Investigation into the theoretical properties of, and the relationship between, AI planning domain models.

By : Salihin F. Shoeeb
Supervised by Professor Lee McCluskey
A Report Submitted as a Requirement for the Transfer to Second Year

Department of Informatics
UNIVERSITY OF HUDDERSFIELD
School of Computing and Engineering

TABLE OF CONTENTS

TABLE OF FIGURES

1. ABSTRACT

Representation of knowledge (KR) is an essential problem in artificial intelligence systems. The primary requirement of KR is the development of a sufficiently precise notation for representing knowledge. The importance of KR comes from the current design model of intelligent systems need for expert knowledge. Mainly, there are two methods to represent knowledge in artificial Intelligence systems, which are encoded within an application and encoding within a domain model. The former method describes the problem as a part of AI application and on the other hand the domain model method insulates the AI application from its problem description. Recently, artificial intelligence systems are developed separately from their problem domains in order to facilitate the process of updating, maintaining and repairing the domain models as well as the possibility to reuse them for other AI purposes. Consequently, number of planning domain languages have been developed (e.g STRIPS, ADL, PDDL, OCL) to describe a real problem domain formally. This project aims to develop a set of planning domain metrics and comparison techniques based on the domain model languages' criteria to analyse a domain model's structure, characteristics and properties, in order to investigate and measure domain complexity and equivalence.

2. INTRODUCTION

2.1 Theoretical and terminology

Artificial intelligent (AI): Definitions of artificial intelligence according to (Russell [22]) are divided into two main categories concerned with thought processes and reasoning (i.e think like humans, act rationally). Generally, a planning domain problem depends on three terms which are:

- **Automated planning (AP)** is the processes of making system act or thank autonomously.
- **Knowledge representation (KR)** is a way used to form knowledge in an intelligence system. Knowledge representation and the methods of KR, notation and criteria have not yet met the complexity and requirements of most of realistic problem domain.
- **Machine Learning (ML)** A computer techniques which are concerned with the construction of intelligent algorithms that allow computer systems to learn for instance by example or training. In other word, Learning is any process that increases the performance of an AI system.

2.2 Objective

The objective of this research can be briefly summarized as following (based on the experimental section 6.3), developing a set of measurement tools and comparative techniques to analyse a domain model's specifications and features in order to investigate their complexity, effectives, and equivalence.

2.3 Outline of the report

Below we outline the subsequent chapters of this report

Chapter 3 contains a brief overview of the AP application and historical development of the major techniques for AI planning.

Chapter 4 highlights the problem of knowledge representation in terms of planning domains, providing the theoretical foundation of a number of planning description language (i.e. PDDL).

Chapter 5 presents an overview of the development of planning knowledge representation and formalization, providing several an ML tools which are used to develop and construct a planning domain model.

Chapter 6 introduces an experimental that shows the structurally differences between an automated inductive planning domain model and handcraft one.

Chapter 7 finally, this chapter draws a framework for the next year work plan.

3. AI PLANNING AND APPLICATION

It seems that planning effortlessly, but writing planning engines (planner) is a difficult challenge. Planning, the process of generating a sequence of ordered actions based on a real problem domain description (i.e. initial state) to achieve desired goals.

3.1 Planning history

An automated planning (AP) discipline has appeared in 1950s with the first automated problem attempt, the General Problem Solver (GPS). GPS [1] was the first automated planner introduced in AP literature that expressed human problem solving characteristics in terms of AI algorithms. From that time, it becomes an active research field of an AI with a number of organizations and researchers. Traditionally, AP was considered as part of the problem solving and it has been accused using adaptations of the classical search. The classical planning techniques were used by systems during 1990s include:

- State space search.
- Hierarchical decomposition.
- Heuristic and a range of other techniques developed ad hoc.

The classical approaches in AP presented during that time were assessed on toy problems, such as those used in the IPC, that create real world situations with excessively assumptions and simplifications. Dealing with real planning problems require reasoning about resource and time by planning engine, treat experience knowledge representations, considering dynamic environments, and find way to cooperate with other planners, etc. Although the mentioned issues are crucial for AP, they have been recently introduced to

the planning society (world) as important research directions for the future of AI system. However, most of them are belong to other AI researched areas, such as Constraint Programming, Knowledge Systems, Machine Learning, Intelligent Agents, therefore the ideal way is to utilize the effort already put into them.

3.2 Planning problem

Plans (denoted as a tuple P= (O, s0, g)) are produced by searching a space of actions until a sequence of feasible actions are reached that can carry out the given tasks. In other word, an AP problem comprises a world description: initial, goal states and a domain theory. A domain theory defines a transition state, the way in which the applicable actions change a state of the domain to a new state and their relations with resource. Based on these inputs, planners produce a sequence of executable actions that can reach the goal state from the initial state.

Mainly, one of the keys design philosophy of planning system is domain-independence. In principle, a domain-independent planner works with any planning domain. However, developing domain independent algorithms in many types of problem domains is not reasonable. As a solution of this, some restrictive assumptions were made [2]:

- System has a finite set of state.
- Problem domain fully observable (complete knowledge).
- The outputs of actions are deterministic.
- The system is static (no dynamics events that can change the problem domain)
- Goals are restricted (planner handles only specified goals)
- A solution plan is a linearly ordered finite sequence of actions.
- Actions have no duration
- Change is not allowed during the plan time (off-line planning).

Such planning techniques that accept these assumptions are formally recognized as classical planning. Unluckily, these attempts to improve the domain independence have decreased the usability of planning systems because they brought many restrictions that are infeasible for real world applications.

Classical planners are based on semantic descriptions (i.e., preconditions of actions) provided by a domain model. Recently, an additional expert knowledge are required by planning problems which is may not be fully accessible due to the limitation and complexities of the domains for the experts to provide such knowledge. Hence, it is difficult to develop learning systems in order to learn knowledge as human contributions are restricted [7].

A typical example of a classical planning problem is a Dock Worker Robot (DWR). This problem involves a number of cranes, locations, robots, containers, and piles. Robot starts moving out from one of the locations. The goal is to transport each container to its final destination in a desired order. Consider, for example, an instance of a simple DWR problem with only one robot, two container, two crane and two locations 1 and 2 as given in Figure 1.

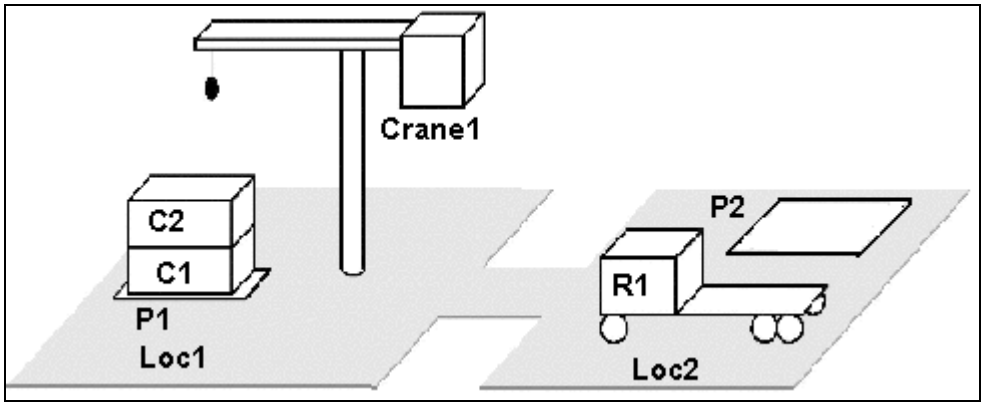

Figure 1: Dock Worker Robot Domain

In this particular example the robot at location 2 and the containers at location 1 in the initial state and the goal is to have the containers to location2. The actions in the DWR domain describe driving the robot from location 2 to location1,take a container off of the

location1 by the crane1, loading a container into the robot at that location1, driving the robot from location1 to location2, unloading a container from the robot by crane2, and put a container on location2. These actions are given in their standard planning domain description language (PDDL) [10] in Figure 2.

```
;; moves a robot between two adjacent locations
   (:action move
     :parameters (?r - robot ?from ?to - location)
     :precondition (and (adjacent ?from ?to)
        (at ?r ?from) (not (occupied ?to)))
     :effect (and (at ?r ?to) (not (occupied ?from))
        (occupied ?to) (not (at ?r ?from)) ))
;; loads an empty robot with a container held by a nearby cr
 (:action load
     :parameters (?k - crane ?c - container ?r - robot)
     :vars (?l - location)
     :precondition (and (at ?r ?l) (belong ?k ?l)
        (holding ?k ?c) (unloaded ?r))
     :effect (and  (loaded ?r ?c) (not (unloaded ?r))
        (empty ?k) (not (holding ?k ?c))))
;; unloads a robot holding a container with a nearby crane
 (:action unload
     :parameters (?k - crane ?c - container ?r - robot)
     :vars (?l - location)
     :precondition (and (belong ?k ?l) (at ?r ?l)
                (loaded ?r ?c) (empty ?k))
     :effect (and (unloaded ?r) (holding ?k ?c)
                (not (loaded ?r ?c))(not (empty ?k))))
;; takes a container from a pile with a crane
 (:action take
     :parameters (?k - crane ?c - container ?p - pile)
     :vars (?l - location ?else - container)
     :precondition (and (belong ?k ?l)(attached ?p ?l)
                        (empty ?k) (in ?c ?p)
                        (top ?c ?p) (on ?c ?else))
     :effect (and (holding ?k ?c) (top ?else ?p)
                (not (in ?c ?p)) (not (top ?c ?p))
                (not (on ?c ?else)) (not (empty ?k))))
;; puts a container held by a crane on a nearby pile
 (:action put
     :parameters (?k - crane ?c - container ?p - pile)
     :vars (?else - container ?l - location)
     :precondition (and (belong ?k ?l) (attached ?p ?l)
                        (holding ?k ?c) (top ?else ?p))
     :effect (and (in ?c ?p) (top ?c ?p) (on ?c ?else)
                 (not (top ?else ?p)) (not (holding ?k ?c))
                 (empty ?k))))
```

Figure 2: DWR Domain model in PDDL

The assumptions of classical planning, and the usual mechanisms for solving it, are rather restrictive, and most real problems are neoclassical. The main differences between classical and neoclassical planning techniques are: in classical planning, for any problem domain consists of group of nodes (search space), every node mapped a partial plan, whereas in neoclassical planning node considered as a set of several partial plan [2].

Many well-known approaches to relax classical planning assumptions have been made, including HTN planning [3,7,8,9], MDPs(Markov decision processes) [4], temporal planning [5], and so on. Nonetheless, classical planning algorithms are still restricted to limited categories of planning domains as most of practical planning problems do not satisfy the early mention assumptions of classical planning [2].

3.3 Planning Application

In the past, planning has been successfully applied in various areas including space exploration, robotics, transportation, finance, crisis management, etc.

AI planning has often been intimately connected with robotics [25]. Some researchers even seem to equate these two issues. For example, [23] introduce their work on planning with the words: *"We are conducting research on robot decisional abilities taking into account explicit reasoning on the human environment and on the robot capacities to achieve its tasks in such a context"*.

While planning is very important for robotics, the converse is not necessarily right. Other authors have a broader view of planning, not only considering robotics and replication of human planning. [24] Write *"Ideally, the set of actions so produced is then passed on to a robot, a manufacturing system, or some other form of effector, which can follow the plan and produce the desired output"*.

Even though many authors do not explicitly mention robots, they often seem to tacitly assume applications having the same or very similar properties. The problem is normally to synthesize plans for some agent situated in a physical environment, sometimes of dynamic and uncertain nature. Such an agent can typically perform a small number of very general and complex operators that can be applied in a large number of states and objects. The plans are typically not very long but may be hard to find because of the complexity of a world model employed.

An attractive application for AP is when a process collapse or is blocked in an emergency situation. Following such an incident, the system may be in any number of states and therefore requires a very complex plan to revive the system again to resume work normally. It is not realistic to have precompiled plans for how to start up the operation again from each such state. Furthermore, involved costs are considered as large industrial process terminated, such as a paper mill, is inactive necessitate a prompt response from the planner. Another example of applications is a traffic system, for example a motorway or underground network. In such case a planner could be used raptly and regularly, to prevent break downs and easy the flow of traffics. For example, as a support system for network operators to decide how to direct the aeroplanes traffic.

Additionally, automated planning techniques have been successfully applied to wide range of real domain problems: space missions [29], management of fire extinctions [30] or control of underwater vehicles [31]. Although these are very successful application adopting AP techniques, but still suffering from various knowledge representation.

4. PLANNING DOMAIN MODEL LANGUAGES

In current planning research quite a lot of formal work has been produced to understand and precisely characterize the power of automated planning frameworks ([2, 3, 8.13] and many others).the first issue arises in this field is how and in which form knowledge should be represented in. Hence, several notations (domain representation languages) and criteria have been developed over time.

4.1 STRIPS Language

In this section, we first sketch the basic planning domain model representation language of classical planning problems, identified as the STRIPS language. Possible variations in STRIPS-like languages will be described later on. STRIPS (STanford Research Institute Problem Solver) [20,21] is one of the oldest, simplest, and most used planning language, even if not too flexible. A problem in Strips is a tuple <A,O,I,G>. STRIPS planning was introduced in [20] as a model of the kind of planning problems that

people appear to solve in common sense reasoning. STRIPS planning corresponds to a certain formal graph search problem.

It is important to distinguish between the original STRIPS planner, and the STRIPS representation language. Due to its power of representation, the language developed for describing STRIPS' operators and world domain models was since then used, with minor modifications, in a large number of traditional planners.

Planning was historically motivated by the need of robotics. Indeed, STRIPS was designed to act as the planner for another pioneering project: "Shakey the robot", developed at SRI International. Shakey was supposed to wander in his world (figure 3), push boxes around, and turn light switches on/off (the former action also required to climb on top of a box, since the robot was short and the switches were high on the wall). It is important to note that this environment was quite idealized compared to a real one, given the primitive sensors and effectors available at that time.

STRIPS represents world domain models as a collection of first order predicate calculus formulas. In this context, "problem solving" means finding a sequence of operators in the space of world models that will transform an initial model into another model in which a given formula (goal) can be proven to be true.

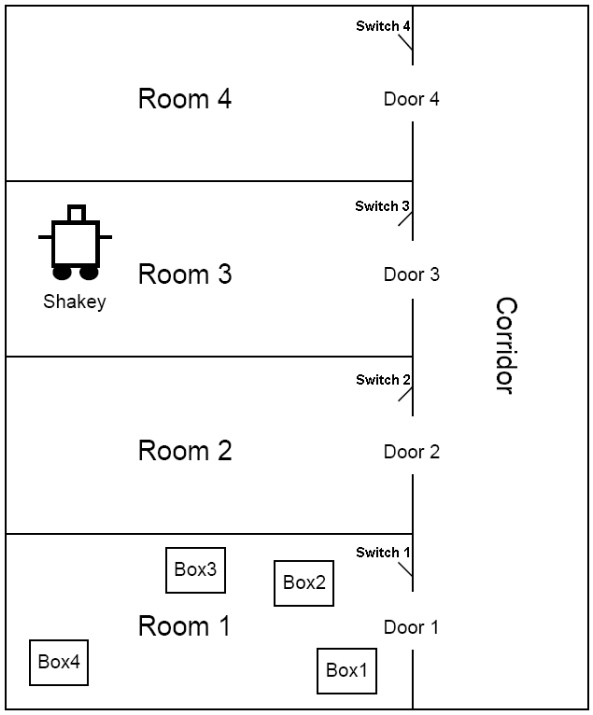

Figure 3: Shakey the robot's world (from [22])

The restrictions forced by the STRIPS were chosen to make planning algorithms simpler and able to express real world problems. One of the most important restrictions is that literals be function-free [22]. With this restriction, an action schema for any problem can be propositionalized (represented into a finite collection of propositional action that is free of variables).

Recently, STRIPS has become inadequately expressive planning description language for representing various real world domains. As a result, many languages have been developed.

4.2 Action Description Language (ADL)

This part, briefly describes an important planning description language, the Action Description Language (ADL). The ADL (10) relaxed some of the restrictions assumptions in the STRIPS language and provided some flexibility to encode more realistic world domains. The Problem Domain Description Language [2] was introduced as a standardized syntax for representing ADL, STRIPS, and other languages

4.3 Planning Domain Definition Language (PDDL)

Specification languages like STRIPS or ADL can be used to describe a real problem domain. Another possibility to describe real domains is the planning domain definition language (PDDL). It was developed by Drew McDermott in 1998 and later evolved continuously by **International Planning Competitions** to standardize planning domain and problem description languages. The latest version is PDDL3.1 which introduced two new features, state trajectory and preference constraints [32]. PDDL is supported by most planning engine. It is used well to identify the domain properties and specifications in detail, the predicates which are used and the action definition. Additionally, it includes the properties of some planning domain language such as STRIPS, ADL, and the hierarchical task networks (HTN)[2,3,9]. PDDL Is descended from the ancestors of several notations: ADL, SIPE-2, UMCP, Unpop, and UCPOP [10].

Although, The STRIPS and ADL demonstrated that are sufficient to express several of real world domains, there are still some important limitations (restrictions). The most obvious is that they cannot represent in a natural way the ramifications of actions. For example, if there are people, luggage, etc in the ship, then they change location when the ship sails. Changes can be represented as the direct effects of sailing, whereas it seems more natural to represent the location of the ship's contents as a logical consequence of the location of the ship. More examples (e.g Block world and Air cargo transport) are described by [10].

4.4 Object-Centred Language (OCL)

OCL has been developed by (Donghong Liu and T.L.McCluskey) at the University of Huddersfield. [26] Described OCL as a tool-supported language for domain developers. OCL aims to provide a language that representing the domain models in the classical tradition of AI Planning considering the structure and dynamics of domains.

5. PLANNING KNOWLEDGE-BASE & ML

Basically, an AI planning algorithm requires a problem domain model to be given beforehand, for example FF[19], is an independent-domain planner that is required two separated files to be executed: problem file and domain file. Though, it is both boring and time-consuming to encode the domain models by hand using a formal language e,g PDDL [10], OCL [11]. On other way, a domain model can be revised using interactive systems such as GIPO[18] by obtaining the training plans through monitoring devices such as sensors and cameras. Another way to extract and induce a problem domain model by using ML mechanism [14, 17].

5.1 Knowledge acquisition

Knowledge acquisition is the major challenge in AI planning. The most recently used techniques to extract and acquire a planning knowledge are the machine learning tools. Therefore, some works have been proposed in automatically learning domain models from plan traces with intermediate observations. In many real world applications, such intermediate states are usually not fully observable. Most of these works require an amount of training data as input [27]. To address this limitation, [13] Present approach to learning domain model by using other planning domains and transferring shared parts

from an existing domain models to a new domain model by finding a logical relationship between these parts. For instance, in transportation domain and DWR domain, a passenger should be in a bus and a container should be on the robot which means that the bus, the passenger, the robot and the container must be at desired location Loc in order to achieve this action. Consequently, the presence of the bus, the passenger, the container and the robot at the location represents a logical relationship of the two domains.

[14] Develop a learning algorithm known as Simultaneous Learning and Filtering (SLAF). The algorithms learn action that a sufficient number of training traces are available in partially observable domains. SLAF algorithms cannot take advantage of other related planning domains to help the learning task.

Most of learning algorithms are required huge amount of training data to be achieved. From AI planning history, the planning problem is considered in a general concept where actions can be deterministic, non-deterministic, or probabilistic, and actions' effects can be fully [15] or partially observable [14, 16]. Recently, there are several attempts deal with non-observable environment. A notable example of non-observable environment is Learning Object-Centred Model (LOCM) [17]. LOCM is an inductive tool which is automatically induces a domain model from set of training example of plans. LOCM requires no knowledge in advanced (e.g description of states, predicates, actions, etc…) to be executed.

5.2 Knowledge representation

Designing planning domain models is not easy job even for professionals. The process of encoding is arduous. Questions such as: how to use domain knowledge, what are the optimal methods to represent a real problem domain, are the existing domain model languages claim the purpose of domain model representation, and what features that are needed to be attached with the languages to express various world problem with less or non-faults, are yet to be accurately answered. Some of the existing planners use domain independent search heuristics, and some others depend on domain specific knowledge. Planners that use domain independent are relatively slow compared with domain dependent planners. This is the answer of the question of why most successful

real world planning systems are domain specific (or domain dependent). However, it's not practically reasonable to build a whole new planner for every domain.

5.3 Domain-dependent

Nowadays many logic based representations formalisms are available for knowledge engineering tasks. However, most of them do not support the representation of domain- specific knowledge. On one hand building special systems, which supports these representation tasks is a costly and cumbersome process. For specific planning applications, domain-specific methods are well justified. They are greatly successful in most of the application areas (e.g path and motion planning, navigation planning). However, they are restricted for several reasons [2].

Commonalities to all of applications of planning are not addressed.

More costly to address each planning problem anew.

Not satisfactory for designing an autonomous intelligent machine.

Although, the automated planning is interested in domain-independent but it is not meant to be opposed to domain-dependant planning techniques. The direct implementation of inference mechanisms reflecting the semantics available in a specific domain leads to an inflexible system: any miniature change in the input language results in changes of the whole system.

5.4 Domain-independent. Why?

There are several reasons for planning domain-independent.

- A domain-independent reduce development, debug and maintenance cost (knowledge are located in specified place).
- Reusable (it can be reused over time).
- Domain-independent easier to swap planning engines, prove properties of engine and knowledge base (e.g engine always produces optimal solutions when the knowledge base is consistent)
- Additionally, Planning engines can be developed independent of their application in a "clean" fashion (separate efforts developing engines and tools for knowledge engineering domain models).

6. DOMAIN MODEL INVESTIGATIVE TOOLS

6.1 Research Issue

The focus of this project is to develop a set of planning domain metrics and comparison techniques to analyse a domain model's structure, characteristics and properties, in order to investigate and measure domain complexity and equivalence. Briefly, the research study will go through these stages:

- Forming domain model, inductive domain model tools and domain model languages criteria.
- Developing measurement tools based on high quality of planning domain models.

6.2 Research questions

The investigation should answer research questions such as, what makes a domain model structurally sound, efficient, simple or complex? How does impurities (bug) planning domain model affect an AI system, comparing to a standard notation of the domain language? How possible to make the domain models that are produced by machine learning tools to meet the standard domain model language notation?

6.3 Experimental

We implemented our experimental as an extension to the FF planning engine and LOCM tool on extensive testing performed to evaluate how different, similar, complex are the domain models that induced by ML tools comparing to the standard ones.

A. Experimental Design

This experimental can be logically described as having three components:

Planner: FF planner [19], independent-domain planner works with any planning domain.

Inductive tool: LOCM [17], an ML tool that automatically generate a planning domain model from example training plans. In this experimental, the required examples are provided by FF planner.

Converter component: a piece of code has been written and added to FF planner in order to collect and reformat the output of the planner into OCL notation. figure 4 describe the operation of the converter component.

Run FF

Problem file

Domain file

For every domain

Print: **domain (**domain name**)**.

Print: sequence_task (#,[

End

Read x

If x is an action & not the last action then

Print: name of action **(** parmeter$_1$, par$_2$,…,par$_n$ **),**

Figure 4: Pseudocode of Converter Component

Using several known world domain models (i.e Depot, Rover, DWR) as examples to show the difference between tow domain models for the same world domain and same description domain model language. The following diagram illustrates the mechanism of obtaining a world domain model using FF and LOCM planning tools.

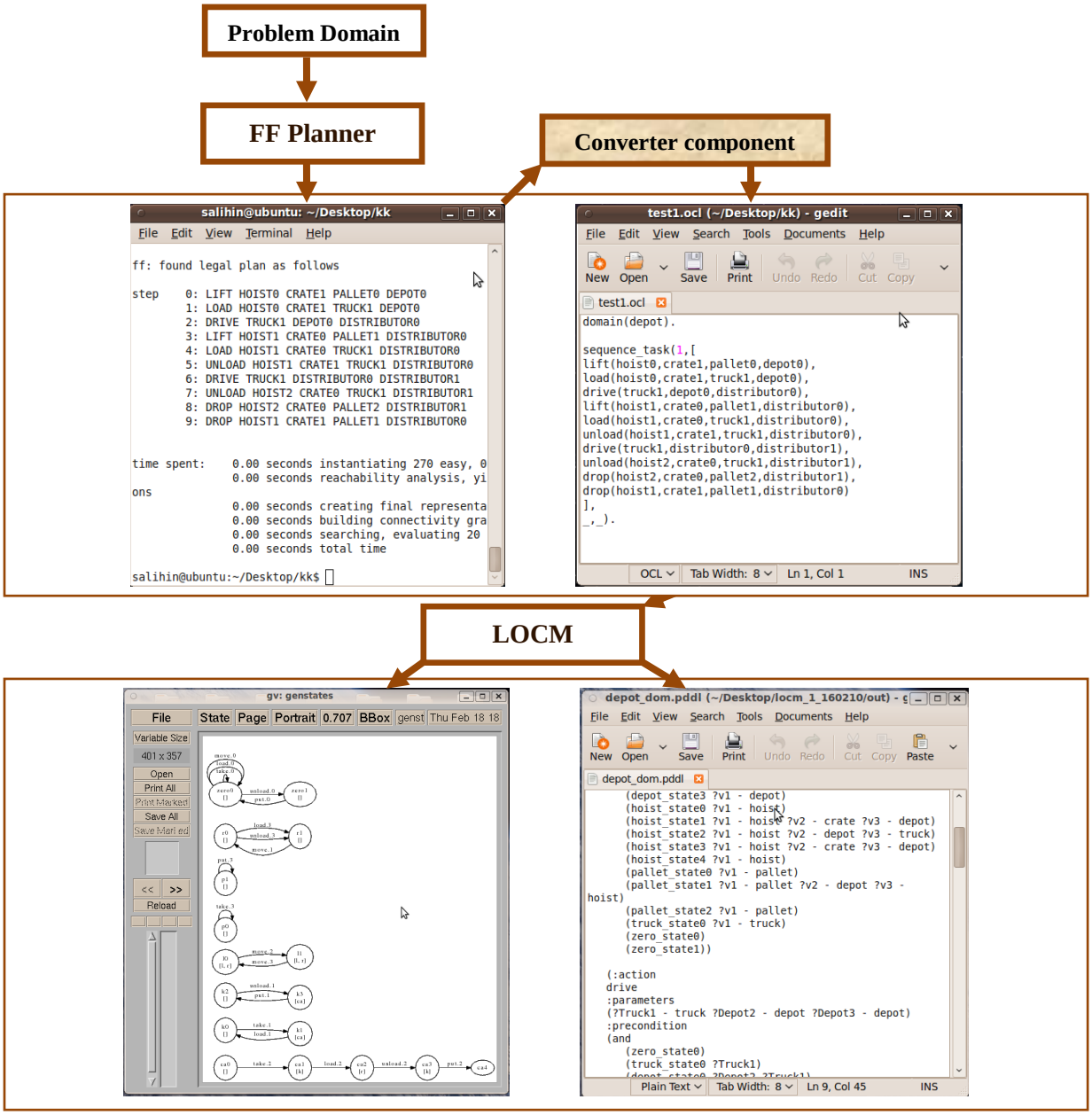

Figure 5: Inducing Domain Model using FF & LOCM

B. Experimental implementation and result

Several diverse types of experiment examples could be performed in this manner. Here is one example:

Step1: run FF with problem file called (pfile01) and domain file (domain.pddl) .

This figure shows the actions training sequence of Depot domain produced by FF planning engine.

```
0: lift hoist0 crate1 pallet0 depot0
1: load hoist0 crate1 truck1 depot0
2: drive truck1 depot0 distributor0
3: lift hoist1 crate0 pallet1 distributor0
4: load hoist1 crate0 truck1 distributor0
5: unload hoist1 crate1 truck1 distributor0
6: drive truck1 distributor0 distributor1
7: unload hoist2 crate0 truck1 distributor1
8: drop hoist2 crate0 pallet2 distributor1
9: drop hoist1 crate1 pallet1 distributor0
```

Figure 6: Depot Plan Produced by FF

Step 2: convert the FF result into OCL notation

This figure presents the output of FF planner translated into OCL notation which is accepted by LOCM.

```
domain(depot).
sequence_task(1,[
lift(hoist0,crate1,pallet0,depot0),
load(hoist0,crate1,truck1,depot0),
drive(truck1,depot0,distributor0),
lift(hoist1,crate0,pallet1,distributor0),
load(hoist1,crate0,truck1,distributor0),
unload(hoist1,crate1,truck1,distributor0),
drive(truck1,distributor0,distributor1),
unload(hoist2,crate0,truck1,distributor1),
drop(hoist2,crate0,pallet2,distributor1),
drop(hoist1,crate1,pallet1,distributor0)
],
_,_).
```

Figure 7: Depot Plan Converted into OCL Notation

Step 3: run LOCM using the example in step 3 as input.

Full detailed description of the domain is not possible due to the limitation of the report. Consequently, figure 8 shows a part of the Depot domain model to be compared with standard one.

```
(define
   (domain depot)
   (:requirements :typing)
   (:types crate depot hoist pallet truck zero)
   (:predicates
      (crate_state0 ?v1 - crate)
      (crate_state1 ?v1 - crate ?v2 - depot ?v3 - hoist)
      (crate_state2 ?v1 - crate ?v2 - truck)
      (crate_state3 ?v1 - crate ?v2 - depot ?v3 - hoist)
      (crate_state4 ?v1 - crate)
      (depot_state0 ?v1 - depot ?v2 - truck)
      (depot_state1 ?v1 - depot ?v2 - crate ?v3 - hoist)
      (depot_state2 ?v1 - depot)
      (depot_state3 ?v1 - depot)
      (hoist_state0 ?v1 - hoist)
      (hoist_state1 ?v1 - hoist ?v2 - crate ?v3 - depot)
      (hoist_state2 ?v1 - hoist ?v2 - depot ?v3 - truck)
      (hoist_state3 ?v1 - hoist ?v2 - crate ?v3 - depot)
      (hoist_state4 ?v1 - hoist)
      (pallet_state0 ?v1 - pallet)
      (pallet_state1 ?v1 - pallet ?v2 - depot ?v3 - hoist)
      (pallet_state2 ?v1 - pallet)
      (truck_state0 ?v1 - truck)
      (zero_state0)
      (zero_state1))

   (:action
   drive
   :parameters
   (?Truck1 - truck ?Depot2 - depot ?Depot3 - depot)
   :precondition
   (and
      (zero_state0)
      (truck_state0 ?Truck1)
      (depot_state0 ?Depot2 ?Truck1)
      (depot_state2 ?Depot3))

   :effect
   (and
      (depot_state0 ?Depot3 ?Truck1)
      (not (depot_state2 ?Depot3)))
)
```

Figure 8: Depot Domain Model Produced by LOCM

Step 4: standard domain

Figure 9 shows the final result of the experimental which is clearly conclude The differences between the two domain models(step 3 and step 4)

```
(define (domain Depot)
(:requirements :typing)
(:types place locatable - object
      depot distributor - place
        truck hoist surface - locatable
        pallet crate - surface)

(:predicates (at ?x - locatable ?y - place)
             (on ?x - crate ?y - surface)
             (in ?x - crate ?y - truck)
             (lifting ?x - hoist ?y - crate)
             (available ?x - hoist)
             (clear ?x - surface))

(:action Drive
:parameters (?x - truck ?y - place ?z - place)
:precondition (and (at ?x ?y))
:effect (and (not (at ?x ?y)) (at ?x ?z)))
```

Figure 9: Depot Domain Model Produced by Handcraft

7. FUTURE WORK

A problem with the current designing of the planning domain models is its complexity of development. It may be necessary to look at ways to measure the domain models, domain model inductive tools and domain model representation languages to find out what make them different structurally and if they affect the final AI systems. This may involves considerably different optimisation techniques, or simply writing a code in a more efficient manner. In general, the current target is to develop a technique that can define optimal (or near to optimal) standard notation to build non-bugged planning domain model considering its description language notations. Based on the result of the experimental, a set of planning domain metrics and comparison techniques to analyse a domain model's structure, characteristics and properties will be developed, in order to investigate and measure domain complexity and equivalence.

7.1 Second Year Plan

A. **Literature Review:** during the first year, I have failed to cover some important topics that related to the area of research. consequently, within the first three mounts important topics will be revised once again.

- Run and evaluate some other planning algorithms that are presented in ICAPS and ICKEPS.
- Comparative study and evaluation of most known LM domain models inductive tools comparing to LOCM.
- Review latest related research paper (ICAPS and ICKEPS)
- Revise using available material to fill some holes in knowledge engineering background.
- Continue improving skills of using Prolog.
- Intensive survey of domain model languages (advantage/disadvantage)

B. Model Design:

- Perform experiments using different planning algorithms and domain model inductive tools.
- Develop a new independent (can be used with any planner) component to convert planning example into OCL notation.

- Identify standard criteria to be used with our matrix measurement tools
- Create and design feasible standard domain model metrics tools.

C. Publication:

- School conference
- ICAPS
- First journal paper

Reference

[1] A. Newell and H. Simon. Computers and Thought, chapter GPS, a Program that simulates human thought, pages 279-293. McGraw Hill, NY, 1963.

[2] M. Ghallab, D. Nau, and P. Traverso. Automated Planning: Theory and Practice. Morgan Kaufmann, 2004.

[3] D. Nau, T. Au, O. Ilghami, U. Kutter, W. Murdock, D. Wu, and F. Yaman. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*, 20:379-404, December 2003.

[4] L. Kaelbling, M. Littman, and A. Cassandra. Planning andacting in partially observable stochastic domains. *Artificial Intelligence*, Volume 101, pages: 99-134, 1998.

[5] Bacchus and Kabanza, 2000,F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116:123-191, 2000.

[6] Chapman and Agre, 1987, D. Chapman and P. Agre. Pengi: An implementation of a theory of activity. AAAI-87, 1987.

[7] C.Hogg, U.Kuter, H.Munoz-Avila. From Plan Traces to Hierarchical Task Networks Using Reinforcements: A Preliminary Report. Workshop on Learning Structural Information from Traces (STRUCK-09). AAAI Press. IJCAI-09,

[8] H. Zhuo, D. Hao Hu, C. Hogg, Q. Yang, H. Munoz-Avila.Learning HTN Method Preconditions and Action Models from Partial Observations. *Artificial Intelligence (IJCAI 2009)***,** Pasadena, California, USA, Pages 1804-1809.

[9] C. Hogg, H. Mu˜noz-Avila, and U. Kuter. HTN-MAKER: Learning HTNs with minimal additional knowledge engineering required. In Proceedings of AAI, pages 950–956, 2008.

[10] AIPS-98 Planning competition Committee. PDDL – the Planning Domain Definition Language. Technical Report CVC TR-98-003/DCS TR-1166, Yale Center for Computational Vision and Control, 1998.

[11] D. Liu and T.L.McCluskey. The OCL Language Manual, Version 1.2 Technical report, Department of Computing Science, University of Huddersfield, 2000.

[12] Zimmerman, T. and Kambhampati, S. (2003). Learning-assisted automated planning: looking back, taking stock, going forward. AI Magazine, 24:73 – 96.

[13] Hankui Zhuo, Qiang Yang, Derek Hao Hu and Lei Li, Transferring Knowledge from Another Domain for Learning Action Models. PRICAI 2008. Pp 1110 -1115.

[14] Amir, E. 2005. Learning partially observable deterministic action models. In Proceedings of IJCAI'05, 1433–1439. Benson, S. 1995. Inductive learning of reactive action models. In Proceedings of ICML'95, 47–54.

[15] Xuemei Wang: Learning by Observation and Practice: An Incremental Approach for Planning Operator Acquisition. ICML 1995: 549-557

[16] Allen Chang, Eyal Amir: Goal Achievement in Partially Known, Partially Observable Domains. ICAPS 2006: 203-211

[17] S. N. Cresswell and T. L. McCluskey and M. M. West. Acquisition of object-Centred Domain Models from Planning Examples. ICAPS 2009: 338-341

[18] T. Leo McCluskey, D. Liu, R.M. Simpson, GIPO II: HTN planning in a tool-supported knowledge engineering environment, in: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS 2003), Trento, Italy, 2003, pp. 92–101.

[19] J¨org Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research, 14:263–302, 2001.

[20] Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. Artificial Intelligence, 2:198-208, 1971.

[21] R.Frikes and N. nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. Artificial intelligence, 2:189-208, 1971.

[22] S. Russell and P. Norvig. Artificial Intelligence: A Modern Approach. Prentice Hall, 1995.

[23] Rachid Alami, Aur´elie Clodic, Vincent Montreuil, Emrah Akin Sisbot and Raja Chatila. Task planning for Human-Robot Interaction. P81-85, 2005.

[24] Moataz Ahmed, Ernesto Damiani, and David Rine. Fast Recall Of Reusable Fuzzy Plans Using Acyclic Directed Graph Memory.272-276, 1998.

[25] Alami, R.; Chatila, R.; Clodic, A.; Fleury, S.; Herrb, M.; Montreuil, V.; and Sisbot, E. A. 2006. Towards human aware cognitive robots. In AAAI-06, Stanford Spring Sympoium.

[26] Liu, D., and McCluskey, T. L. 2000. The OCL Language Manual, Version 1.2. Technical report, Department of Computing and Mathematical Sciences, University of Hudderseld.

[27] T.L.McCluskey, S.N.Cresswell, N.E.Richardson, R.M.Simpson and M.M.West. Acquisition of Planning Operator Schema using Opmaker.

[28] Jorge A. Baier and Fahiem Bacchus and Sheila A. McIlraith. A Heuristic Search Approach to Planning with Temporally Extended Preferences. IJCAI07. P 1808 – 1815

[29] Nayak, P., Kurien, J., Dorais, G., Millar, W., Rajan, K., and Kanefsky, R. (1999). Validating the ds-1remote agent experiment. In Artificial Intelligence, Robotics and Automation in Space.

[30] Castillo, L., Fdez.-Olivares, J., Garc´ıa-P´erez, O., and Palao, F. (2006). Bringing users and planning technology together. experiences in SIADEX. In International Conference on Automated Planning and Scheduling (ICAPS 2006).

[31] Bellingham, J. and Rajan, K. (2007). Robotics in remote and hostile environments. Science,318(5853):1098–1102.

[32] Stefan Edelkamp, On the Compilation of Plan Constraints and Preferences. 2006, American Association for Artificial Intelligence. P 374-377