Report
Assignment 2
AI Planning for Autonomy
COMP90054_2023_SM1

ID & Email

Unimelb Student ID: 1262480
Student Email Address: shengqim@student.unimelb.edu.au

Link to the URL

http://editor.planning.domains/#read_session=2zdHbCnn2y

PDDL Domain Files

Domain File of Task 1:

```
(define (domain MakeTea)

   (:requirements
      :typing
   )

   (:types
      tapioca
      syrup
      ices
      teas
   )

   (:constants
      ;ice and tea do not have various types, so they are treated as constants
      ice - ices
      tea - teas
   )

   (:predicates
      (cup-exist)
      (iced)
      (ingredient-in-cup ?i)
      (tapioca-exist)
      (syrup-exist)
   )

   (:functions)
```

```
    (:action add-tapioca
        :parameters (?i - tapioca)
        :precondition (and
            (not(tapioca-exist))
                )
        :effect (and
            (ingredient-in-cup ?i)
            (tapioca-exist)
                )
    )

    (:action add-syrup
        :parameters (?i - syrup)
        :precondition (and
            (not(syrup-exist))
                )
        :effect (and
            (ingredient-in-cup ?i)
            (syrup-exist)
                )
    )

    (:action add-ice
        :parameters (?i - ices)
        :precondition (and
            (not(ingredient-in-cup ?i))
                )
        :effect (and
            (ingredient-in-cup ?i)
            (iced)
                )
    )

    (:action add-tea
        :parameters (?i - teas)
        :precondition (and
            (not(ingredient-in-cup ?i))
                )
        :effect (and
            (ingredient-in-cup ?i)
                )
    )
)
```

Domain File for Task 2:

```
(define (domain MakeHeatedMixedTea)

   (:requirements
      :typing
   )

   (:types
      tapioca
      syrup
      ices
      teas
   )

   (:constants
      ;ice and tea do not have various types, so they are treated as constants
      ice - ices
      tea - teas
   )

   (:predicates
      (cup-exist)
      (tapioca-exist)
      (syrup-exist)
      (iced)
      (heated ?i)
      (mixed)
      (ingredient-in-cup ?i)
   )

   (:functions)

   (:action add-tapioca
      :parameters (?i - tapioca)
      :precondition (and
         (not(tapioca-exist))
                  )
      :effect (and
         (ingredient-in-cup ?i)
         (tapioca-exist)
            )
   )
```

```
(:action add-syrup
   :parameters (?i - syrup)
   :precondition (and
      (not(syrup-exist))
            )
   :effect (and
      (ingredient-in-cup ?i)
      (syrup-exist)
         )
)

(:action add-ice
   :parameters (?i - ices)
   :precondition (and
      (not(ingredient-in-cup ?i))
            )
   :effect (and
      (ingredient-in-cup ?i)
      (iced)
         )
)

(:action add-tea
   :parameters (?i - teas)
   :precondition (and
      (not(ingredient-in-cup ?i))
            )
   :effect (and
      (ingredient-in-cup ?i)
         )
)

(:action heat-ingredients
   ;no parameter since modelling situation of the entire cup
   :parameters ()
   :precondition (and
      (cup-exist)
            )
   :effect (and
      ;effect after heating iced tea
      (when(ingredient-in-cup ice)(and(not(ingredient-in-cup ice))(not(iced))))

      ;effect after heating un-iced ingredients, such as tea, tapioca and syrup
      ;effect after heating tea
```

```
        (when(and(not(ingredient-in-cup ice))(ingredient-in-cup tea))(heated tea))
        ;effect after heating tapioca, and effect applies to all kinds of tapioca
        (forall(?i - tapioca)
           (when(and(not(ingredient-in-cup ice))(ingredient-in-cup ?i))(heated ?i))
        )
        ;effect after heating syrup, and effect applies to all kinds of syrup
        (forall(?i - syrup)
           (when(and(not(ingredient-in-cup ice))(ingredient-in-cup ?i))(heated ?i))
        )
           )
  )

  (:action mix-ingredients
     ;no parameter since modelling situation of the entire cup
     :parameters ()
     :precondition (and
        (cup-exist)
              )
     :effect (and
        ;no more unheated tapioca ball after mixing it
        (forall(?i - tapioca)
           (when(and(tapioca-exist)(not(heated ?i)))(not(tapioca-exist)))
        )

        ;the drink is now mixed by mixing tea and syrup
        (when(and(ingredient-in-cup tea)(syrup-exist))(mixed))

        ;the drink is now mixed by mixing unheated tapioca into syrup
        ;assume certain flavour of unheated tapioca turns to exactly the same flavour of syrup
by mixing
        (forall(?i - tapioca ?j - syrup)
           (when(and(ingredient-in-cup ?i)(not(heated ?i)))(and(not(ingredient-in-cup
?i))(ingredient-in-cup ?j)))
        )

        ;the following statement does not generalize to all kinds of tapioca and syrup, but
keep it to show my idea is derived from here
        ;(when(and(ingredient-in-cup mango-tapioca)(not(heated
mango-tapioca)))(and(not(ingredient-in-cup mango-tapioca))(ingredient-in-cup
mango-syrup)))
        ;(when(and(ingredient-in-cup lime-tapioca)(not(heated
lime-tapioca)))(and(not(ingredient-in-cup lime-tapioca))(ingredient-in-cup lime-syrup)))
           )
  )
```

)

Domain File for Task 3:

```
(define (domain MakeTwoCupTea)

   (:requirements
      :typing
   )

   (:types
      tapioca
      syrup
      ices
      teas
      cup
   )

   (:constants
      ;ice and tea do not have various types, so they are treated as constants
      ice - ices
      tea - teas
   )

   (:predicates
      ;add one more parameter to each applicable predicate to account for cups
      (cup-exist ?c - cup)
      (tapioca-exist)
      (syrup-exist)
      (iced ?c - cup)
      (heated ?i ?c)
      (mixed ?c - cup)
      (ingredient-in-cup ?i ?c)
      (heated-cup ?c - cup)
   )

   (:functions)

   (:action add-tapioca
      :parameters (?i - tapioca ?c - cup)
      :precondition (and
         (not(tapioca-exist))
                  )
      :effect (and
         (ingredient-in-cup ?i ?c)
```

```
        (tapioca-exist)
            )
)

(:action add-syrup
    :parameters (?i - syrup ?c - cup)
    :precondition (and
        (not(syrup-exist))
                )
    :effect (and
        (ingredient-in-cup ?i ?c)
        (syrup-exist)
            )
)

(:action add-ice
    :parameters (?i - ices ?c - cup)
    :precondition (and
        (not(ingredient-in-cup ?i ?c))
                )
    :effect (and
        (ingredient-in-cup ?i ?c)
        (iced ?c)
            )
)

(:action add-tea
    :parameters (?i - teas ?c - cup)
    :precondition (and
        (not(ingredient-in-cup ?i ?c))
                )
    :effect (and
        (ingredient-in-cup ?i ?c)
            )
)

(:action heat-ingredients
    :parameters (?c - cup)
    :precondition (and
        (cup-exist ?c)
                )
    :effect (and
        ;hot cup
        (heated-cup ?c)
```

```
        ;effect after heating iced tea
        (when(ingredient-in-cup ice ?c)(and(not(ingredient-in-cup ice ?c))(not(iced ?c))))

        ;effect after heating un-iced ingredients, such as tea, tapioca and syrup
        ;effect after heating tea
        (when(and(not(ingredient-in-cup ice ?c))(ingredient-in-cup tea ?c))(heated tea ?c))
        ;effect after heating tapioca, and effect applies to all kinds of tapioca
        (forall(?i - tapioca)
           (when(and(not(ingredient-in-cup ice ?c))(ingredient-in-cup ?i ?c))(heated ?i ?c))
        )
        ;effect after heating syrup, and effect applies to all kinds of syrup
        (forall(?i - syrup)
           (when(and(not(ingredient-in-cup ice ?c))(ingredient-in-cup ?i ?c))(heated ?i ?c))
        )
           )
  )

  (:action mix-ingredients
     :parameters (?c - cup)
     :precondition (and
        (cup-exist ?c)
              )
     :effect (and
        ;no more unheated tapioca ball after mixing it
        (forall(?i - tapioca)
           (when(and(tapioca-exist)(not(heated ?i ?c)))(not(tapioca-exist)))
        )

        ;the drink is now mixed by mixing tea and syrup
        (when(and(ingredient-in-cup tea ?c)(syrup-exist))(mixed ?c))

        ;the drink is now mixed by mixing unheated tapioca into syrup
        ;assume certain flavour of unheated tapioca turns to exactly the same flavour of syrup
by mixing
        (forall(?i - tapioca ?j - syrup)
           (when(and(ingredient-in-cup ?i ?c)(not(heated ?i ?c)))(and(not(ingredient-in-cup ?i
?c))(ingredient-in-cup ?j ?c)))
        )
           )
  )

  (:action tip-ingredients
     :parameters (?from - cup ?to - cup)
```

```
        :precondition (and
            (cup-exist ?from)
            (cup-exist ?to)
            ;situations of 2 cups need to be different, otherwise no point for tipping
            (not(= ?from ?to))
                    )
        :effect (and
            ;transfer ingredients from one cup to another, the logic is same for all 4 ingredients
            ;transfer tapioca
            (forall(?i - tapioca)
                (when(ingredient-in-cup ?i ?from)(and(ingredient-in-cup ?i
?to)(not(ingredient-in-cup ?i ?from))))
            )
            ;transfer syrup
            (forall(?i - syrup)
                (when(ingredient-in-cup ?i ?from)(and(ingredient-in-cup ?i
?to)(not(ingredient-in-cup ?i ?from))))
            )
            ;transfer ice
            (when(ingredient-in-cup ice ?from)(and(ingredient-in-cup ice
?to)(not(ingredient-in-cup ice ?from))))
            ;transfer tea
            (when(ingredient-in-cup tea ?from)(and(ingredient-in-cup tea
?to)(not(ingredient-in-cup tea ?from))))
                )
    )
)
```

PDDL Problem Files

Problem File for Task 1 a:

```
(define (problem task1a)
  (:domain MakeTea)
  (:objects
    tea - teas
    ice - ices
    mango-tapioca - tapioca
    mango-syrup - syrup
  )
  (:init
    (cup-exist)
  )
  (:goal (and
    (iced)
    (ingredient-in-cup tea)
```

```
    (ingredient-in-cup mango-tapioca)
    (ingredient-in-cup mango-syrup)
        )
  )
)
```

Problem File for Task 1 b:

```
(define (problem task1b)
  (:domain MakeTea)
  (:objects
    tea - teas
    ice - ices
    mango-syrup - syrup
    apple-syrup - syrup
  )
  (:init
    (cup-exist)
  )
  (:goal (and
    (iced)
    (ingredient-in-cup tea)
    (ingredient-in-cup apple-syrup)
    (ingredient-in-cup mango-syrup)
        )
  )
)
```

Problem File for Task 2 a:

```
(define (problem task2a)
  (:domain MakeHeatedMixedTea)
  (:objects
    tea - teas
    ice - ices
    mango-syrup - syrup
    lime-syrup - syrup
    lime-tapioca - tapioca
    mango-tapioca - tapioca
  )
  (:init
    (cup-exist)
  )
  (:goal (and
    (mixed)
    (not(heated tea))
```

```
    (heated mango-tapioca)
    (ingredient-in-cup mango-syrup)
    (not(heated mango-syrup))
    (not(ingredient-in-cup ice))
        )
  )
)
```

Problem File for Task 2 b:

```
(define (problem task2b)
  (:domain MakeHeatedMixedTea)
  (:objects
    tea - teas
    ice - ices
    mango-syrup - syrup
    lime-syrup - syrup
    lime-tapioca - tapioca
    mango-tapioca - tapioca
  )
  (:init
    (cup-exist)
  )
  (:goal (and
    (mixed)
    (ingredient-in-cup mango-syrup)
    (ingredient-in-cup lime-syrup)
    (ingredient-in-cup ice)
    (not(heated mango-syrup))
    (not(heated lime-syrup))
        )
  )
)
```

Problem File for Task 3 a:

```
(define (problem task3a)
  (:domain MakeTwoCupTea)
  (:objects
    tea - teas
    ice - ices
    mango-syrup - syrup
    lime-syrup - syrup
    lime-tapioca - tapioca
    mango-tapioca - tapioca
    cup1 - cup
```

```
    cup2 - cup
  )
  (:init
    (cup-exist cup1)
    (cup-exist cup2)
  )
  (:goal (and
    (mixed cup2)
    (not(heated tea cup2))
    (heated mango-tapioca cup2)
    (ingredient-in-cup mango-syrup cup2)
    (not(heated mango-syrup cup2))
    (not(ingredient-in-cup ice cup2))
    (not(heated-cup cup1))
        )
  )
)
```

Problem File for Task 3 b:

```
(define (problem task3b)
  (:domain MakeTwoCupTea)
  (:objects
    tea - teas
    ice - ices
    mango-syrup - syrup
    lime-syrup - syrup
    lime-tapioca - tapioca
    mango-tapioca - tapioca
    cup1 - cup
    cup2 - cup
  )
  (:init
    (cup-exist cup1)
    (cup-exist cup2)
  )
  (:goal (and
    (mixed cup2)
    (ingredient-in-cup mango-syrup cup2)
    (ingredient-in-cup lime-syrup cup2)
    (ingredient-in-cup ice cup2)
    (not(heated mango-syrup cup2))
    (heated lime-syrup cup2)
    (not(heated-cup cup1))
        )
```

)
)

Output Generated by Problem Files

Problem 1 a:

(add-tapioca mango-tapioca)
(add-syrup mango-syrup)
(add-ice ice)
(add-tea tea)

Problem 1 b:

Suspected timeout.

warning: parameter ?I of op ADD-TAPIOCA has unknown or empty type TAPIOCA.
skipping op --- OK.
Match tree built with 4 nodes.

PDDL problem description loaded:
Domain: MAKETEA
Problem: TASK1B
#Actions: 4
#Fluents: 11
Landmarks found: 4
Starting search with IW (time budget is 60 secs)...
rel_plan size: 4
#RP_fluents 6
Caption
{#goals, #UNnachieved, #Achieved} -> IW(max_w)

{4/4/0}:IW(1) -> [2]rel_plan size: 3
#RP_fluents 4
{4/3/1}:IW(1) -> [2]rel_plan size: 2
#RP_fluents 3
{4/2/2}:IW(1) -> [2];; NOT I-REACHABLE ;;
Total time: -7.59959e-10
Nodes generated during search: 11
Nodes expanded during search: 5
IW search completed
Starting search with BFS(novel,land,h_add)...
--[4294967295 / 4]--
--[2 / 4]--
--[2 / 3]--
--[2 / 2]--

Total time: -7.59959e-10
Nodes generated during search: 5
Nodes expanded during search: 4
Plan found with cost: 6.0379e-39
BFS search completed

Problem 2 a:
(add-tapioca mango-tapioca)
(heat-ingredients)
(add-syrup mango-syrup)
(add-tea tea)
(mix-ingredients)

Problem 2 b:
(add-syrup lime-syrup)
(add-ice ice)
(add-tapioca mango-tapioca)
(mix-ingredients)
(add-tea tea)
(mix-ingredients)

Problem 3 a:
(add-tapioca mango-tapioca cup2)
(heat-ingredients cup2)
(add-syrup mango-syrup cup2)
(add-tea tea cup2)
(mix-ingredients cup2)

Problem 3 b:
(add-syrup lime-syrup cup2)
(heat-ingredients cup2)
(add-ice ice cup2)
(add-tapioca mango-tapioca cup2)
(mix-ingredients cup2)
(add-tea tea cup2)
(mix-ingredients cup2)

Self Evaluation

Task 1:
Self evaluated marks are 3. I can successfully generate correct plans for the goals. In order to check the correctness of my plan, I came up with a plan to make a specified bubble tea by my human judgment, and see if the plan derived from my human brain is the same as the plan generated by the problem file. If they are the same, then the plan generated by the problem file should be correct. Furthermore, I checked all the requirements in the problem

specification, if I meet all requirements, then the plan generated by the problem file should be correct. In addition, I tested my planners with different problems to further prove the correctness of my plans. For example, change types of tapioca balls and syrup, add heated or unheated ingredients, add iced or un-iced ingredients.

I believe my model is robust. My model generalizes to all kinds of syrup and tapioca balls, the parameter of each action applies to all types of each ingredient, the robot is able to make bubble teas with different flavors, which are not only the ones specified in the assignment. My model is clear, concise, and at the 'right' level of abstraction, I only kept the necessary predicates, actions, parameters, preconditions and effects in my model. Here is my way to ensure the ‘right’ level of abstraction: I keep reducing predicates, actions, parameters, preconditions and effects in my model, as long as the model is able to generate correct plans for problems. In this way, I give my model the autonomy to plan at its own discretion.

I learned that I need to build a qualified model in my mind before I can build a qualified model in PDDL. I firstly thought about how I would make bubble tea if I were a robot. Once I come up with the plan, I transform the plan in my mind to a plan in PDDL. To make the transformation correct, I need to learn the grammar in PDDL, so the solver understands what I mean. About abstraction, I learned that a PDDL solver is smart, I do not have to teach it to the full extent of domain and problem specification, in many cases the solver is able to make correct plans with only some necessary information given by me. My first assumption is that ice and tea do not have various types, so they are treated as constants. My second assumption is that only one of each ingredient can be added into a single drink.

Task 2:

Self evaluated marks are 5. I can successfully generate correct plans for the goals. In order to check the correctness of my plan, I came up with a plan to make a specified bubble tea by my human judgment, and see if the plan derived from my human brain is the same as the plan generated by the problem file. If they are the same, then the plan generated by the problem file should be correct. Furthermore, I checked all the requirements in the problem specification, if I meet all requirements, then the plan generated by the problem file should be correct. In addition, I tested my planners with different problems to further prove the correctness of my plans. For example, change types of tapioca balls and syrup, add heated or unheated ingredients, add iced or un-iced ingredients.

I believe my model is robust. My model generalizes to all kinds of syrup and tapioca balls by applying ‘forall’ statements, which ensures the robot is able to make bubble teas with different flavors, not only the ones specified in the assignment. The parameter of each action also applies to all types of each ingredient. My model is clear, concise, and at the 'right' level of abstraction, I only kept the necessary predicates, actions, parameters, preconditions and effects in my model. Here is my way to ensure the ‘right’ level of abstraction: I keep reducing predicates, actions, parameters, preconditions and effects in my model, as long as the model is able to generate correct plans for problems. In this way, I give my model the autonomy to plan at its own discretion.

I learned that a model can never be perfect, it would encounter problems that I cannot solve. In order to solve it, assumptions of the model need to be modified. The above situation applies to the task-2-a, more details will be explained in assumptions. About abstraction, I learned that a PDDL solver is smart, I do not have to teach it to the full extent of domain and problem specification, in many cases the solver is able to make correct plans with only some necessary information given by me. About assumptions, my first assumption is that ice and tea do not have various types, so they are treated as constants. My second assumption is that only one of each ingredient can be added into a single drink. My third assumption is that tapioca balls cannot be mixed after they are heated, this assumption is to accommodate task-2-a, for task-2-a, if I heat mango tapioca balls before mixing, the balls will not exist after mixing; if I mix before I heat mango tapioca balls, tea and mango syrup will be heated after heating mango tapioca balls since they are in the same cup. In either way, the problem is unsolvable, so I introduce the third assumption here.

Task3:

Self evaluated marks are 1.5. I can successfully generate correct plans for the goals under my assumptions. In order to check the correctness of my plan, I came up with a plan to make a specified bubble tea by my human judgment, and see if the plan derived from my human brain is the same as the plan generated by the problem file. If they are the same, then the plan generated by the problem file should be correct. Furthermore, I checked all the requirements in the problem specification, if I meet all requirements, then the plan generated by the problem file should be correct. In addition, I tested my planners with different problems to further prove the correctness of my plans. For example, change types of tapioca balls and syrup, add heated or unheated ingredients, add iced or un-iced ingredients. However, I did not make use of the action ‘tip-ingredients’ in my plans, since I am able to create one bubble tea using only one cup under my assumptions. Moreover, the other cup is unheated since it is not used at all.

I believe my model is robust. My model generalizes to all kinds of syrup and tapioca balls by applying ‘forall’ statements, which ensures the robot is able to make bubble teas with different flavors, not only the ones specified in the assignment. The parameter of each action also applies to all types of each ingredient. My model is clear, concise, and at the 'right' level of abstraction, I only kept the necessary predicates, actions, parameters, preconditions and effects in my model. Here is my way to ensure the ‘right’ level of abstraction: I keep reducing predicates, actions, parameters, preconditions and effects in my model, as long as the model is able to generate correct plans for problems. In this way, I give my model the autonomy to plan at its own discretion.

About modeling, I learned that there are many ways to model a scenario, such as making bubble tea. Each solver is giving plans under certain assumptions, assumptions have an impact on how solvers generate plans. About abstraction, I learned that a PDDL solver is smart, I do not have to teach it to the full extent of domain and problem specification, in many cases the solver is able to make correct plans with only some necessary information

given by me. About assumptions, my first assumption is that ice and tea do not have various types, so they are treated as constants. My second assumption is that only one of each ingredient can be added into a single drink. My third assumption is that tapioca balls cannot be mixed after they are heated. Due to my third assumption, I did not make use of action ‘tip-ingredients’ in my plans for task-3-a.