What is partial order planning? Explain with shoe example.

Any planner that maintains a partial solution as a totally ordered list of steps found so far is called a **total-order planner**, or a **linear planner**. Alternatively, if we only represent partial-order constraints on steps, then we have a **partial-order planner**, which is also called a **non-linear planner**. In this case, we specify a set of temporal constraints between pairs of steps of the form S1 < S2 meaning that step S1 comes before, but not necessarily immediately before, step S2. We also show this temporal constraint in graph form as

S1 +++++++++> S2

STRIPS is a total-order planner, as are situation-space progression and regression planners

Partial-order planners exhibit the property of least commitment because constraints ordering steps will only be inserted when necessary. On the other hand, situation-space progression planners make commitments about the order of steps as they try to find a solution and therefore may make mistakes from poor guesses about the right order of steps.

Representing a Partial-Order Plan A partial-order plan will be represented as a graph that describes the temporal constraints between plan steps selected so far. That is, each node will represent a single step in the plan (i.e., an instance of one of the operators), and an arc will designate a temporal constraint between the two steps connected by the arc. For example,

• S1 ++++++++> S2 ++++++++++> S5

• |\ ^

• | \++++++++++++++++| |

• | v |

++++++> S3 ++++++> S4 ++++++

graphically represents the temporal constraints S1 < S2, S1 < S3, S1 < S4, S2 < S5, S3 < S4, and S4 < S5. This partial-order plan implicitly represents the following three total-order plans, each of which is consistent with all of the given constraints:

[S1,S2,S3,S4,S5], [S1,S3,S2,S4,S5], and [S1,S3,S4,S2,S5].

In the following example, we will show how the planning algorithm derives a solution to a problem that involves putting on a pair of shoes. In this problem scenario, Pat is walking around his house in his bare feet. He wants to put some shoes on to go outside.

*Note: There are no threats in this example and therefore is no mention of checking for threats though it a necessary step*

To correctly represent this problem, we must break down the problem into simpler, more atomic states that the planner can recognize and work with. We first define the Start operator and the Finish operator to create the minimal partial order plan. As mentioned before, we must simplify and break down the situation into smaller, more appropriate states. The Start operator is represented by the effects: ~LeftSockOn, ~LeftShoeOn, ~RightSockOn, and ~RightShoeOn. The Finish operator has the preconditions that we wish to meet: LeftShoeOn and RightShoeOn. Before we derive a plan that will allow Pat to reach his goal (i.e. satisfying the condition of having both his left and right shoe on) from the initial state of having nothing on, we need to define some operators to get us

Version 2 CSE IIT,Kharagpur

there. Here are the operators that we will use and the possible state (already mentioned above).

**Operators**

Op(ACTION: PutLeftSockOn() PRECOND:~LeftSockOn EFFECT: LeftSockOn)

Op(ACTION: PutRightSockOn() PRECOND:~RightSockOn EFFECT: RightSockOn)

Op(ACTION: PutLeftShoeOn() PRECOND:LeftSockOn EFFECT: LeftShoeOn)

Op(ACTION: PutRightShoeOn() PRECOND:RightShoeOn EFFECT: RightShoeOn)

**States**

LeftSockOn, LeftShoeOn, RightSockOn, RightShoeOn

**Creating A Plan**

From the states listed above, we first create a minimal partial order plan. We can represent bare feet (Start operator) by saying that Pat is not wearing any socks or shoes and shoes on (Finish operator) with the two shoe on states. Here is the minimal partial order plan.

Initially we have two preconditions to achieve; RightShoeOn and LeftShoeOn. Let's start with the condition of having our right shoe on. We must choose an operator that will result in this condition. To meet this condition we need to the operator 'PutRightShoeOn()'. We add the operator and create a causal link between it and the Finish operator. However, adding this operator results a new condition (i.e. precondition of PutRightShoeOn()) of having the right sock on. Start ~LeftSockOn, ~LeftShoeOn, ~RightSockOn, ~RightShoeOn PutRShoeOn( ) RightSockOn, LeftSockOn Finish RightShoeOn LeftShoeOn, RightShoeOn

Version 2 CSE IIT,Kharagpur

At this point we still have two conditions to meet: having our left shoe on and having our right sock on. We continue by selecting one of these two preconditions and trying to achieve it. Let's pick the precondition of having our right sock on. To satisfy this condition, we must add another step, operator 'PutRightSockOn()'. The effects of this operator will satisfy the precondition of having our right sock on. At this point, we have achieved the ‘RightSockOn’ state. Since the precondition of the ‘PutRightSockOn()’ operator is one of the effects of the Start operator, we can simply draw a causal link between the two operators. These two steps can be repeated for Pat’s left shoe. The plan is complete when all preconditions are resolved.

The Partial Order Planning algorithm can be described as a form of regression planning that use the principle of least commitment. It starts with a minimal partial order plan that consists of a Start operator (initial state) and a Finish operator (goal state). It then chooses a precondition that has not been resolved and chooses an operator whose effect matches the precondition. It then checks if any threats were created by the addition of the operator and if one is detected, resolves it either by demoting the operator, promoting the operator, or backtracking (removing the operator). It continues to choose operators until a solution is found (i.e. all preconditions are resolved).

Solutions created by the Partial Order Planning algorithm are very flexible. They may be executed in many ways. They can represent many different total order plans (partial order plans can be converted to total order plans using a process called linearization). Lastly they can more efficiently if steps are executed simultaneously.