   :parameters (
     ?an - carnivore
     ?aniloc - animallocation
     ?f - meat
     ?rob - robot
     ?tro - trolley
   )
   :precondition (
     and
       (hungry ?an)
       (robot-at ?rob ?aniloc)
       (animal-at ?an ?aniloc)
       (pushing-trolley ?rob ?tro)
       (trolley-holding ?tro ?f)
       (trolley-at ?tro ?aniloc)
       (not(free ?rob))
       (no-visitors ?aniloc)
   )
   :effect (
     and
       (not(hungry ?an))
       (not(trolley-holding ?tro ?f))
   )
```

```
 )
```

We initialize a problem or environment by instantiating objects and asserting predicates. Let's look at Problem0.pddl for example:

```
(:objects
   tiger - carnivore
   tigerpen1 - animallocation
   pantry1 - pantry
   robot1 - robot
   trolley1 - trolley
   meat1 - meat
 )
```

Recall the hierarchy defined in the domain, we can see in yellow the instances of objects that exist in this problem.



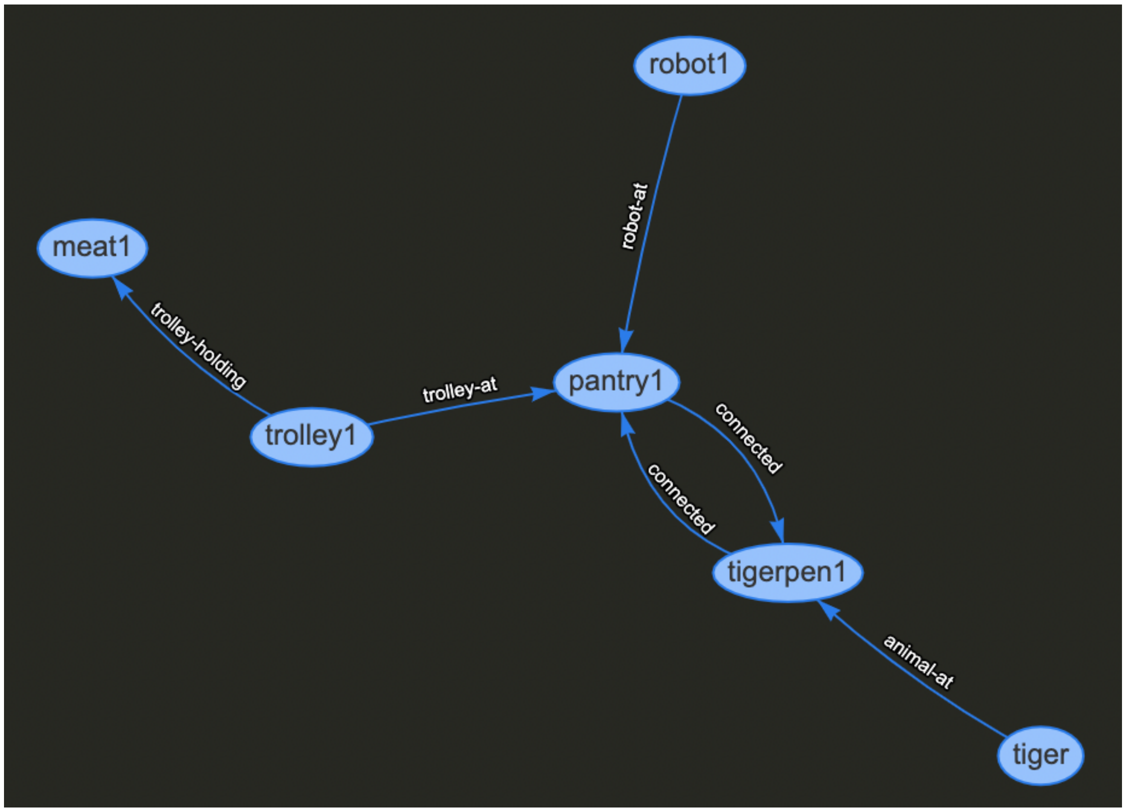
We also initialize predicates for this problem by inputting the instances of objects into predicates defined in the domain.

```
(:init
    (connected tigerpen1 pantry1)
    (connected pantry1 tigerpen1)
    (robot-at robot1 pantry1)
    (no-robot tigerpen1)
```

```
   (trolley-at trolley1 pantry1)
   (no-trolley tigerpen1)
   (animal-at tiger tigerpen1)
   (trolley-holding trolley1 meat1)
   (hungry tiger)
   (free robot1)

   (no-visitors tigerpen1)
   (no-visitors pantry1)
 )
```

Again, we can visualize this better as a graph:



Finally, we express a goal, as follows:

```
(:goal
   (not(hungry tiger))
 )
```

Given the domain and the problem/environment, PDDL will find a plan to achieve the goal expressed in the problem.

We run the problem0.pddl file and get the following output:

```
0.00100: (grab-trolley robot1 trolley1 pantry1)
0.00200: (push-trolley robot1 pantry1 tigerpen1 trolley1)
0.00300: (feed-carnivore tiger tigerpen1 meat1 robot1 trolley1)
Planner found 1 plan(s) in 0.532secs.
```

grab-trolley and push trolley are just other actions which we defined in our domain. You can find the details of all the actions we defined in the domain.pddl file. They are: feed-carnivore, feed-herbivore, feed-omnivore, free-move, push-trolley, grab-trolley, pick-up-food, grab-animal, move-animal, release-animal, visitors-move.

Luke Salamone, Simon Benigeri, Renpin Luo, William Ansehl
KRR Final Report - Zoo Planning

How Did You Reason Using Your Represented Knowledge?

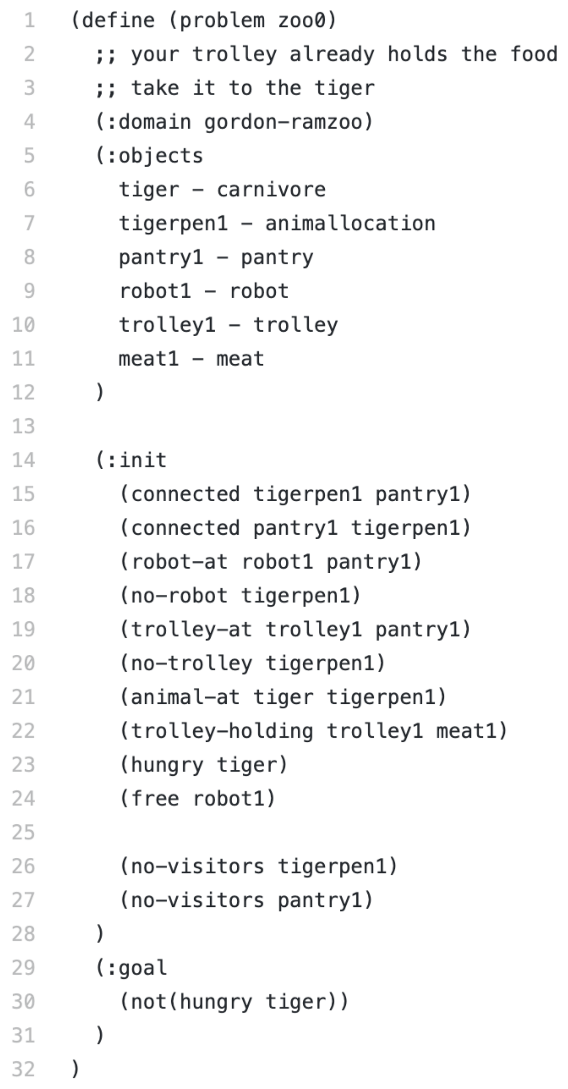
In the domain file, we create 16 predicates and 12 actions that can be utilized to generate plans. PDDL editor tries all possible actions in each step to achieve the goal that is expressed in the problem. A feasible plan can navigate predicates from the initial state to the final goal step by step.
In order to be able to reason and plan meaningfully in a given scenario, we specify some base settings that describe what robots can and cannot do. These settings include: robots will only feed animals as long as the animal is in the cage, robots will not feed in front of visitors, only one robot and one trolley can be present at a location at the same time, etc. We write these settings into specific activities. As an example, we divided all animals into three categories: carnivores, herbivores, and omnivores. Therefore, we need to define three different feeding activities. The main difference between these three activities is the type of food: meat - carnivores, vegetable - herbivores, meat or vegetable - omnivores. Some common preconditions include: the animal is hungry, the robot holds a trolley with food on it, the robot is in the animal location, and the animal is in the cage and surrounded by no visitors as mentioned above. The effect of all three activities is that the animal is not hungry.

```
(:action feed-carnivore
  :parameters (
    ?an - carnivore
    ?aniloc - animallocation
    ?f - meat
    ?rob - robot
    ?tro - trolley
  )

  :precondition (
    and
      (hungry ?an)
      (robot-at ?rob ?aniloc)
      (animal-at ?an ?aniloc)
      (pushing-trolley ?rob ?tro)
      (trolley-holding ?tro ?f)
      (trolley-at ?tro ?aniloc)
      (not(free ?rob))
      (no-visitors ?aniloc)
  )

  :effect (
    and
      (not(hungry ?an))
      (not(trolley-holding ?tro ?f))
  )
)
```

Using a similar approach, we set up the remaining activities, including robot movement with and without the trolley, robot picking up and putting down the trolley, robot picking up, moving, and putting down the animal, and so on. Since we could not determine the movement of the visitors, we also specified the activity of moving the tour bus. All these activities can adequately satisfy all the questions we provide in a given scenario.
Let's use problem0 as an example. In this scenario, as shown in the picture below, we have the following information: there are two locations: the pantry and the tiger cage, the robot and the trolley are in the pantry, the pantry has meat on it, and the tiger is in the cage and there are no visitors around. The goal of the problem is to feed a hungry tiger. Based on this information, we can get a plan: the robot firstly picks up the trolley, then the robot moves to the cage, and the robot feeds the tiger as the last step. After each activity, some of the predicates are affected and updated, and the updated predicates satisfy the preconditions for the next activity. Throughout this process, the predicates in the initial conditions can be guided by the individual activities to reach the goal.



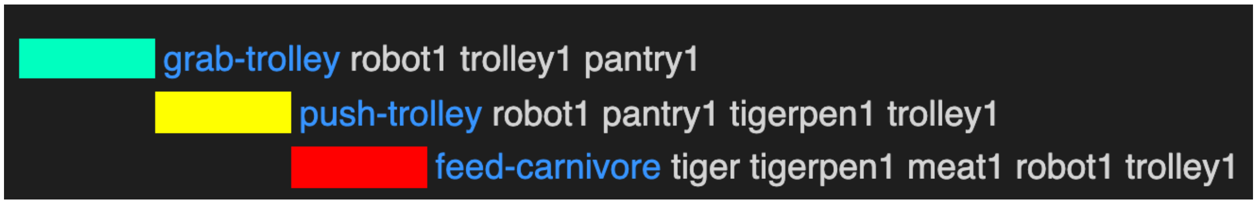
```
(define (problem zoo0)
  ;; your trolley already holds the food
  ;; take it to the tiger
  (:domain gordon-ramzoo)
  (:objects
    tiger - carnivore
    tigerpen1 - animallocation
    pantry1 - pantry
    robot1 - robot
    trolley1 - trolley
    meat1 - meat
  )

  (:init
    (connected tigerpen1 pantry1)
    (connected pantry1 tigerpen1)
    (robot-at robot1 pantry1)
    (no-robot tigerpen1)
    (trolley-at trolley1 pantry1)
    (no-trolley tigerpen1)
    (animal-at tiger tigerpen1)
    (trolley-holding trolley1 meat1)
    (hungry tiger)
    (free robot1)

    (no-visitors tigerpen1)
    (no-visitors pantry1)
  )
  (:goal
    (not(hungry tiger))
  )
)
```

```
0.00100: (grab-trolley robot1 trolley1 pantry1)
0.00200: (push-trolley robot1 pantry1 tigerpen1 trolley1)
0.00300: (feed-carnivore tiger tigerpen1 meat1 robot1 trolley1)
Planner found 1 plan(s) in 0.532secs.
```

Show Us How Your Knowledge Works.
In the previous section, we walked through one simple example of the actions a robot needs to take in order to accomplish its goal. Each problem has objects, an initialized state and a goal. The sequence of actions for problem 0 appears as follows:

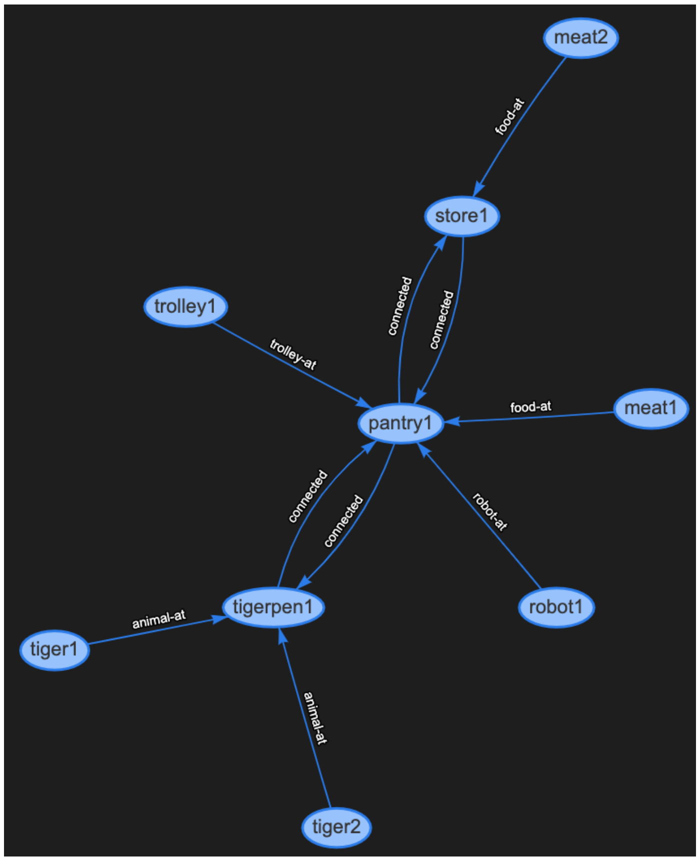


```
grab-trolley robot1 trolley1 pantry1
        push-trolley robot1 pantry1 tigerpen1 trolley1
                feed-carnivore tiger tigerpen1 meat1 robot1 trolley1
```

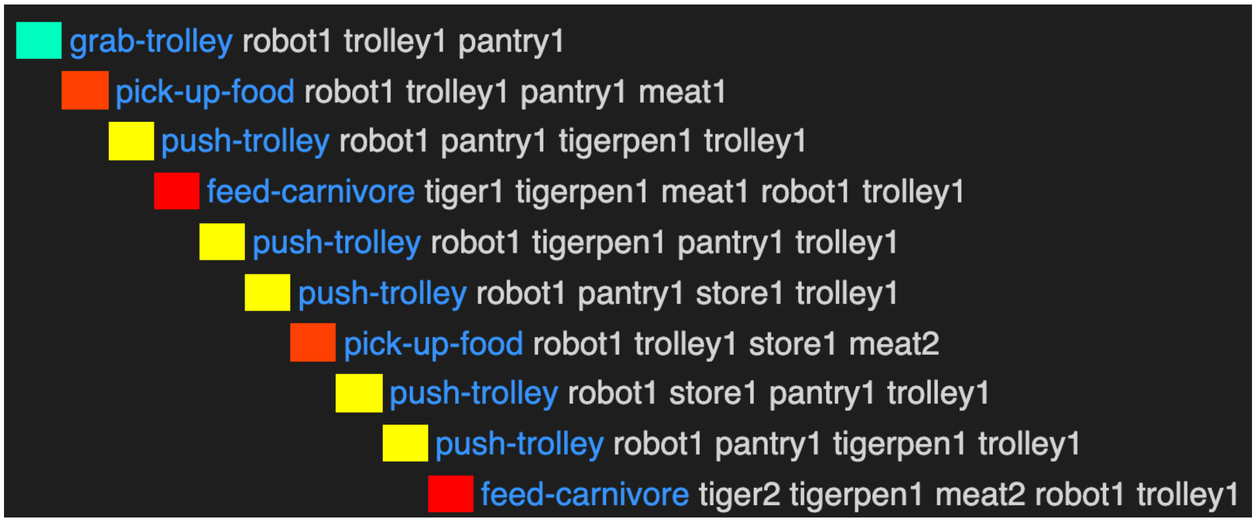
We see that the robot grabs the trolley, pushes the trolley from the location "pantry1" to the location "tigerpen1" and feeds the tigers with meat found on the trolley. Each problem statement has at minimum one robot, one animal and one item of food. Some environments are instantiated with a variety of foods - veggies and meats in higher quantities - as well as multiple different types of hungry animals - giraffes, bears, tigers etc. Problem 4 initializes an additional tour bus agent carrying tourists while problem 5 initializes animals outside of their respective pens. In this section, we will individually walk through each problem to better understand the actions taken by the robot to feed animals, and thereby the reasoning used by the agent. Consistent across all problems is the goal state: all animals have been fed.

Problem 1:

Luke Salamone, Simon Benigeri, Renpin Luo, William Ansehl
KRR Final Report - Zoo Planning



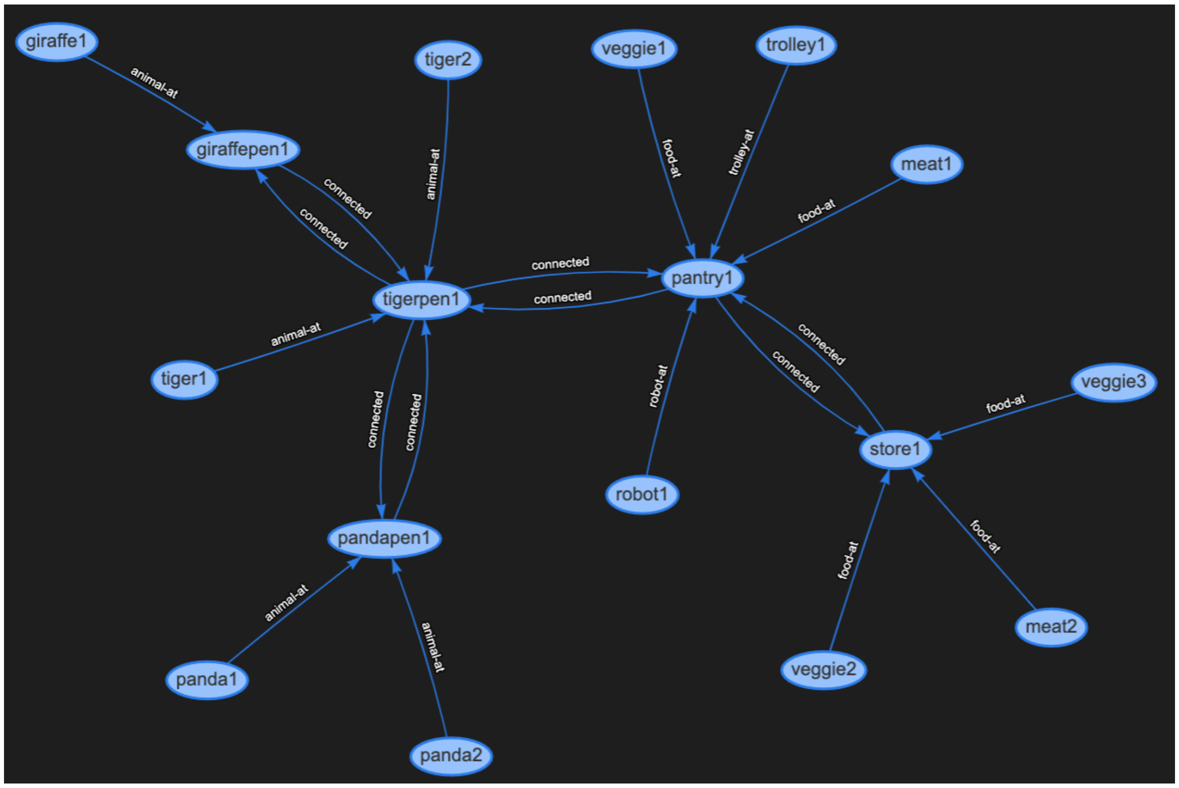
Problem 1 is initialized with a robot, a trolley and one meat item located at the pantry. 2 tigers are hungry residents to a tiger pen. The robot must grab the food from the pantry and feed a tiger. However, there is only enough food in the pantry for one tiger. Therefore, the robot must travel to the store to pick up an additional food item to accomplish its goal state.



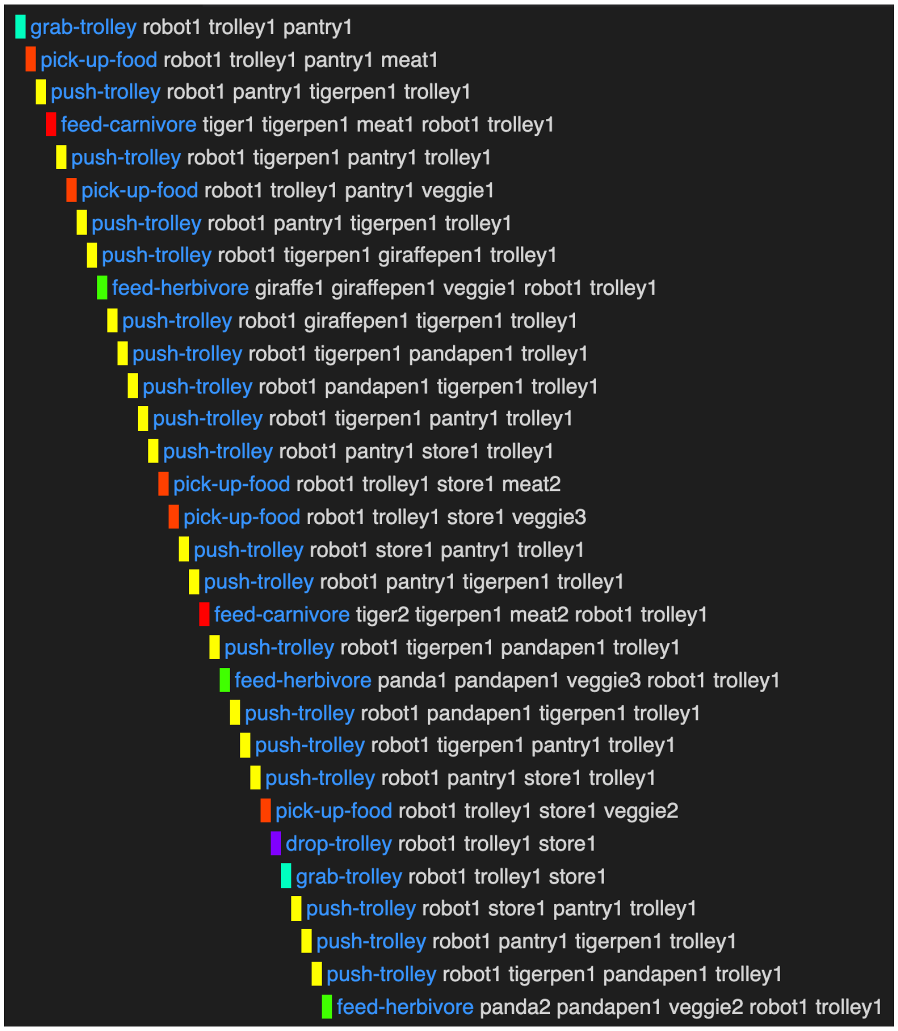
```
🟩 grab-trolley robot1 trolley1 pantry1
    🟥 pick-up-food robot1 trolley1 pantry1 meat1
        🟨 push-trolley robot1 pantry1 tigerpen1 trolley1
            🟥 feed-carnivore tiger1 tigerpen1 meat1 robot1 trolley1
                🟨 push-trolley robot1 tigerpen1 pantry1 trolley1
                    🟨 push-trolley robot1 pantry1 store1 trolley1
                        🟥 pick-up-food robot1 trolley1 store1 meat2
                            🟨 push-trolley robot1 store1 pantry1 trolley1
                                🟨 push-trolley robot1 pantry1 tigerpen1 trolley1
                                    🟥 feed-carnivore tiger2 tigerpen1 meat2 robot1 trolley1
```

When the program is run, we see exactly what we have previously described. The robot picks up food from the pantry and feeds one tiger. Then, the robot agent travels to the store to pick up more food and feed the remaining tiger. After both tigers are fed, the goal state is reached.

Problem 2:



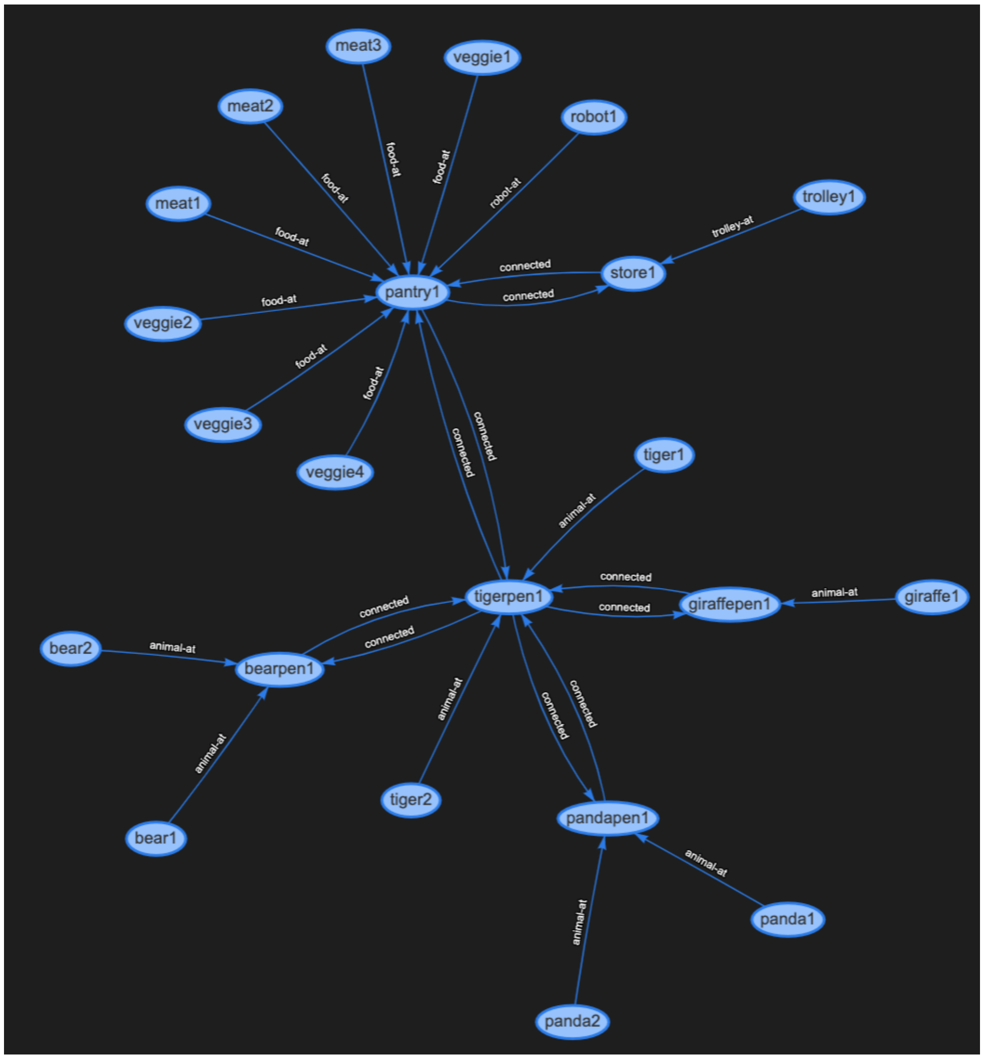
Problem 2 is notably more complex in the initialization graph than problem 1. In addition to carnivores such as tigers, there are herbivores - pandas and giraffes. Similarly to the last problem, the robot must venture to the store to get any additional food required to accomplish the goal state, should the pantry not suffice. The addition of dietary restrictions adds planning complexity and further reasoning to the problem. For example, the robot agent is unable to feed meat to an herbivore, or veggies to a carnivore. It could, however, feed veggies or meat to an omnivore (as we'll later see in problem 3). Therefore, if an agent is at a pen, it will need to distinguish if it has the appropriate food type to feed a respective animal.



```
grab-trolley robot1 trolley1 pantry1
 pick-up-food robot1 trolley1 pantry1 meat1
  push-trolley robot1 pantry1 tigerpen1 trolley1
   feed-carnivore tiger1 tigerpen1 meat1 robot1 trolley1
    push-trolley robot1 tigerpen1 pantry1 trolley1
     pick-up-food robot1 trolley1 pantry1 veggie1
      push-trolley robot1 pantry1 tigerpen1 trolley1
       push-trolley robot1 tigerpen1 giraffepen1 trolley1
        feed-herbivore giraffe1 giraffepen1 veggie1 robot1 trolley1
         push-trolley robot1 giraffepen1 tigerpen1 trolley1
          push-trolley robot1 tigerpen1 pandapen1 trolley1
           push-trolley robot1 pandapen1 tigerpen1 trolley1
            push-trolley robot1 tigerpen1 pantry1 trolley1
             push-trolley robot1 pantry1 store1 trolley1
              pick-up-food robot1 trolley1 store1 meat2
               pick-up-food robot1 trolley1 store1 veggie3
                push-trolley robot1 store1 pantry1 trolley1
                 push-trolley robot1 pantry1 tigerpen1 trolley1
                  feed-carnivore tiger2 tigerpen1 meat2 robot1 trolley1
                   push-trolley robot1 tigerpen1 pandapen1 trolley1
                    feed-herbivore panda1 pandapen1 veggie3 robot1 trolley1
                     push-trolley robot1 pandapen1 tigerpen1 trolley1
                      push-trolley robot1 tigerpen1 pantry1 trolley1
                       push-trolley robot1 pantry1 store1 trolley1
                        pick-up-food robot1 trolley1 store1 veggie2
                         drop-trolley robot1 trolley1 store1
                          grab-trolley robot1 trolley1 store1
                           push-trolley robot1 store1 pantry1 trolley1
                            push-trolley robot1 pantry1 tigerpen1 trolley1
                             push-trolley robot1 tigerpen1 pandapen1 trolley1
                              feed-herbivore panda2 pandapen1 veggie2 robot1 trolley1
```

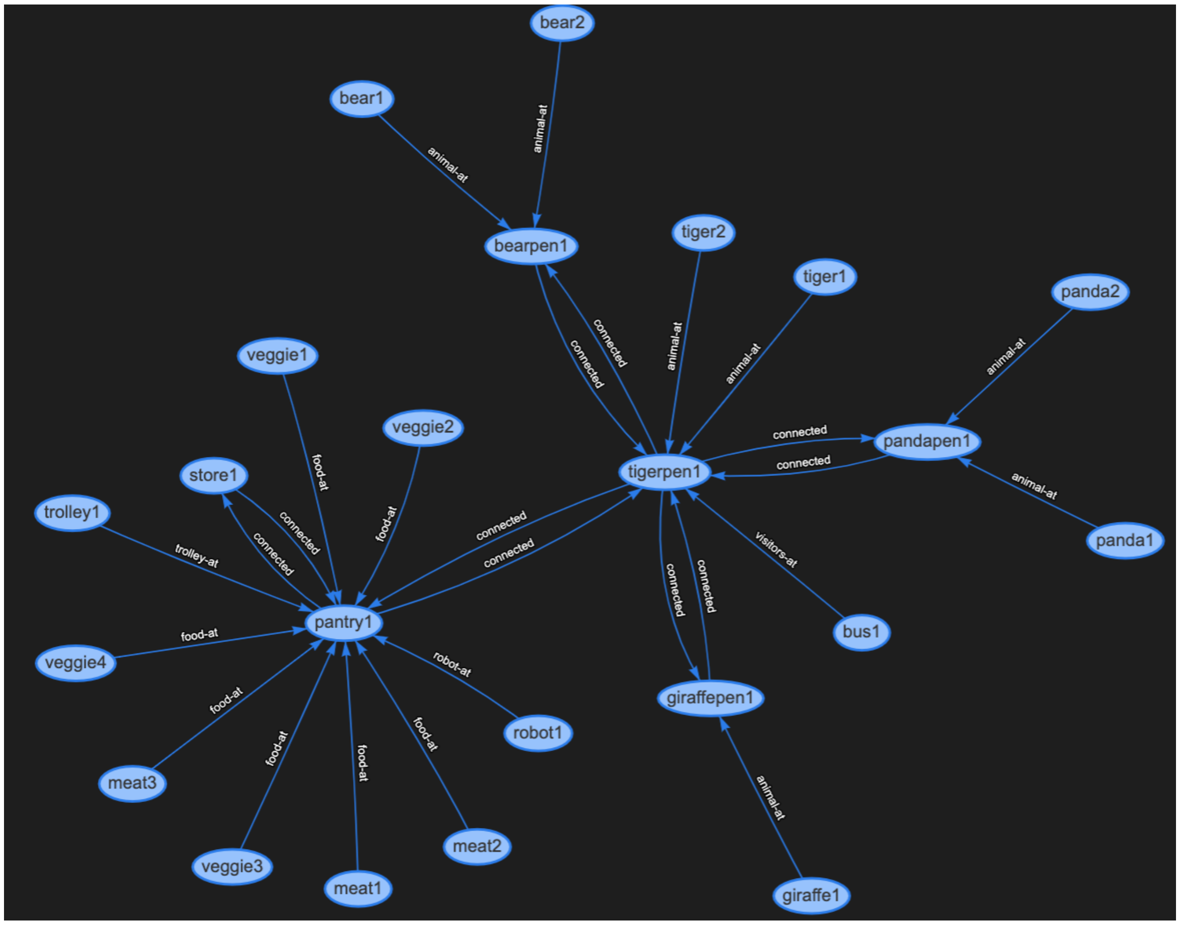
This example serves as a good reminder that PDDL simply returns one of multiple potential plans that could accomplish the goal state. We see that at times the robot picks up a single food item, feeds an animal and returns to pick up another food item. In another instance, the robot picks up multiple food items and then proceeds to leave that particular location. There is no particular reasoning that states the robot is limited to one food item at a time, nor is there reasoning that states the robot must treat the pantry or the store differently as inventories for food. PDDL simply returns one path to success.

Problem 3:

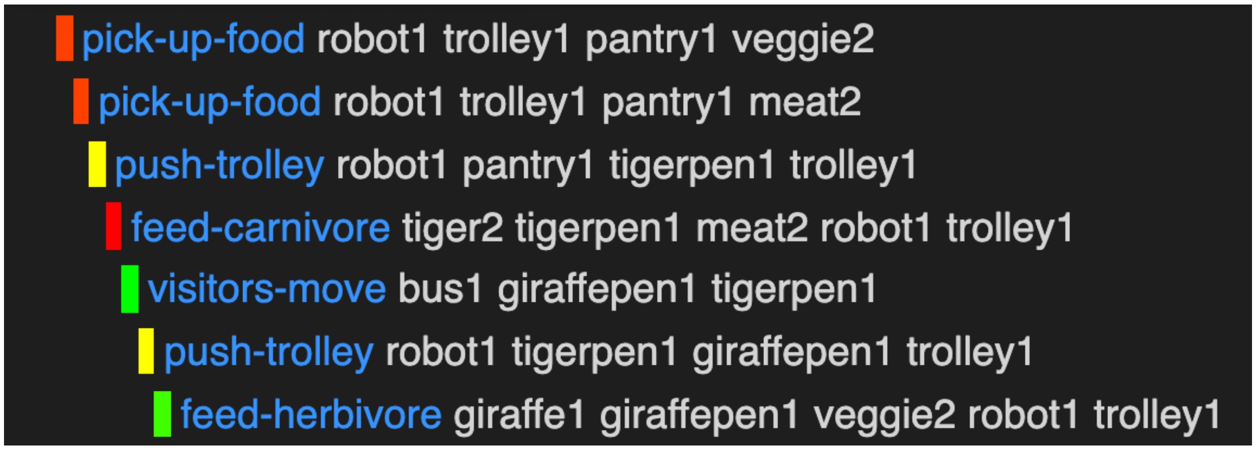


Question 3 is similar to question 2 except with the small difference that the trolley is no longer at the same location as the robot. The robot agent can only pick up food items if it has a trolley. Therefore, the agent must move to the store to grab the trolley prior to picking up food items at the pantry. The findings from running the problem are similar to problem 2. The agent is able to pick up multiple items at a time, but chooses not to pick up all the items at once. This decision demonstrates that pddl doesn’t necessarily return the most efficient path, simply a path.

Problem 4:



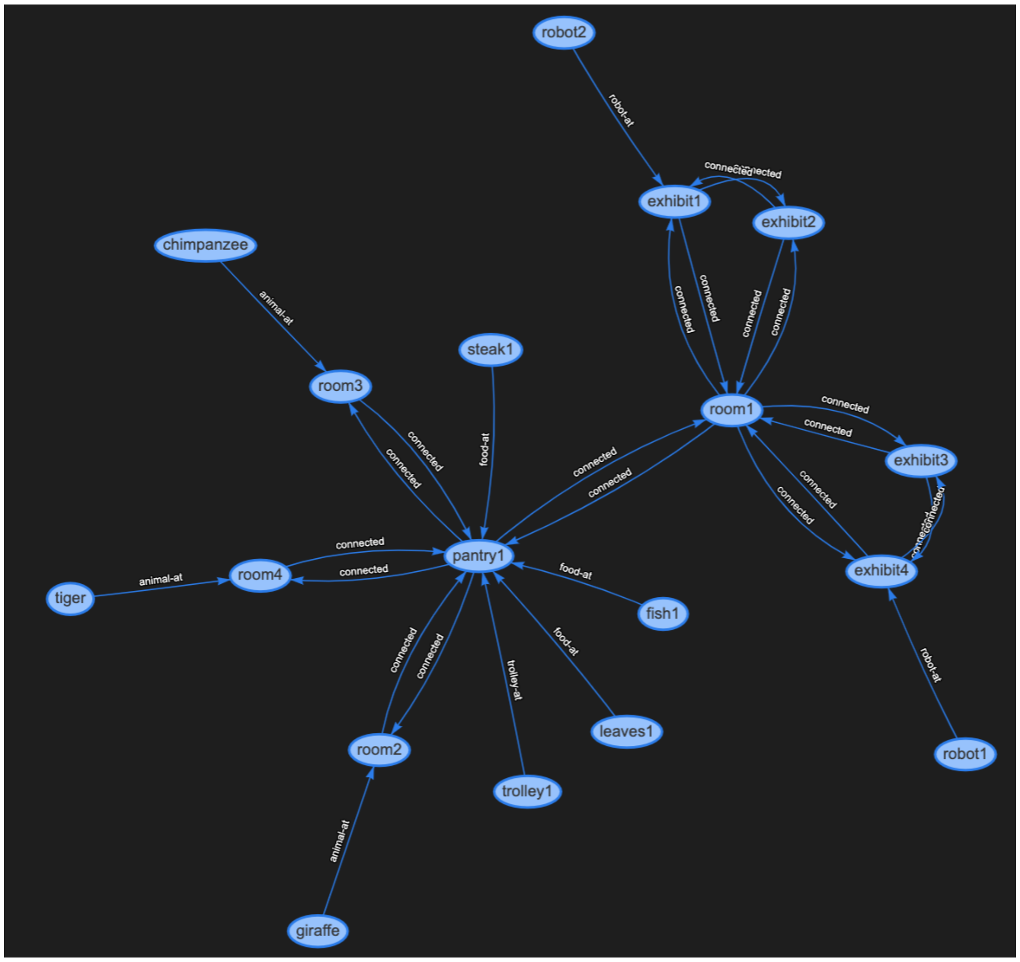
Problem 4 adds an extra layer of complexity and reasoning with the addition of a tour bus agent, labeled as “bus1.” The tour bus is able to roam all the locations except those that are reasonably off-limits to visitors - the pantry and the store. A tour bus is able to be at the same location as a robot agent, and vice versa. However, the presence of a tour bus at a given location prevents the robot from feeding an animal at the same location. Therefore, the robot agent must accomplish the same goal under an added restriction.


```
pick-up-food robot1 trolley1 pantry1 veggie2
 pick-up-food robot1 trolley1 pantry1 meat2
  push-trolley robot1 pantry1 tigerpen1 trolley1
   feed-carnivore tiger2 tigerpen1 meat2 robot1 trolley1
    visitors-move bus1 giraffepen1 tigerpen1
     push-trolley robot1 tigerpen1 giraffepen1 trolley1
      feed-herbivore giraffe1 giraffepen1 veggie2 robot1 trolley1
```

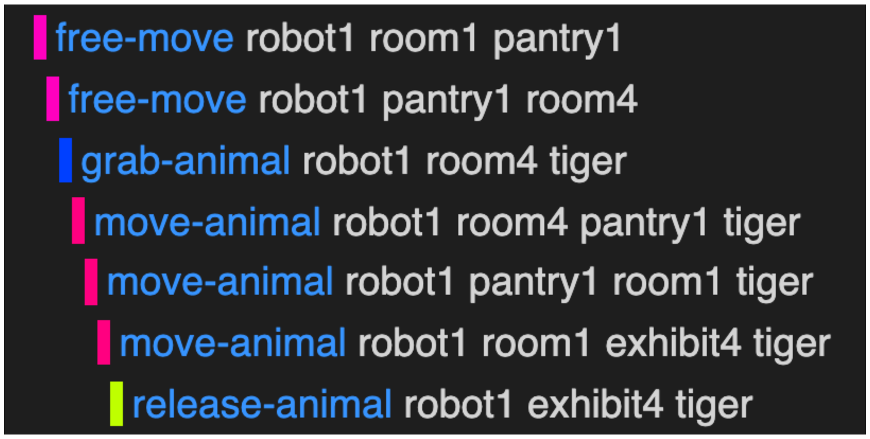
Luke Salamone, Simon Benigeri, Renpin Luo, William Ansehl
KRR Final Report - Zoo Planning

The plan for this problem is particularly long. As such, I have provided a screenshot of only a few of the steps in the robot's path to goal completion. In this instance, we see that the robot feeds animals at location giraffepen1 only when the bus has moved elsewhere, despite the robot having veggies available to it.

Problem 5:



In Problem 5, we introduce two new concepts. One is the potential for new locations, described as rooms, separate from animal pens/exhibits. The other concept is the possibility of animals not in their appropriate pens. In this scenario the robot must first ensure an animal has been placed in its pen/exhibit prior to being able to feed an animal. As such, the robot must pick up animals, put them in exhibits, grab the trolley and find food and finally feed all animals in order to accomplish the goal state.



```
free-move robot1 room1 pantry1
 free-move robot1 pantry1 room4
  grab-animal robot1 room4 tiger
   move-animal robot1 room4 pantry1 tiger
    move-animal robot1 pantry1 room1 tiger
     move-animal robot1 room1 exhibit4 tiger
      release-animal robot1 exhibit4 tiger
```

The above plan is but a segment of its entirety. For an animal that is out of its exhibit, the robot must grab it, move it to its appropriate exhibit and release it at that location. Only then can the animal be fed. Animals are unable to move on their own accord and will stay where they are placed.