Derin Gezgin | Russell Kosovsky | Jay Nash
Fall 2024 | COM316: Artificial Intelligence | Problem 9:
Planning & Current-Best Learning & Representation

A. Planning

Use the STRIPS planning approach to make a plan to solve the blocks problem given below. The set of facts that describes an initial condition and the set of facts that describes a goal condition are provided. In your solution you should describe how you handled all links including when they were generated. In addition, discuss how things might change if the ordering of the preconditions of the rules changed. Your answers show be both specific to the problem given and general enough to handle any initial/goal situation. For extra credit, write a pseudocode algorithm that shows how the planner works.

Our Solution for Planning

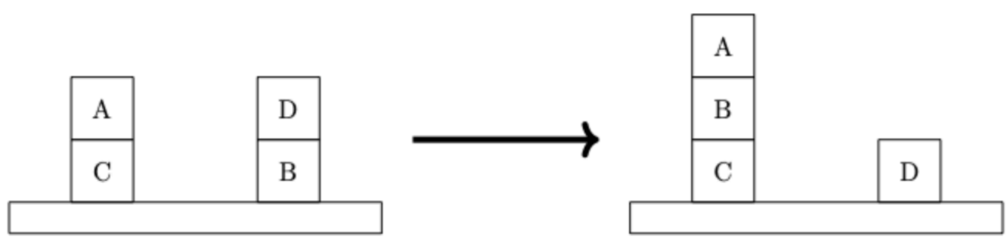
In this problem, we are given the following facts:

on(A C) | clear(A) | on(D B) | clear(D) | on(C Table) | on(B Table)

and the following goals:

clear(A) | on(A B) | on(B C) | on(D Table) | on(C Table)

We can visualize our start and the goal states like the following:



To create a plan using the *STRIPS* planning approach, we are given three rules:

	Rule 1	Rule 2	Rule 3
if	on(x y) clear(x) clear(z)	on(x y) clear(x)	on(x Table) clear(x) clear(z)
add	on(x z) clear(y)	on(x Table) clear(y)	on(x z)
delete	on(x y) clear(z)	on(x y)	on(x Table) clear(z)

To solve this problem, we can start by backtracking from the final state and aiming for the starting position. We can backtrack from our goals and return to our initial solution. We can check for our current facts and goals for each goal to choose the right rule to fire.

Goal 1: on(C Table)

Our initial state already satisfies this goal. We should not do anything extra to satisfy this goal but keep in mind that we should not affect it.

Goal 2: on(D Table)

Our current rule system does not satisfy this goal. We can fire ***Rule II*** to satisfy this goal.

- To fire ***Rule II***, the facts we need are on(D B) and clear(D)
- As a result of ***Rule II***, we add on(D Table) and clear(B) to our fact list.
- On the other hand, we should delete on(D B).

Goal 3: on(B C)

Our current rule system does not satisfy this goal. We could fire ***Rule I*** or ***Rule III*** to satisfy this goal, but we do not have the conditions for either, which require B to be on another block or C to be clear. For simplicity's sake, we choose to fire ***Rule III***, which has more fulfilled prerequisites. However, the other path is also achievable with the same backtracking logic.

- To fire ***Rule III***, the facts we need are: on(B Table), clear(B), and clear(C)
- As a result of ***Rule III***, we add on(B C)
- On the other hand, we delete on(B Table), clear(C)

The issue with this rule is that we do not have clear(C) in our current rule base. We can add this as a sub-goal.

Sub-Goal: clear(C)

Our current rule system does not satisfy this goal. We can fire ***Rule I*** or ***Rule II*** to satisfy this goal. Both will have similar outcomes—while one moves A onto the table, the other would move A onto D, which would affect the next rule we fire depending on the situation.

- To fire ***Rule II***, the facts we need are on(A C) and clear(A)
- As a result of ***Rule II***, we add on(A Table) and clear(C)
- On the other hand, we delete on(A C)

This rule satisfied all the conditions, so we can continue our path by firing ***Rule III*** to satisfy the third goal. ***Rule II*** should fire before ***Rule III*** as it prepares the necessary conditions for ***Rule III***.

Goal 4: on(A B)

Our current rule system does not satisfy this goal. We can fire ***Rule I*** or ***Rule III*** to move A on top of B. However, since A was already on the table from the previous rules, firing ***Rule III*** would shorten this process as it would satisfy more preconditions. If we chose to fire ***Rule I*** in the previous goal, we would also choose ***Rule I*** in this goal, considering A would be on D.

- To fire ***Rule III***, the facts we need are: on(A Table), clear(B), and clear(A)
- As a result of ***Rule III***, we add on(A B)
- On the other hand, we delete on(A Table), clear(B)

Goal 5: clear(A)

Our initial state already satisfied this goal. We should not do anything extra to satisfy it, but we should also remember not to affect it.

This is a walk-through of how we can start from goals to achieve the starting setup of the blocks.

In this case, we fire ***Rule II*** to move D onto the table, ***Rule II*** to clear C, ***Rule III*** to move B onto C, and ***Rule III*** to move A onto B. While this is a single solution for this problem, there are multiple other solutions, such as:

- ***Rule II*** to move D onto the table
- ***Rule I*** to move A onto D

- ***Rule III*** to move B onto C
- ***Rule III*** to move A onto B

With this approach, it is possible to create a detailed rule map, but with our current simulation, this should be the order of firing of the rules:

Step 1: Start
Step 2: ***Rule II*** (D on table)
Step 3: ***Rule II*** (A on table)
Step 4: ***Rule III*** (B on C)
Step 5: ***Rule III*** (A on B)
Step 6: Goal Achieved

The order can change slightly as we can execute Step 3 before Step 2. On the other hand, we have to execute Step 2 before Steps 4 and 5 to clear B.

In this system, the order of the preconditions might matter. In a case with two rules with three preconditions, the rule with the fulfilled precondition in the first order can be traced rather than the rule that has the true precondition at the end, considering one would already be traceable compared to the other, which reveals it is also on the right track in the last precondition.

,

B. Current-Best Learning

Describe how the robot can use learning through induction and the semantic representation of an object to learn important concepts that will help in its search. The primary concept that is needed for search as we do it now, is passability. We have several possible objects that can be obstacles (chairs, desks, book shelves, buildings, monsters, kittens, etc.). We need the robot to learn the concept of passability. Initially think of this as you showing the robot objects and telling it whether they are passable or not. What objects would you use and in what order? What will change about the robot's concept of passable with each example? Show an example of a learning session. Draw each new object shown to the robot along with its semantic net representation and the new semantic net representing the concept of a passable object. Once trained, would it be able to sense the attributes of some new obstacle and determine if it is passable? How useful is this for the robot's ability to search in our more complicated environment?

Our Solution for Current-Best Learning

In a training session, we can introduce our robot to very simple aspects of the grid. We would start by showing stable and unstable squares. After this, we introduce it to low and high obstacles, making it reconsider its passable and unpassable classification. This can be a sample learning session.

Training Session

To teach the robot the concept of passability, we would present it with a series of objects in a specific order that gradually introduces complexity:

- ➔ **Open Space:** An empty area with no obstacles (Passable)
- ➔ **Wall:** A solid barrier (Impassable)
- ➔ **Curtain:** A hanging fabric (Passable)
- ➔ **Glass Door (Closed):** Transparent but solid (Impassable)
- ➔ **Archway:** An open architectural structure (Passable)

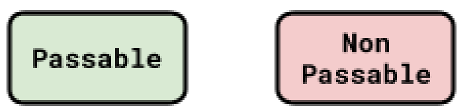
Changes to the Robot's Concept of Passability with Each Example

- **After Open Space:** The robot's initial concept is that areas with no objects are passable.
- **After Wall:** Updates concept to recognize that solid, opaque barriers are not passable.
- **After Curtain:** Learns that not all opaque objects are not passable; factors like solidity and opacity matter.
- **After Glass Door:** Learns that transparency does not guarantee passability; solidity is also important.
- **After Archway:** Learns openings within obstacles can be passable despite surrounding obstacle parts.

Step by Step Learning Session

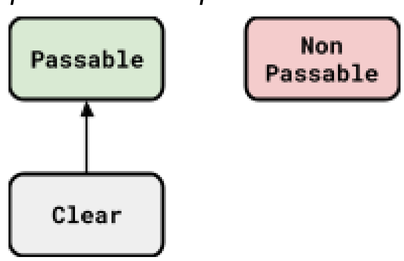
Initial Step:c

We do not have any information about what is passable and what is not.



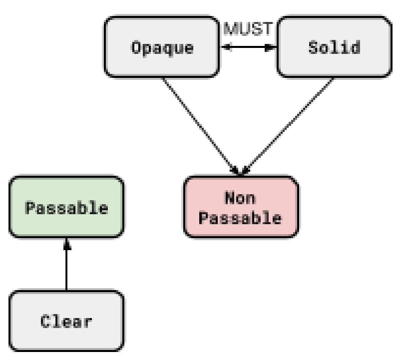
***Step 1:** Introducing the open space to robot*

We introduce that clear spaces mean passable

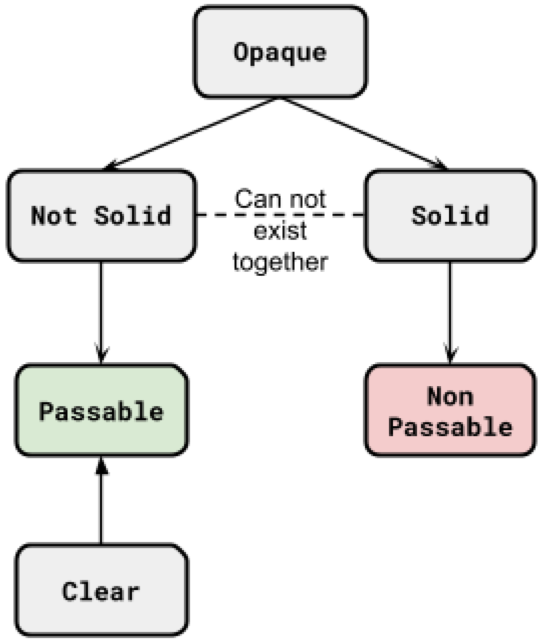


***Step 2:** Introducing the wall to robot*

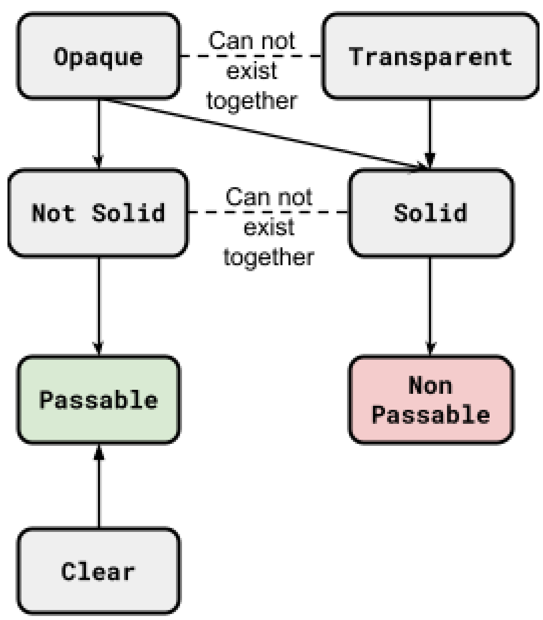
If something is solid and opaque, it is not passable.



Step 3: *Introducing the curtain to robot*
If something is opaque but not solid, it is passable.

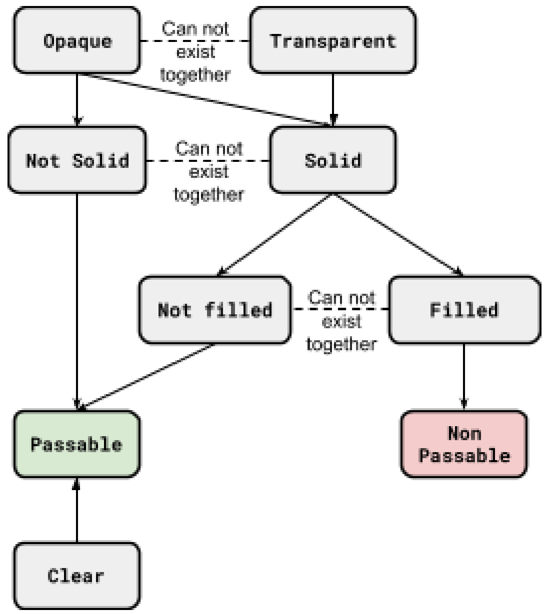


Step 4: *Introducing the glass door to robot*
If something is transparent but also solid, it is not passable.



***Step 5:** Introducing the archway to robot*

If something is opaque, and solid but not filled, it is still passable.



While we can extend this training session to present more alternative obstacle types to the robot, with the current representation of the rule base, it can not unlock the ability to sense the new obstacle types around it.

For example, if we ask the robot to decide on a transparent and non-solid obstacle -which did not appear in the training session- it might not be able to conclude this obstacle as we did not establish a connection for this specific combination. On the other hand, it can make guesses better than random, considering the elements of an unknown obstacle type. For example, in the case of our obstacle, the robot can **guess** the obstacle being non-solid, making it **passable**, but cannot make a 100% sure final verdict about it, considering there can be an edge case that makes it **unpassable** when the object is transparent. At the same time, the structure of our network and how detailed it is can be crucial in this case, as we should have a directly transparent link (rather than following the rule network) that will conclude passable. This can also create discrepancies in the network structure.

If the robot can overcome any obstacle and is penalized if it encounters an impassable block, it could still use inductive learning. It could be its instructor who could update the semantic network if penalized. The only issue is that the robot might be unable to create the most optimal network in the current search space and might have a more complicated network with the same functionality.

Example of a Learning Session with More Detailed Attributes

First Object: Open Space

Attributes:
Type: Space
Solidity: None
Visibility: Clear
Material: Air
Passability: Yes

Robot's Concept Update:

Objects of type space are passable
Objects with no solidity are passable
Objects with clear visibility are passable
Objects made of air are passable

Second Object: Wall

Attributes:
Type: Barrier
Solidity: Solid
Visibility: Opaque
Material: Brick
Passability: No

Robot's Concept Update:

Objects of type Barrier are impassable
Solid objects are impassable
Opaque objects are impassable
Brick objects are impassable

Third Object: Curtain

Attributes:
Type: Barrier
Solidity: Flexible
Visibility: Opaque
Material: Cloth
Passability: Yes

Robot's Concept Update:

Objects of type Barrier may or may not be passable
Flexible objects are passable
Opaque objects may or may not be passable
Cloth objects are passable

Ability to Determine Passability of New Obstacles

Once trained with diverse examples, the robot can analyze the attributes of new obstacles and predict passability. For instance, if it encounters a "Glass Wall":

Attributes:
Type: Barrier
Solidity: Solid
Visibility: Transparent
Material: Glass

The robot can determine that the “Glass Wall” is impassable as solid objects are impassable. All other facts say the object might be passable or has not yet been learned, so it’s more likely that “Glass Wall” is impassable.

C. Representation

Although we have some more organization with semantic networks, we still have a scattering of links. Objects and classes of objects can be linked by relationship links -- we can designate an object as Chair5 with an isa link to wheeled-chairs and a color link to red. But what if there was no color link or even a default color link in the hierarchy? How would we know there was even the possibility of color for a chair? Another issue is that it's a problem to represent information that is not discrete. More structure in our organization would help us use the isa links for inheritance and possibly make it easier for data to be continuous. Suggest some possibilities for this more structured method of representation.

Our Solution for Representation

To solve the problem of representation, we can use an inheritance structure with fields to ensure that we are aware of an object's known and unknown attributes. We can start by using a boilerplate recipe to generate the object and its fields when initializing a new object. Each recipe can have default values, which indicate that those fields are the attributes of an object but do not yet have a value. For example, we can start by defining a recipe for a regular object:

Recipe:	`Object`
Slots:	
`height`	default value `0cm`
`width`	default value `0cm`
`weight`	default value `0kg`
`material`	default value `soil`

After this object recipe, we can add more sub-types of objects to assert default values and have more object-specific fields.

Recipe:	`Chair`
isa:	`Object`
Slots:	
`Price`	default value `$0`
`Type`	default value `Normal`

Recipe:	`Wheeled-Chair`
isa:	`Chair`
Slots:	
`Wheel count`	default value `0`

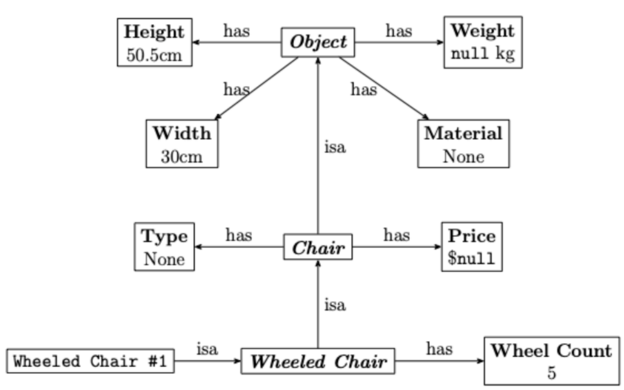
Right now, this representation of objects can be helpful for information about a specific kind of object and knowledge about different fields, regardless of whether we have a value. The only issue is that sometimes default values can also be possible in our rule base, so our program might get confused. As a solution, the default value can be a value that cannot happen in the real world. After this modification, our recipes will look like this:

Recipe:	`Object`
Slots:	
`height`	default value `null` cm
`width`	default value `null` cm
`weight`	default value `null` kg
`material`	default value `None`

Recipe:	`Chair`
isa:	`Object`
Slots:	
`Price`	default value `$null`
`Type`	default value `None`

Recipe:	`Wheeled-Chair`
isa:	`Chair`
Slots:	
`Wheel count`	default value `null`

This new set of recipes shows that a wheeled chair will have a height, width, weight, material, price, type, and wheel count. In our initial setup, we do not have to have values for these fields, as we know they exist with default placeholders. The placeholders are impossible values, so they will never get confused with a standard value. A semantic network would look like this when we have a wheeled chair with a known height, width, and wheel count:



This semantic network representation is created using the recipes and our knowledge base. We can see that despite some of the values being missing, they are set to the default values not to lose information about the existence of a field. Lastly, as our fields can have any value, this representation allows us to have continuous values in different fields.