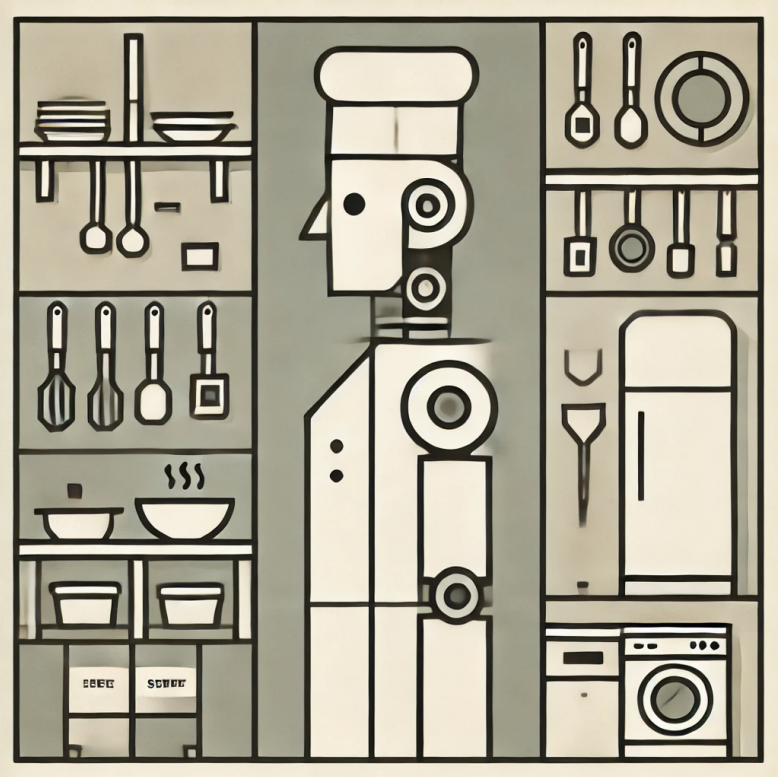
Robot Chef Task: PDDL Planner


[2]

Víctor Carballo
Júlia Amenós
Oriol Miró

October 2024

Planning and Approximate Reasoning (MIA-MESIIA)
Practical Exercise 2: PAR

Contents

1 Introduction to the problem

In this project we must develop a PDDL program to coordinate a robot chef's operations in a Japanese restaurant's kitchen. The robot must plan all actions regarding the preparation of various dishes, where he must gather ingredients, perform actions on them (e.g. cut them, mix them...), assemble ingredients to create new ones...

The kitchen is structured into seven discrete areas: storage (SA), preparation (PA), cooking (CA), serving (SVA), dishwashing (DWA), cutting (CTA), and mixing (MIXA). Movement between adjacent areas is possible unless blocked by walls. Each order specifies a dish, requiring the robot to perform diferent actions, each of which has a specific room. For example, already-made ingredients are stored in SA, all cooking must be done in CA, etc. Tools used during cooking must be cleaned in DWA before reuse.

The assignment description proposes a series of predicates and actions; we however opted for designing a more general system, which more closely resembles real life. For example, our more general system permits creating a plan to create more than one dish (further discussed in Section 1.1), or crafting intermediate ingredients (such as first making noodles when preparing ramen). We also generalise on possible actions and tools, permitting the extension of our work in a future situation.

1.1 Assumptions

Such a large planner left some constraints open to interpretation; our own generalisation of the proposed system also implied the modification of some other proposed assumptions. For this purpose, the following section will detail all our assumptions:

1. There is only one robot.
2. The robot has only one hand, therefore can only carry an ingredient or tool at a time.
3. All operators have the same cost.
4. There exists an ordering for the actions needed for each ingredient (e.g. when preparing vegetables, first cut and then cook).
5. An order for more than one dish can arrive.

6. When receiving an order for multiple dishes, the preparation of these can be done in parallel (more closely resembling real life), even though one action is done at a time.

7. When receiving an order for multiple dishes requiring ingredients in common, each dish will use a different instance of each ingredient (e.g. when making two ramen dishes, we have two instances of the ingredient "noodles")

8. When receiving an order for multiple dishes, there does not exist a priority between them — that is, any can be served first.

9. Tools must be washed between each use, even if used successively on different instances of the same ingredient.

2 Problem Analysis

After reviewing the problem, and introducting the assumptions, we will follow by an analysis of the search space and a description of how we model the planner. We will provide in-depth explanations on the objects, predicates, and operators used, with emphasis on motivating the choices made.

As a note, we model all food — dishes and ingredients — as "ingredients"; this is due to the capacity of our system for creating intermediate ingredients, to be used for making even more complex food. Therefore, for simplicity, we model dishes as "ingredients" too, which could cause confusion if not taken into account.

2.1 Search Space

The search space represents all possible configurations (states) that the system can be in. In the context of our problem, this includes every combination of:

- States of the robot (S_R)
- States of the tools (S_T)
- States of the ingredients (S_I)

Let L be the number of locations ($L = 7$), T be the number of tools ($T = 3$), and I be the number of ingredients. We also have $A = 3$ possible actions to perform on ingredients.

Starting with the robot, it can be at any moment in one of the $L = 7$ locations; moreover, it can be holding either nothing, one of the T tools, or one of the I ingredients. This results in S_R possible robot states:

$$S_R = L \times (1 + T + I) \tag{1}$$

Moving on to tools, each of them can either be at either of L locations or held by the robot, although we will not count the latter possibility as it has already been accounted for above; each tool can also either be clean or dirty. For a single tool $t \in T$, we have $S_t possible states$:

$$S_t = 2 \times L \tag{2}$$

As we have T tools, the total number of tool states (S_T) is:

$$S_T = (2 \times L)^T \tag{3}$$

The calculation gets trickier for ingredients, as there are more permutations:

- They can be in any of L locations (or being held, although that won't be counted for the same argument as for tools)
- They can exist (e.g. from the beginning or when created) or not exist (e.g. when consumed)
- They can require (or have been through) any of A actions (e.g. cut). Some ingredients need no actions, some need more than one; to simplify matters, we will assume all ingredients need all A actions, so our calculation will only provide an upper bound. Therefore there are $A + 1$ possible states for each ingredient regarding actions: still requiring all A actions to be performed, one action has been performed and still requiring $A - 1$ actions, ..., no required actions left.
- They can be served or not. We count all food as ingredients, therefore some are already final product (e.g. sushi), which must be served to the customer.

For each ingredient $i \in I$ we have S_i possible states:

$$S_i \leq L \times 2 \times (A + 1) \times 2 \tag{4}$$
$$S_i \leq 4 \times L \times (A + 1) \tag{5}$$

And the total state space for all ingredients I is S_i:

$$S_I \leq (4 \times L \times (A + 1))^I \tag{6}$$

Therefore the total number of states is:

$$\text{Total States} \leq S_R \times S_T \times S_I \tag{7}$$
$$\text{Total States} \leq L \times (1 + T + I) \times (2 \times L)^T \times (4 \times L \times (A + 1))^I \tag{8}$$

Substituting the fixed values $L = 7$, $T = 3$, and $A = 3$:

$$\text{Total States} \leq 7 \times (1 + 3 + I) \times (2 \times 7)^3 \times (4 \times 7 \times (3 + 1))^I \tag{9}$$
$$\text{Total States} \leq 7 \times (4 + I) \times 2744 \times (112)^I \tag{10}$$
$$\text{Total States} \leq 19208 \times (4 + I) \times (112)^I \tag{11}$$
$$\tag{12}$$

Note this is an upper bound given the assumption made on actions to be performed on ingredients.S

2.2 Types

The types encapsulate the diversity of elements that the problem captures. In this problem we have:

- **Location**: it is used to declare the different parts of the kitchen. The properties of these parts are encoded via predicates.
- **Action**: encapsulates the different aspects of cooking a dish, for example cutting, boiling or mixing.
- **Item**: generic type that is used in actions such as pick up, as both ingredients and tools can be held by the robot.
- **Tool** (subtype of item): the cookware that the robot uses to prepare the dishes (knife, pot, spoon...) falls under this category.
- **Ingredient** (subtype of item): this type encapsulates both the raw ingredients as the dishes themselves, as those can be used to prepare other more complex dishes (for example, cooking noodles and then preparing ramen with them).

There is no need for a robot object, as there is only one and its behaviour and state can be fully modelled though predicates and actions.

2.3 Predicates

Predicates tell us more information about the state of the objects (whether a dish is already served, if the ingredient is cooked...) and the world itself. For this problem we used the following predicates

- `(robot-at ?loc - location)`: specifies that the robot is currently in a given part of the kitchen.
- `(item-at ?item - item ?loc - location)`: indicates the location of a given item:
- `(tool-clean ?tool - tool)`: for a specific tool, it tells whether it is clean or dirty (has been used and not still washed).
- `(robot-holding ?item - item)`: for a specific item (ingredient or tool), indicates that the robot is holding it.
- `(is-robot-holding-something)`: generic predicate that indicates that the robot is holding some item.
- `(adjacent ?loc1 - location ?loc2 - location)`: are two locations connected, that is, can the robot go from one to the other.
- `(location-is-PA ?loc - location)`: predicate that specifies that an object of type location is the preparing area.
- `(location-is-SVA ?loc - location)`: predicate that specifies that an object of type location is the serving area.
- `(location-is-DWA ?loc - location)`: predicate that specifies that an object of type location is the dish washing area.
- `(dish-is-served ?dish - ingredient)`: for an ingredient (recall that dishes are also ingredients) indicates whether it is served.
- `(needs-ingredient ?dish - ingredient ?ingredient - ingredient)`: this predicate specifies that, in order to prepare some dish, a given ingredient is needed.
- `(already-crafted ?ing - ingredient)`: this predicate indicates wether an ingredient is present or has been prepared yet. For example, the problem has an ingredient object of type sushi, but it does not *exist* in the kitchen until it is prepared from its ingredients. All the base ingredients of a recipe are in a crafted state and stop being crafted once they are consumed to prepare something.

- `(ingredient-needs-action ?ingredient - ingredient ?a - action)`: specifies that a given ingredient needs an action (such as cooking, cutting, mixing...) before it is ready to be used in a recipe.
- `(ingredient-is-ready ?ingredient - ingredient)`: an ingredient is considered ready once all actions needed by it are performed. For example, vegetables might need to be cut and boiled before being ready.
- `(action-predecence-for-ingredient ?act_pre -action ?act_post - action ?ingredient - ingredient)`: some actions on a given ingredient need to be done before others, which is what this predicate indicates. For example, it does not make sense to first mix and then boil rice.
- `(loc-for-action ?loc - location ?a - action)`: indicates that a given action has to be performed in a specific location. For example, the action cutting has to be done in the location object that corresponds to the cutting area.
- `(tool-for-action ?tool - tool ?a - action)`: specifies that an action has to be done while the robot holds a type of tool. For instance, the action cutting can only be done while the robot is holding a knife.

2.4 Actions

Actions in a PDDL domain specify the evolution and allowed change in the state of the problem. The design choices made in the domain that we have designed strive for the greatest level of generalization possible. Thus, some of the actions can result a bit complex (specially `action-on-ingredient` and `prepare`). The list of actions is:

- **Move**
  - **Action**: `move(?from - location ?to - location)`
  - **Description**: the robot moves from location `?from` to an adjacent location `?to`, as long as those are adjacent.
- **Pickup item**
  - **Action**: `pick-up-item(?loc - location ?item - item)`
  - **Description**: the robot holds `?item` if both are at `?loc`. `is-robot-holding-something` is set.
- **Drop off item**

  - **Action**: `drop-off-item(?loc - location ?item - item)`
  - **Description**: the robot drops `?item` at `?loc`, as long as it was holding it. `is-robot-holding-something` is cleared.

- **Clean tool**
  - **Action**: `clean-tool(?tool - tool ?loc - location)`
  - **Description**: as long as the robot and the tool are in the dishwashing area and the tool is not already clean, the robot cleans the tool and `tool-clean` is set.

- **Action on ingredient**
  - **Action**: `action-on-ingredient(?loc - location ?ingredient - ingredient ?tool - tool ?a - action)`
  - **Description**: as long as the robot and the ingredient are in the appropriate location for the action, the robot is holding the appropriate tool and all the previous actions on the ingredient (if any) have been done, the robot performs the needed action on the ingredient.

- **Prepare**
  - **Action**: `prepare(?loc - location ?result - ingredient)`
  - **Description**: as long as all the necessary actions have been done on the ingredients needed and the robot is at the preparing area, the required ingredients get consumed to produce the resulting dish.

- **Serve dish**
  - **Action**: `serve-dish(?loc - location ?dish_ingredint - ingredient)`
  - **Description**: if the ingredient that represent the dish exists and is being held by the robot, who has to be in the serving area, then the robot can serve the dish.

3 PDDL Implementation

Before showing the code itself, we want to emphasize that we strive for the greatest level of generalization possible with this domain. The original state indicates that the actions are mixing, cutting and cooking, but nothing forbids from adding another one, such as amassing, which can also have its own specific tool, without

modifying the domain. Similarly, any dish with any number of intermediate steps and ingredients can be encoded in the problem without changing the domain. This robot can work in any kind of restaurant with this problem definition.

The following is the PDDL domain file that defines the actions, predicates, and functions for the rescue drone problem as discussed above:

Listing 1: PDDL Domain Definition for Rescue Drone

```
(define (domain robot-chef)
  (:requirements :strips :typing :negative-preconditions)

  (:types
    item
    tool ingredient - item

    location
    action
  )

  (:predicates
    (robot-at ?loc - location)
    (item-at ?item - item ?loc - location)

(tool-clean ?tool - tool)
    (robot-holding ?item - item)
    (is-robot-holding-something)

    (adjacent ?loc1 - location ?loc2 - location)


    ; for locations
    (location-is-PA ?loc - location)
    (location-is-SVA ?loc - location)
    (location-is-DWA ?loc - location)

    ; dish
    (dish-is-served ?dish - ingredient)
    (needs-ingredient ?dish - ingredient ?ingredient -
       ingredient)
    (already-crafted ?ing - ingredient)

    ; for preparing ingredient
    (ingredient-needs-action ?ingredient - ingredient ?a -
       action)
```

```
    (ingredient-is-ready ?ingredient - ingredient)

    ; logistics
    (action-predecence-for-ingredient ?act_pre -action ?
       act_post - action ?ingredient - ingredient)
    (loc-for-action ?loc - location ?a - action)
    (tool-for-action ?tool - tool ?a - action)
  )

  (:action move
      :parameters (
        ?from - location ?to - location
      )
      :precondition (and
        (robot-at ?from)
        (or
          (adjacent ?from ?to)
          (adjacent ?to ?from)
        )
      )
      :effect (and
        (not (robot-at ?from))
        (robot-at ?to)
      )
  )

  (:action pick-up-item
      :parameters (?loc - location ?item - item)
      :precondition (and
        (robot-at ?loc)
        (item-at ?item ?loc)
        (not (is-robot-holding-something))
      )
      :effect (and
        (not (item-at ?item ?loc))
        (robot-holding ?item)
        (is-robot-holding-something)
      )
  )

  (:action drop-off-item
      :parameters (?loc - location ?item - item)
      :precondition (and
```

```
77          (robot-at ?loc)
78          (robot-holding ?item)
79          (is-robot-holding-something)
80        )
81        :effect (and
82          (item-at ?item ?loc)
83          (not (robot-holding ?item))
84          (not (is-robot-holding-something))
85        )
86    )
87
88    (:action clean-tool
89        :parameters (?tool - tool ?loc - location)
90        :precondition (and
91          (robot-at ?loc)
92          (location-is-DWA ?loc)
93          (robot-holding ?tool)
94          (not (tool-clean ?tool))
95        )
96        :effect (and
97          (tool-clean ?tool)
98        )
99    )
100
101
102   (:action action-on-ingredient
103       :parameters (?loc - location ?ingredient - ingredient ?
              tool - tool ?a - action)
104       :precondition (and
105         (robot-at ?loc)
106         (item-at ?ingredient ?loc)
107         (robot-holding ?tool)
108         (tool-clean ?tool)
109
110         (ingredient-needs-action ?ingredient ?a)
111         (tool-for-action ?tool ?a)
112         (loc-for-action ?loc ?a)
113
114         ; Check that all prerequisite actions for this one are
              fulfilled
115         (not
116           (exists (?pre_action - action)
117             (and
```

```
118                 (action-predecence-for-ingredient ?pre_action ?a
                        ?ingredient)
119                 (ingredient-needs-action ?ingredient ?pre_action)
120               )
121             )
122           )
123         )
124         :effect (and
125           (not (tool-clean ?tool))
126           (not (ingredient-needs-action ?ingredient ?a))
127           ; If there are no other actions left fot this
                ingredient, mark it as ready
128           (when ;when (a) => (b)
129             (not
130               (exists (?action_to_do - action)
131                 (and (ingredient-needs-action ?ingredient ?
                        action_to_do)
132                   (not (= ?action_to_do ?a)) ; current action
                          is not deleted imediately
133                 )
134               )
135             )
136             (ingredient-is-ready ?ingredient)
137           )
138         )
139     )
140
141     (:action prepare
142         :parameters (?loc - location ?result - ingredient)
143         :precondition (
144           and
145             (not (already-crafted ?result))
146             (robot-at ?loc)
147             (location-is-PA ?loc)
148             ; The result requires at least one ingredient (to
                  prevent the solver from crafting stuff
149             ; from nothing)
150             (exists (?ing - ingredient) (needs-ingredient ?result
                    ?ing))
151             ; Check that all all ingredients are either ready or
                  do not require any action on them
152             (forall (?ingredient - ingredient)
153               (or
```

```
154                 (not (needs-ingredient ?result ?ingredient))
155                 (and
156                   ; Either the ingredient is ready or it does not
                          need any action
157                   (or (ingredient-is-ready ?ingredient)
158                       (not (exists (?a - action) (ingredient-
                              needs-action ?ingredient ?a)))
159                   )
160                   (item-at ?ingredient ?loc)
161                 )
162               )
163             )
164       )
165       :effect (
166         and
167           ; "Remove" items by not placing them anywhere (not
                 accesible by robot)
168           (forall (?ingredient - ingredient)
169             (when
170               (needs-ingredient ?result ?ingredient)
171               (not (item-at ?ingredient ?loc)) ; disappears
172             )
173           )
174           (robot-holding ?result) ; hold == prepared!!
175           (already-crafted ?result)
176       )
177   )
178
179   (:action serve-dish
180       :parameters (?loc - location ?dish_ingredint - ingredient
              )
181       :precondition (
182         and
183           (robot-at ?loc)
184           (location-is-SVA ?loc)
185           (robot-holding ?dish_ingredint)
186
187       )
188       :effect (
189         and
190           (dish-is-served ?dish_ingredint)
191           (not (robot-holding ?dish_ingredint))
192       )
```

```
    )
)
```

It is worth noting that we have chosen the number of steps as the metric to minimize. We decided not to introduce a specific cost variable for optimization, as we assumed that all actions have equal weight. For example, while we considered minimizing the number of robot movements, this goal is inherently addressed by reducing the total number of steps.

4 Experiments

The code was tested using a variety of problem cases, including different dishes, ingredients and tool usages. This experimentation aims to assess the planner's ability to navigate action dependencies, respect conditional requirements and optimize its search for feasible solutions under varied scenarios. Each case provides different insights into the planner's scalability and its robustness in managing complex tasks. The planner used in our experiments is Metric-FF [1].

4.1 Test Case 1 - Sashimi

The aim of the first test was to evaluate the correct operation of the domain. We will focus on location, tool and item constraints. We have chosen a dish of just one ingredient: sashimi. The preparation consists in picking the fish and cut it. The problem's code can be found in section 5.

Results and Analysis

```

ff: parsing domain file
domain 'ROBOT-CHEF' defined
 ... done.
ff: parsing problem file
problem 'SASHIMI' defined
 ... done.


no metric specified. plan length assumed.

task contains conditional effects. turning off state domination.



checking for cyclic := effects --- OK.

ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
    metric is  plan length

```

```
advancing to distance:   13
                         12
                         10
                          9
                          8
                          7
                          6
                          5
                          4
                          3
                          2
                          1
                          0

ff: found legal plan as follows

step    0: MOVE CA PA
        1: MOVE PA MIXA
        2: MOVE MIXA SA
        3: PICK-UP-ITEM SA FISH
        4: MOVE SA CTA
        5: DROP-OFF-ITEM CTA FISH
        6: PICK-UP-ITEM CTA KNIFE
        7: ACTION-ON-INGREDIENT CTA FISH KNIFE CUTTING
        8: DROP-OFF-ITEM CTA KNIFE
        9: PICK-UP-ITEM CTA FISH
       10: MOVE CTA MIXA
       11: MOVE MIXA PA
       12: DROP-OFF-ITEM PA FISH
       13: PREPARE PA SASHIMI
       14: MOVE PA CA
       15: MOVE CA SVA
       16: SERVE-DISH SVA SASHIMI


time spent:    0.00 seconds instantiating 81 easy, 16 hard action templates
               0.00 seconds reachability analysis, yielding 66 facts and 90 actions
               0.00 seconds creating final representation with 58 relevant facts, 0
                    relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.00 seconds searching, evaluating 46 states, to a max depth of 0
               0.00 seconds total time
```

The robot navigates only between adjacent locations: i.e. it starts at PA and before reaching SA, it passes through MIXA (steps 0 to 2). Specific actions are restricted to specific areas: it can only pick up fish within the SA area (steps 4 and 5). The robot selects tools prior to executing tasks; it respects the conditional order of the actions. For example, it picks up a knife before cutting the fish (steps 6 and 7). Finally, the robot completes the process by preparing the sashimi and serving it in the SVA (steps 12 to 16).

Notably, the solver employs an heuristic-driven best-first search. The function g(s)+5*h(s) balances the current cost with the estimated future costs. In specific, $g(s)$ represents the actual cost in order to reach the current state s, while $h(s)$ provides an estimate of the remaining cost to reach the goal.

The planner has evaluated 46 states. It can be seen that the maximum depth was 0, which means that a solution was quickly found, without searching deeply into alternative paths.

4.2 Test Case 2 - Sushi

The second test was proposed in the assignment statement: preparing and serving sushi. This dish requires 3 ingredients: fish, rice and seaweed, which require prior action preparation. The robot needs to boil and mix the rice (in that order), cut the fish and assemble both with seaweed. The problem's code can be found in section 5.

Results and Analysis

```
1
2  ff: parsing domain file
3  domain 'ROBOT-CHEF' defined
4   ... done.
5  ff: parsing problem file
6  problem 'SUSHI' defined
7   ... done.
8
9
10 no metric specified. plan length assumed.
11
12 task contains conditional effects. turning off state domination.
13
14
15
16 checking for cyclic := effects --- OK.
17
18 ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
19     metric is  plan length
20
21 advancing to distance:   22
22                          21
23                          20
24                          19
25                          18
26                          17
27                          16
28                          15
29                          14
30                          13
31                          12
32                          11
33                          10
34                           9
35                           8
36                           7
37                           6
38                           5
39                           4
40                           3
41                           2
```

```
                        1
                        0

ff: found legal plan as follows

step    0: MOVE CA PA
        1: MOVE PA MIXA
        2: MOVE MIXA SA
        3: PICK-UP-ITEM SA RICE
        4: MOVE SA MIXA
        5: DROP-OFF-ITEM MIXA RICE
        6: MOVE MIXA SA
        7: PICK-UP-ITEM SA FISH
        8: MOVE SA CTA
        9: DROP-OFF-ITEM CTA FISH
       10: PICK-UP-ITEM CTA KNIFE
       11: ACTION-ON-INGREDIENT CTA FISH KNIFE CUTTING
       12: DROP-OFF-ITEM CTA KNIFE
       13: PICK-UP-ITEM CTA FISH
       14: MOVE CTA MIXA
       15: MOVE MIXA PA
       16: DROP-OFF-ITEM PA FISH
       17: MOVE PA MIXA
       18: MOVE MIXA SA
       19: PICK-UP-ITEM SA SEAWEED
       20: MOVE SA MIXA
       21: MOVE MIXA PA
       22: DROP-OFF-ITEM PA SEAWEED
       23: PICK-UP-ITEM PA POT
       24: MOVE PA CA
       25: DROP-OFF-ITEM CA POT
       26: MOVE CA PA
       27: MOVE PA MIXA
       28: PICK-UP-ITEM MIXA RICE
       29: MOVE MIXA PA
       30: MOVE PA CA
       31: DROP-OFF-ITEM CA RICE
       32: PICK-UP-ITEM CA POT
       33: ACTION-ON-INGREDIENT CA RICE POT COOKING
       34: DROP-OFF-ITEM CA POT
       35: PICK-UP-ITEM CA RICE
       36: MOVE CA PA
       37: MOVE PA MIXA
       38: DROP-OFF-ITEM MIXA RICE
       39: PICK-UP-ITEM MIXA SPOON
       40: ACTION-ON-INGREDIENT MIXA RICE SPOON MIXING
       41: DROP-OFF-ITEM MIXA SPOON
       42: PICK-UP-ITEM MIXA RICE
       43: MOVE MIXA PA
       44: DROP-OFF-ITEM PA RICE
       45: PREPARE PA SUSHI
       46: MOVE PA CA
       47: MOVE CA SVA
       48: SERVE-DISH SVA SUSHI


time spent:    0.00 seconds instantiating 105 easy, 21 hard action templates
               0.00 seconds reachability analysis, yielding 97 facts and 126 actions
               0.00 seconds creating final representation with 85 relevant facts, 0
                   relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
```

```
                0.00 seconds searching, evaluating 486 states, to a max depth of 0
                0.00 seconds total time
```

Similar to the previous test, the robot moves through different areas, picks up ingredients, uses tools and finally serves the prepared dish. The robot collects all the ingredients needed before cooking the meal and the action precedence is followed.

The final path contains 49 steps, as the number of dependencies for the final dish is higher than in the Sashimi case (more ingredients and preparation steps). Likewise, the number of evaluated states has been extended to 486.

4.3 Test Case 3 - Unsolvable

This experiment tests the planner's response to an unsolvable scenario. The problem's code can be found in section 5. The sushi requires to have seaweed as one of its ingredients, but there is no initial state defining the availability of seaweed. The goal of this test is to ensure the planner correctly identifies the situation as infeasible and terminates efficiently.

Results and Analysis

```

ff: parsing domain file
domain 'ROBOT-CHEF' defined
 ... done.
ff: parsing problem file
problem 'SUSHI' defined
 ... done.


ff: goal can be simplified to FALSE. No plan will solve it
```

The planner successfully identified the position is not solvable and returned **FALSE**. Unfortunately no extra information (such as how long it took to evaluate, or the nodes expanded until the realisation it was unsolvable) is provided, limiting the depth of the current analysis.

4.4 Test Case 4 - Ramen

In this fourth test, the dish to be served is ramen. Noodles are not in the initial state and need to be created by mixing flour and boiling water. In addition, the robot has to cook the noodles, cut the vegetables and boil them, as well as the broth. Ramen is made as a combination of all those ingredients. We will examine the requirement of tools needed to be cleaned whenever is used more than once. This constraint adds an additional layer of complexity. The problem's code can be found in section 5.

Results and Analysis

```
1
2  ff: parsing domain file
3  domain 'ROBOT-CHEF' defined
4   ... done.
5  ff: parsing problem file
6  problem 'RAMEN' defined
7   ... done.
8
9
10 no metric specified. plan length assumed.
11
12 task contains conditional effects. turning off state domination.
13
14
15
16 checking for cyclic := effects --- OK.
17
18 ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
19     metric is  plan length
20
21 advancing to distance:   32
22                          31
23                          30
24                          29
25                          28
26                          27
27                          26
28                          25
29                          24
30                          23
31                          22
32                          21
33                          20
34                          19
35                          18
36                          17
37                          16
38                          14
39                          13
40                          12
41                          11
42                          10
43                           9
44                           8
45                           7
46                           6
47                           5
48                           4
49                           3
50                           2
51                           1
52                           0
53
54 ff: found legal plan as follows
55
56 step    0: MOVE CA PA
57         1: MOVE PA MIXA
58         2: MOVE MIXA SA
59         3: PICK-UP-ITEM SA VEGETABLES
```

```
 4: MOVE SA CTA
 5: DROP-OFF-ITEM CTA VEGETABLES
 6: PICK-UP-ITEM CTA KNIFE
 7: ACTION-ON-INGREDIENT CTA VEGETABLES KNIFE CUTTING
 8: DROP-OFF-ITEM CTA KNIFE
 9: PICK-UP-ITEM CTA VEGETABLES
10: MOVE CTA MIXA
11: MOVE MIXA PA
12: DROP-OFF-ITEM PA VEGETABLES
13: MOVE PA MIXA
14: MOVE MIXA SA
15: PICK-UP-ITEM SA FLAVOUR
16: MOVE SA MIXA
17: DROP-OFF-ITEM MIXA FLAVOUR
18: PICK-UP-ITEM MIXA SPOON
19: ACTION-ON-INGREDIENT MIXA FLAVOUR SPOON MIXING
20: DROP-OFF-ITEM MIXA SPOON
21: PICK-UP-ITEM MIXA FLAVOUR
22: MOVE MIXA PA
23: DROP-OFF-ITEM PA FLAVOUR
24: PICK-UP-ITEM PA POT
25: MOVE PA MIXA
26: DROP-OFF-ITEM MIXA POT
27: MOVE MIXA SA
28: PICK-UP-ITEM SA WATER
29: MOVE SA MIXA
30: MOVE MIXA PA
31: DROP-OFF-ITEM PA WATER
32: MOVE PA MIXA
33: MOVE MIXA SA
34: PICK-UP-ITEM SA BROTH
35: MOVE SA MIXA
36: MOVE MIXA PA
37: DROP-OFF-ITEM PA BROTH
38: MOVE PA MIXA
39: PICK-UP-ITEM MIXA POT
40: MOVE MIXA PA
41: MOVE PA CA
42: DROP-OFF-ITEM CA POT
43: MOVE CA PA
44: PICK-UP-ITEM PA WATER
45: MOVE PA CA
46: DROP-OFF-ITEM CA WATER
47: PICK-UP-ITEM CA POT
48: ACTION-ON-INGREDIENT CA WATER POT COOKING
49: MOVE CA DWA
50: CLEAN-TOOL POT DWA
51: MOVE DWA CA
52: DROP-OFF-ITEM CA POT
53: PICK-UP-ITEM CA WATER
54: MOVE CA PA
55: DROP-OFF-ITEM PA WATER
56: PREPARE PA NOODLES
57: MOVE PA CA
58: PICK-UP-ITEM CA POT
59: DROP-OFF-ITEM CA NOODLES
60: ACTION-ON-INGREDIENT CA NOODLES POT COOKING
61: PICK-UP-ITEM CA NOODLES
62: MOVE CA PA
63: DROP-OFF-ITEM PA NOODLES
64: PICK-UP-ITEM PA VEGETABLES
65: MOVE PA CA
```

```
       66: DROP-OFF-ITEM CA VEGETABLES
       67: MOVE CA DWA
       68: CLEAN-TOOL POT DWA
       69: MOVE DWA CA
       70: ACTION-ON-INGREDIENT CA VEGETABLES POT COOKING
       71: MOVE CA DWA
       72: CLEAN-TOOL POT DWA
       73: MOVE DWA CA
       74: PICK-UP-ITEM CA VEGETABLES
       75: MOVE CA PA
       76: DROP-OFF-ITEM PA VEGETABLES
       77: PICK-UP-ITEM PA BROTH
       78: MOVE PA CA
       79: DROP-OFF-ITEM CA BROTH
       80: ACTION-ON-INGREDIENT CA BROTH POT COOKING
       81: PICK-UP-ITEM CA BROTH
       82: MOVE CA PA
       83: DROP-OFF-ITEM PA BROTH
       84: PREPARE PA RAMEN
       85: MOVE PA CA
       86: MOVE CA SVA
       87: SERVE-DISH SVA RAMEN


time spent:    0.00 seconds instantiating 135 easy, 32 hard action templates
               0.00 seconds reachability analysis, yielding 130 facts and 167 actions
               0.00 seconds creating final representation with 114 relevant facts, 0
                    relevant fluents
               0.00 seconds computing LNF
               0.00 seconds building connectivity graph
               0.04 seconds searching, evaluating 4996 states, to a max depth of 0
               0.04 seconds total time
```

Preparing ramen requires multiple ingredients to be boiled in a pot. For this reason, the robot must incorporate extra steps to clean the tool before each reuse. An example is in the case of cooking noodles and later on, vegetables (steps from 61 to 71).

The proposed solution comprises a total of 88 steps. The planner explored 4996 states, indicating a more complex search space due to the multiple stage of tool reuse. However, the time complexity is only of 0.04s, remaining very efficient.

4.5 Test Case 5 - Multiple

In this final test case, the robot must prepare and serve both sashimi and sushi. We will analyse how it handles having multiple orders. The problem's code can be found in section 5.

Results and Analysis

```
ff: parsing domain file
domain 'ROBOT-CHEF' defined
ff: parsing problem file
problem 'MULTIPLE' defined
```

```
... done


no metric specified. plan length assumed.

task contains conditional effects. turning off state domination.


checking for cyclic := effects --- OK.

ff: search configuration is  best-first on 1*g(s) + 5*h(s) where
    metric is  plan length

advancing to distance:   30
                         29
                         28
                         27
                         26
                         25
                         24
                         23
                         22
                         21
                         20
                         19
                         18
                         17
                         16
                         15
                         14
                         13
                         12
                         11
                         10
                          9
                          8
                          7
                          6
                          5
                          4
                          3
                          2
                          1
                          0

ff: found legal plan as follows

step    0: MOVE CA PA
        1: MOVE PA MIXA
        2: MOVE MIXA SA
        3: PICK-UP-ITEM SA FISH1
        4: MOVE SA CTA
        5: DROP-OFF-ITEM CTA FISH1
        6: MOVE CTA SA
        7: PICK-UP-ITEM SA RICE
        8: MOVE SA MIXA
        9: DROP-OFF-ITEM MIXA RICE
       10: MOVE MIXA SA
       11: PICK-UP-ITEM SA FISH2
       12: MOVE SA CTA
       13: DROP-OFF-ITEM CTA FISH2
       14: MOVE CTA SA
```

```
15: PICK-UP-ITEM SA SEAWEED
16: MOVE SA MIXA
17: MOVE MIXA PA
18: DROP-OFF-ITEM PA SEAWEED
19: MOVE PA MIXA
20: MOVE MIXA CTA
21: PICK-UP-ITEM CTA KNIFE
22: ACTION-ON-INGREDIENT CTA FISH1 KNIFE CUTTING
23: DROP-OFF-ITEM CTA KNIFE
24: PICK-UP-ITEM CTA FISH1
25: MOVE CTA MIXA
26: MOVE MIXA PA
27: DROP-OFF-ITEM PA FISH1
28: PREPARE PA SASHIMI
29: MOVE PA CA
30: MOVE CA SVA
31: SERVE-DISH SVA SASHIMI
32: MOVE SVA CA
33: MOVE CA PA
34: MOVE PA MIXA
35: MOVE MIXA CTA
36: PICK-UP-ITEM CTA KNIFE
37: MOVE CTA MIXA
38: MOVE MIXA PA
39: MOVE PA CA
40: MOVE CA DWA
41: CLEAN-TOOL KNIFE DWA
42: MOVE DWA CA
43: MOVE CA PA
44: MOVE PA MIXA
45: MOVE MIXA CTA
46: ACTION-ON-INGREDIENT CTA FISH2 KNIFE CUTTING
47: DROP-OFF-ITEM CTA KNIFE
48: PICK-UP-ITEM CTA FISH2
49: MOVE CTA MIXA
50: MOVE MIXA PA
51: MOVE PA CA
52: DROP-OFF-ITEM CA FISH2
53: MOVE CA PA
54: PICK-UP-ITEM PA POT
55: MOVE PA CA
56: ACTION-ON-INGREDIENT CA FISH2 POT COOKING
57: MOVE CA DWA
58: CLEAN-TOOL POT DWA
59: MOVE DWA CA
60: DROP-OFF-ITEM CA POT
61: PICK-UP-ITEM CA FISH2
62: MOVE CA PA
63: DROP-OFF-ITEM PA FISH2
64: MOVE PA MIXA
65: PICK-UP-ITEM MIXA RICE
66: MOVE MIXA PA
67: MOVE PA CA
68: DROP-OFF-ITEM CA RICE
69: PICK-UP-ITEM CA POT
70: ACTION-ON-INGREDIENT CA RICE POT COOKING
71: DROP-OFF-ITEM CA POT
72: PICK-UP-ITEM CA RICE
73: MOVE CA PA
74: MOVE PA MIXA
75: DROP-OFF-ITEM MIXA RICE
76: PICK-UP-ITEM MIXA SPOON
```

```
129         77: ACTION-ON-INGREDIENT MIXA RICE SPOON MIXING
130         78: DROP-OFF-ITEM MIXA SPOON
131         79: PICK-UP-ITEM MIXA RICE
132         80: MOVE MIXA PA
133         81: DROP-OFF-ITEM PA RICE
134         82: PREPARE PA SUSHI
135         83: MOVE PA CA
136         84: MOVE CA SVA
137         85: SERVE-DISH SVA SUSHI
138
139
140 time spent:    0.00 seconds instantiating 135 easy, 25 hard action templates
141                0.00 seconds reachability analysis, yielding 127 facts and 160 actions
142                0.00 seconds creating final representation with 110 relevant facts, 0
                       relevant fluents
143                0.00 seconds computing LNF
144                0.00 seconds building connectivity graph
145                0.03 seconds searching, evaluating 4590 states, to a max depth of 0
146                0.03 seconds total time
```

The robot successfully prepares and serves both dishes. Similar to the previous test, it must clean tools before reusing it. In this case, the knife is used to cut `fish1` and `fish2`, treated as distinct ingredients. It is worth to note that one same ingredient can have different preparation methods, depending on the dish. Such is the case of fish.

In terms of complexity, the solution involved 86 steps, with the planner evaluating a total of 4590 states. The computation time was of 0.03, confirming the planner's efficiency in finding a solution.

5 Final Analysis and Conclusion

In this practice, we developed a PDDL program that models the actions of a robotic chef tasked with preparing and serving dishes within a restaurant environment. The domain was designed to allow the robot to move, collect ingredients, perform preparation actions and serve the completed dishes. We conducted a series of experiments across varied scenarios, each introducing different levels of complexity and dependencies among actions. The experiments tested the efficiency of the planner and allowed us to explore the complexity of the problem as well as its search space.

A global analysis of the tests reveals how different factors - number of ingredients or dishes, action dependencies and tool cleaning requirements - impact the complexity and efficiency of the solution. Simpler tasks with fewer dependencies, like in the test case 1 (*sashimi*), led to shorter solution paths and fewer nodes expanded, while more complex tasks in the cases 4 and 5 (*ramen* and *multiple*, respectively) significantly increased the number of steps and states explored.

During the design phase we also considered other alternatives to our assumptions. One option was to define meal substitutes for cases when certain ingredient

were unavailable, as seen in test 3. Another improvement could involve using *fluents* to represent ingredient quantities and define predicates to help reduce the need for tool cleaning when working consecutively with the same type of ingredient. This approach would optimize the process by soften redundant constraints, enhancing the robot's efficiency in meal preparation.

This modification would widen the scope of the assignment excessively, not permitting depth of analysis. Nevertheless, it could be interesting to explore these in future work.

References

[1] Joerg Hoffmann. The metric-ff planning system. https://fai.cs.uni-saarland.de/hoffmann/metric-ff.html, 2024. Top Performer in the Numeric Track of the 3rd International Planning Competition.

[2] OpenAI. This image was created with the assistance of dall·e 2. https://openai.com/dall-e-2, 2024. Accessed: 12-10-2024.

Annex A: Complete PDDL Problem Files

Below are the pddl problem files for each one of the examples shown before.

A.1: Example 1

Listing 2: Problem definition for example 1

```
(define (problem sashimi)
  (:domain robot-chef)

  (:objects
    ; GENERIC OBJECTS
    knife pot spoon - tool
    SA PA CA SVA DWA CTA MIXA - location
    cutting mixing cooking - action

    ; PROBLEM SPECIFIC OBJECTS
    fish sashimi - ingredient
  )

  (:init
    ; GENERIC PREDICATES
    (robot-at CA)

    (adjacent SA MIXA)
    (adjacent SA CTA)
    (adjacent MIXA PA)
    (adjacent MIXA CTA)
    (adjacent PA MIXA)
    (adjacent PA CA)
    (adjacent CA DWA)
    (adjacent CA SVA)

    (location-is-PA PA)
    (location-is-SVA SVA)
    (location-is-DWA DWA)

    (tool-for-action knife cutting)
    (tool-for-action pot cooking)
    (tool-for-action spoon mixing)

    (tool-clean knife)
```

```
    (tool-clean pot)
    (tool-clean spoon)

    (item-at knife CTA)
    (item-at pot PA)
    (item-at spoon MIXA)

    (loc-for-action CTA cutting)
    (loc-for-action CA cooking)
    (loc-for-action MIXA mixing)

    ; PROBLEM SPECIFIC PREDICATES
    (already-crafted fish)

    (item-at fish SA)

    (ingredient-needs-action fish cutting)

    (needs-ingredient sashimi fish)
  )

  (:goal
    (and
      (dish-is-served sashimi)
    )
  )
)
```

A.2: Example 2

Listing 3: Problem definition for example 2

```
(define (problem sushi)
  (:domain robot-chef)

  (:objects
    ; GENERIC OBJECTS
    knife pot spoon - tool
    SA PA CA SVA DWA CTA MIXA - location
    cutting mixing cooking - action

    ; PROBLEM SPECIFIC OBJECTS
```

```
    sushi rice seaweed fish - ingredient
)

(:init
    ; GENERIC PREDICATES
    (robot-at CA)

    (adjacent SA MIXA)
    (adjacent SA CTA)
    (adjacent MIXA PA)
    (adjacent MIXA CTA)
    (adjacent PA MIXA)
    (adjacent PA CA)
    (adjacent CA DWA)
    (adjacent CA SVA)

    (location-is-PA PA)
    (location-is-SVA SVA)
    (location-is-DWA DWA)

    (tool-for-action knife cutting)
    (tool-for-action pot cooking)
    (tool-for-action spoon mixing)

    (tool-clean knife)
    (tool-clean pot)
    (tool-clean spoon)

    (item-at knife CTA)
    (item-at pot PA)
    (item-at spoon MIXA)

    (loc-for-action CTA cutting)
    (loc-for-action CA cooking)
    (loc-for-action MIXA mixing)

    ; PROBLEM SPECIFIC PREDICATES

    (already-crafted rice)
    (already-crafted fish)
    (already-crafted seaweed)

    (item-at rice SA)
```

```
    (item-at fish SA)
    (item-at seaweed SA)

    (ingredient-needs-action fish cutting)
    (ingredient-needs-action rice cooking)
    (ingredient-needs-action rice mixing)

    (action-predecence-for-ingredient cooking mixing rice)

    (needs-ingredient sushi fish)
    (needs-ingredient sushi rice)
    (needs-ingredient sushi seaweed)
  )

  (:goal
    (and
      (dish-is-served sushi)
    )
  )
)
```

A.3: Example 3

Listing 4: Problem definition for example 3

```
(define (problem sushi)
  (:domain robot-chef)

  (:objects
    ; GENERIC OBJECTS
    knife pot spoon - tool
    SA PA CA SVA DWA CTA MIXA - location
    cutting mixing cooking - action

    ; PROBLEM SPECIFIC OBJECTS
    sushi rice seaweed fish - ingredient
  )

  (:init
    ; GENERIC PREDICATES
    (robot-at CA)

```

```
    (adjacent SA MIXA)
    (adjacent SA CTA)
    (adjacent MIXA PA)
    (adjacent MIXA CTA)
    (adjacent PA MIXA)
    (adjacent PA CA)
    (adjacent CA DWA)
    (adjacent CA SVA)

    (location-is-PA PA)
    (location-is-SVA SVA)
    (location-is-DWA DWA)

    (tool-for-action knife cutting)
    (tool-for-action pot cooking)
    (tool-for-action spoon mixing)

    (tool-clean knife)
    (tool-clean pot)
    (tool-clean spoon)

    (item-at knife CTA)
    (item-at pot PA)
    (item-at spoon MIXA)

    (loc-for-action CTA cutting)
    (loc-for-action CA cooking)
    (loc-for-action MIXA mixing)

    ; PROBLEM SPECIFIC PREDICATES

    (already-crafted rice)
    (already-crafted fish)

    (item-at rice SA)
    (item-at fish SA)

    (ingredient-needs-action fish cutting)
    (ingredient-needs-action rice cooking)
    (ingredient-needs-action rice mixing)

    (action-predecence-for-ingredient cooking mixing rice)

```

```
    (needs-ingredient sushi fish)
    (needs-ingredient sushi rice)
    (needs-ingredient sushi seaweed)
  )

  (:goal
    (and
      (dish-is-served sushi)
    )
  )
)
```

A.4: Example 4

Listing 5: Problem definition for example 4

```
(define (problem ramen)
  (:domain robot-chef)

  (:objects
    ; GENERIC OBJECTS
    knife pot spoon - tool
    SA PA CA SVA DWA CTA MIXA - location
    cutting mixing cooking - action

    ; PROBLEM SPECIFIC OBJECTS
    noodles broth vegetables ramen flavour water - ingredient
  )

  (:init
    ; GENERIC PREDICATES
    (robot-at CA)

    (adjacent SA MIXA)
    (adjacent SA CTA)
    (adjacent MIXA PA)
    (adjacent MIXA CTA)
    (adjacent PA MIXA)
    (adjacent PA CA)
    (adjacent CA DWA)
    (adjacent CA SVA)

```

```
    (location-is-PA PA)
    (location-is-SVA SVA)
    (location-is-DWA DWA)

    (tool-for-action knife cutting)
    (tool-for-action pot cooking)
    (tool-for-action spoon mixing)

    (tool-clean knife)
    (tool-clean pot)
    (tool-clean spoon)

    (item-at knife CTA)
    (item-at pot PA)
    (item-at spoon MIXA)

    (loc-for-action CTA cutting)
    (loc-for-action CA cooking)
    (loc-for-action MIXA mixing)

    ; PROBLEM SPECIFIC PREDICATES

    (already-crafted vegetables)
    (already-crafted broth)
    (already-crafted flavour)
    (already-crafted water)

    (item-at flavour SA)
    (item-at water SA)
    (item-at broth SA)
    (item-at vegetables SA)

    (ingredient-needs-action vegetables cutting)
    (ingredient-needs-action vegetables cooking)
    (ingredient-needs-action noodles cooking)
    (ingredient-needs-action flavour mixing)
    (ingredient-needs-action broth cooking)
    (ingredient-needs-action water cooking)

    (action-predecence-for-ingredient cutting cooking
       vegetables)

    (needs-ingredient noodles flavour)
```

```
69      (needs-ingredient noodles water)
70
71      (needs-ingredient ramen vegetables)
72      (needs-ingredient ramen noodles)
73      (needs-ingredient ramen broth)
74    )
75
76    (:goal
77      (and
78        (dish-is-served ramen)
79      )
80    )
81  )
```

A.5: Example 5

Listing 6: Problem definition for example 5

```
1   (define (problem multiple)
2     (:domain robot-chef)
3
4     (:objects
5       ; GENERIC OBJECTS
6       knife pot spoon - tool
7       SA PA CA SVA DWA CTA MIXA - location
8       cutting mixing cooking - action
9
10      ; PROBLEM SPECIFIC OBJECTS
11      fish1 sashimi sushi rice seaweed fish2 - ingredient
12    )
13
14    (:init
15      ; GENERIC PREDICATES
16      (robot-at CA)
17
18      (adjacent SA MIXA)
19      (adjacent SA CTA)
20      (adjacent MIXA PA)
21      (adjacent MIXA CTA)
22      (adjacent PA MIXA)
23      (adjacent PA CA)
24      (adjacent CA DWA)
```

```
    (adjacent CA SVA)

    (location-is-PA PA)
    (location-is-SVA SVA)
    (location-is-DWA DWA)

    (tool-for-action knife cutting)
    (tool-for-action pot cooking)
    (tool-for-action spoon mixing)

    (tool-clean knife)
    (tool-clean pot)
    (tool-clean spoon)

    (item-at knife CTA)
    (item-at pot PA)
    (item-at spoon MIXA)

    (loc-for-action CTA cutting)
    (loc-for-action CA cooking)
    (loc-for-action MIXA mixing)

    ; PROBLEM SPECIFIC PREDICATES

    ; Sashimi instructions
    (already-crafted fish1)
    (item-at fish1 SA)
    (ingredient-needs-action fish1 cutting)
    (needs-ingredient sashimi fish1)

    ; Sushi instructions
    (already-crafted rice)
    (already-crafted fish2)
    (already-crafted seaweed)

    (item-at rice SA)
    (item-at fish2 SA)
    (item-at seaweed SA)

    (ingredient-needs-action fish2 cutting)
    (ingredient-needs-action fish2 cooking)
    (ingredient-needs-action rice cooking)
    (ingredient-needs-action rice mixing)
```

```
68
69      (action-predecence-for-ingredient cooking mixing rice)
70      (action-predecence-for-ingredient cutting cooking fish2)
71
72      (needs-ingredient sushi fish2)
73      (needs-ingredient sushi rice)
74      (needs-ingredient sushi seaweed)
75    )
76
77    (:goal
78      (and
79        (dish-is-served sashimi)
80        (dish-is-served sushi)
81      )
82    )
83  )
```