Media Engineering and Technology Faculty
German University in Cairo

A Virtual Environment for Partial-Order Planning

Bachelor Thesis

Author:	Mohamed Ayman Tammaa
Supervisors:	Assoc. Prof. Haythem Ismail
Submission Date:	19 May, 2024

Media Engineering and Technology Faculty
German University in Cairo

A Virtual Environment for Partial-Order Planning

Bachelor Thesis

Author:	Mohamed Ayman Tammaa
Supervisors:	Assoc. Prof. Haythem Ismail
Submission Date:	19 May, 2024

This is to certify that:

(i) the thesis comprises only my original work toward the Bachelor Degree

(ii) due acknowlegement has been made in the text to all other material used

Mohamed Ayman Tammaa
19 May, 2024

Acknowledgments

Text

Abstract

Abstact

Contents

Acknowledgments **V**

1 Introduction **1**
1.1 Section Name 1
1.2 Another Section 1

2 Background **3**
2.1 Introduction to Planning 3
2.2 Partial Order Planning 3
2.2.1 Formal Definition of Partial Plans 4
2.2.2 Consistency of Partial Plans 4
2.2.3 Threats in Partial Plans 4
2.2.4 POP Algorithm 5

3 Design and Implementation **7**
3.1 Introduction 7
3.2 Design of the Algorithm & Class Diagram 7
3.2.1 Nondeterministic Achievers & Threat Search 7

4 Conclusion **9**

5 Future Work **11**

Appendix **12**

A Lists **13**
List of Abbreviations 13
List of Figures 14

References **15**

Chapter 1

Introduction

1.1 Section Name

Some sample text with an Acronym Without Citation (AC), some citation [1], and some more Acronym With Citation [2] (AC2).

1.2 Another Section

Reference to Section 1.1, and reuse of AC nad AC2 with also full use of Acronym With Citation [2] (AC2).

Chapter 2

Background

2.1 Introduction to Planning

Planning in Artificial Intelligence (AI) is a fundamental aspect in the field that allows agents to formulate sequences of actions and strategies to achieve a specific goal. It used in a wide range of fields where agents need to make decisions and take actions based on the current state of the environment.

Classical planning is a type of planning that is used in AI to solve problems that can be represented as a set of states and actions. It deals with straightforward actions & with predictable and deterministic environments, where the agent can predict the outcome of its actions. The challenge in classical planning is to construct a sequence of actions that will transform the initial state of the environment into a desired goal state, while dealing with exponential growth in the search space, and dealing with the actions and steps in chronological order.

Among the different types of planning, we have **state space planning**, which is a type of planning that is used in AI to solve problems searching through a set of states and actions, and **plan space planning**, which is a type of planning that is used in AI to solve problems searching through a set of plans and actions. In this thesis, we will focus on plan space planning, and more specifically on Partial Order Planning (POP).

2.2 Partial Order Planning

Partial Order Planning (POP) is a plan-space search algorithm. Unlike other planning algorithms, POP is partially ordered plan search. This gives it the advantage over total order planning algorithms that it can use problem decomposition, work on several subproblems in parallel independently, solve them with several subplans, and then merge the subplans into a final plan.

POP uses least commitment strategy. It is a type of planning that does not require the planner to commit to a specific order of actions. Instead, the planner can choose to leave some actions unordered, and the planner can choose to order actions only when necessary. This allows the planner to explore a larger space of possible plans, and it allows the planner to find plans that are more flexible and more robust. Partial Order Planning is a powerful and flexible planning algorithm that has been used in a wide range of applications, including robotics, natural language processing, and automated planning.

2.2.1 Formal Definition of Partial Plans

A partial order plan is a tuple $\pi = (A, \prec, B, L)$ where:

- A is a set of actions, or partially instantiated Operators.
- $\prec$ is a set of ordering constraints between actions in the form of $a_i \prec a_j$
- B is a set of bindings.
- L is a set of causal links.

are the components of a partial plan.

2.2.2 Consistency of Partial Plans

A partial order plan $\pi = (A, \prec, B, L)$ is consistent if it satisfies the following conditions:

- The transitive closure of the ordering constraints $\prec$ is a strict partial order.
- every subsitution σ which binds a variable x to a value in its allowed domain D_x is consistent with the bindings in all other constraints in B.

2.2.3 Threats in Partial Plans

Last thing to mention before we show the algorithm, is the concept of threats in partial plans. A threat in a partial plan is an action (partially instantiated operator) that could potentially undo the effects of another action. We can say an action a_k threatens a causal link $(a_i \xrightarrow{P_j} a_j)$ if it has the following properties:

- $e_k(\in$ effect of $a_k)$ unifies with $\neg p_j$.
- the MGU of e_k and $\neg p_j$ is consistent with the bindings in B.
- $\prec \cup \{a_i \prec a_k, a_k \prec a_j\}$ is consistent.

Once the 3 conditions are met, we can say that a_k threatens the causal link
$(a_i \xrightarrow{P_j} a_j)$.

2.2.4 POP Algorithm

The Partial Order Planning (POP) algorithm is a plan-space search algorithm that uses a partial order plan representation. The algorithm works by incrementally building a partial order plan, and then refining the plan by adding ordering constraints, bindings, and causal links to resolve threats and make the plan consistent.

Before we show the algorithm, we need to look at some terminologies and objects used in the algorithm:

- **Operator**: is a tuple $o = (name, preconds, effects)$ where:
  - $name$ is the name of the operator.
  - $preconds$ is a set of preconditions.
  - $effects$ is a set of effects.
- **Action**: as mentioned before, an action is a partially instantiated operator, (i.e. any ground instance of an operator with some of its variables instantiated).
- **Causal Link**: is in the form of $(a_i \xrightarrow{P_j} a_j)$ where:
  - a_i is an action.
  - a_j is another action linked to a_i.
  - P_j is a precondition of a_j, and at the same time an effect of a_i.

  An action a_i is said to achieve a precondition p_j if $p_j \in$ effects(a_i) and $p_j \in$ preconds(a_j).
- **Binding**: is a set of constraints that bind variables to values. For example, if we have a binding $B = \{x \leftarrow \{1, 5\}, y \leftarrow 2\}$, this means that the variable x is bound to the values 1 or 5, and the variable y is bound to the value 2.

Variables used in the algorithm:

- O is the set of operators.
- s_0 is the initial state.
- g is the goal state.
- $\pi = (A, L, \prec, B)$ is a partial order plan.
- $agenda$ is a set of pairs (a_i, p_i) where a_i is an action and p_i is a precondition of a_i.
- $achievers$ is the set of operators that can achieve a precondition p_i.
- $failure$ is a special value that indicates that the algorithm has failed to find a plan.

Now we can show the POP algorithm in Algorithm 1:

Algorithm 1 POP Algorithm

```
  Function POP($O$, $s_0$, $g$)
Ensure: a plan
  return POP1($\{a_0, a_\infty\}, \{a_0 \prec a_\infty\}, \emptyset, \emptyset, \{a_\infty\} \times$ Preconds($a_\infty$))
```

```
  Function POP1($\pi = (A, L, \prec, B)$, agenda)
Ensure: a plan
  if agenda == $\emptyset$ then
    return $\pi$
  end if
  Select any pair $(a_i, p_i)$ and remove it from agenda
  achievers $\leftarrow$ the set of operators achieving $(a_i, p_i)$
  if achievers == $\emptyset$ then
    return $failure$
  end if
  Nondeterministically choose some operator $a_j \in$ achievers
  $L \leftarrow L \cup \{\langle a_j \xrightarrow{P_i} a_i \rangle\}$
  Update $\prec$ with $a_j \xrightarrow{P_i} a_i$
  Update B with binding constraints of this link
  if $a_j \notin A$ then
    $A \leftarrow A \cup \{a_j\}$
    Update $\prec$ with $a_0 \prec a_j$ and $a_j \prec a_\infty$
    agenda $\leftarrow$ agenda $\cup\{(a_j, p_j) | p_j \in$ preconds$(a_j)\}$
  end if
  $\pi \leftarrow$ RESOLVE-THREATS($\pi$, $a_j$, $\langle a_j \prec a_i \rangle$)
  return POP1($\pi$, agenda)
```

```
  Function RESOLVE-THREATS($\pi = (A, L, \prec, B)$, $a_l$, $L$)
Ensure: a plan
  for each threat $a_k$ on $(a_i \xrightarrow{P_j} a_j)$, where $a_k = a_l$ or $(a_i \xrightarrow{P_j} a_j) = l$ do
    Nondeterministically choose one of the following:
    - Update $\prec$ with $a_k \prec a_i$
    - Update with $a_j \prec a_k$
    - Add a binding constraint to $B$ which renders $p_i$ nonunifiable with any threatening
    effect of $a_k$.
  end for
  return $\pi$
```

Chapter 3

Design and Implementation

3.1 Introduction

In this chapter, we will discuss the design and implementation of the POP algorithm. We will start by discussing the design of the algorithm, and then we will discuss the implementation of the algorithm in C#.

3.2 Design of the Algorithm & Class Diagram

3.2.1 Nondeterministic Achievers & Threat Search

Partial Order Planning (POP) needs to be able to handle some form of nondeterminism in the planning domain. This is because the planner needs to be able to handle situations where there are multiple ways to achieve a precondition of an action. This cannot be achieved in practice without trying to search all possible ways of the choices that can be made. So, to model nondeterminism in the planning domain, we need to introduce the concept of **A* Search**. A* Search is a search algorithm that is

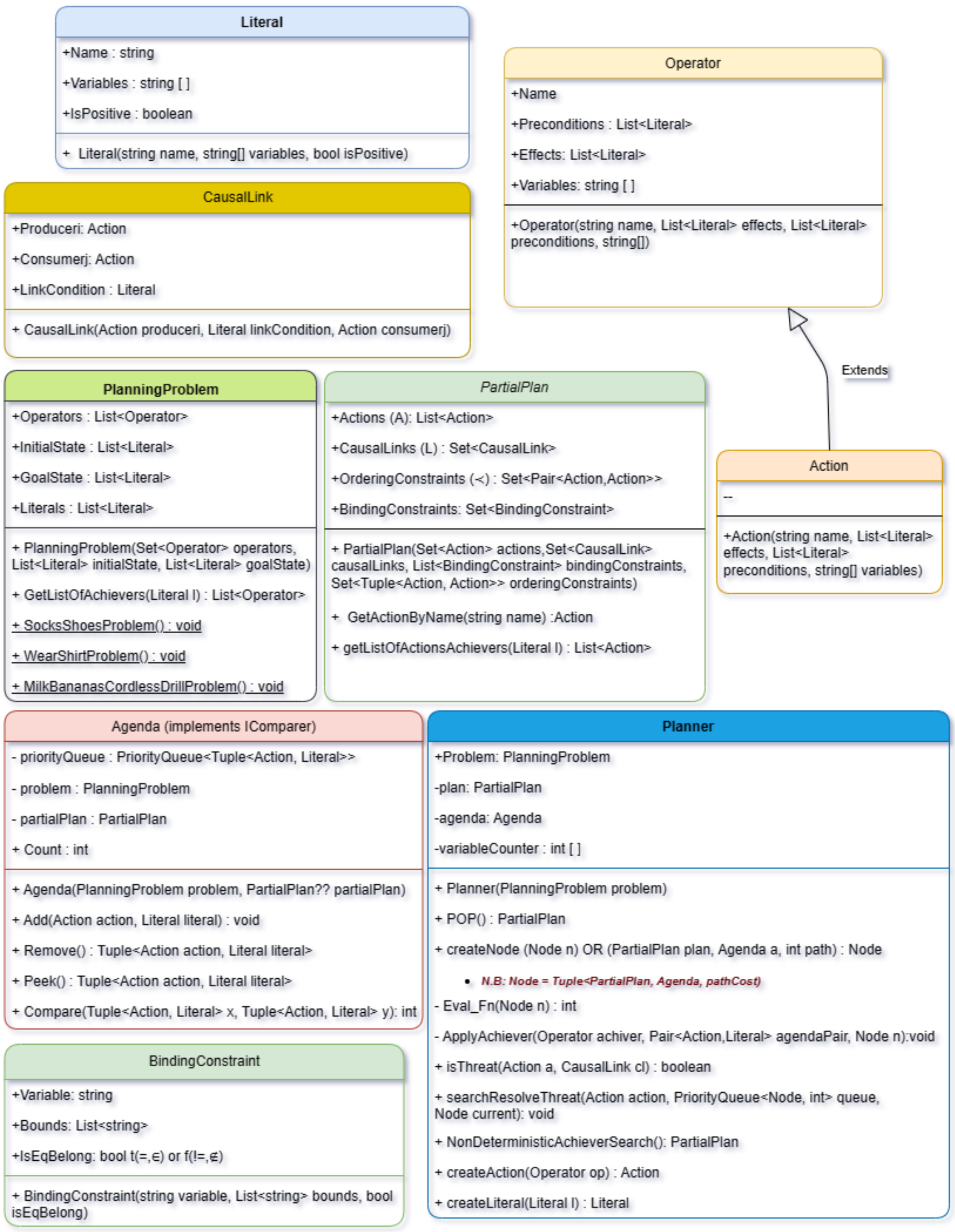
Literal
+Name : string
+Variables : string [ ]
+IsPositive : boolean
+ Literal(string name, string[] variables, bool isPositive)
CausalLink
+Produceri: Action
+Consumerj: Action
+LinkCondition : Literal
+ CausalLink(Action produceri, Literal linkCondition, Action consumerj)
Operator
+Name
+Preconditions : List<Literal>
+Effects: List<Literal>
+Variables: string [ ]
+Operator(string name, List<Literal> effects, List<Literal> preconditions, string[])
Extends
PlanningProblem
+Operators : List<Operator>
+InitialState : List<Literal>
+GoalState : List<Literal>
+Literals : List<Literal>
+ PlanningProblem(Set<Operator> operators, List<Literal> initialState, List<Literal> goalState)
+ GetListOfAchievers(Literal l) : List<Operator>
+ SocksShoesProblem() : void
+ WearShirtProblem() : void
+ MilkBananasCordlessDrillProblem() : void
PartialPlan
+Actions (A): List<Action>
+CausalLinks (L) : Set<CausalLink>
+OrderingConstraints (≺) : Set<Pair<Action,Action>>
+BindingConstraints: Set<BindingConstraint>
+ PartialPlan(Set<Action> actions,Set<CausalLink> causalLinks, List<BindingConstraint> bindingConstraints, Set<Tuple<Action, Action>> orderingConstraints)
+ GetActionByName(string name) :Action
+ getListOfActionsAchievers(Literal l) : List<Action>
Action
--
+Action(string name, List<Literal> effects, List<Literal> preconditions, string[] variables)
Agenda (implements IComparer)
- priorityQueue : PriorityQueue<Tuple<Action, Literal>>
- problem : PlanningProblem
- partialPlan : PartialPlan
+ Count : int
+ Agenda(PlanningProblem problem, PartialPlan?? partialPlan)
+ Add(Action action, Literal literal) : void
+ Remove() : Tuple<Action action, Literal literal>
+ Peek() : Tuple<Action action, Literal literal>
+ Compare(Tuple<Action, Literal> x, Tuple<Action, Literal> y): int
Planner
+Problem: PlanningProblem
-plan: PartialPlan
-agenda: Agenda
-variableCounter : int [ ]
+ Planner(PlanningProblem problem)
+ POP() : PartialPlan
+ createNode (Node n) OR (PartialPlan plan, Agenda a, int path) : Node
• N.B: Node = Tuple<PartialPlan, Agenda, pathCost)
- Eval_Fn(Node n) : int
- ApplyAchiever(Operator achiver, Pair<Action,Literal> agendaPair, Node n):void
+ isThreat(Action a, CausalLink cl) : boolean
+ searchResolveThreat(Action action, PriorityQueue<Node, int> queue, Node current): void
+ NonDeterministicAchieverSearch(): PartialPlan
+ createAction(Operator op) : Action
+ createLiteral(Literal l) : Literal
BindingConstraint
+Variable: string
+Bounds: List<string>
+IsEqBelong: bool t(=,∈) or f(!=,∉)
+ BindingConstraint(string variable, List<string> bounds, bool isEqBelong)

Figure 3.1: Class Diagram of the POP Algorithm

Chapter 4

Conclusion

Conclusion

Chapter 5

Future Work

Text

Appendix

Appendix A

Lists

POP	Partial Order Planning
AI	Artificial Intelligence
AC	Acronym Without Citation
AC2	Acronym With Citation [2]

List of Figures

3.1 Class Diagram of the POP Algorithm 8

Bibliography

[1] W.G. Campbell. *Form and style in thesis writing.* Houghton Mifflin, 1954.

[2] S. Wenkang. An analysis of the current state of English majors' BA thesis writing [J]. *Foreign Language World*, 3, 2004.