NL to PDDL

One-Shot Learning of Planning Domains from Natural Language Process Manuals

S. Miglani

NL to PDDL

One-Shot Learning of Planning Domains from Natural Language Process Manuals

by

S. Miglani

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday August 27, 2019 at 15:00.

Student number: 4723058
Project duration: November 1, 2018 – August 14, 2019
Thesis committee:

Dr. Neil Yorke-Smith,	Algorithmics, TU Delft, supervisor
Prof. Catholijn Jonker,	Interactive Intelligence, TU Delft
Dr. Casper Poulsen,	Programming Languages, TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Abstract

Automated Planning (AP) is a key component of Artificial General Intelligence and has been successfully employed in applications ranging from scheduling observations of Hubble Space Telescope to generating dialogue agents. A significant bottleneck for its widespread adoption is acquiring accurate domain models which formally encode the planning problem's environment. Traditionally, these domain models have been hand-coded by human experts and knowledge engineers. However, manually encoding domain models is an increasingly difficult task when one moves away from toy domains towards complex real-world problem scenarios.

To resolve this, the AP community has developed several systems to automatically acquire domain models from valid sequences of actions called plans. This approach has two significant issues. First, the generated domain models might be incomplete, error-prone, and hard to understand and/or modify. Second, most domain learning approaches are based on data-intensive inductive learning, which needs large quantities of structured data (plans) to converge. This data is seldom available without an accurate domain model, which leads to a causality dilemma.

To mitigate these issues, we take advantage of readily available and easy to craft Natural Language (NL) data. We present a pipeline called NLtoPDDL, which takes as input a classical domain's process manual written in a natural language and outputs its Planning Domain Definition Language (PDDL) model. Specifically, NLtoPDDL does this in two steps: first, it combines pre-trained contextual embeddings with an approach developed in previous research [33] that extracted structured plans from NL data using Deep Reinforcement Learning (DRL), and a consequence, NLtoPDDL beats the Feng et al. [33]'s model which is the current state-of-the-art on action sequence extraction problem; second, it uses the trained DRL model from the first step to extract structured plans from a domain process manual and employs a modified Learning Object Centered Models (LOCM2) algorithm [25] to one-shot learn a PDDL model. Finally, we showcase the effectiveness of our pipeline on four planning domains of varying complexities, by evaluating our learned domain models for soundness, completeness, validity and intuitiveness.

S. Miglani
Delft, August 2019

Preface

Imagine giving instructions to Siri or Google Assistant on your phone, and they suddenly get the context of your words and can rationally make further decisions for you. This thesis has been inspired by this teacher-student learning setting and working on it has been a journey through many fields of Artificial Intelligence, from natural language transfer learning and deep reinforcement learning to automated planning and knowledge engineering. It has in many ways, enhanced my understanding of AI, introduced me to the very recent happenings in it and has also sparked my interest in automated planning and its use in space missions. I started with a relatively simpler problem of learning planning domain models from structured data, which was hard to get for real-world problems. In a quest to truly apply my research to real-world scenarios, the horizons were broadened to unstructured data in the form of natural language and challenges associated with it.

I am deeply grateful to Dr. Neil Yorke-Smith, who has been my supervisor during the last year. I immensely appreciate the freedom that you gave me to follow my curiosity and interests. I thank you for your guidance, support, patience and invaluable feedback. I would also like to thank Wenfeng Feng and Julie Porteous who shared the code for their research, and the amazing open-source community in AI, for the techniques reused in this thesis.

Suhasni and Raghu have been invaluable during my time at TU Delft, and I thank them for their continuous encouragement. Last but certainly not the least, I sincerely thank my beloved father and sister for their unwavering and everlasting support.

S. Miglani
Delft, August 2019

Contents

1 Introduction

Automated Planning (AP) is a branch of Artificial Intelligence (AI) which concerns the autonomous realisation of a **plan**: a *sequence of actions* that transforms the environment from an initial state to the desired goal state while adhering to specified performance measures and constraints. AP allows for an autonomous agent to computationally think and anticipate future events to reach its objectives and thus, forms an important component of Artificial General Intelligence.

The very first application of AP, a robot named Shakey [94] uses **ST**anford **R**esearch **I**nstitute **P**roblem **S**olver (STRIPS) [34] as a planner to determine plans for moving boxes in an office environment. Since the days of Shakey, the AP community has made huge advances in developing fast and robust **domain-independent planners** such as BFWS [37, 77], FF [60], and FD [56, 57, 110, 111] which work for a wide range of domains. For instance, domain-independent planning has been successfully employed in space mission operations [21, 66], urban traffic control [22, 122], and generating dialogues [16]. Furthermore, major breakthroughs have happened in AI by combining planning with deep learning. For example, the famous Go[1] playing computer program AlphaGo [113] used a planning technique called Monte-Carlo tree search with a neural network acting as a heuristic generator, to select the next best action.

As we know the rules of Go game, it may be trivial to encode its domain knowledge into the system for using Monte-Carlo tree search. Yet, as we move away from the toy and game-like environments to real-world environments like controlling a space-aircraft, *it becomes increasingly difficult and time-consuming to model the dynamics of a domain for computational use*. Thus, despite the accomplishments, the amount and sophistication of Knowledge Engineering (KE) required limits the widespread adoption of AP systems. In particular, we need the **domain model**, which is a formal representation of the system in which we want to plan, and the **problem definition**, which comprises of our planning problem's initial and goal state, to use a domain-independent planner as shown in Figure 1.1. For this purpose, the AP community has developed **P**lanning **D**omain **D**efinition **L**anguage (PDDL) [36, 45], which has become the de facto standard to encode domain knowledge. Once we have the domain model in PDDL, we can use it without changes to solve several problems, which are specified using a problem definition (a tuple of initial and the desired goal state).

Figure 1.2 shows the PDDL model for a simple `gripper` domain, which comprises of two parts:

[1] `https://en.wikipedia.org/wiki/Go_(game)`

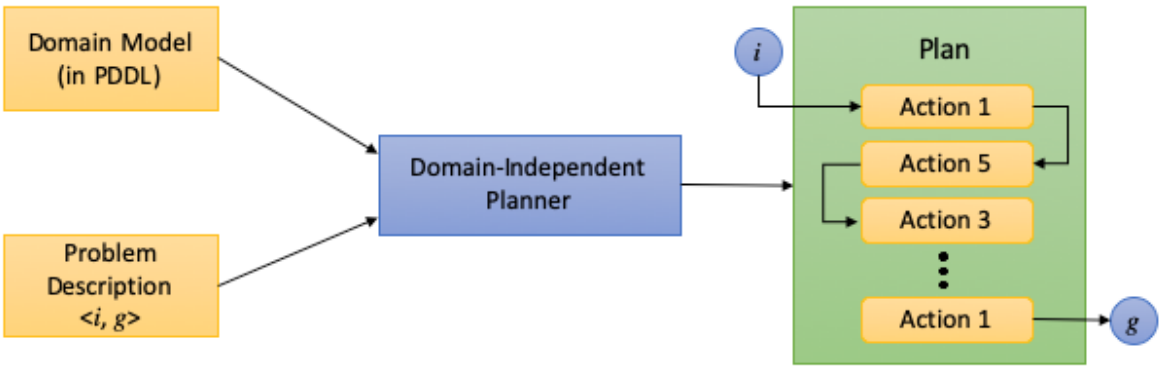


Figure 1.1: Automated Planning System: A domain-independent solver or planner takes two inputs: 1) the domain model written in a planning language and 2) a problem definition that describes the initial state i and the desired goal state g using the domain model's terminology; and produces as output a plan, that is, a sequence of actions that takes the agent from the initial state to the goal state.

- the ***predicates*** that describe the state of an agent, and
- an ***action model*** which defines what are the action choices available for an agent, the *preconditions* of when a specific action can be performed, and the *effects* of performing that particular action.

This work concentrates on the problem of acquiring such domain models for complex real-world problems. We present an automated KE tool called **NLtoPDDL**, which learns a valid and intuitive PDDL model from a natural language process manual describing an example sequence of actions taken in the domain. Imagine, describing how you planned your last week to a virtual assistant like Siri[2] in a natural language, for the virtual agent to create the domain model about what you have described and plan the next week for you while fulfilling your mentioned goals. Apart from this practical application in teacher-student settings, the scientific motivation of knowledge engineering and modelling the world around us for computational use is at the heart of our approach. In the next section, we describe our rationale behind and need for such an approach in terms of the work that has already been done in the automated planning community.

1.1. Motivation

There are two methods for acquiring domain models: the traditional way of manually writing it, or an automated way of learning the domain model from some input data. We look at both paradigms and motivate our approach through their shortcomings.

1.1.1. Manual Domain Model Acquisition

Traditionally for complex real-world problems, domain models are written manually through the collaboration among human Subject Matter Experts (SMEs) and human Knowledge Engineers (KEs). The SMEs know the dynamics of the domain in some natural language but do not necessarily have expertise in representing it in a formal planning language. The KEs extract this knowledge from the SMEs to formulate a syntactically and semantically valid PDDL domain model to be used by the off-the-shelf planner. This process is an iterative process in which multiple verification and validation (V&V) loops are incorporated via

[2] https://www.apple.com/siri/

```
(define (domain gripper-strips)
 (:predicates (room ?r)  (ball ?b) (gripper ?g) (at-robby ?r)
              (at ?b ?r) (free ?g) (carry ?o ?g))
 (:action move
  :parameters (?from ?to)
  :precondition (and  (room ?from) (room ?to) (at-robby ?from))
  :effect (and (at-robby ?to) (not (at-robby ?from))))
 (:action pick
  :parameters (?obj ?room ?gripper)
  :precondition (and (ball ?obj) (room ?room) (gripper ?gripper)
                     (at ?obj ?room) (at-robby ?room) (free ?gripper))
  :effect (and (carry ?obj ?gripper) (not (at ?obj ?room))
               (not (free ?gripper))))
 (:action drop
  :parameters (?obj ?room ?gripper)
  :precondition (and (ball ?obj) (room ?room) (gripper ?gripper)
                     (carry ?obj ?gripper) (at-robby ?room))
  :effect (and (at ?obj ?room) (free ?gripper)
               (not (carry ?obj ?gripper)))))
```

Predicates

Action Model

Figure 1.2: Gripper domain PDDL model: A robot moves balls from one room to another using actions move, pick and drop.

simulations and debugging, which makes it time-consuming. Figure 1.3 summarises the manual domain model acquisition.

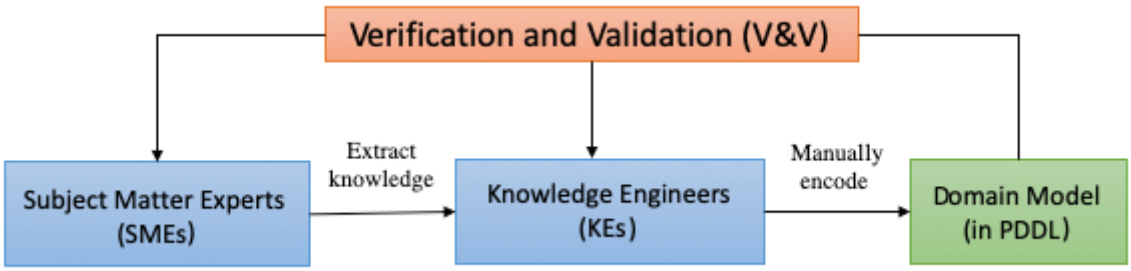


Figure 1.3: Manual Domain Model Acquisition: the paradigm works in real-world scenarios but is time-consuming and non-scalable.

Many knowledge engineering tools have been developed to help SMEs and KEs. These range from Integrated Development Environments (IDE) like myPDDL [116] to Graphical User Interface (GUI) tools such as itSIMPLE [124]. Despite these tools, the traditional way to acquire domain models is time-consuming, error-prone, labour-intensive and difficult to scale for bigger planning problems because of the enlisted challenges:

- **The inherent complexity of real-world problems**: Automated Planning is usually required in complex tasks for which humans can't plan effectively. This might be due to problem's large-scale such as planning logistics delivery of orders placed on an e-commerce site for a city, or due to constraints like the degree of autonomy required, like in space missions owing to limited data transfer link capacity. The scale of such problems make it very time-consuming or even infeasible to create an action model

with all its preconditions and effects.

- **Human beings are prone to errors and omissions**: The decision problem of whether a plan exists for a problem definition is a PSPACE hard problem [18, 31, 46] for most cases. Thus, the quality of plans generated from a domain-independent planner is dependent on the quality of domain models. Effective communication between the KEs and SMEs is required to build an accurate domain model. It is possible that the intricacies of the domain cannot be described in words. Omissions could lead to additional iterations of validation and coding, making it an increasingly time-consuming task. Thus, the quality of the domain model varies with 'expertness' of the SMEs and KEs.

- **Limitations related to manual KE tools**: Although IDEs like myPDDL [116], help with syntax highlighting, syntax-checking and easy collaboration through a version control system (VCS), they still are inadequate due to the scale of real-world problems, and sophistication of knowledge engineering required. itSimple [124], an award-winning GUI method takes it a step further by improving the quality of domain models produced by automating certain domain modelling aspects through dynamic analysis but still, requires encoding of the domain in Unified Modelling Language (UML) standard [103] and is much less flexible [112]. Also, there are many other drawbacks such as bugs, no updates, and limited expressivity in terms of PDDL features. This necessitates the V&V cycles, and thus, there is no significant reduction in time taken for modelling.

Thus, to alleviate the bottleneck of time-consuming manual domain model acquisition, a growing body of work has emerged in the AP community which provides us with several domain acquisition algorithms that learn the domain model from some input data.

1.1.2. Automated Domain Model Acquisition

For the past decade, the AP community has been researching on automatically *learning domain models* from input data such as previously generated plans or logs of planning missions. This paradigm, shown in Figure 1.4, is motivated to mitigate the challenges associated with manually writing domain models. A variety of domain learning algorithms with varying complexity and capabilities have been devised. Section 3.2 presents a taxonomy and survey of such domain learning methods.

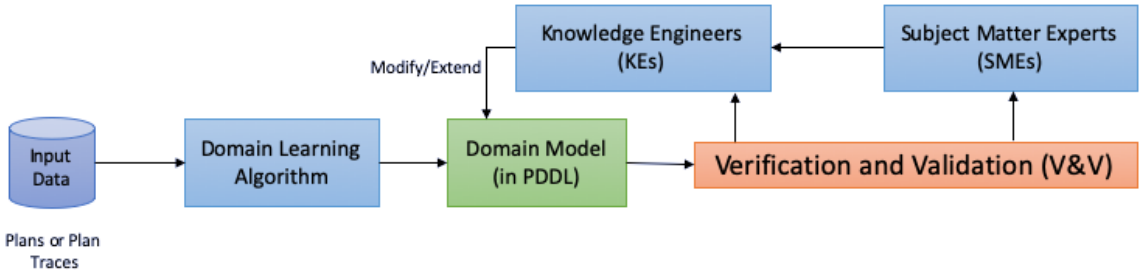


Figure 1.4: Automated Domain Model Acquisition: the paradigm produces an initial domain model which SMEs and KEs can modify or refine, or it can be used in model-lite planning [138, 143] which works with shallow and incomplete models.

However, these automated methods suffer from four major issues:

1. **Quality of the domain model**: The generated domain models are incomplete, inconsistent and error-prone. This drawback makes automated domain learning an

assistive technology, in which human SMEs and KEs take the initially created domain model and verify, validate, and modify it, to perfect the domain model. Nonetheless, it saves time and effort to have an initial domain model if other issues are resolved. A new paradigm called **model-lite planning** [138, 143] which works with such initial and incomplete models has also emerged on the scene and is an active field of research.

2. **Explainability of the domain model**: Explainability is an important aspect of Automated Planning. Why a particular action was chosen or to whom the responsibility is to be assigned for safety-critical autonomous systems, say a medical expert system, are important questions that must be answered. This process of explainability starts from intuitive domain models. If the learned domain model is not intuitive and uses variables like $c1, c2$ for say, a car object, it is hard to understand, modify, and even validate it, as it is difficult to formulate initial and goal states in terms of such variables.

3. **Lack of structured input datasets**: As most of the domain learning approaches are forms of data-intensive inductive learning, huge dumps of structured data in the form of plans (sequence of actions) is required to learn any meaningful models. This data is seldom available without an accurate domain model, which leads to a causality dilemma and defeats the purpose of learning domain models. Some of them even require the (partial) state information which is not always accessible for real-world domains, especially if we do not have the domain model. Transfer Learning in learning domains has been researched upon [135, 136, 142] but is still immature and has not been proven to work outside the simpler domains of International Planning Competitions (IPC)[3].

4. **Limited Reproducibility**: A significant drawback is that the bulk of the domain-learning algorithms are not open-sourced or well-maintained. Thus, their lack of community support makes them either obsolete or hard to reproduce.

Considering the prevalent issues in the automatic domain acquisition paradigm, we motivate and formulate our problem statement in the next section.

1.2. The Problem Statement

Natural Language (NL) provides the most innate way to deal with issues pertaining to lack of structured input data and explainability. Figure 1.5 describes the paradigm of learning domain models through natural language data.

In this research, we explicitly want to handle the challenges mentioned above by combining research from various disciplines to formulate an automated KE tool. Specifically, we want to leverage natural language as a solution to above-mentioned problems of explainability and lack of structured input data.

A simpler version of this learning paradigm was presented in recent research by Lindsay et al. [76] which showcased a pipeline called Framer, which generates PDDL domain models from restrictive natural language templates using Stanford CoreNLP's dependency parsing [80] and LOCM [26] domain learning algorithm. However, Framer is too restrictive to be applied in the real-world scenarios as the input is limited to certain templates of natural language.

On the other hand, extracting action sequences from natural language instructions is not a new problem. Recently, Feng et al. [33] presented an approach called EASDRL, which uses Deep Reinforcement Learning to achieve state-of-the-art results in extracting structured action sequences from *free* natural language instructional data like WikiHow cooking recipes[4].

[3] `http://www.icaps-conference.org/index.php/Main/Competitions`
[4] `https://www.wikihow.com/Category:Recipes`

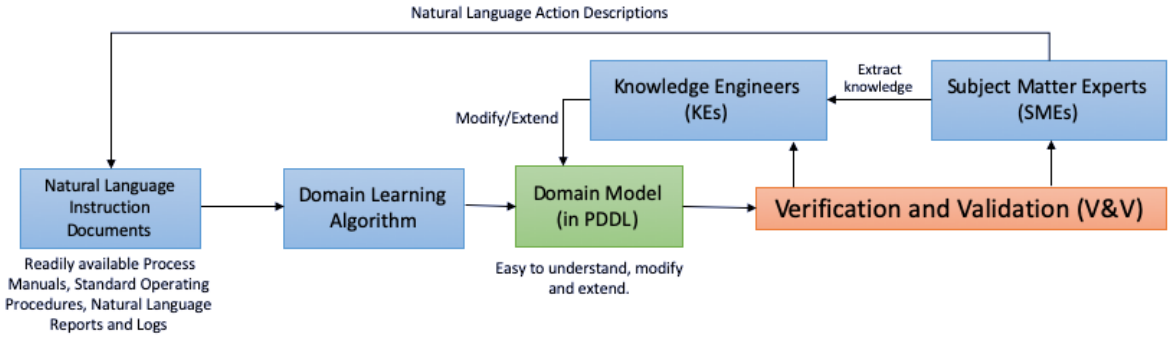


Figure 1.5: Automated Domain Model Learning from Natural Language instructional data: the paradigm takes in natural language instructions and presents an initial domain model to be used by SMEs or KEs, or as an input in model-lite planning [138, 143].

This kind of natural language instructional data is also readily available in the form of process manuals and interaction between SMEs and KEs, and is much easier to generate than the structured set of plans.

Taking inspiration from the above studies, we restrict our problem setting of learning domain models from natural language data by visualising it as two separate problems: the **action sequence extraction problem** and **the domain learning problem**. The action sequence extraction problem is the problem of extracting correct structured action sequences from natural language instructional data as defined in [33]. The domain learning problem concerns itself with automatically learning domain models from valid action sequences, as explained earlier in Subsection 1.1.2. In Chapter 2, we revisit our problem statement to formalise and illustrate it in depth using examples.

1.3. Research Objectives and Scope

The main hypothesis of the thesis is:

Can we sufficiently solve the Action Sequence Extraction Problem to extract structured data from freely written natural language process manuals of real-world problems, and then use it to solve the Domain Acquisition Problem to induce syntactically valid and meaningful PDDL models?

Overall, we lay out the following sub-research questions that will be addressed by the NLtoPDDL pipeline proposed in this thesis:

1. Can we integrate dynamic contextual word embeddings generated from pretrained language models learned from recent NLP transfer learning techniques, like BERT [29], ELMo [99], and Flair [4], into Feng et al. [33]'s EASDRL to push the state-of-the-art in Action Sequence Extraction? If yes, which contextual embeddings work the best and why?

2. Will the dynamic contextual embeddings mitigate the problems of out-of-vocabulary (OOV) words (represented by `UNK`), polysemy, shared representation, and infinite word senses caused by Word2Vec model [85] that was used in EASDRL [33]?

3. Would we be able to generalise our action sequence extraction approach on real-world data better than the current state-of-the-art, EASDRL [33]?

4. Can we use the extracted action sequences as valid structured input for domain model learning technique LOCM2 [25] and induce PDDL models?

5. Can we use NLtoPDDL as a one-shot algorithm, i.e., can it also work with only a single natural language plan to produce a best-possible valid output?

6. What is the quality of domain models learned in terms of their completeness, soundness, validity, and consistency? Are the learned domain models intuitive (meaningful) enough that they can be used as initial domain models by the SMEs and KEs?

7. Can we extend the scope of NLtoPDDL to more expressive features of PDDL, like durative actions?

Using unstructured natural language data to learn domain models makes it an even harder task than the already tough task of learning domain models from structured data. Thus, we limit our research to domains which have **deterministic effects** with **no state observability**. No state observability is a strong assumption and is rarely followed by existing domain learning algorithms. LOCM family of algorithms [25, 26] do not assume any state information and thus, we opted to use modified LOCM2 [25] in the pipeline. In terms of the expressiveness of the planning language, we support `strips` and `typing` features of PDDL2.1 [36].

Except "full state observability" assumption of what is known as **classical planning**, we follow its restrictive assumptions such as deterministic, sequentially taken, and durationless actions. We describe the full set of restrictions in Section 2.1.

Regarding input data, we used instructional documents which contain instructions about "How to do something?". Specifically, these contain a sequence of action descriptions written in a natural language. We refer to these documents as **process manuals**.

Considering our scope of research, the goal is to contribute an open-source pipeline which combines best performing techniques from the fields of NLP and AP, to learn valid and intuitive PDDL domain models from natural language instructional texts. We learn the domain models in PDDL, to allow for their direct use in off-the-shelf *domain-independent* planners.

1.4. Contributions

The contributions of this thesis are enlisted below:

- we present a novel pipeline NLtoPDDL that pushes and combines state-of-the-art from various disciplines to learn PDDL domain models from *free* natural language instructional texts (Chapter 4),

- we present a way to employ transfer-learning in NLP with deep reinforcement learning to learn generalisable models, which in turn help in transfer-learning in domain model acquisitions (Chapter 4, Chapter 5),

- and we reimplement the LOCM2 algorithm in Python and enhance it by integrating some user interaction. We open-source all our code for reproducibility[5].

1.5. Thesis Outline

In Chapter 2, we introduce the formal framework of automated planning in order to describe our domain learning problem. We then explain the action sequence extraction problem through an example. Subsequently, we review basic concepts of the deep reinforcement

[5] `https://github.com/Shivam-Miglani/contextual_drl`

learning and natural language word embeddings, which are used in NLtoPDDL.

In Chapter 3, we present an extensive literature study on the existing domain learning approaches covering their strengths and weaknesses. Later, we broadly overview model-free Reinforcement Learning and delve more into Deep Q-Network research. We then review the transfer learning in natural language, specifically, the research related to the character-based and contextual embeddings.

Chapter 4 describes our two-phased NLtoPDDL pipeline's architecture and implementation. It motivates our decisions behind choosing specific components and illustrates the working of the pipeline through numerous examples.

In Chapter 5, we experimentally evaluate both phases of our pipeline. We formulate various hypotheses and try to test them through empirical experiments. We also perform experiments to see if NLtoPDDL is flexible enough to be extended to real-world NL data and durative actions.

Chapter 6, finally concludes our work by answering our main and sub-research questions. It also discusses potential enhancements to our approach and presents an outlook of the future.

2

Background and Problem Description

In this chapter, we build the necessary terminology and background to set the stage for subsequent chapters. Section 2.1 reviews the automated planning framework that enables us to formally describe the domain learning problem in Section 2.2. In Section 2.3, we introduce the reader to a particular type of action sequence extraction problem that we opt to tackle in our research. We subsequently delve into the basics of deep reinforcement learning and natural language embeddings, which form key components of the NLtoPDDL pipeline.

2.1. Automated Planning Framework

To understand the domain learning problem, we first review the conceptual model of planning presented by Nau [92]. As shown in Figure 2.1, it consists of three important components:

1. a *state-transition system* Σ, which is a formal model of a real-world system which we want to create plans. Nau [92] define state transition system to be a four-tuple $\Sigma = (S, A, E, T)$, where

   - $S = \{s_0, s_1, s_2 \ldots\}$ is a set of **states**;
   - $A = \{a_1, a_2, a_3 \ldots\}$ is a set of **actions**, that is, the state transitions that an agent can take,
   - $E = \{e_1, e_2, e_3 \ldots.\}$ is a set of *events*, that is, the state transitions which happen because of events happening in the environment,
   - $T : S \times (A \cup E) \Rightarrow 2^S$ is a function that defines these state-transitions.

2. a *planner*, which produces the plans and policies to be executed by the controller in order to behave intelligently; and

3. a *controller* which executes the actions resulting in a change of system's state.

Also, Figure 2.1 illustrates the difference between offline and online planning. In **offline planning**, planning and execution of actions happen in a disconnected fashion, i.e., the planner receives no feedback or observations about the current state of the system. It relies on the formal domain model and the provided initial state to anticipate which state the system might occupy to generate a plan, but it does not care about the actual state of the system. When the environment is dynamic or the encoded domain model is significantly different from the environment, online planning comes into the picture. In **online planning**, the planning and acting (execution) activities are interleaved and planner needs to monitor

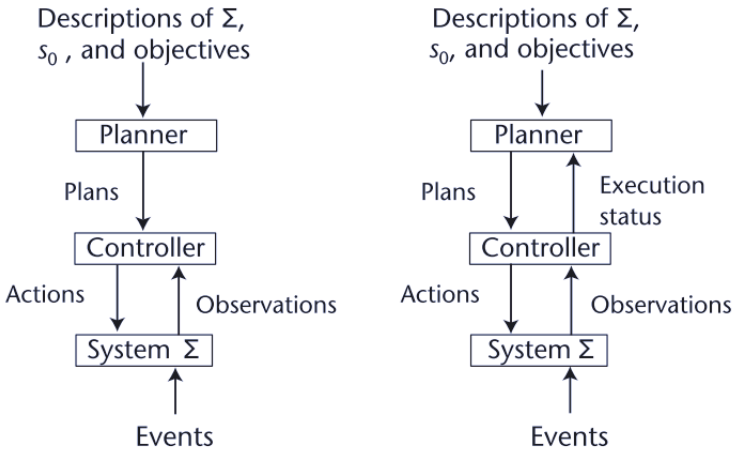


Figure 2.1: Conceptual Model for an (a) *Offline* and (b) *Online* Automated-Planning system, taken from [92]. s_o represents the initial state and objectives refer to our goal state.

the state of the system to actively revise, refine and regenerate the plans [92]. Our work concentrates on the offline planning paradigm because it is more popular and our problem statement of learning domain models fits well into it.

The reason for the popularity of offline planning is the fundamental assumption made by AP community that there should be a logical dissociation between the planner and the domain model. Due to the existence of this assumption, the field of automated planning has been dominated by research on **domain-independent planning**, which aims at building domain-independent planners that would work for all planning domains. The complexity of building such a planner that works for all problems has restricted the majority of the research to **classical planning** domains [92]. Classical planning is the simplest possible planning problem which makes the following restrictive assumptions [92]:

1. The state-transition system Σ, i.e., the domain has a finite set of states and is fully observable. There is a unique known initial state.

2. The actions are deterministic in nature.

3. The state-transition system Σ does not change, i.e., no external events are happening. This makes state system a triple $\Sigma = (S, A, T)$ or a 4-tuple $\Sigma = (S, A, T, cost)$, where cost represents a *cost function* which could mean monetary cost or time or anything else which one wants to minimise [47].

4. The plans are sequential, i.e., no concurrent actions could occur.

5. The actions are durationless, i.e., they happen instantly.

6. Planning occurs in an offline fashion.

Classical planning imposes very restrictive assumptions due to which not many real-world problems fall under this category. For example, a mars rover does not has full state observability and has both, concurrent actions and actions with duration. Thus, we do not follow the full state observability assumption of classical planning. We assume **no state observability** in NLtoPDDL, which makes it closer to real-world problems. We also try to extend it to durative actions in Section 5.2.4.

2.2. The Domain Learning Problem

If one has enough memory to form a look-up table of all the transitions in a domain, then the classical planning problem can be envisioned as a trivial problem of finding a path in a state-transition graph. However, Ghallab et al. [46] demonstrate that even for such easy classical problems, the number of states and actions could explode and the transition-graph cannot be represented in the memory. Thus, we need to encode state information into predicates and define an action model, like in PDDL, to formulate a compact representation of a problem. The problem of learning this domain model from some input data like previously executed plans is called the domain learning problem. We formally define some terminology next, which is taken from [109].

Terminology for Domain Learning

The domain model representing the state-transition system Σ can be represented as a conjunction of fluents. A **fluent** $p(arg_1, arg_2, \ldots, arg_n)$ represents a logic predicate p acting on arg_i objects of the world [109]. Each fluent has an associated value: boolean for literal fluents and numeric for numeric fluents [109]. We restrict our problem to boolean fluents, i.e., each predicate representing the state can either be true or false. The objects are often clustered by their types (`typing` requirement in PDDL2.1). Types are analogous to "data types" in functional programming.

We directly take the definitions of Planning Domain, and PDDL planning action from [109]:

"A **Planning Domain** can be defined as a two-tuple $\langle Ontology, Actions \rangle$, where $Ontology$ describes the predicates and objects of the world, and $Actions$ is a collection of PDDL actions" [109]. Here, by PDDL actions, we mean the uninstantiated PDDL actions, i.e., the action models.

"A **PDDL planning action** is a four-tuple

$$\langle Name, Parameters, Preconditions, Effects \rangle,$$

where $Name$ is the action's name or identifier, $Parameters$ are the parameters of the action, $Preconditions$ are the necessary conditions that must be met to allow for execution of the action, and $Effects$ are the changes in the state of the world after the action has been executed" [109].

A template of PDDL planning action is called an **action model**, i.e., it is the non-instantiated version of PDDL action. Following the above definitions, A **plan** is a sequence of instantiated PDDL planning actions that gives a path from an initial state s_0 to a goal state g. Finally, a **plan trace** $PT = \langle s_0, a_0, s_1, a_1, \ldots s_n, a_n, g \rangle$ is a sequence of state-action interleavings, i.e., it is a plan with state information interleaved into it. The state information present in a plan trace might be partial state information.

Ideal Domain Learning Algorithm

Based on our motivation in Chapter 1, we now describe the characteristics of our ideal domain learning algorithm:

1. *Input and Output*: Ideally, the domain modelling algorithm should work with the least amount of input data and generate the most expressive form of PDDL models. It should not demand large amounts of structured data, as it is difficult to gather. Also, the outputs should be meaningful and intuitive enough for further modifications and extensions.

2. *One-shot algorithm*: The domain learner should work even with a single instance of data (a plan or plan trace) and should reflect the best possible outcome that is representative of that data.

3. *Scope not limited to IPC problems*: Ideally, any real-world problem should be modelled by the domain learner. The approach should be easily scalable to non-IPC domains.

4. *Efficiency*: The time-consuming processes like training a neural network classifier should be done offline once. The algorithm itself should display the first results of domain to the user without much delay.

Taking into account our ideal model's characteristics, we reformulate our domain learning problem to be: **Given an input in terms of a few plans (or a single plan) and no interleaved state information, the domain learning algorithm should quickly output a valid and intuitive PDDL model, which should be easy to understand, extend or modify by the end-user.**

2.3. Action Sequence Extraction Problem

The problem of extracting a correct sequence of actions that occur in a natural language process manual document is described as Action Sequence Extraction Problem. It comprises of three aspects:

1. extract the words that represent the *actions*,

2. extract the words that represent the objects on which action is performed, i.e, the *arguments* corresponding to the extracted actions, and

3. while extracting actions and their arguments, maintain the *sequence* of actions that occur in a process manual. This aspect of maintaining the correct sequence separates action sequence extraction problem from the Information Extraction or Dependency Parsing problem present in the field of Natural Language Processing (NLP) and makes it an even harder problem than its counterparts [33]. Nonetheless, it formulates the basis of learning domain models from natural language.

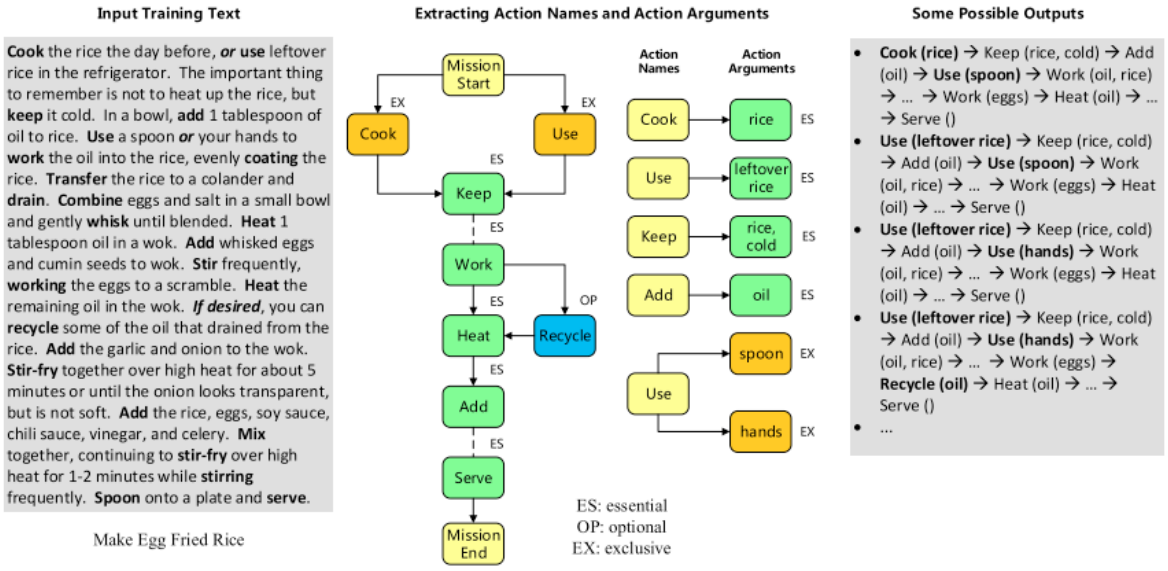


Figure 2.2: An example of Action Sequence Extraction problem, taken from [33]

Formally, we use the version of action sequence extraction problem described in [33]. To extract the correct sequence of actions, the authors explicitly consider *exclusive* (EX) and

optional (OP) actions in addition to the *essential* (ES) actions. For example, in an action description document taken from the same paper [33] represented by Figure 2.2, the first sentence “Cook the rice the day before, or use leftover rice in the refrigerator” represents an exclusive action. We either extract “cook” or “use” as actions but not both. The essential actions are represented by the words “keep”, “work”, and “heat” etc. The sentence “if desired, you can recycle some of the oil that drained from the rice” gives an example of an optional action “recycle”. As a consequence of considering exclusive and optional actions, there are multiple possible action sequences from a single process manual as shown on the right-hand side of Figure 2.2. Ideally, **we should be able to extract all correct and meaningful sequences, and use them to learn a domain model.**

2.4. Deep Reinforcement Learning

This section serves a background material for understanding Deep Q-Networks (DQNs) which used in the NLtoPDDL pipeline. Deep Reinforcement Learning as the name suggests, is a combination of Reinforcement Learning (RL) with Deep Learning (DL). Reinforcement Learning deals with the problem of an agent acting in an environment for the purpose of maximising a scalar reward. It is different from supervised learning as we do not provide direct supervision to the agent but instead, reward or punish the agent for its good or bad behaviour, respectively to make the agent learn good behaviours for reaching its goals. Figure 2.3 showcases that at each discrete time step t, the environment provides the agent with an observation s_t and a reward r_t for its performed RL action a_t. Note, that we differentiate between RL actions and actions of the planning domain model by always prefixing RL in front of reinforcement learning action.

This environment is mathematically described with a Markov Decision Process (MDP), which is a five-tuple $\langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $T(s, a, s') = P(s'|s, a)$ is the transition function which gives a conditional probability of landing in state s' at time $t+1$, if we took an action a in state s at time t, $R(s_t, a_t)$ is the reward function which gives an expected return as a function of current state and action, $\gamma \in [0, 1]$ is a discount factor [1, 58]. An **episode** $\tau = (s_0, a_0, s_1, a_1, ..)$ is defined to be a sequence of RL states and RL actions. In deep RL, MDPs are episodic with a constant γ except on the terminal states [39, 58].

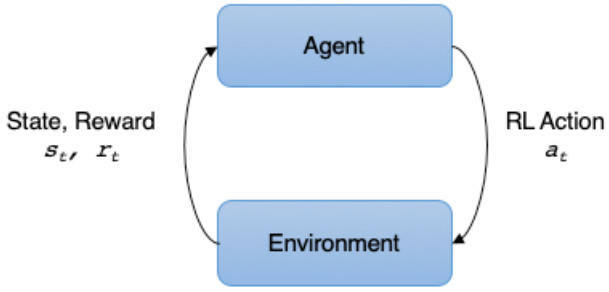


Figure 2.3: Reinforcement Learning: Agent Environment Interaction, adapted from [1]

An agent decides what action to take in a state based on a **policy** π. It could be deterministic or stochastic. In stochastic case, policy outputs a probability distribution over actions instead of an action value. In DRL, we have **parameterized policies**, whose output is a function of a neural network’s parameters (weights and biases). These parameters can be adjusted via normal deep learning optimization techniques like adaptive moment estimation (Adam) [71], and thus learn a good policy [1]. Deep learning’s ability to

approximate functions well [74] makes it possible to learn parameterized policies.

The infinite horizon discounted return is defined in [1, 118] as

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$$

It sums the discounted rewards obtained by the agent in an episode. The RL agent's goal is to *maximise the expected discounted return*.

2.4.1. Taxonomy of DRL methods

A taxonomy of DRL methods taken from [1] is presented in Figure 2.4 which differentiates between model-free and model-based RL methods. Deep RL algorithms can be divided into two model-free vs model-based methods based on their access to the environment, i.e., a function which predicts state-transitions and rewards. Further, the algorithms can be divided by the entities they learn: policy, Q-functions, value functions or environment models [1].

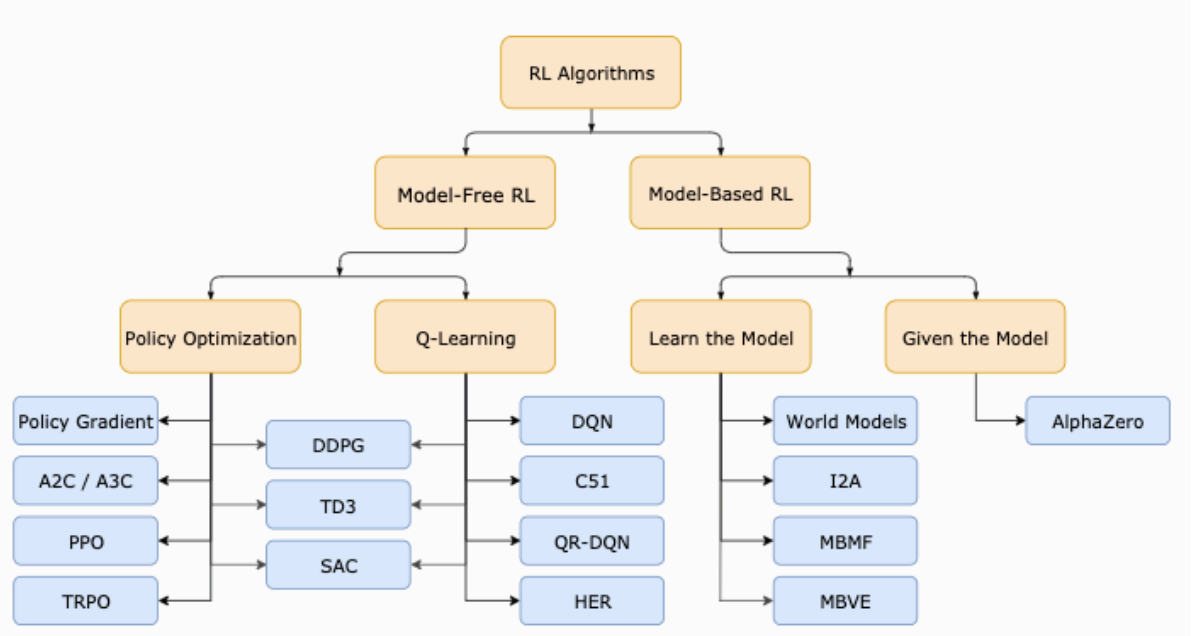


Figure 2.4: Non-exhaustive taxonomy of DRL methods, taken from [1].
Citations: Policy Gradient [119], A2C/A3C [88], PPO [107], TRPO [106], DDPG [75], TD3 [41], SAC [54], DQN [87], CS51 [13], QR-DQN [27], HER [7], Word Models [53], I2A [127], MBMF [91], MBVE [32], AlphaZero [113]. These citations are taken from [1].

As our goal was not to implement a DRL method from scratch, but only to reuse and improve an existing method EASDRL [33], we followed the taxonomy presented in [1], to non-exhaustively review the types and enhancements used in DQNs, and these are presented in Section 3.5.

2.5. Contextual Word Embeddings

Our NLtoPDDL pipeline integrates contextual word embeddings into EASDRL [33] to tackle limitations of Word2Vec [85] and to achieve better performance and generalisability. In this section, we review the key differences in the usage of contextual word embeddings like BERT [29] and a non-contextual word embeddings like Word2Vec [85].

Word embeddings, or neural word embeddings, refer to a short and dense representation of words in a low-dimensional vector space. By short, we mean that a vector is 50-500 dimensions long, which is relatively small compared to the size of the vocabulary. By dense, we imply that most of the values in the vector are non-zero (unlike one-hot encoded approaches) [67]. The usefulness of word embeddings come from their clustering of similar words in a vector space. "$king - man + woman = queen$" is a quintessential example that is overused to represent such semantic properties of word embeddings.

Famous **Word2Vec** [85, 86] models are shallow two-layer neural networks that process text and convert it into such word embeddings. As explained above, it groups similar words in the vector space based on their past appearances in the corpus. Figure 2.5 describes the two ways from which Word2Vec achieves this. The *continuous bag of words (CBOW)* method uses the context of nearby words to predict the target word [85]. On the other hand, the *skip-gram* method uses the word to predict a target context [85]. This task of understanding language by training to predict the target word or predicting the context is called **Language Modeling**.

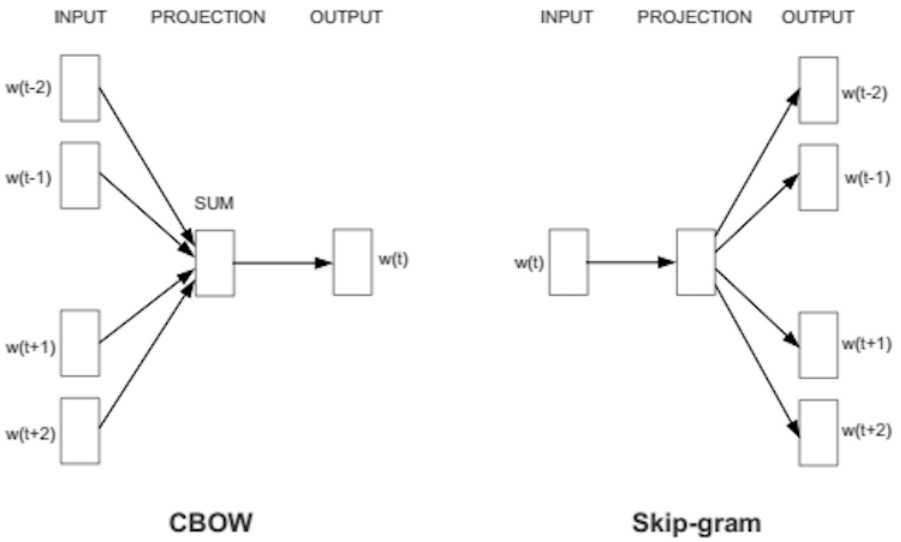


Figure 2.5: CBOW and skip-gram methods to learn Word2Vec embeddings, taken from [85]

The Word2Vec model averages all the contexts of the word that it saw, into a single dense vector. As a result, Word2Vec doesn't address the problem of **polysemy**, which means that a word has different meanings in different contexts and the meanings can co-exist. Another problem with Word2Vec is out-of-vocabulary (OOV) words. Vectors of each word in vocabulary are stored into a dictionary data structure, and thus, Word2Vec embeddings are **static** in nature and only work for the words present in vocabulary.

Contextual word embeddings solve for the issue of polysemy by including information about the context (preceding and succeeding words) into the vector representation of a particular instance of the word. These embeddings are computed **dynamically** at the run-time by passing as input the sentence in which the instance of the word occurs. Although it is computationally expensive than just looking into a dictionary, the results obtained on various NLP tasks are much better [4, 29, 99]. The OOV words issue is solved by learning character-level or sub-word level embeddings. This helps in computing vectors for misspelt, rare or internet slang words. A simple way to learn character-level embeddings is to decompose a word into character n-grams and then use the skip-gram model from a large corpus of data [15]. A sophisticated way would be to represent character n-grams by a

convolutional neural network (CNN) and a highway network and then pass it to LSTM to learn a language model [70], which is used in ELMo [99].

BERT [29] gains its language understanding and nuances on a sub-word level from semi-supervised pre-trained language model. Just applying this language model to a downstream task by using a task-specific architecture is called **feature-based** strategy. The fine-tuning of parameters of pre-trained language model without introducing task-specific parameters is called **fine-tuning** [29]. Figure 2.6 [6] illustrated the process of language modelling and fine-tuning in BERT.

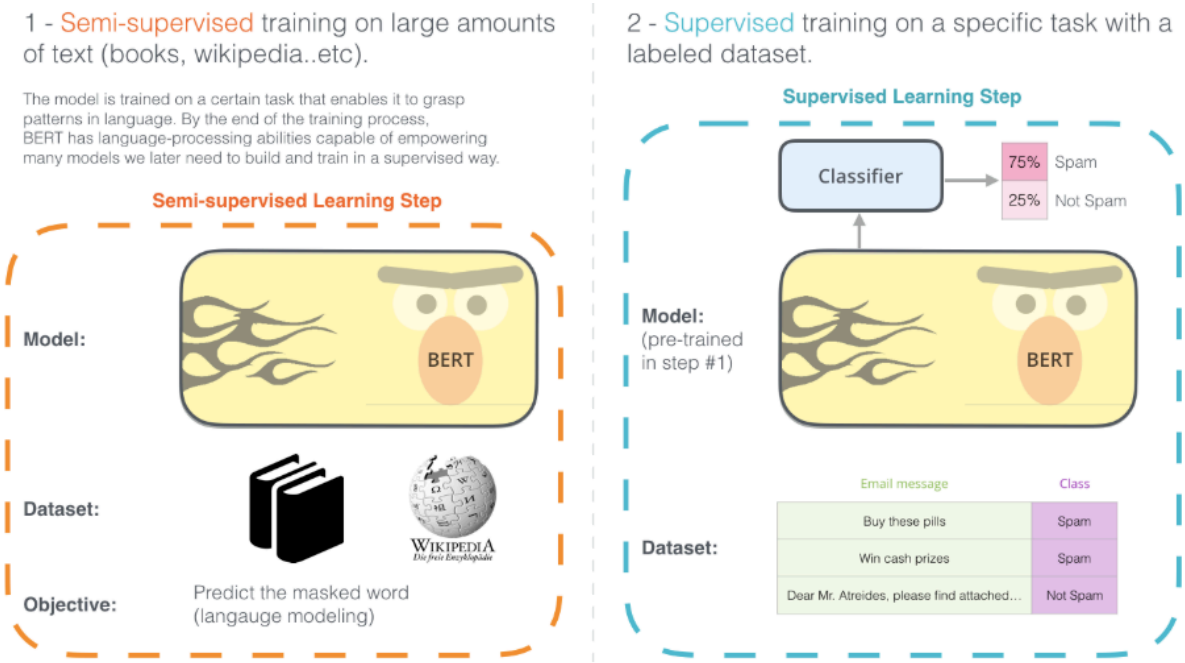


Figure 2.6: A semi-supervised training is performed on large amounts of text (wikipedia, news dataset etc.) to build a language model. The language model is fine-tuned for a particular task such as classification of spam and non-spam emails, taken from [6]

Technically, using the same universal language model and fine-tuning it to task is referred to as transfer learning in NLP. However, **we use feature-based strategy as we already had the task specific architecture of EASDRL [33] to extract action sequences.** This inherently does some transfer learning as we have learned a lot of information from pretrained language models and we are applying that information in our downstream task (not related to the pretrained models). Section 3.6 reviews the literature of contextual embeddings in NLP.

3

Literature Review

In this chapter, we review the relevant literature associated with our action sequence extraction and domain learning problems. We first present various ways to categorise domain learning algorithms, then in Section 3.2 present an extensive review of existing domain learning approaches. In Section 3.3, we discuss the existing planning languages and the versions of PDDL.

In Section 3.4, we review various action sequence extraction approaches and also, the type of problems in NLP to which action sequence extraction belongs. Thereafter, we switch our focus on model-free Deep reinforcement Learning and specifically to Deep Q-Networks. In the end, we review the research related to transfer learning in NLP, which is deemed as the "ImageNet" moment (inflection point) in NLP research.

3.1. Taxonomies of Domain Learning Algorithms

The domain learning approaches can be divided based upon many factors such as the kind of input provided, the learning technique they employ, and the kind of problem they tackle. Arora et al. [8] present a guide to choose a domain learning algorithm based on such factors, which is illustrated in Figure 3.1. We explain each of the factors in the following subsections.

Based on the Environment. The characteristics of the environment in which a planning agent acts has led to various groups of domain learning systems. Jiménez et al. [65] categorised them into four groups:

1. **Deterministic** effects, **Full** state observability (D, FO): These are the classical planning problems that we discussed in Chapter 2, which are too restrictive in their assumptions to represent any interesting real-world problems. We can use plan traces (state-action interleavings) in this scenario, to learn the domains. The learning complexity is theoretically bounded [65].

2. **Deterministic** effects, **Partial** state observability (D, PO): These learning methods consider partial or no state observability. LOCM [25, 26, 52] is a family of methods that assume no state observability.

3. **Probabilistic** effects, **Full** state observability (P, FO): This scenario of full observability but stochastic action effects corresponds to probabilistic planning task. Probabilistic PDDL [132] is the representational language for such tasks.

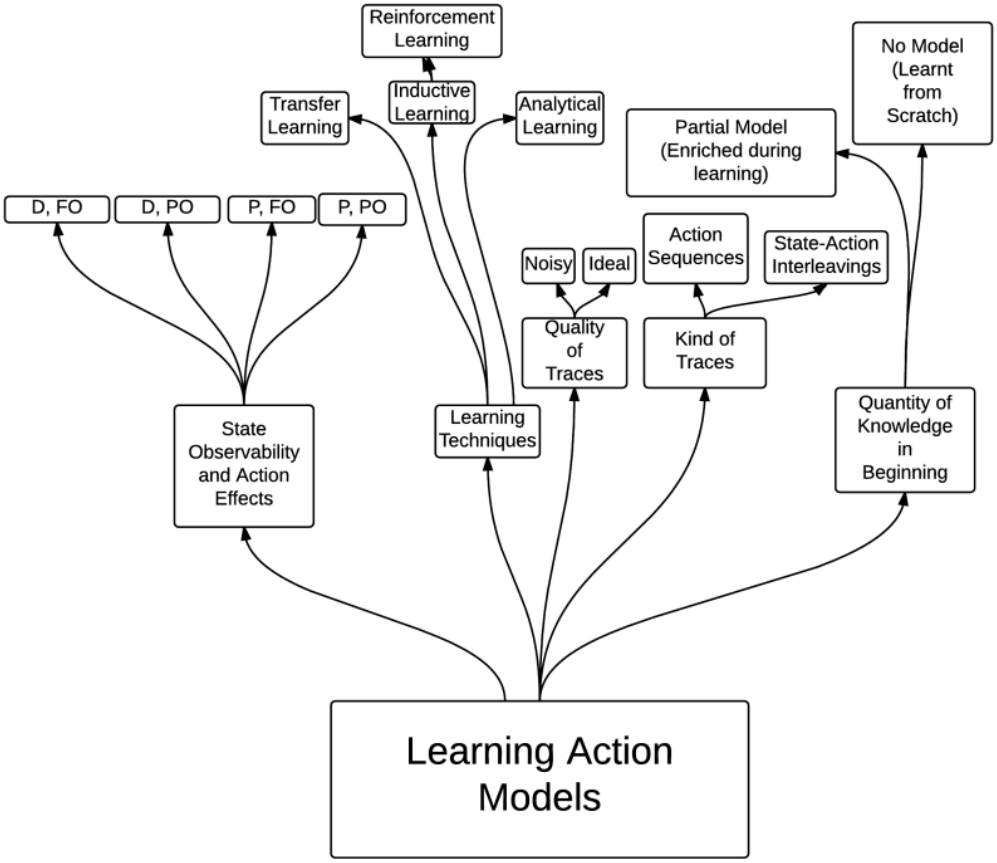


Figure 3.1: A taxonomy of learning planning domains models, taken from [10]. D=Deterministic, P=Probabilistic, FO=Fully Obersvable, PO=Partially Observable.

4. **Probabilistic** effects, **Partial** state observability (P, PO): In case of partial/incorrect state observability and stochastic effects, not many studies have been done for the modelling task. However, an important area of work called **model-lite planning** [131, 138, 143] has emerged which tries to search for not only the perfect plans but also most plausible solution plans[65].

We consider only D,PO environment with no noise in our implemented methods.

Based on the Input and Output. The domain learning algorithms can be categorised on the basis of input they take, and the output they produce. The inputs can be plans (sequence of actions), plan traces (state-action interleavings) and partial domain models. The state-information in plan traces varies from partial state information to no state information. The input plans or plan traces may be noisy or incorrect as well. Learning methods can also be categorised based on the output they generate. The output could be partial models, STRIPS [34] model, or a subset of PDDL [36, 45]. Our work concentrates on plans as an input, which means, that there is no state observability, and output as PDDL2.1 models supporting `strips, typing` and `durative-actions` requirements. We assume that the input is valid and there is no noise.

Based on the Learning Technique. Finally, the domain learning algorithms can be categorised based on the learning technique they employ. For example, inductive learning, and maximum satisfiability are some of the popular techniques employed in domain learning

approaches. We discuss these along with their implementations in the next section.

3.2. Existing Domain Learning Approaches

Chapter 2 described a specific type of domain learning problem. However, the area of domain learning is much broader and thus, we define generalised domain learning problem as:

Given an input in the form plans, plan traces or a partial domain model, learn a domain model containing an action model, which describes the applicable actions, their conditions for applicability and their effects, which best describes the input data.

We categorise domain learning methods into the following types of learning and discuss their strengths and weaknesses alongside their implementations. Most of these types are taken from a survey of Arora et al. [8], but we update it with the most recent methods as well.

1. **Inductive Learning**: This the most popular form of learning domain models. The task is to find the best hypothesis that satisfies a given set of traces [8]. Most inductive approaches are data-intensive and require huge dumps of structured data to learn planning domain models. This can further be divided into various types:

    - **LOCM family of algorithms**: Learning Object-Centric Models or LOCM uses object-centric notation and assumes that objects in a domain belong to certain classes and their behaviour can be modelled by finite state machines[26]. On the basis of valid input plans, it inductively learns the state machines and parameterises them with object associations. LOCM2 generalises LOCM by allowing for an object to have multiple behaviours through multiple finite state machines (FSMs) per class [26]. LOP system induces the static conditions which were not captured by LOCM and LOCM2 [50]. NLOCM [51] extends the scope of LOCM methods from classical planning to numeric planning. It uses the finite state automata generated from LOCM to learn action costs of object transitions and state parameters [51]. LC_M [52] is a variant of LOCM which deals with missing and noisy information by segregating plans into valid and invalid parts and applying LOCM[26] to them. Finally, Framer [76] and StoryFramer [55] are two approaches that use LOCM to learn domain models from natural language action descriptions. We use LOCM1 and LOCM2 in our thesis.
    - **Regression trees**: Planning, Execution and Learning Architecture or PELA [64], which does all three tasks described in its name, updates STRIPS [34] action model with probabilities learned from plan execution. The probabilities are learned by inducing first-order decision trees [65]. Thus, PELA works with probabilistic effects using regression trees.
    - **Mixed-Initiative**: Opmaker2, based on Opmaker [114] take in a partial domain model and a few hand-crafted training examples to learn the domain models [82]. Recently, Li and Zhuo [73] presented an integrated development environment which asks the user to create a graphical model, does consistency detection using MAX-SAT and weighted MAX-SAT over a knowledge base and applies domain learning algorithm called AMAN [137] to learn the domain models. If the modelling is incorrect, the end-user corrects the generated plans and then KAVI learns the domain model again.

2. **Analytical Learning**: In analytical learning, extra background knowledge is available to draw inferences from. Zimmerman and Kambhampati [145] present a survey of analytical approaches which shows how techniques such as memoization, explanation-based learning, and statistical analysis can help in domain model acquisition.

3. **Genetic and Evolutionary algorithm-based approaches**: Genetic algorithms have been applied in model-learning approaches. Colledanchise et al. [24] learn behaviour trees instead of restrictive FSMs to learn behaviours of an autonomous using genetic algorithms. In an a priori unknown dynamic environment, using rewards and binary observations, the agent uses genetic programming to learn a switching structure in the form of behaviour tree [24]. Another approach called LOUGA (Learning planning operators using genetic algorithms) used ‘a genetic algorithm to learn action effects and an ad-hoc environment to learn action preconditions [72].

4. **Uncertainty-based approaches**: Uncertainty-bases approaches use Markov Logic Network (MLN) [102], which is a combination of first order logic (which handles uncertainty) and probabilistic graphical model (which models a markov network) [8]. Learning Actions from Plan Traces (LAMP) [140] is a domain learning method that uses MLNs to model complex action models with logical implications and quantifiers.

5. **Noisy plan trace dealing approaches**: Action-Model Acquisition from Noisy Plan Traces (AMAN) makes a graphical model to capture relations between actions and states [137]. PlanMiner O2 [109] is another algorithm that handles noise and learns numerical action models. It uses a classification algorithm called “NSLV genetic algorithm” to learn the pre and post-states of actions. On the basis of results mentioned, the models of IPC problems that learned are robust to noise LC_M [52] is a variant of LOCM, which deals with missing and noisy information by segregating plans into valid and invalid parts and applying LOCM[26] on them. Probabilistic planning operators are learned from noisy traces in [89]. The input to the learning mechanism uses a vector representation that encodes a description of the action being performed and the state at which the action is applied. Then a transition function is learned between states in the form of a set of classifiers. Using classifier’s parameters a STRIPS model is learned [89].

6. **Transfer Learning**: An example of transfer learning-based domain learning approach is TRAMP (Action-model acquisition for planning via transfer learning) [139]. It assumes that action models in source domains are already created by SMEs and can be transferred to the target domain. TRAMP uses Markov Logic Networks (MLN) for selecting most likely subsets of candidate formulas, an idea previously researched in LAMP [140]. t-LAMP [142] is a similar approach which uses web search to bridge the knowledge gap between the source and target domain.

7. **MAX-SAT based approaches**: Classical Planning is reduced into a (weighted) MAX-SAT problem and MAX-SAT solvers are used to solve it. ARMS (Action-Relation Modeling System) [129] learns action models from state-action interleavings with partial state information. It forms information and action constraints that are followed by the predicates and pattern mines frequent action pairs to learn action models [129][8]. A transfer learning approach LAWS [136] builds a weighted MAX-SAT to find similarity between source models and target models using web search. Learning Models for multi-agent environments (LAMMAS) extends ARMS to multia-agent setting [141][8].

8. **Deep Learning-based approaches:** Deep Learning has recently being employed as domain learning approaches. Classical planning is done in deep latent space in [11], a state auto-encoder which uses unlabelled training image pairs representing correct states to encode correct behaviour, and then at the testing time takes in a pair of initial and final state images to find a plan using variational autoencoder using Gumbel-Softmax activation.

9. **Model-Lite approaches**: [68, 131] apply probabilistic logic to infer incomplete and dynamic domain models. They represent planning problem as the most plausible explanation (MPE) problem and reduce it to MAX-SAT to use MAX-SAT solvers. A library of plan cases is used augment the extracted plans from an incomplete model in [143]. A case-based (Model-Lite Case Based Planning) and a model-based (Refining Incomplete Models [144]) approach are presented and compared in [138].

10. **Classical Planning approaches**: Recently, [2] used classical planning with conditional effects to build up generative models representing STRIPS action models from less amount of plan traces. In Aineto et al. [2], the authors stress test their approach to do the learning in one-shot, i.e., from one plan trace.

A summary of important existing domain learning approaches is shown in Table 3.1

3.3. The Planning Languages

In this section, we review the existing planning languages and subsequently, delve into the versions of PDDL which has become de facto standard for modelling domains through various International Planning Competitions. A planning language is a representation language to encode domain models. An overview of representation languages in Automated Planning is shown in Table 3.2. PDDL is the standard language to be used in the domain-independent planners. Each iteration of PDDL made it more expressive and useful. There have been many adaptions of PDDL to make it more suitable for the task at hand. For example, New Domain Definition Language (NDDL) was developed by NASA to use it in EUROPA2 planning system [14]. NDDL replaces states and actions with timelines and constraints between those timelines, which makes it much more practical for search control [14]. Another example is HTN-PDDL [49], which uses a `task` element to introduce hierarchical tasks. The most recent version of PDDL is PDDL 3.1 [42], which has introduced functions and object fluents. We only support strips, typing and durative-actions subset of PDDL2.1 [36] version in our thesis.

3.4. Existing Action Sequence Extraction Methods

Extracting procedural knowledge or a sequence from a natural language in machine-interpretable format has been a popular problem in the field of Natural Language Processing (NLP). It is similar to translating a natural language to structured machine language, and just like successful machine translation methods, it comes under the category of sequence-to-sequence (seq2seq) problems [117]. The sequence-to-sequence models are based on encoder-decoder architecture, possibly with an inclusion of attention. The encoder is a neural network architecture that understands the input sequence, and creates a compact representation, i.e., encodes the input in less dimensions. Afterwards, the encoder passes on this representation to the decoder which is a separate network that generates a sequence representing the output.

Regarding action sequence extraction problem, most of the work in this direction has been performed in mapping navigational route instructions to a predefined set of actions, to allow autonomous agents and robots to follow the natural language instructions. Earlier

Existing Domain Learning Algorithms					
Algorithms	**Input**	**Output**	**Technique**	**Environment**	**Noise robustness**
Inductive learning-based approaches					
PlanMiner O2 [109]	State-Action Interleavings	PDDL (Action Model)	Classification based on genetic algorithm	D,PO	Y
LOCM [26]	Action sequences (no state info)	PDDL	Inductive Learning on Finite State Machines	D,PO	N
LOCM2 [25]	Action sequences (no state info)	PDDL	Inductive Learning on Finite State Machines	D,PO	N
NLOCM [51]	Action sequences (no state info)	PDDL	Inductive Logic Programming	D,FO	N
LC_M [52]	Action sequences (no state info)	PDDL	Inductive Learning on Finite State Machines		Y
OpMaker [114]	Partial model + action sequences	Operators/OCL	Operator induction by mixed-initiative	D,PO	Y
OpMaker2 [82]	Partial model + action sequences (OCL)	Operator/OCL. Domain model and heuristics	Computes intermediate states using heuristics, inference from PDM, training tasks and solutions	D,PO	Y
PELA [64]	Initial and goal State + PDM + Action Sequences	Enriched action models/PDDL	Top-down induction of decision trees	P,PO	N
Genetic Algorithm-based approaches					
Newton et al. [93][8]	Domain and example problems	Macros/PDDL	Genetic algorithm	D,FO	N
Colledanchise et al. [24]	Actions, reward and boolean sensing	Behavior Trees	Genetic algorithm learning Behavior trees	D, FO	N
LOUGA [72]	Valid Action Sequence	STRIPS	Genetic algorithm	D, FO	N
Uncertainty-based approaches					
AMAN [137]	Action sequence	Operators/STRIPS	PGM, reinforcement learning	D,PO	Y
LAMP [140]	State-action interleavings	Action models/PDDL	Markov logic network	D,FO	N
Pasula et al. [96][8]	State-action interleavings	Probabilistic STRIPS like	Parameter estimation.	P,FO	N
Transfer learning-based approaches					
t-LAMP [142]	State-action interleavings	Action models/PDDL	Markov logic network + Web Search	D, FO	N
TRAMP [139]	Action schemas, predicate set, few plan traces from target domain. Set of action models from source domain.	Action model in target domain. (STRIPS)	Transfer Learning	D,FO	N
MAX-SAT based approaches					
ARMS [129]	Action sequence (partial/no state information)	Operators/STRIPS	MAX-SAT problem	D,FO	N
LAWS [136]	Action schemas, predicate set, few plan traces from target domain. Set of action models from source domain.	Action models in target domain (STRIPS)	Transfer Learning + KL divergence	D,FO	N
Lammas [141][8]	Action sequences	Operators/MA-STRIPS	MAX-SAT solver	D,FO	N
Supervised learning-based approaches					
LSO-NIO [89]	PDM, Noisy Plans	Operators/PPDDL	Kernel Perceptrons + sequential covering	P,FO	Y
Deep learning-based approaches					
LatPlan [11]	State Auto-Encoder + Unlabelled training image pairs for allowed actions. Pair of intial and goal state images.	PDDL	Variational Autoencoder + Gumbel-Softmax activation	D,FO	N
PDeepLearn [9]	State-Action interleavings	PDDL	Exhaustive model generation + Pattern mining and LSTM.	D,FO	N
Classical Planning-based approaches					
Bandres et al. [12]	Visible screen pixel features	Atari Game models	Classical Planning	D,FO	N
Aineto et al. [2]	Plan Sequences and/or initial and goal state and/or partial domain model	STRIPS	Synthesis of generative models using Classical Planning with conditional effects	D,FO	N
Aineto et al. [3]	One Plan Sequence and/or initial and goal state and/or partial domain model	STRIPS	Synthesis of generative models using Classical Planning with conditional effects.	D,FO	N

Table 3.1: Existing Domain Learning Approaches, adapted and updated from [8]

Languages	Features
PDDL [83]	Standardized syntax for STRIPS (types, constants, predicates, and actions.)
PDDL+ [35]	Models continuous time-dependent effects.
PDDL2.1 [30, 36]	Extension of PDDL to numeric fluents and temporal planning
PDDL3.0 [43, 44]	Introduced hard and soft constraints for preference-based planning.
PDDL3.1 [42]	Introduced functions and object fluents.
PPDDL [132]	Extension of PDDL2.1 for probabilistic planners. Supports probabilistic effects.
HTN-PDDL [49]	Extension of PDDL for hierarchical task networks, uses “task” for compound tasks
STRIPS [34]	Sublanguage of PDDL. Unknown literals are false (closed-world)
OCL [81, 82]	Object-Centered representation, inspired by Object Oriented Programming
ANML [115]	Used by NASA in space missions. Combines best aspects of PDDL and NDDL.
RDDL [104]	STRIPS + functional terms, leading to higher expressiveness.
ADL [8]	An extension of STRIPS to include quantifiers and negative conditions.
NDDL [14]	Intervals and constraints between those intervals as states and actions.

Table 3.2: Planning representation languages: PDDL is the de facto standard for classical planning and has been used in many International Planning Competitions (IPCs). ANML = Action Notation Modeling Language, PDDL = Planning Domain Definition Language. PPDDL = Probabilistic PDDL. RDDL = Relational Dynamic Influence Diagram. ADL = Action Description Language. NDDL = New Domain Definition Language. HTN-PDDL = Hierarchical Task Network-PDDL

approaches [20, 79] used semantic parsers, context-grammars, and statistics of corpus to extract actions from natural language data and map them to instruction-set [84]. In a famous paper of ”listen, attend and walk”, Mei et al. [84] used an encoder-decoder architecture to map “free” natural language instructions to an executable action sequence. Both encoder and decoder structures were long short-term memory (LSTM) [59] RNNs. Although it performs well for single sentences, the weakness of this approach is that it cannot handle multi-sentence texts well. Despite the success of these approaches, they all require a finite set of actions as input, as they essentially solve the mapping or a sequence labelling problem instead of extraction problem [33]. Feng et al. [33] defined a version of action sequence extraction problem in which they exclusive tackle exclusive “or” instructions. Either one of the instructions is extracted and not both which maintains the sequence of actions. Keeping the action sequence intact is important for domain learning approaches. Feng et al. [33] do this action sequence extraction by using a Deep Q-Network to learn action sequences from *free* natural language data, without the need of any mapping action set. In an unrelated research, it is demonstrated that using DRL instead of encoder-decoder architectures reduces the exposure bias and input-output mismatch problems, which are prevalent in NLP [69]. We use Feng et al. [33]’s version of problem definition and DRL model in our research and take it a step further by incorporating state-of-the-art contextual embeddings like ELMo [99], BERT [29] and Flair [4] into it. As we will see later in Chapter 5, this pushes the state-of-the-art further, and even helps in resolving problems of using static embedding like Word2Vec [85].

There also has been some research on learning planning domain models directly from natural language data. Goldwasser and Roth [48] use natural language instructions to learn partial game dynamics and map it onto the structural actions [48]. Framer [76] extracts verbs (action names) and objects (action arguments) in a sentence using Stanford CoreNLP [80] library and clusters them into action templates. The type of input data used is in the form of restricted templates. They later build domain models using LOCM [26]. StoryFramer [55] uses natural language stories to generate domain models in a mixed-initiative fashion. The user selects the correct actions templates for LOCM in StoryFramer. In this research, we extended the Framer’s [76] architecture to work with “free” language data.

3.5. Deep Q-Networks and their variants

Deep Q-Networks (DQNs) combine Q-learning [126] with deep neural networks. Mnih et al. [87] presented for the first a reinforcement learning approach combined with convolutional neural network with a variant of Q learning to get control policies from pixel level input [87]. This was important because large state spaces made it intractable to learn Q value estimates in original Q-learning from a lookup table. DQN changed this by representing values $q(s, a)$ with deep neural networks [58]. However, there were similar limitations in DQN which are tackled by its extensions. Double Q-learning [123] showed that the original DQN overestimated action values under certain conditions, and used two value functions one of which determined how to select actions and the other to determine its value. Prioritised experience-replay was shown select important transitions and replays them more frequently in the network [105]. In EASDRL [33], which we use in our research uses both double Q-learning by building two Q-networks (base and target), and prioritised experience-replay by selecting positive experiences more [33].

3.6. Contextual Embeddings: Natural Language Transfer Learning

The rise of contextual embeddings and universal language models has marked a new era in NLP. These embeddings are trained in an unsupervised or semi-supervised fashion on a huge corpus of data such as books corpus[1], and then used on downstream task on a feature-based or fine-tuning based approach [29]. Embeddings from Language Model (ELMo)[99] learns a bidirectional Language Model (biLM) by training on huge corpora in an unsupervised fashion. ELMo can be used as a feature-based approach. Flair embeddings [4] are similar to ELMo embeddings which train the biLM a character-level with no notion about words. Cross-View training (CVT) [23] combines two separate training stages that ELMo needs for pre-training and task-specific training into a unified semi-supervised procedure. The biLSTM encoder improves its encoding using both labeled and unlabelled data [128]. ULMFiT [62], which stands for Universal Language Modelling and Fine Tuning was the first paper which looked into using a pretrained language model with task-specfic fine-tuning. The techniques introduced in ULMFiT which made task-specfic fine-tuning possible were discriminative fine-tuning (tuning different layers of LM) and Slanted triangular learning rates (the learning rate increases for short time, then decreases) [62, 128]. OpenAI-GPT [100], which is short for Generative Pre-training transformer used a transformer (an attention based encoder-decoder neural net [125]) decoder to train on a giant collection of data. On the other hand, BERT uses a stack of transformer encoders [29]. BERT beats OpenAI-GPT by a novel idea of masked input which encouraged bi-directional prediction and sentence-level understanding [128]. XLNet [130] and OpenAI-GPT2 [101] are other notable transformer based methods. We used ELMo, Flair and BERT embeddings in our research.

[1] https://googlebooks.byu.edu/

4

NLtoPDDL

The formal architecture of NLtoPDDL pipeline is shown in Figure 4.1. It is divided into two phases which are based on the type of data they use:

1. **Training the DRL model with annotated training data**: In Phase 1, a Deep Q-Network (DQN) based on the architecture presented in [33] trains on annotated datasets. The DQN learns to extract words that represent action names, action arguments, and the sequence of actions in the natural language process manual. An important thing to note here is that instead of using static `Word2Vec` embeddings [85], we incorporate dynamic contextual embeddings like BERT [29], ELMo [99] and Flair [4] into the Feng et al. [33]'s model.

2. **Learning the domain model with unseen test data**: In Phase 2, we learn the domain model of the domain represented by unseen process manuals. First, we extract the action sequences by feeding our unseen test data to our trained Deep Q-Network of Phase 1. Second, we preprocess the extracted action sequences to make them suitable for assumptions made by domain learning algorithm LOCM2 [26]. Lastly, the preprocessed extracted action sequences are then used by LOCM2 to learn the PDDL domain model.

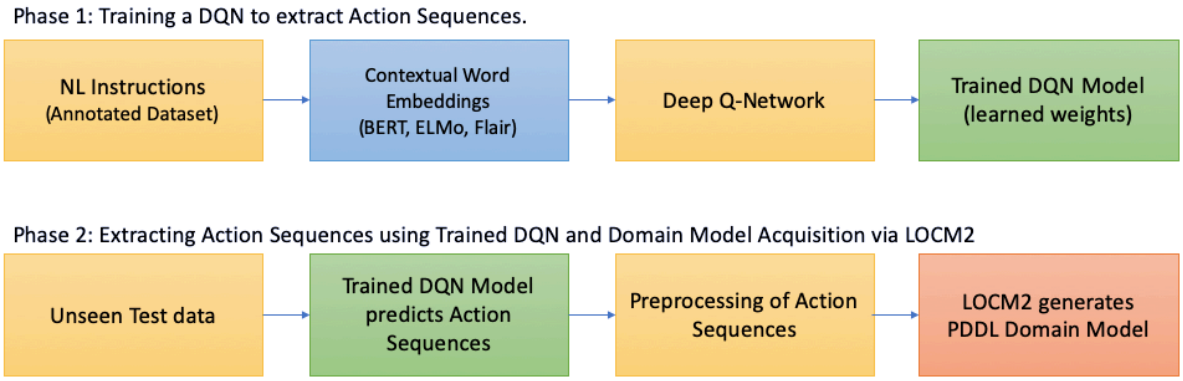


Figure 4.1: Formal pipeline architecture to learn PDDL domain models from natural language instructions. It is formed by combining from [76], [4] and [33]

In this chapter, we look at both the phases of NLtoPDDL in detail and also highlight the decision-making process behind the selection of specific components and their modifications.

4.1. Training the Deep Q-Network

Feng et al. [33] presented a method named **E**xtracting **A**ction **S**equences from texts using **D**eep **R**einforcement **L**earning (EASDRL), which uses Deep Q-networks to extract action sequences out of natural language instructional data. The reason for using this architecture is four-fold:

1. EASDRL produces state-of-the-art results for extracting correct sequence order from *free* natural language text. This means that the user doesn't need to adhere to any restrictions while expressing the actions in a natural language [33].
2. EASDRL doesn't require a prior action set to be specified [33]. This makes it suitable for learning domain models from scratch as we do not know *a priori* the set of actions.
3. DQNs, along with their improvements like prioritized-replay and double q-learning work well with discrete action spaces, and have proven to be sample efficient and achieve state-of-the-art performance on Atari 2600 benchmarks among other domain-free DRL algorithms [38, 58]

4.1.1. Training Dataset

Feng et al. [33] define the training data as a sequence $\langle words, annotations \rangle$ pairs, where $words = \langle w_1, w_2, \ldots, w_N \rangle$ is the sequence of words in a process manual, and $annotations = \langle y_1, y_2, \ldots y_N \rangle$ is the corresponding sequence of annotations. If w_i is not an action name, the corresponding y_i is ϕ. If a word w_i is an action name, the corresponding y_i can be defined by Feng et al. [33] as a triple:

$$(ActType, \{ExActId\}, \{\langle ArgId, ExArgId \rangle\})$$

Here, $ActType$ tells the type of action the word is: essential (ES), optional(OP) or exclusive(EX), as defined previously in Section 2.3. $\{ExActId\}$ is a list of word indexes that represent a list of exclusive actions that are alternatives to the action in consideration. $\{\langle ArgId, ExArgId \rangle\}$ represents the list of sequences of arguments and their alternatives for the action in consideration. Figure 4.2 illustrates an example of such a $\langle words, annotations \rangle$ pair. In the example, "Hang" and "opt" are the exclusive actions, "engraving" and "lithograph" are exclusive arguments for the action "hang". Since "frame" has no exclusive argument, it is written as a tuple $\langle 9, \rangle$ with no second value. The annotations y_i for non-action words is ϕ.

4.1.2. Sequence Extraction as Reinforcement Learning Problem

Feng et al. [33] represent the action sequence extraction problem as a reinforcement learning problem. This is done because of arbitrary long length of complex annotations, which makes it hard to formulate a supervised learning setting. The allowed RL actions in EASDRL Feng et al. [33] are "select" and "reject". The RL states are a concatenation of word embedding and the RL action performed. This was done to distinguish between two states as the RL agent can either select or reject a word to indicate whether the word is an action name (or an action argument) or not. An example of the RL state and RL action is shown in Figure 4.3. The RL action NULL in the figure is a default action used to represent an unparsed sequence. Note that we distinguish between RL action and the action of the domain model by always prefixing RL to the reinforcement learning action.

Feng et al. [33] solve the action sequence extraction problem by using two DQNs. The first DQN learns to extract the sequence of action names $actions = \langle a_1, a_2, \ldots, a_N \rangle$. Its model can be represented by:

$$\mathcal{F}^1(actions|words; \theta_1)$$

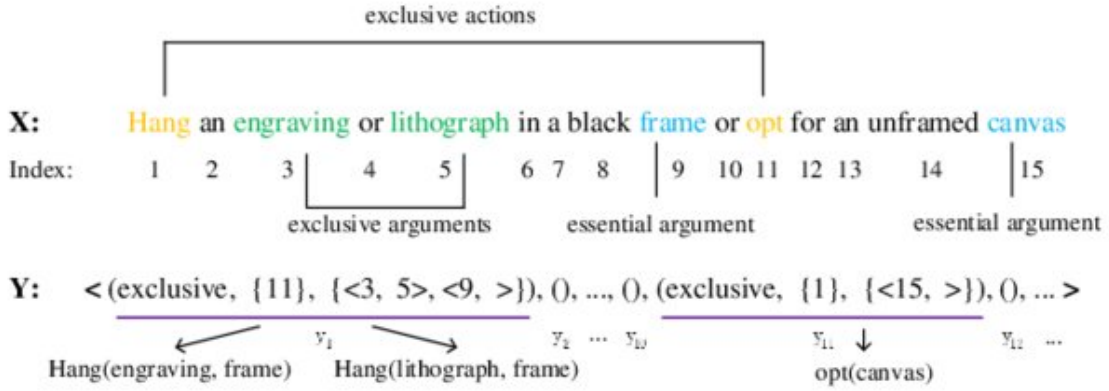


Figure 4.2: An illustration of $\langle words, annotations \rangle$ pair of the training datasets, taken from [33]

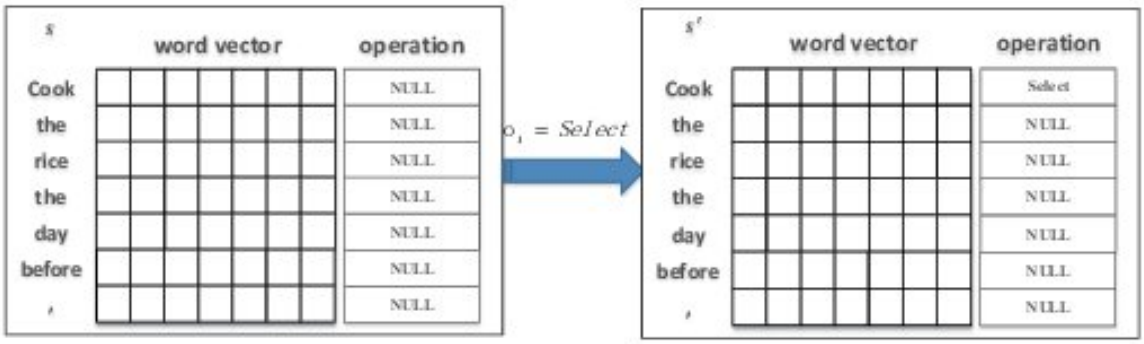


Figure 4.3: A schematic example of RL states and RL actions (denoted as operation here), taken from [33]

Then the action names are used to train the second DQN which has the same framework as one to extract the arguments of actions.

$$\mathcal{F}^2(actions|words; \theta_2)$$

Here, θ_1 and θ_2 are the parameters that the DQN learns for predicting action names and arguments [33]. Just like in [33], $\mathcal{F}^2$ is trained using ground-truth action labels, and while testing the extracted action names from $\mathcal{F}^2$ are used.

4.1.3. Repeat Representation in States

Feng et al. [33] use a repeat representation while representing states. For action DQN, the RL state is represented by the word vector of d dimensions and the RL action is one of $\{0, 1, 2\}$, which corresponds to $\{NULL, Select, Reject\}$, but is repeated d times. Similarly, in argument DQN, the state is represented by a triple $\langle word\_vector, distance, RL\ action \rangle$, where the distance is the distance between the word in consideration (w_j) and the

Schematic representation of states in an argument DQN is shown in Figure 4.4. We can infer that repeat representation is not scalable. For Word2Vec embeddings of 100 dimensions, the state representation is 300 dimensional long, but for BERT embeddings of 3072 dimensions, one state is represented by 9216 dimensions. This causes memory issues in training even with smaller batch sizes. We explain what we tried to resolve this in the next

section.

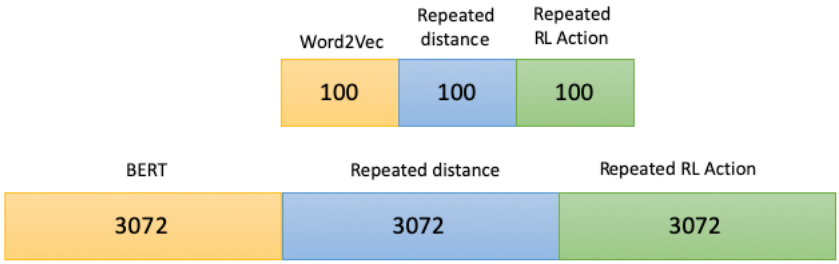


Figure 4.4: A schematic illustration of repeat representation for 100-dimensional Word2Vec embedding and 3072-dimensional BERT embedding.

4.1.4. Deep Q-Network for RL Action execution

The two Q-networks construct an action sequence by repeatedly accepting or rejecting the words on current states to achieve new states. The Q-function is iteratively updated using the following two Bellman equations, taken from [33]:

$$Q_{i+1}(s, a; \theta_1) = E\{r + \gamma \max_{a'} Q_i(s', a'; \theta_1)|s, a\}$$

$$Q_{i+1}(\hat{s}, a; \theta_2) = E\{r + \gamma \max_{a'} Q_i(\hat{s'}, a'; \theta_2)|\hat{s}, a\}$$

where, $Q_{i+1}(s, a; \theta_1)$ and $Q_{i+1}(\hat{s}, a; \theta_2)$ correspond to the action DQN and argument DQN respectively. $\hat{s}$ denotes that the information about extracted actions is incorporated into it through the repeat representation of "distance" described in the last subsection.

Feng et al. [33] used MGNC-CNN [133] to reduce the dimensionality of the state-information encoded by the repeat representation. Four types of kernels that were used in the CNN were bigram, trigram, fourgram and five-gram, following the results of [33]. We used the same architecture mainly due to the following two reasons:

1. We attempted first to couple MGNC-CNN with LSTM layers by encoding it into a time-distributed (`keras.layers.TimeDistributed(layer)`) layer. This didn't work due to memory constraints caused by repeat representation. Repeat representation is not scalable and also caused problems in training with contextual embeddings as discussed above. Thus, applying sequence-to-sequence architectures from Keneshloo et al. [69] was out of the question for large dimensional embeddings.

2. We tried to get rid of repeat-representation by trying out various combinations of convolving the word vectors and other information (distances, RL actions) separately. But the concatenation or average of decoupled convolutions didn't converge the DQNs.

4.1.5. Reward Model and Training the DQN

We did not change the reward model and the training procedure from [33]. Reward r is comprised of two parts: a **basic reward** at time-step t and an **additional reward**. The values basic reward at time-step t are defined to be : +50 when the RL action of select or reject is correct and -50 for incorrect action, +100 when the word is an essential (ES) item, and -100 for missing ES item, +100 for getting optional (OP) item and 0 for not extracting OP item, +150 for correctly extracting exclusive items and -150 when the action is incorrect. An additional reward is determined based on the *real-time extraction* rate of actions or arguments. The

real-time extraction rate is compared to the ground-truth and if it was less than ground-truth rate, an additional reward is given. Otherwise, a negative reward is assigned to control the precision [33].

For *training* the EASDRL, we used the same mini-batched sampling which [33] applied. Specifically, the transitions $\langle s, a, r, s' \rangle$ and $\langle \hat{s}, a, r, \hat{s}' \rangle$ were stored in two replay memories Ω and $\hat{\Omega}$. Authors used positive experience-based replay instead of random sampling for faster convergence. Thus, we stuck to the parameters used in Feng et al. [33]'s CNN model and used the repeat representation as well. Although Feng et al. [33] achieved state-of-the-art-results, EASDRL exhibits a variety of problems caused by the used input representation of Word2Vec [85], which hinders its applicability to real-world tasks. We demonstrate these problems in the next section.

4.2. cEASDRL: Incorporating Contextual Embeddings into DQN

First, we illustrate the problems caused by using Word2Vec model, and then discuss state-of-the-art contextual embedding frameworks used in our report:

4.2.1. Problems with Word2Vec

The DRL model in [33] uses Word2Vec [85] embeddings. Word2Vec model is a shallow two-layer neural network that takes in as input a text corpus and outputs a dense vector per word. The word's meaning is distributed across the whole vector. As discussed in Chapter 2, the intuition behind the training of this neural net is that similar words would occur in a similar context in the corpus. Despite the capturing of semantics, Word2Vec showcased the following problems:

1. **Inability to handle out of vocabulary (OOV) words**: The never-seen-before words are not handled by the word2vec neural network as it has no idea about how to represent such OOV words. Especially, in planning domains, there is domain-specific knowledge and new technical words are introduced according to the task that has to be performed. For example,
   Sentence: `Use spoon to work the oil into the rice evenly coating the rice`
   EASDRL Sequence output: `<1> Use (UNK) <2> work (oil) <3> coating (rice)`
   Here, **UNK** represents the OOV word "spoon". We might initialize a random vector for it, which is far from ideal.

2. **Ignores Context/Polysemy**: Word2Vec is a **static embedding** average out the meaning of two polysemous words. For example, if "Apple" (the company and apple (the fruit) both occur in the corpus, the word "apple" would be represented by a single vector which averages out the meaning of two words. This makes Word2Vec representation too general and this might lead to the extraction of wrong actions/arguments.

3. **No shared representation at the sub-word level**: For Word2Vec, each of the words "eat", "eating" and "ate" are different, although they have the same word-stem and are used in the similar context. Word2Vec is too specific in this case. This would lead to duplicate actions or arguments.

4. **A number of word senses is not finite and thus, Word2Vec is biased**: There is large number of contexts in which a word can be used and doesn't always fit the dictionary representations.

4.2.2. Contextual Embeddings to the Rescue

Contextual word vectors include both the semantics of the word as well as some neural network parameters that describe their context. As we already have a task-specific architecture, we use the **feature-based** approach described in Section 2.5 to take advantage of pre-trained language models. To solve above-mentioned issues, we used the following contextual embeddings:

1. **POS/NER Embeddings**: Inspired from Sense2Vec[121], these embeddings stem the word (reduce it to its base form) and append parts of speech(POS) and named-entity recognition(NER) information to the words and then a model like Word2Vec is trained on it. For example, “going” becomes “going.verb” with lemma “go”. This embedding is not a true contextual embedding as it only captures some sense of the word and it is dependent on the accuracy of the dependency tree extracted from probabilistic POS/NER tagger. This is used as a baseline in Section 5.1.2. It is fast as at its core it’s still a dictionary lookup, but OOV words are still a problem.

2. **ELMo Embeddings**: ELMo[99] which stands for “Embeddings from Language Models” generates **contextual word vectors** that come from internal states of a bidirectional-LSTM [108] that is trained in an unsupervised fashion on large scale text corpus in order to learn a language model [99]. As the training is done in both forward and backward direction, a bidirectional Language Model (biLM) is learned. These are defined in [99] as,

$$p(w_1, w_2, \ldots w_N) = \prod_{k=1}^{N} p(w_k | w1, w_2, \ldots w_{k-1})$$

, which is the forward language model predicting the next word based on previous history, and

$$p(w_1, w_2, \ldots w_N) = \prod_{k=1}^{N} p(w_k | w_{k+1}, w_{k+2}, \ldots w_N)$$

which is the backward language model predicting the target word using future context.

These models are combined by jointly maximising the log-likelihood of the forward and backward models [99] and parameters for embedding layer θ_e and softmax layer θ_s are shared between them. A schematic view of biLM is shown in Figure 4.5. ELMo uses a character-based model which are based on character convolutions [15, 99]. These essentially remove the problems of OOV words and polysemy.

3. **Flair Embeddings**: Flair Embeddings[4] are very similar to ELMo in the sense that they train a biLM using LSTMs but they are trained without an explicit notion of what a word is. Figure 4.6 represents a schematic overview of Flair embeddings. They report comparable results to ELMo, that’s why we incorporated them. Also, the Flair API [5] is very easy to use and incorporates APIs for ELMo and BERT as well.

4. **BERT Embeddings**: BERT [29] or Bidirectional Encoder Representations from Transformers, is another method to train language models, which takes ideas from semi-supervised sequence learning [28], ELMo [128] and ULMFit [62]. As previously seen, ELMo and Flair train two representations of each word (left-to-right and right-to-left) and concatenate them together to have a single representation. Instead, BERT combines the left and right context by masking out 15% of the input words and

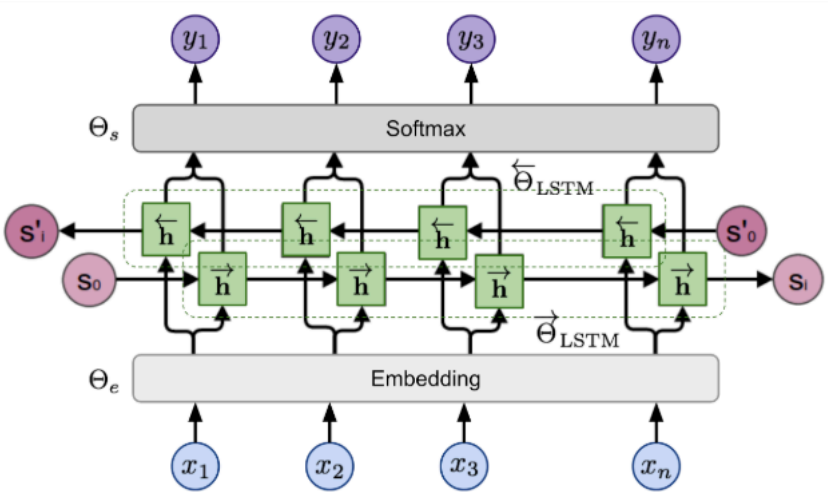


Figure 4.5: The biLSTM in ELMo trains on a joint objective of learning a bidirectional Language Model. The image is taken from [128], which is recreated from [95]

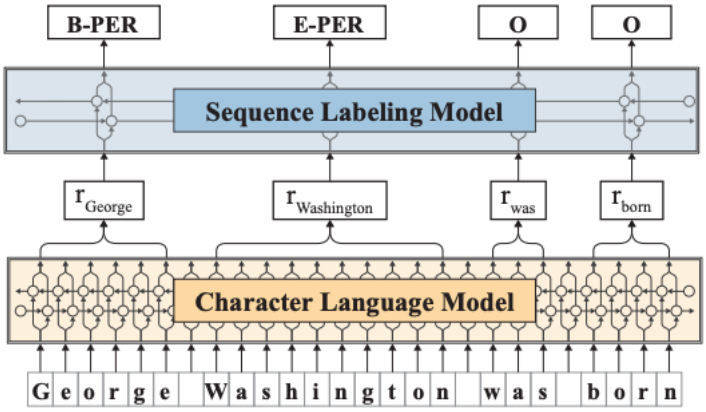


Figure 4.6: “Flair embeddings: Input as a character sequence of a sentence into a pre-trained biLM (like ELMo). The contextual word embeddings from biLM are passed into the sequence labelling model to perform named-entity representation [4].

then running entire sequence through multilayer bidirectional Transformer [1][125] encoder, then predict only the masked words. According to ablation study in [29], not having

5. **Stacked Embeddings**: Most papers recommend to combine the forward and backward contextual embeddings with traditional word embeddings (like GloVe, Word2Vec). An example of `StackedEmbedding` object from Flair API library [5] is shown below:

```
StackedEmbedding = [Glove,
                    flair-X-forward,
                    flair-X-backward]
```

Here, X represents the dataset used for pretraining.

4.3. Learning domain model using LOCM

Learning **O**bject **C**entred **M**odels (LOCM) is a family of algorithms [25, 26, 50–52] developed to learn planning domain models using only the plans, i.e., a sequence of actions $\langle a_0, a_1, a_2, ..., a_n \rangle$. They **do not require any intermediate state information** in the plans in order to perform learning but rather depend on certain assumptions that hold true for most of the domains. The assumptions made by LOCM mentioned in [25] are:

1. Each instantiated action changes the state of objects which are its arguments, and that each time an action is executed, the preconditions and effects on an object are the same.
2. The behaviour of an object can be described by a single Finite State Machine (FSM)
3. Objects belong to classes and all objects in a class behave in a similar way. In other words, each class of objects has a defined set of states that their objects can occupy. An object's state may change (state transition) as a result of action instance execution [81].
4. The position of each argument in an instantiated action always takes object of the same class, i.e., the action model/template never changes.
5. Each transition only appears once in the FSM. This assumption was made to avoid the use of conditional effects.

LOCM2 [25] is a more generalised version of LOCM [26] and allows for multiple (parameterised) FSMs per class of objects. By doing this, it relaxes assumption 2 and accommodates multiple behaviours per objects, which is usually the case in real-world. As a consequence, **an object occupies one state in each state machine.**

4.3.1. Preprocessing Action Sequence to satisfy the assumptions made by LOCM2

The assumptions of LOCM2 make it necessary that there should be no noise in the input. In other words, LOCM2 requires an action to be described with a fixed template. The action name should be unique and the action parameters should always be instantiated in the same order and types. This is analogous to a fixed functional prototype with no variable arguments

[1] an encoder-decoder attention based model with positional encodings, described in a famous paper called "Attention is all you need" [125]

in languages like C++ or Java. The extracted action sequences might have actions with the same action name but a different number of parameters. We need to cluster these actions into one or differentiate between them in order to satisfy the assumptions of LOCM2. First, we describe an approach of achieving fixed templates that **did not work** but was very intuitive and has scope for further research. We thought of preprocessing extracted action sequences in two steps:

1. cluster similar sentences to identify different ways in which an action prototype has been extracted, and

2. choose the best action prototype by doing a frequency analysis.

Lindsay et al. [76] used a similarity metric based on various thesaurus resources to compare words in order to cluster similar actions. One can say that this approach has the same problems as Word2Vec as it averages out the various thesaurus senses. Therefore, we attempted to make use of our contextual embeddings to find similarity between input sentences. Specifically, we created sentence embeddings by averaging out the contextual word embeddings of words of the sentence, normalized them, and compared them using pairwise `cosine_similarity` method from `scikit-learn` library [97]. Figure 4.7 shows a heatmap to showcase the similarity of sentences for an example of four sentences:

```
0. Make a gluten-free sandwich.
1. Make a sandwich which contains no gluten
2. Make a sandwich.
3. Make a sandwich which contains gluten.
```

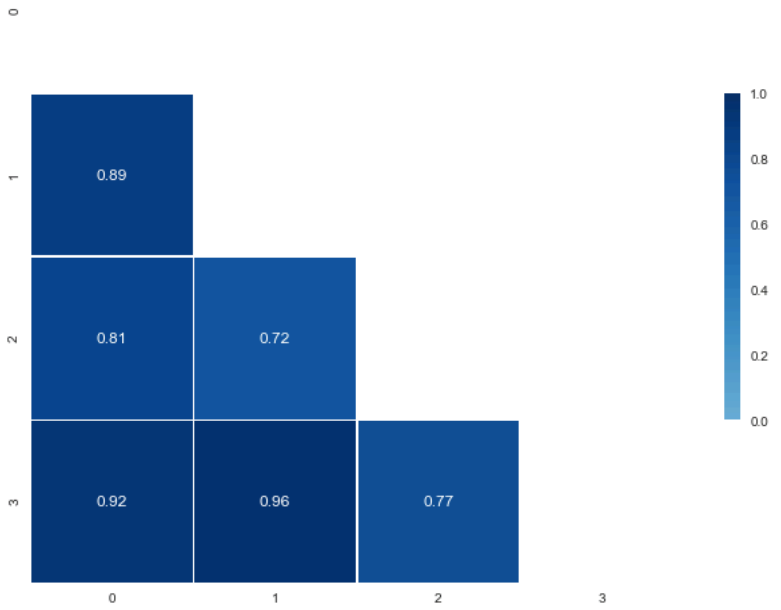


Figure 4.7: The heatmap shows the similarity matrix of sentences for an example extracted sequence

According to contextual similarity, the most similar sentences are 1 and 3, which is completely wrong. They are similar in terms of words and context but a “no” makes them opposite in meaning. Also, BERT Devlin et al. [29] embeddings are not trained for sentence similarity task and thus, doesn’t show good results. Thus, we learned that simply aggregating (contextual) word embeddings is a bad idea and the sentences lose their meaning.

In the end, **we employed a simple heuristic** which says if two actions have the same names but a different number of arguments, we differentiate between them by appending the first argument’s name that differs. For example, if the extracted action

prototypes were: `add(water,cup)` and `add(milk)`, then we would convert them to `add-water(water,cup)` and `add-milk(milk)`. This is not ideal but by doing this, we do not omit any actions or useful information represented by data and finally, the end-user (KE or SME) would have a choice to select between them.

4.3.2. Implementation of Interactive-LOCM2 along with step-wise illustrations

We reimplemented LOCM [26] and its extension LOCM2 [25] in an interactive Jupyter[2] Notebook environment in Python as the original source-code was not available. We call this interactive-LOCM2 because we need minor user-input from the user at three points in the procedure, and the interactive style of Jupyter notebooks makes it easier for the user to enter it. The inputs are optional but do increase the quality and intuitiveness of learned domains. We have already seen in Chapter 1 that automated KE tools are an assistive technology to the KEs and SMEs and thus, by providing such interaction we aim to reduce the effort and time required in modelling real-world domains.

The re-implemented LOCM1+2 [25, 26] combined algorithm is stated in Algorithm 1.

Algorithm 1: LOCM1+2

Input: action training sequence
Output: PDDL domain model

1 Determine classes of objects, and make state machines with transitions as states.
2 Get transition sets from `select_transition_sets` routine of LOCM2 algorithm
3 Use transition sets from `select_transition_sets` and create multiple Finite State Machines
4 Create and test hypothesis for state parameters
5 Create and merge state parameters
6 Remove parameter flaws
7 Get static preconditions from user (optional)
8 Form Action Schemas and PDDL model
9 end

Step 2 of this algorithm allows for multiple behaviours (parameterised FSMs) per object. The step 2 is described in Figure 4.8. We illustrate the algorithm step-by-step with the example driverlog domain from IPC 2002[3].

Step 1.1: Determine classes of objects

Input is extracted plans (action sequences) from the trained cEASDRL. LOCM2 assumes that each object in the argument list of an action undergoes a transition and objects of the same class behave in a similar way. We follow what transitions an object undergoes in a sequence. An example sequence for object `driver1` of class `driver` in `driverlog` domain is shown below:

```
walk(driver1, ewi, 3me),
walk(driver1, 3me, aula),
board-truck(driver1, truck3, aula),
load-truck(package5, truck3, aula),
drive-truck(truck3, aula, sports-center, driver1),
unload-truck(package5, truck3, sports-center),
disembark-truck(driver1, truck3, sports-center),
walk(driver1, sports-center, ewi)
```

[2] https://jupyter.org/
[3] http://ipc02.icaps-conference.org/

Procedure *select_transition_sets*
Input:
T_{all} – set of observed transitions for sort
H – set of holes - each hole is a set of one or two transitions.
P – set of pairs $\langle t_1, t_2 \rangle$, meaning t_1 and t_2 occur as consecutive transitions.
E – set of example sequences of actions
Output:
S – set of transition sets.

begin
1. $S \leftarrow \emptyset$

/* Ensure each hole is included */
2. **for each** $h \in H$
3. **If** there is no set $s' \in S$ such that $h \subseteq s'$ **then**
4. By breadth-first search, form s, the smallest set such that $h \subseteq s \subset T_{all}$ and
5. s is valid with respect to P, E by Definition 2
6. $S \leftarrow S \cup \{s\}$
end If
end for

/* Remove redundant sets */
7. **If**, for any two sets s_1 and s_2 in S $s_1 \subset s_2$ **then** $S \leftarrow S \setminus \{s_1\}$
end If

/* Include all-transitions machine, even though it might not be well-formed */
8. $S \leftarrow S \cup \{T_{all}\}$
return S
end

Figure 4.8: Procedure for selecting transition sets, taken from [25]

Here, we label `walk.0` as the transition undergone by the object of class `driver` as `driver1` is the first argument of `walk` action. From the assumption of fixed action templates, we can say that always an object of class `driver` would appear as the first argument. From the action sequence, we can directly observe the action names, the action arguments, and the transitions that are happening in our system. For our example,

```
Actions: {'load-truck', 'walk', 'disembark-truck',
          'board-truck', 'drive-truck', 'unload-truck'}
Objects/Arguments: {'sports-center', 'truck3', '3me',
                    'package5', 'ewi', 'aula', 'driver1'}
Transitions for walk action are walk.0, walk.1 and walk.2.
Similarly, each object of all actions undergoes a different transition.
```

Next, we look at the actions that have the same name. Using the assumption that all actions have fixed templates, we can cluster the set of objects that occur at a specific location into a class. In this way, we determine all the classes of objects present in the domain. The extracted classes and their names from the sequence are:

```
Classes:
Sorts/Classes
[{'driver1'}, {'truck3'}, {'package5'}, {'sports-center', '3me', 'ewi', 'aula'}]

Extracted class names
['driver', 'truck', 'package', 'sports-center']
```

Here, we need input from the user (**user-input no. 1**) to rename some of the classes, so that they can make sense later. Using “sports-center” instead of “location” would confuse the end-user. The interactive style of LOCM has this advantage that the end-user can improve the quality of the learned model by monitoring the procedure by executing it step-by-step. Here, a user can rename the fourth class to “location” and the resulting output would be:

```
Renamed class names
['driver', 'truck', 'package', 'location']
```

Step 1.2: Make transition state machines.

We first define two concepts: consecutive actions and consecutive transitions. Subsequently, we formulate the transition state-machines in which transitions formulate the states.

Definition: Consecutive Actions: Cresswell et al. [26] define two actions a_i and a_j to be consecutive if they operate on the same object and no other action in-between operates on the same object. For example, `walk(driver1, ewi, 3me)` and `walk(driver1, 3me, aula)` are consecutive with respect to objects `driver1` and `3me`, whereas `walk(driver1, ewi, 3me)` and `boardtruck (driver1 , truck3 , aula)` are not consecutive with respect to any objects because `driver1` was operated by another action in-between.

Definition: Consecutive Transition: Following the definition of consecutive actions and the assumption that classical plans are sequential we can deduce that for a pair of consecutive actions $\langle a_i, a_j \rangle$ with respect to object O (belonging to class G) occurring at positions k and l in their arguments respectively, **end-state of transition $a_i.k$ will be same as the start state of $a_j.l$.** These type of transitions are called **consecutive transitions** [26]. Consecutive transitions in our example sequence lead to following equivalence of states:

```
end(walk.0) = start(walk.0)
end(walk.2) = start(walk.1)
end(walk.2) = start(board_truck.2) ... and so on.
```

The next step is to make transition state machine of the sequences. Assuming transitions as “states” and if they are consecutive transitions or not as an indication of a “transition/edge” in FSM, we build a transition-graph. An example of such transition-graph learned from 5 plans of driverlog domain for the `driver` class is shown in Figure 4.9.

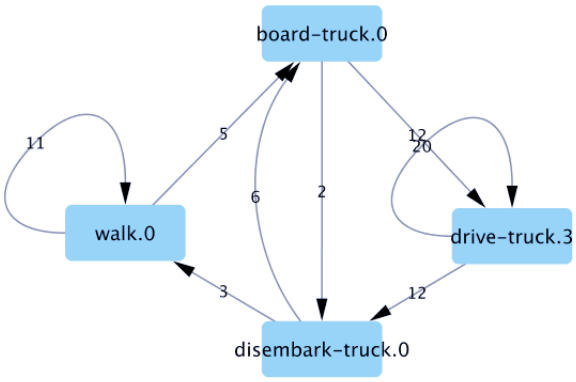


Figure 4.9: Transition state machine for `driver` class in `driverlog` domain. The weights represent how many times the consecutive transitions were observed.

Unlike LOCM2 [25], we retain the weights which represent how many times the consecutive transitions with respect to the object of class G were observed in the dataset. If we have data in abundance, we can do some noise-reduction techniques like discretisation to make LOCM

work with noisy data. However, as we are aiming one-shot learning, we do not use these weights in this work.

Instead, we use user interaction (**optional user-input #2**) to observe the transition FSMs and correct them using an interface we made from an open-source web graphing software called Cytoscape.js[4] [40]. The end-user can add/remove the missing/noisy transition-edges and start the rest of the pipeline with corrected transition FSMs. Cytoscape provides many other network analysis utilities like looking for connected components, self-loops, edge betweenness. These could be helpful in noise-reduction if the input data is noisy. Right now, the process of opening and saving graph files is manual but we intend to make an automated user-interface in future.

An equivalent representation of transition FSM is a transition matrix. Figure 4.10 shows transition FSM for the class **truck**. The equivalent Transition matrix for the class **truck** shown below:

	board-truck.1	load-truck.1	drive-truck.0	unload-truck.1	disembark-truck.1
board-truck.1	hole	6	7	1	1
load-truck.1	1	hole	9	1	1
drive-truck.0	hole	5	10	8	9
unload-truck.1	1	1	6	hole	4
disembark-truck.1	5	1	hole	1	hole

The numbers show the weights and valid transitions. The keyword `hole` indicates a behaviour that might get overfit by LOCM1. For a pair consecutive transitions $\langle T1, T2 \rangle$, LOCM1 would unify the states that should not be unified. **LOCM's assumption that each transition appears only once in the state-based representation is violated if two rows in the transition matrix have overlapping values but are non-identical.** These are referred to as holes. For example, the driver cannot disembark a truck after disembarking it already. The reason for this stated in [25] is that transition FSM is more expressive notation than STRIPS [34] and we need to eliminate such holes by allowing for multiple behaviours per class of objects object.

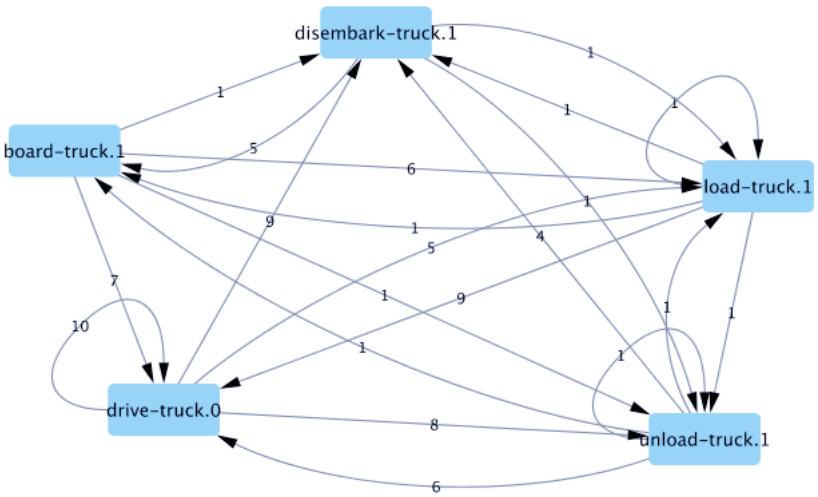


Figure 4.10: Transition state machine for `truck` class in `driverlog` domain. The weights represent how many times the consecutive transitions were observed.

[4] `http://js.cytoscape.org/`

Step 2: Get Transitions Sets per class from LOCM2

The step aims to get valid transition sets that define multiple behaviours per class and avoid over-fitting (over-generalization) of LOCM1 that is caused by its unification of states. As discussed, we need valid transition sets because we want to find equivalent state-representations which have more restrictive expression power than transition FSM. Figure 4.8 shows the original procedure `select_transition_sets` taken from [25]. It tries to partition the transition matrix into different parts so that there are no 'holes' (representing potential wrong state unification) are left. It does so by forming sets via Breadth-First Search starting from smallest to find the largest subset which doesn't have holes.

For the truck class, final transition set that we got was:

```
{{'disembark-truck.1', 'drive-truck.0', 'board-truck.1'},
{'unload-truck.1', 'disembark-truck.1', 'drive-truck.0', 'board-truck.1', 'load-truck.1'}}
```

Thus, by inducing a separate state machine for `disembark-truck.1`, `drive-truck.0` and `board-truck.1`, the hole in the transition matrix at $\langle disembark - truck.1, disembark.truck.1\rangle$ stays empty, as this smaller transition FSM doesn't allow it.

Step 3: Algorithm for induction of state machines

Algorithm 2 takes in the transition set and for each transition defines states **start(T1)** and **end(T1)**. Then for each pair of consecutive transitions T1,T2 in TS, it unifies states **end(T1)** and **start(T2)**

Algorithm 2: Step 1

Input: action training sequence of length N
Output: transition set TS, set of object states OS

1 Initialize state set OS and transition set TS to empty
2 Iterate through $A_i, i \in 1,\ldots,N$ and $j \in 1,\ldots,m[i]$ as follows:
3 Add state identifiers $start(A_i, j)$ and $end(A_i, j)$ to OS
4 Add $A_i.j$ to TS
5 For each pair of consecutive transitions T_1, T_2 in TS
6 Unify states $end(T_1)$ and $start(T_2)$ in set OS.
7 end

Induced State FSM1 for the class truck is shown in Figure 4.11. Here `state0` can be considered as being "driver outside the truck", and `state 1` can be considered as being "driver inside the truck". This state-machine imposes a restriction that either the driver is inside or outside the truck, and thus combined with other state-machine representing class truck correctly models the class behaviour.

Before the next discussed step, LOCM uses another step called "zero analysis" [26] to learn an implicit background object. We do not use this as this is a part of static preconditions in a domain which a user can easily specify. These could also be learned from systems like [50] but they will require more data than just one action sequence.

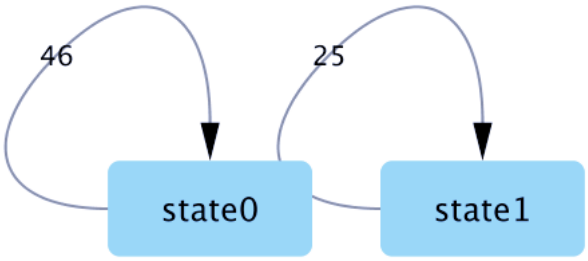


Figure 4.11: State machine for `truck` class in `driverlog` domain. State0 is `{'start(board-truck.1)', 'end(disembark-truck.1)}`, and state 1 is `{'end(board-truck.1)', 'end(drive-truck.0)', 'start(disembark-truck.1)', 'start(drive-truck.0)'}`

Step 4: Induction of parameterised FSM

LOCM parameterises the state machines to record a pairwise dynamic association between objects [26]. Algorithm 3 taken from [26], describes this step:

Algorithm 3: Induction of Parameterised FSM (Step 4)

Input: Action Sequence Seq, Transition Set TS, Object set Obs
Output: HS retained hypothesis for state parameters

Form Hypotheses from state machines
For each pair $B.k$ and $C.l$ in TS
    such that $end(B.k) = S = start(C.l)$
For each pair $B.k'$ and $C.l'$ sharing sort G'
    and $k \neq k', l \neq l'$
    Store in hypothesis set HS the hypothesis H = $< S, B, k, k', C, l, l', G, G' >$
end
Test hypotheses against example sequences
For each object O occurring in O_u
    For each pair of transitions $A_p.m$ and $A_q.n$ consecutive for O in Seq
        For each hypothesis H = $< S, B, k, k', C, l, l', G, G' >$
            matching $A_p = B, m = k, A_q = C, n = l$
            if $O_{p,k'} = O_{q,l'}$
                then flag H as having a positive instance
                else remove H from hypothesis set HS
            endif
        end
    end
end
Remove any hypothesis H from HS without a positive instance.

Here, $B.k$ and $C.l$ are two consecutive transitions with respect to object O from class G in our transition set, and we check whether the same object O' from class G' always appears in them. If it does, there is a strong chance that G and G' are related classes. After checking the whole dataset, the LOCM retains such a hypothesis.

Step 6: Creation and merging of state parameters

Since many hypotheses are redundant, this step tries to reduce the number of hypotheses found. The redundancy comes from the fact that there are multiple transitions that go through a state and each one doesn't need their own parameter. They `set` and `read` the same parameters of the state, much like mutex locks in Operating Systems. We create parameters

bindings related to all hypotheses and then merge them if two hypotheses operate on the same state [26].

Step 7: Removing parameter flaws

There might still be parameters that enter into the transition but `set` the value of a parameter of that state. Such bindings are called parameter flaws and are removed from the parameter bindings set. For example, fault removed parameter bindings were found between some location states and object of type driver.

Step 6: Extraction of static preconditions

This is the last input required by the user (**user input #3**). These are relatively easy to write in the final PDDL model itself. Automated methods such as [50, 63] have been devised to detect static preconditions, but require more data to converge.

Step 7: Formulation of PDDL action schema

This step creates one predicate each for an object state. Action schema is induced by using the action prototype from plans. Preconditions and Effects are determined in “and” form from various induced finite state machines. The full learned model of driverlog domain is presented in Section B.3.

In the end, to use the extracted domain model we have to specify the states in terms of start(T1), end(T2), ..etc., where T1 and T2 are transitions. This is very tedious to do and this representation of state knowledge in terms of transitions is not human-readable. Since we do one-shot learning from a single process manual, this is still manageable for the purposes of our thesis but we do need a better approach to show the user state knowledge.

4.4. Summary

We once again refer to our formal modal in Figure 4.1 in order to solidify the discussions about individual components. Our main contributions in this chapter are to combine all three components: contextual embeddings [4, 29, 99], EASDRL [33] and LOCM [25, 26] into one seamless module called NLtoPDDL, which takes as input a natural language process manual and generates as output a PDDL model. In terms of implementation, we reimplemented LOCM1+2 in an interactive fashion to incorporate user input. The reasoning behind this is that LOCM2 is **heuristic** in nature [25]. Although LOCM2 requires no background information, it usually requires many plan traces for synthesising meaningful domain models. The output represents to its best efforts what is given in the example sequences. If the sequences are always goal-oriented, the LOCM2 might miss certain aspects of the domain. For best results, both exploratory and goal-oriented actions should be incorporated. As we want to one-shot learn the domains from natural language process manuals, we use user-interaction alongside our reimplementation of LOCM to mitigate this limitation. However, while evaluating the learned domain models in Chapter 5, we do not use this optional user input, except for changing detected class names.

5

Experimental Evaluation

Since the NLtoPDDL pipeline described in Chapter 4 happens in two stages, we did an experimental analysis for each phase separately. We first evaluated the performance of our trained contextual-DQN approaches in order to select the best approach for action sequence extraction task.

Later, in Section 5.2, we employed this trained-DQN to extract action sequences from unseen process manuals and learn the domain models using LOCM2 [25]. We evaluated the learned domain models on their accuracy, robustness, completeness and intuitiveness.

5.1. Evaluating Trained DQN

Our evaluation of the trained DQN model aimed to confirm the following hypothesis:

- *Hypothesis 1.* Transfer Learning in NLP, which uses unsupervised pretraining of language models on huge corpora to generate dynamic contextual embeddings will seamlessly integrate into DQN architecture of EASDRL [33] and will push the state-of-the-art by improving upon the F1-scores for extraction of action names and action arguments.
- *Hypothesis 2.* Use of dynamic contextual embeddings will resolve the issues caused by the use of non-contextual embeddings, namely, out-of-vocabulary (OOV) words, polysemy, shared representation, and infinite word senses.
- *Hypothesis 3.* By having a 20% unseen test set, we can detect overfitting and thus, have an unbiased estimate about the generalisation of our approaches to real-world user data.
- *Hypothesis 4.* Employing Transfer Learning through contextual embeddings would make the DQN converge faster with the same amount of data, i.e., the training would converge in less number of epochs.

5.1.1. Experimental Setup

The training of models was done on TU Delft's `insy-cluster`[1] to utilise the available GPUs that were needed for training the DQNs. Following are the details of our experimental setup related to action sequence extraction problem:

Annotated Datasets. For the training of the DRL models, the annotated datasets are taken from Feng et al. [33]'s open-source repository[2] for three real-world domains:

[1] `http://insy.ewi.tudelft.nl/content/hpc-cluster`
[2] `https://github.com/Fence/EASDRL`

1. “Microsoft Windows Help and Support” (`WinHelp`) documents [17]
2. “CookingTutorial” (`Cooking`)[3]
3. “WikiHow Home and Garden” (`WikiHG`)[4]

The datasets are of increasing complexity in the context of the task of finding action names and arguments as described in Table 5.1. “Labelled texts” row shows the total number of documents that represent different process manuals or recipes. “Input-Output pairs” row depicts the number of $\langle word, annotation \rangle$ type training pairs which were detailed in Section 4.1. Action name and action argument rates represent the frequency of occurrence of actions and arguments in respective datasets.

	WinHelp	Cooking	WikiHG
Labelled texts	154	116	150
Input-Output pairs	1.5K	134K	34M
Action name rate (%)	19.47	10.37	7.61
Action argument rate (%)	15.45	7.44	6.30

Table 5.1: Annotated Datasets used to train the Deep RL models. Values taken from [33]

Introduction of the Test Set. Although Feng et al. [33] used cross-validation to average out scores on different validation folds, they did not use a test dataset to get an unbiased estimate of the performance of EASDRL. We held-out 20% of our data as unseen test data to test Hypothesis 3. We then used 80% of the remaining data as training data and 20% of the remaining data as validation data. Thus, the final ratio of training-validation-test data split was 64-16-20. We did not use cross-validation folds to avoid out-of-memory issues caused by large embeddings on limited resources of the `insy-cluster`, as well as to save computational time.

Number of Epochs. The number of epochs we trained the contextual-DQNs was set to 1. The primary reason was limited memory resource quota on `insy-cluster`. However, this led us to test the faster convergence rate of contextual-embeddings (Hypothesis 4).

Hyperparameters and Reproducibility. We used the same hyperparameters for the ConvNet (Q-estimator of the DQNs) as mentioned in the [33] except for changes in number of epochs and embedding dimensions. The authors of EASDRL took these parameters from MGNC-CNN [133]. An instance of the architecture of the CNN used is shown in Appendix E. We varied the input dimension according to the embedding used, for example, ELMo embeddings used (500 x 868 x 2) for action names and (100 x 868 x 3) for action arguments. For reproducibility, the source code is available on `https://github.com/Shivam-Miglani/contextual_drl`.

Evaluation Metrics. For the evaluation of approaches, the validation set’s F1-scores were computed to compare with the results in [33]. In addition, we computed the F1-scores on test dataset to gauge the generalisability of approaches (Hypothesis 3). F1-scores were computed the same way as in [33]. Specifically,

$$precision = \frac{\#TotalRight}{\#TotalTagged},$$

[3] `http://cookingtutorials.com/`
[4] `https://www.wikihow.com/Category:Home-and-Garden`

$$recall = \frac{\#TotalRight}{\#TotalTruth},$$
$$and\ F1 = \frac{2 \times precision \times recall}{precision + recall}$$

where, $\#TotalRight$ is the number of correctly extracted action names or action arguments; $\#TotalTagged$ is the number of extracted action names or action arguments; and $\#TotalTruth$ is the number of ground truth action names or action arguments from the annotations.

5.1.2. Baselines and cEASDRL Contenders

To compare with our contextual-EASDRL (cEASDRL) approaches, we used variants of non-contextual approaches like Framer [76] and EASDRL [33] as **baselines**:

1. **StanfordCoreNLP**: Stanford CoreNLP [80], a NLP library was used to parse the texts and tag parts-of-speech (POS) to the words. Using these tags the sequences of actions was extracted by selecting a verb as an action name, and the objects as action arguments in [76]. This was not implemented in our research and results shown are reported from [33]

2. **EASDRL** and **rEASDRL**: EASDRL from [33], which uses word2vec embeddings [85] was used as a baseline because it produced state-of-the-art results for action sequence extraction problem. As the paper reported only cross-validation results, we compared it with the results of contenders on our validation set. However, the results of EASDRL are not directly comparable as they are averaged out on 10 folds of the cross-validation but still give some indication of relative performance on F1 score.

   For direct comparisons on both validation and test datasets, Reproduced-EASDRL (rEASDRL) was used. rEASDRL is same as EASDRL[33] except that it employed our experimental setup of 64-16-20 train-val-test split without cross-validation, instead of EASDRL's 10 fold-cross validation without a hold-out test set. In essence, rEASDRL trained on less data, i.e., 64% instead of 80% and so did our contender approaches.

   We trained both EASDRL and rEASDRL methods for 20 epochs each for action name DQN and action argument DQN. The value 20 epochs was the original setting used in the EASDRL paper.

3. **GloVe + rEASDRL**: In GloVe + rEASDRL baseline, the 50-dimensional word2vec embeddings [85] were replaced by the 100-dimensional GloVe embeddings [98].

4. **POS-GloVe + rEASDRL**: In POS-GloVe + rEASDRL baseline, we added some context to the words by appending parts-of-speech (POS) information extracted from the Stanford CoreNLP library [80] and then retrained the GloVe embeddings. For example, if the word was "cheese" is replaced by "cheese|NN", where NN stands for "Noun, singular or mass".

   This approach is inspired by and is a simplified version of researches presented in Sense2Vec [121] and syntax-tree embeddings [78]. The simplified version was used because both approaches use their own tokenizers, which compound multiple words into a single joined word. This messes up the word index order of the training dataset and consequently, the annotations are no longer applicable.

Our Approach. We now specify the variants of our contextual-EASDRL (cEASDRL) approach which are distinguished by the choice of contextual embedding they use. Since the papers

of contextual embeddings suggest appending non-contextual embeddings like word2vec or GloVe to them, we append 100-dimensional GloVe vectors to each of the embeddings to generate stacked embeddings. The following were our choice of stacked-embeddings based on empirical evidence of their performance in their respective research papers [4, 29, 99].

1. **GloVe + ELMo + cEASDRL**: This cEASDRL's variant uses 100 dimensional GloVe embeddings [98] and 768 dimensional ELMo embeddings [99]. The pre-trained dataset used by these embeddings is 1B Word Benchmark [19][99].

2. **GloVe + BERT + cEASDRL**: In this variant of cEASDRL, we use the `bert-base-uncased` version of BERT embeddings [29] which are 3072-dimensional long vectors stacked with 100 dimensions of glove making it a 3172-dimensional stacked embedding. The datasets used in pretraining the `bert-base-uncased` are e BooksCorpus (800M words) [134] and English Wikipedia (2,500M words)[5][29].

3. **GloVe + Flair-f-b + cEASDRL**: In this variant of cEASDRL, we use stacked `mix-forward` and `mix-backward` Flair character embeddings [4], each of which is 2048 dimensions. Stacked with GloVe, this makes a 4196-dimensional word embedding. The `mix-forward` and `mix-backward` versions are pre-trained on the mixed corpus (Web, Wikipedia, Subtitles) [5]. Due to high memory requirements, we also limit the character limit per word to 128 characters by setting the `chars_per_chunk` in the optional arguments of the Flair-Stacked embedding.

All the contextual embeddings are implemented using Flair NLP library[6] [5] using its `StackedEmbedding` class.

5.1.3. Results of Comparison with Baselines

F1-scores of all three datasets for action names and action arguments are reported for the validation dataset and test dataset.

Validation dataset results. Validation dataset allowed us to iterate over it repeatedly and get the best possible model weights and parameters. In Table 5.2, we can see from the validation results of the contextual-approaches are better than that of baselines just in 1 epoch. Specifically, there is an improvement of 4-7% in the F1-score from the best baseline for action names in all three datasets but barely any significant improvements in F1-score for action arguments.

On the other hand, the baseline POS-GloVe+rEASDRL had terrible results as the average loss of DQN generally increased, and the RL agent became too conservative to act. It neither selected or rejected a word to avoid negative rewards. The blame may be given to appended parts-of-speech tags which distinguish the context in which word is used. This creates different entries in the lookup-table of GloVe or Word2Vec and struggles to generalise when the same word is used in a new context. The results of this baseline also indicate that the results are highly dependent on the accuracy of dependency parsing.

The validation results give us an idea of the performance of cEASDRL variants compared to non-cEASDRL variants but is a biased estimate as it underestimates the true test error substantially. Thus, we look at the performance of all approaches on 20% of held-out test dataset to get an unbiased estimate that is closer to the real-world usage.

[5] `https://dumps.wikimedia.org/`
[6] `https://github.com/zalandoresearch/flair`

			Action names			Action Arguments		
Method	Epochs	Cross-val	WinHelp	Cooking	WikiHG	WinHelp	Cooking	WikiHG
StanfordCoreNLP*	1	No	62.66	67.39	62.75	38.79	43.31	42.75
EASDRL	20	10-fold	93.46	84.18	75.40	**95.07**	74.80	75.02
word2vec+rEASDRL	20	No	92.03	81.99	74.02	94.90	74.05	73.70
GloVe+rEASDRL	20	No	94.08	80.41	64.58	94.35	74.21	73.69
POS-GloVe+rEASDRL	20	No	32.33	0.0	0.0	73.24	38.82	42.66
GloVe+ELMo+cEASDRL	1	No	92.75	87.29	79.22	92.06	**75.81**	**76.99**
GloVe+BERT+cEASDRL	1	No	96.22	**89.18**	**82.59**	92.78	73.23	76.19
GloVe+Flair-f-b+cEASDRL	1	No	**97.32**	88.86	79.16	83.43	62.41	72.40

Table 5.2: F1-scores on 16% of validation dataset in 64-16-20 training-validation-test split. * indicates the result taken from [33]. Note that, extraction of action arguments uses ground-truth action names.

			Action names			Action Arguments		
Method	Epochs	Cross-val	WinHelp	Cooking	WikiHG	WinHelp	Cooking	WikiHG
word2vec+rEASDRL	20	No	91.98	80.17	73.77	85.15	69.65	68.27
GloVe+rEASDRL	20	No	93.96	78.44	57.87	**94.02**	71.79	47.87
POS-GloVe+rEASDRL	20	No	32.68	0.0	0.0	74.55	38.63	51.27
GloVe+ELMo+cEASDRL	1	No	92.75	85.18	78.43	92.47	**76.50**	**77.12**
GloVe+BERT+cEASDRL	1	No	96.15	**88.42**	**82.95**	90.56	72.98	74.75
GloVe+Flair-f-b+cEASDRL	1	No	**97.46**	86.19	80.09	83.64	64.40	72.82

Table 5.3: F1-scores on 20% of test dataset in 64-16-20 training-validation-test split. Note that, extraction of action arguments uses ground-truth action names.

Test dataset results. In Table 5.3, we can clearly notice the advantage of using contextual embeddings. For the action names, we see an improvement over the best baseline by 5-8% in all datasets.

For the action arguments, the biased estimate of the baselines on validation tests is revealed, i.e., they were underestimating the true test error. Hence, the performance of baselines drops significantly, especially in complex datasets (Cooking and WikiHG). On the other hand, the performance of cEASDRL approaches is even better on the test sets than validation results, making them more generalisation to real-world settings. In particular, GloVe+ELMo+cEASDRL beat the best baselines by 5-9% in Cooking and WikiHG datasets.

Thus, we can confirm that contextual embeddings did integrate well into the DQN architecture, and beat the current state-of-the-art results presented in EASDRL [33] for the action sequence extraction problem. This confirms our Hypothesis 1,3 and 4.

Chosen model. Based on test results, we choose **GloVe+BERT+cEASDRL** as our final model for extracting action names. It is chosen because it performs better than GloVe+Flair-f-b+cEASDRL on harder datasets of Cooking and WikiHG. Similarly, **GloVe+ELMo+cEASDRL** was chosen as our final model for extracting the related action arguments.

5.1.4. Qualitative analysis of the Extracted Sequences

F1-score is a qualitative measure of improvement in the quality of sequences, but what does a 5% increase in F1-score actually mean. Intuitively, it should much more significant than 5% improvement in accuracy, because it gives equal weights to precision and recall.

To determine this **qualitatively** assessed the extracted action sequences from unseen data. Figure 5.1 shows an example of action descriptions taken from Florida Atlantic University's website[7] to emulate the user-input. The extracted action sequences of Word2Vec + rEASDRL and our chosen cEASDRL model are compared and shown in Figure

[7] http://www.fau.edu/ehs/info/fire-safety-manaul.pdf

5.1.

We can see that cEASDRL produces coherent and correct action sequences, unlike Word2Vec + rEASDRL. In sentence no. 2, Word2Vec model initialises unseen words as zero vectors and thus, is unable to extract essential actions. On the other hand, cEASDRL does extract correct action from unseen data and even recognises an exclusive-or between “hazardous experiments” and “procedures”.

We see a safety-critical scenario in sentence no. 5, Word2Vec + rEASDRL, incorrectly extracts an action “go()” and misses the ES argument “heat”, whereas the chosen cEASDRL model correctly extracts the actions described in the statement. Some arguments are missed by both approaches, for example, “nature of emergency” in sentence no. 8. The reason behind might be that the models are trained to extract single word arguments. Also, cEASDRL correctly skips sentence no. 1 and 10 as they do not represent any ES action names.

Thus, we can conclude that even 5-7% improvement in F1-score corresponds to huge improvements in the overall sequence quality. We can also easily deduce that the contextual embeddings do solve for problems of out-of-vocabulary (OOV), polysemy, and shared representation. This is due to the fact that language models are pretrained on character level (e.g., ELMo) or sub-word level (e.g., BERT), and these algorithms take in the whole sentence as input to consider the context of the word and then generate a dynamic word-embedding rather than a fixed vector. As the language models for contextual-embeddings are trained on huge corpora, they also solve for a large extent the problem of infinite word senses. This section qualitatively confirms our hypotheses 2 and 3.

In the next section, we evaluate the domain models which were learned using the chosen model’s extracted action sequences using LOCM2.

5.2. Learning of Domain Models

To learn a domain, we apply NLtoPDDL’s second phase to its natural language process manual containing action descriptions describing a valid plan. We test our domain learning approach on IPC problems related to our datasets and couple of process manuals taken from the real-world settings.

We aim to confirm the following hypothesis in this section:

- *Hypothesis 5.* We can one-shot induce a valid PDDL domain model from an extracted action sequence belonging to a natural language process manual.
- *Hypothesis 6.* The learned domain model’s robustness (precision) and completeness (recall) would correspond to the F1-scores of the chosen sequence extracting approach.
- *Hypothesis 7.* The learned domain models would be intuitive which makes them easy to extend and modify for the user.
- *Hypothesis 8.* We can extend NLtoPDDL to durative actions by learning durative domains, a type of non-classical domain.

5.2.1. Learning IPC Domains

.

Reference Models. We did not have the liberty to pick popular domains from the IPC competitions because they do not relate to our training datasets. Nevertheless, we selected the following *classical* domains which seem to be related to our training datasets:

1. child_snack[8] : We used sequential, optimal version of the child_snack domain which was used in IPC 2014. We chose this domain because it is about cooking and we can use weights learned from the cooking dataset to extract action sequences.

   The authors Raquel Fuentetaja and Tomás de la Rosa Turbides describe the domain as:

   "This domain is to plan how to make and serve sandwiches for a group of children in which some are allergic to gluten.

   Problems in this domain define the ingredients to make sandwiches at the initial state. Goals consist of having all kids served with a sandwich to which they are not allergic."

2. woodworking: A subset of woodworking domain is taken from the sequential optimal track of IPC 2008. We chose this domain because it might be related to the home and gardening dataset's distribution. It is described in IPC 2008[9] as:

   "Simulates the works in woodworking workshop where there is some quantity of wood that has to be polished, coloured etc. using different tools with different cost."

   Although the domain is quite large with six kinds of machines with action costs, we attempted to learn only a small subset ignoring the action-costs.

The reference PDDL models of the domains can be found in Appendix A.

Unseen Datasets. We crafted the action descriptions for the domain by thinking of an instance of a valid sequence of actions. We also used PLANNING.DOMAINS website [90], which includes a cloud planner to solve the original domains to get example plans in order to think of our action descriptions in English. The input action descriptions, the extracted action sequences, and the learned domain model are presented in Appendix B.

Evaluation Metrics. We checked for learned models completeness and soundness (robustness) compared to the reference model. To evaluate the soundness and completeness of the learned domain model, we calculate **precision** and **recall** in the same manner as used in [2]. Intuitively, precision gives a notion of soundness or robustness by telling us how many selected domain items were relevant. On the other hand, recall gives a notion of the completeness of learned domain models by telling us how many relevant items were selected. Formally,

$$Precision = \frac{tp}{tp + fp},$$

where tp is the number of true positives, i.e., the predicates that correctly appear in the action model, and fp is the number of false positives, i.e., predicates that appeared in the learned action model but should not appear. Recall is formally defined as

$$Recall = \frac{tp}{tp + fn},$$

where fn is the number of false negatives, i.e., predicates that should appear in the learned action model but are missing.

We also employed precision and recall on the detected action names and each action's arguments, which will always be high if we are learning from structured data but not necessarily high when learning from natural language data.

[8] https://github.com/potassco/pddl-instances/tree/master/ipc-2014/domains/child-snack-sequential-optimal

[9] http://icaps-conference.org/ipc2008/deterministic/Domains.html

Domain	Action names		Action Parameters		Pre-conditions		Effects	
	P	R	P	R	P	R	P	R
child_snack	0.80	0.66	0.60	0.38	0.66	0.43	0.77	0.50
woodworking	0.40	0.10	0.5	0.53	-*	-*	-*	-*

Table 5.4: Precision (P) and Recall (R) averaged over all learned actions, representing soundness and completeness of the learned PDDL models, respectively. The precision and recall of preconditions and effects are based on the preconditions and effects of the arguments that were extracted. Otherwise, they would be zero as we were not able to extract the full set of arguments in the first phase of NLtoPDDL. These results are only indicative of performance for one instance of process manuals, and can't be generalised. -* means that learned actions were too different to compare.

5.2.2. Results on IPC Domains

Soundness and Completeness. Compared to the reference model, the precision and recall for action names, action arguments, action preconditions and action effects are stated in Table 5.4. The precision and recall scores were calculated manually and were averaged over all actions. The precision and recall over action names are similar to the F1-scores learned by cEASDRL model. However, the precision and recall over action parameters are way less than F1-scores of cEASDRL. This is because the instructions provided are not representative of the annotated dataset.

As we were not able to learn all the arguments, the preconditions and effects would never exactly match their counterparts in the reference model. Therefore, while calculating precision and recall for preconditions and effects, we ignored the unlearned arguments by assuming they are always present and then evaluated the rest of the behaviour. In other words, the results related to preconditions and effects are representative of the predicates that the learned model got right in terms of arguments that are present in it. Moreover, these results are only indicative of the performance for the type of natural language instructions that we crafted. These cannot be compared with PDDL models learned from structured data or previous research because of the "free" form of natural language that is used as input. As the same instructions could be said in a hundred different ways, we played around a little bit with the style of crafted instructions by changing it using an online paraphrasing tool, called QuillBot[10] to see changes in the learned PDDL model. The paraphrased process manuals generated a different domain model each time, and qualitatively evaluating it by manually calculating precision and recall was not feasible. Thus, we opted for a different approach of measuring the validity and intuitiveness of the learned PDDL model. These are the aspects that are important for the end-user (KEs or SMEs), especially, when they want to understand, modify or extend the learned PDDL models.

Domain	Syntax	Level of Intended meaning /Semantics captured?	Easy to fix?
child_snack	✓	intermediate	✓
woordworking	✓	low	✗

Table 5.5: Validity and intuitiveness of domains.

Validity and Intuitiveness. We imported the learned domain model in `mypddl-IDE` [116]

[10] `quillbot.com`

plugin of Sublime Text[11]. The context-aware syntax highlighting feature of `mypddl` [116] distinctly represents missing brackets, missing expressions and misspelled keywords. As expected, we didn't find any syntactical mistakes because our PDDL code is automatically generated through simple rules based on LOCM output, which avoids such mistakes.

We discuss the learned behaviours and compare it with intended semantics to measure the intuitiveness.

- **child_snack**: The input action descriptions, the extracted action sequences and the learned domain model are presented in Section B.1. The learned domain model combined the `make_sandwich_no_gluten` and `make_sandwich` actions of original domain into single action `make` of making a sandwich with an extra argument for sandwich-type. The learned `make` action is shown below:

```
(:action make
:parameters (?gluten-free - sandwich-type ?sandwich - sandwich )
:precondition (and
                (sandwich-type_fsm0_state0)
                (sandwich-type_fsm0_state1 ?v0 - sandwich ?v1 - sandwich)
                (sandwich-type_fsm0_state2)
                (sandwich_fsm0_state0)
                (sandwich_fsm0_state1)
                (sandwich_fsm1_state0)
                )
:effect (and
                (sandwich-type_fsm0_state0)
                (sandwich-type_fsm0_state1 ?v0 - sandwich ?v1 - sandwich)
                (sandwich-type_fsm0_state2)
                (sandwich_fsm0_state0)
                (sandwich_fsm0_state1)
                (sandwich_fsm1_state0)
        ))
```

  Here, the precondition and effect with state `sandwich-type_fsm0_state1` has parameters `v0` and `v1` of the sandwich class, which represents that sandwich-type is always dependent on a sandwich. There are two parameters, one for `gluten-sandwich` and other for `gluten-free sandwich`. Learned domain model has an extra action `take`, which takes the ingredients of a sandwich before making it. This showcases that LOCM only considers **atomic actions** and cannot combine multiple actions into a macro-action. In the context of natural language data, this leads to multiple versions of learned domain models based on slight tweaks in natural language data. The learned domain model completely misses out the behaviours of classes `child` and `place`, which are present in the reference model. This happened because the cEASDRL failed to extract arguments related to `child` and `place`.

  The FSM states are simple and intuitive to infer in case of the small input we used. For example, looking at the state dictionary for `sandwich_fsm0_state1`, we obtain the state `{end(make.1), start(put.1)}`, which represents two equivalent states: a state which marks the end of making a sandwich and equivalently, a state which marks the starting point of putting a sandwich into the tray. Thus, this corresponds to the state "sandwich is prepared". However, inferring this information manually for each state predicate is tedious and not a scalable approach.

- **woodworking**: The input action descriptions, the extracted action sequences and the learned domain model is presented in Section B.2. Similar behaviour was observed in

[11] `https://github.com/Pold87/myPDDL`

woodworking. Although, a valid PDDL model representing some meaningful constructs of the domain, it again missed some aspects such as types of machines available (planer and glazer), the colour information. The reason behind is two-fold: some of these are static conditions which never appear in the example sequence and requires approaches like [50] (with more data) to learn these, and others are simply due to missed arguments by the DQN. The learned model did incorporate the cost of actions in a separate function called `increase`, but this is not desirable as action-costs must be specified with the actions themselves. To mitigate this, we see an extension of NL2PDDL to a temporal domain (Section 5.2.4) in which time works like an action-cost. An interesting thing that we observed in both IPC domains was that the learned model segregates composite actions which can't be described in a single phrase into two and links them up with state parameters. This might be a desirable property in some applications where only atomic actions are allowed.

From our perspective, the words used to represent classes of objects, objects, action names and arguments are mostly correct and understandable, as they are derived from the natural language process manual, which makes the domain model somewhat intuitive. The problematic aspect is state names which are of the form "`class-name_fsm-no._state-no.`", and one needs to look at the state dictionary and FSMs to see. The results are summarised in Table 5.5. For a smaller domain like child_snack, it was easier to identify problems and fix the intended meaning.

SMEs which do not have experience in planning languages might find it hard to extend such PDDL models. A future user study, comprising of KEs and SMEs, is required to substantiate these claim as intuitiveness of PDDL models is a subjective property.

Despite the inconclusiveness, the main advantage of NLtoPDDL is its generalisability to real-world data. In the next subsection, we evaluate NLtoPDDL to learn an initial PDDL domain model from *free* instructional texts from a fire safety process manual.

5.2.3. Learning PDDL Model from Real-World Process Manual

We take a real-world process manual of our fire safety domain, for which we extracted an action sequence in Figure 5.1. We discuss the learned models strengths and weaknesses below:

- The domain model extracts: `provide`, `call`, `take`, `turn-off`, `secure`, `using`, `proceed`, `inform`, `check`, and `close` as the actions.From manually annotating the input and comparing our extraction to it, we calculated **0.91** and **1.0** to be precision and recall, respectively, on action extraction. Similarly, the precision and recall for extracted action parameters was **0.84** and **0.65**, respectively. We can see that the action extraction rate is better than the domains from IPC. This is because the real-world instructional data is similar to the training datasets used. For the IPC domains, we manually made the dataset with simple instructions. This shows that our method is generalisable to variety of writing styles but is susceptible to perform better for data distribution that is similar to the training set.

- The preconditions and effects of learned actions mostly reflect the previous action's state (which was unified with other states) and reflective of what one could do best from the observed data of one sequence. This is intuitive because there is only sequence of actions happening in the fire safety manual, and the exclusive actions were filtered out while extraction. However, the way of representing preconditions and effects in terms of states of various FSMs requires a graphical user interface (GUI) for user to make sense out of it. Manually determining the current state of an object in all its class's FSMs is

tedious and a non-scalable approach. In future, we do intend to incorporate a GUI for observing state information.

5.2.4. Extension to Durative Actions

To further check the flexibility of our approach, we tried to incorporate durative actions in our learned PDDL models. For this experiment, we used a durative **tea** domain taken from [120]. One advantage of using natural language data is that we can take advantage of existing tools available in NLP in our pipeline. Thus, we can use **Named Entity Recognition** (NER) to extract time phrases. This can also be incorporated into another DQN architecture but it would be an overkill and too specific for this task. We used spaCy [61] library's statistical NER to detect time phrases. Figure 5.2 shows the time phrases detected for the tea domain [120]:

The time phrases are detected with high precision, however, we still need to assign these time phrases as action arguments for some action. We follow a heuristic that the action name's index which is closest to the starting index of time phrase and is in the same sentence as time phrase will exhibit the durative behaviour. This surprisingly worked very well for tea domain as all actions except `visit` got the correct `duration`. Action `visit`'s duration was assigned to action `wait`. The action `wait` in the original Tea domain [120] is a subcomponent of `visit`. Thus, we can say that all time durations were extracted and modelled correctly in the domain model. The input and extracted action sequences, and the learned domain model are presented in Appendix D. The learned domain model closely resembles the reference model and models the behaviour correctly, except for the preconditions and effects related to missing argument `mug` and implicit static `hand` object in all actions.

5.2.5. Summary of Evaluating Domain Models

Through these demonstrations, we can confirm our Hypothesis 5 which states that we can one-shot induce valid PDDL domain models from an extracted action sequence. As the models miss out on static conditions (optional user-input 3) and suffer from some arguments that were not extracted, these can be termed as **shallow models** and can be used in approaches like **model-lite planning** [138, 143], which learn from incomplete models or as an **initial model** by the SMEs and the KEs to learn some interesting models from NL data.

The soundness and completeness of the learned domain model in terms of extracted action names and action parameters is on par with the testing F1 scores of DQN. However, the soundness and completeness of learned domain model in terms of preconditions and effects is not great due to missing arguments and not learning static conditions, and thus, we rejected Hypothesis 6.

The learned domain models are intuitive in terms of action names, action arguments, and the predicates. However, in terms of preconditions and effects, they are hard to read and extend as user has to track multiple finite state machines without any GUI (rejecting Hypothesis 7). Through user-interaction, we can deal with such issues but first, a user study is required to gauge the effectiveness of learned models in reducing time-consumption for PDDL modelling. Perhaps, **no state observability** is too strong an assumption to learn from unstructured data. Relaxing this observation would allow us to use more robust domain learning algorithms than heuristic learning of LOCM2 [25] but would require learning of the state information from natural language data. Nonetheless, the foundations of NLtoPDDL are laid in a very actively research field of NLP, which makes it possible to do much more than what is demonstrated. One such example that confirms Hypothesis 8 (durative actions) is demonstrated above.

Word2Vec + rEASDRL	BERT + ELMo cEASDRL
NO1: Fire(1) Alarm(2) Instructions(3)	NO1: Fire(1) Alarm(2) Instructions(3)
NO2: Turn-off(1) all(2) hazardous(3) experiments(4) or(5) procedures(6) before(7) evacuating.(8)	NO2: Turn-off(1) all(2) hazardous(3) experiments(4) or(5) procedures(6) before(7) evacuating.(8) <1> Turn-off (experiments)
NO3: If(1) possible(2) take(3) or(4) secure(5) all(6) valuables(7) wallets(8) purses(9) keys(10) etc(11) as(12) quickly(13) as(14) possible.(15)	NO3: If(1) possible(2) take(3) or(4) secure(5) all(6) valuables(7) wallets(8) purses(9) keys(10) etc(11) as(12) quickly(13) as(14) possible.(15) <2> take (valuables) <3> secure (valuables, wallets)
NO4: Close(1) all(2) doors(3) behind(4) you(5) as(6) you(7) exit.(8) <1> Close (doors)	NO4: Close(1) all(2) doors(3) behind(4) you(5) as(6) you(7) exit.(8) <4> Close (doors)
NO5: Check(1) all(2) doors(3) for(4) heat(5) before(6) you(7) open(8) or(9) go(10) through(11) them(12) to(13) avoid(14) walking(15) into(16) a(17) fire.(18) <2> Check (doors) <3> go () <4> avoid (walking)	NO5: Check(1) all(2) doors(3) for(4) heat(5) before(6) you(7) open(8) or(9) go(10) through(11) them(12) to(13) avoid(14) walking(15) into(16) a(17) fire.(18) <5> Check (doors, heat) <6> avoid (walking)
NO6: Evacuate(1) the(2) building(3) using(4) the(5) nearest(6) exit(7) or(8) stairway(9)	NO6: Evacuate(1) the(2) building(3) using(4) the(5) nearest(6) exit(7) or(8) stairway(9) <7> Evacuate (building) <8> using (nearest, exit)
NO7: Do(1) not(2) use(3) the(4) elevators.(5)	NO7: Do(1) not(2) use(3) the(4) elevators.(5)
NO8: Call(1) 911(2) from(3) a(4) safe(5) area(6) and(7) provide(8) name(9) location(10) and(11) nature(12) of(13) emergency.(14) <5> Call (911) <6> provide (name, location)	NO8: Call(1) 911(2) from(3) a(4) safe(5) area(6) and(7) provide(8) name(9) location(10) and(11) nature(12) of(13) emergency.(14) <9> Call (911) <10> provide (name, location)
NO9: Proceed(1) to(2) a(3) pre-determined(4) assembly(5) area(6) of(7) building(8) and(9) remain(10) there(11) until(12) you(13) are(14) told(15) to(16) re-enter(17) by(18) the(19) emergency(20) personnel(21) in(22) charge.(23)	NO9: Proceed(1) to(2) a(3) pre-determined(4) assembly(5) area(6) of(7) building(8) and(9) remain(10) there(11) until(12) you(13) are(14) told(15) to(16) re-enter(17) by(18) the(19) emergency(20) personnel(21) in(22) charge.(23) <11> Proceed (pre-determined, assembly, area) <12> remain (there)
NO10: Do(1) not(2) impede(3) access(4) of(5) emergency(6) personnel(7) to(8) the(9) area.(10)	NO10: Do(1) not(2) impede(3) access(4) of(5) emergency(6) personnel(7) to(8) the(9) area.(10)
NO11: Inform(1) Building(2) Safety(3) Personnel(4) or(5) Emergency(6) Personnel(7) of(8) the(9) event(10) conditions(11) and(12) location(13) of(14) individuals(15) who(16) require(17) assistance(18) and(19) have(20) not(21) been(22) evacuated(23) <7> Inform (Building, Safety, Personnel)	NO11: Inform(1) Building(2) Safety(3) Personnel(4) or(5) Emergency(6) Personnel(7) of(8) the(9) event(10) conditions(11) and(12) location(13) of(14) individuals(15) who(16) require(17) assistance(18) and(19) have(20) not(21) been(22) evacuated(23) <13> Inform (Safety, Personnel)

Figure 5.1: Extraction results of Word2Vec + rEASDRL and chosen BERT+ ELMo cEASDRL using the weights of `WikiHG` dataset. Since many words are not seen by Word2Vec + rEASDRL model, it initialises them to zero vectors and thus, it misses to extract many essential (ES) action names in sentence no. 2, 3, 6 and 9. It also fails to detect some ES action arguments. On the other hand, cEASDRL correctly skips Sentence no. 1 and 10 as they do not represent any ES action names. Thus, even the 5-7% improvement in F1-score corresponds to huge improvements in the overall sequence quality.

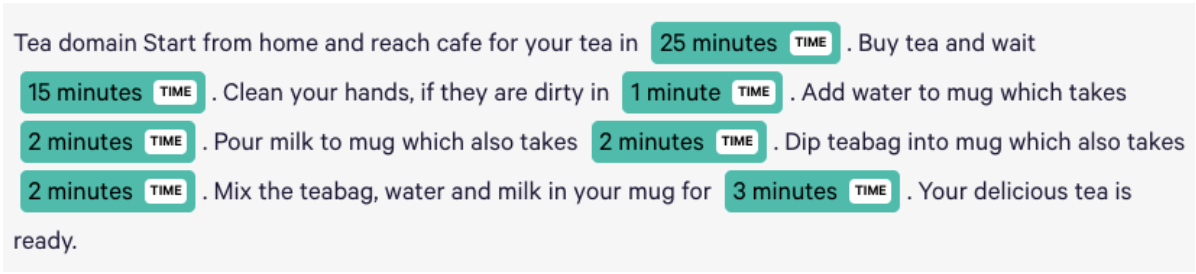


Figure 5.2: Time phrases extracted from NER module of SpaCy library, displayed using $Displacy^{ENT}$ component [61].

6

Conclusion and Future Research

In this chapter, we answer the research questions presented in Section 1.3. Subsequently, the directions for future research are discussed. The main research question of our thesis was:

Can we sufficiently solve the Action Sequence Extraction Problem to extract structured data from freely written natural language process manuals of real-world problems, and then use it to solve the Domain Acquisition Problem to induce syntactically valid and meaningful PDDL models?

We were able to solve for both action sequence extraction and domain acquisition problem by combining promising and state-of-the-art research from natural language processing, deep reinforcement learning and automated planning to induce syntactically valid but **partially** meaningful PDDL models. We achieved the state-of-the-art (SoTA) results in the Action Sequence Extraction Problem by incorporating contextual embeddings into deep reinforcement learning method called EASDRL [33]. In Section 5.1.3, we empirically observed the excellent generalisability of cEASDRL to real-world data. Despite the SoTA results, we were not able to preprocess them into unambiguous structured data. This was the prerequisite to learn domain models from action sequences with no state observability. Nevertheless, we learned meaningful shallow models in one-shot from a natural language process manual which can be used in model-lite planning [138, 143] or as an initial PDDL model by SMEs and KEs to interactively analyse and build the gold-standard PDDL models.

Based on the results, we now answer the sub-research questions:

1. **Can we integrate dynamic contextual word embeddings generated from pretrained language models learned from recent NLP transfer learning techniques, like BERT [29], ELMo [99], and Flair [4], into Feng et al. [33]'s EASDRL to push the state-of-the-art in Action Sequence Extraction? If yes, which contextual embeddings work the best and why?**

   We were able to seamlessly integrate all three dynamic embeddings into Feng et al. [33]'s EASDRL using Flair NLP library [5]. This did push the SoTA F1-scores for extracting action names by 5-7%, and extracting action arguments by 7-9% (Section 5.1.3). Although all contextual embeddings beat the baselines of Word2Vec, GloVe, and POS-GloVe, different embeddings shined for different tasks and datasets. `GloVe+BERT` was the best stacked embedding for extracting action names and `GloVe+ELMo` was the best stacked embedding for extracting action arguments in

`Cooking` and `WinHG` datasets. A possible explanation is that BERT overfits for extracting action arguments due to the large size of RL state-vector, which was caused by the repeat representation explained in Section 4.1.3

2. **Will the dynamic contextual embeddings mitigate the problems of out-of-vocabulary (OOV) words (represented by `UNK`), polysemy, shared representation, and infinite word senses caused by Word2Vec model [85] that was used in EASDRL [33]?**

   The dynamic contextual embeddings did solve the problem of OOV words as they were inherently based on either character convolutions (ELMo and Flair) or sub-word language models (BERT). As contextual embeddings both the context (sentence) of the target word and the language representation learned from unsupervised pretraining on huge corpora, they did solve the problems of polysemy, shared representation and infinite word senses. The only drawback we found was that these could not be saved in a lookup table like Word2Vec.

3. **Would we be able to generalise our action sequence extraction approach on real-world data better than the current state-of-the-art, EASDRL [33]?**

   As shown quantitatively and qualitatively in Sections 5.1.3 and 5.1.4, respectively, we were able to much better generalise our action sequence extraction approach on real-world data than the current state-of-the-art, EASDRL [33]. In other words, contextual embeddings successfully used the knowledge of large scale NL corpus into our downstream task of extracting sequences through DQN, even without task-specific fine-tuning.

4. **Can we use the extracted action sequences as valid structured input for domain model learning technique LOCM2 [25] and induce PDDL models?**

   Yes, we were able to create a valid structured input but it was not representative of the full data that was available in the natural language process manual. Also, “no state observability” assumption of LOCM2[25] closely resembles many real-world scenarios. This also meant that static conditions which can’t be modelled through inductive process could not be learned. As a result, shallow/initial models were induced in one-shot which can be used in model-lite planning [138, 143] or by SMEs and KEs to interactively analyse and build the gold-standard PDDL model from natural language data.

5. **Can we use NLtoPDDL as a one-shot algorithm, i.e., can it also work with only a single natural language plan to produce a best-possible valid output? What is the quality of domain models learned in terms of their completeness, soundness, validity, and consistency? Are the learned domain models intuitive enough that they can be used as initial domain models by the SMEs and KEs?**

   We only performed experiments for one-shot learning. From previous research in AP, we realised that it is too difficult a task to learn domain models in one-shot from natural language data. Hence, we developed an interactive reimplementation of LOCM2, in which the user can follow along the domain learning process and provide simple inputs to improve the quality of the domain. However, in Section 5.2, we did not use these inputs and demonstrated the vanilla NLtoPDDL’s performance for learning domain models. Even without the user input, we were able to induce valid and partially-meaningful shallow PDDL models. The learned PDDL models did not make the best out of the available input. Qualitatively, The precision and recall of the learned action names and action parameters are representative of the F1-scores. The learned

preconditions and effects modelled the behaviour of learned states correctly. Yet, due to missed arguments in extraction and presence of static conditions, many preconditions and effects were not learned.

In Section 5.2.3 and 5.2.4, we moved away from restrictive IPC domains towards real-world domains which were more representative of the data we trained on. We can conclude from the results that our approach can induce one-shot valid PDDL models with reasonable soundness and completeness for real-world instructional data. Despite that, determining the learned state information from state dictionary was found to be a tedious and non-scalable feature, which will be amended in the future by the use of graphical interface that visualises the state of an object in a state machine.

Sometimes, the comparison between the learned domain model and the reference model was not feasible because the domain learner modelled a very different model from the reference model. A further user-study of SMEs and KEs is required to objectively understand if the learned domain model was better.

6. **Can we extend the scope of NLtoPDDL to more expressive features of PDDL, like durative actions?**

   The foundations of NLtoPDDL lie in NLP, which is a very active research field. This makes it possible to do much more than what is demonstrated. One such example was demonstrated in Section 5.2.4, where we learned a temporal domain which contained durative actions. The results were positive for this scenario.

6.1. Future Research

As NLtoPDDL is a flexible pipeline with decoupled components, we have some ideas about enhancing the individual components:

- **Experiments with more data:** The NLtoPDDL approach can work with any amount of natural language data. We only evaluated our pipeline with the least amount of data possible (one-shot learning) and learned simple PDDL models. But, as cEASDRL can extract action sequences from *free* natural language data, it will be interesting to learn domain models form a huge corpus of natural language data and see what they represent or how comprehensible they are. We can validate the quality of generated large domain models by analysing the plans generated from **model-lite** planners [138, 143]. Advantage of having more data is that, we can employ methods like [50] that would also learn the **static conditions** of the domain.

- **Better DRL method**: Although DQNs are sample efficient they do not provide convergence guarantees. For example, in Table 5.3, we saw that DQN failed to converge for POS-GloVe+rEASDRL. Also, memory requirements of DQN were observed to be quite high. Asynchronous architectures like A2C/A3C [88] and architectures that provide strong theoretical guarantees like Trust Region Policy Optimization (TRPO) [106] and Proximal Policy Optimization (PPO) [107] might be better and scalable alternatives.

  One small detail about our model that lowered the domain learning performance was that it was trained on extracting single words and thus, words with adjectives such as “cold milk” or compound nouns such as “training personnel”, were extracted separately which hurt the performance of domain learning algorithm. Thus, by doing some preprocessing and training the model to select multiple words as an action name or argument might facilitate the learning of better domain models.

- **Contextual Embeddings**: The cEASDRL approach that uses EASDRL [33] is fairly generalisable to unseen data, despite the type of contextual embedding applied. Thus, we can customise the contextual embedding to even newer transformer based embeddings which achieve the state-of-the-art results. For example, XLNet [130] which achieves SoTA results in 18 NLP tasks might be a good choice.
- **User-interaction and user-study**: As domain learning algorithms are not mature enough to learn real-world non-classical domains, *mixed-initiative* knowledge engineering is the way to go forward. We took initial steps by incorporating a reimplemented interactive LOCM, but there was not enough time to evaluate it objectively with user input. Our future research would include building a user interface that visualises current state of an object in all the finite state machines that are learned, and then do an extensive user study, with KEs and SMEs as users, to evaluate the different kinds of learned PDDL models which they would build from their interpretation of “free” NL instructions.

Bibliography

[1] Josh Achiam. OpenAI Spinning Up in Deep RL! https://spinningup.openai.com/, 2018. Last accessed: 2019-09-14.

[2] Diego Aineto, Sergio Jiménez, and Eva Onaindia. Learning strips action models with classical planning. In *Twenty-Eighth International Conference on Automated Planning and Scheduling*, 2018.

[3] Diego Aineto, Sergio Jiménez Celorrio, and Eva Onaindia. Learning action models with minimal observability. *Artificial Intelligence*, 275:104–137, 2019.

[4] Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638–1649, 2018.

[5] Alan Akbik, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stefan Schweter, and Roland Vollgraf. FLAIR: An easy-to-use framework for state-of-the-art NLP. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 54–59, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-4010.

[6] Jay Alammar. The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning). http://jalammar.github.io/illustrated-bert/, 2018. Last accessed: 2019-09-14.

[7] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *CoRR*, abs/1707.01495, 2017.

[8] Ankuj Arora, Humbert Fiorino, Damien Pellier, Marc Métivier, and Sylvie Pesty. A review of learning planning action models. *The Knowledge Engineering Review*, 33, 2018.

[9] Ankuj Arora, Humbert Fiorino, Damien Pellier, and Sylvie Pesty. Action model acquisition using lstm. *arXiv preprint arXiv:1810.01992*, 2018.

[10] Ankuj Arora, Humbert Fiorino, Damien Pellier, and Sylvie Pesty. A review on learning planning action models for socio-communicative hri. *arXiv preprint arXiv:1810.09245*, 2018.

[11] Masataro Asai and Alex Fukunaga. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[12] Wilmer Bandres, Blai Bonet, and Hector Geffner. Planning with pixels in (almost) real time. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[13] Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. *CoRR*, abs/1707.06887, 2017.

[14] Sara Bernardini and David E Smith. Developing domain-independent search control for europa2. In *Proceedings of the Workshop on Heuristics for Domain-independent Planning at ICAPS*, volume 7, 2007.

[15] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017. doi: 10.1162/tacl_a_00051.

[16] Adi Botea, Christian Muise, Shubham Agarwal, Oznur Alkan, Ondrej Bajgar, Elizabeth Daly, Akihiro Kishimoto, Luis Lastras, Radu Marinescu, Josef Ondrej, et al. Generating dialogue agents via automated planning. *arXiv preprint arXiv:1902.00771*, 2019.

[17] Satchuthananthavale RK Branavan, Harr Chen, Luke S Zettlemoyer, and Regina Barzilay. Reinforcement learning for mapping instructions to actions. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 82–90. Association for Computational Linguistics, 2009.

[18] Tom Bylander. Complexity results for planning. In *IJCAI*, volume 10, pages 274–279, 1991.

[19] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, and Phillipp Koehn. One billion word benchmark for measuring progress in statistical language modeling. *CoRR*, abs/1312.3005, 2013.

[20] David L Chen and Raymond J Mooney. Learning to interpret natural language navigation instructions from observations. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[21] Steve Chien, Ari Jonsson, and Russell Knight. Automated planning & scheduling for space mission operations. 2005.

[22] Lukáš Chrpa, Daniele Magazzeni, Keith McCabe, Thomas L McCluskey, and Mauro Vallati. Automated planning for urban traffic control: Strategic vehicle routing to respect air quality limitations. *Intelligenza Artificiale*, 10(2):113–128, 2016.

[23] Kevin Clark, Minh-Thang Luong, Christopher D. Manning, and Quoc V. Le. Semi-supervised sequence modeling with cross-view training. *CoRR*, abs/1809.08370, 2018. URL http://arxiv.org/abs/1809.08370.

[24] Michele Colledanchise, Ramviyas Nattanmai Parasuraman, and Petter Ogren. Learning of behavior trees for autonomous agents. *IEEE Transactions on Games*, 2018.

[25] Stephen Cresswell and Peter Gregory. Generalised domain model acquisition from action traces. In *Twenty-First International Conference on Automated Planning and Scheduling*, 2011.

[26] Stephen N Cresswell, Thomas L McCluskey, and Margaret M West. Acquiring planning domain models using locm. *The Knowledge Engineering Review*, 28(2):195–213, 2013.

[27] Will Dabney, Mark Rowland, Marc G. Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. *CoRR*, abs/1710.10044, 2017.

[28] Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. In *Advances in neural information processing systems*, pages 3079–3087, 2015.

[29] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[30] Stefan Edelkamp and Jörg Hoffmann. Pddl2. 2: The language for the classical part of the 4th international planning competition. *4th International Planning Competition (IPC?04), at ICAPS?04*, 2004.

[31] Kutluhan Erol, Dana S Nau, and Venkatramana S Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artificial intelligence*, 76(1-2):75–88, 1995.

[32] Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I. Jordan, Joseph E. Gonzalez, and Sergey Levine. Model-based value estimation for efficient model-free reinforcement learning. *CoRR*, abs/1803.00101, 2018.

[33] Wenfeng Feng, Hankz Hankui Zhuo, and Subbarao Kambhampati. Extracting action sequences from texts based on deep reinforcement learning. *arXiv preprint arXiv:1803.02632*, 2018.

[34] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

[35] Maria Fox and Derek Long. Pddl+: Modeling continuous time dependent effects. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, volume 4, page 34, 2002.

[36] Maria Fox and Derek Long. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research*, 20:61–124, 2003.

[37] Guillem Frances, Hector Geffner, Nir Lipovetzky, and M Ramirez. Best-first width search in the ipc 2018: Complete, simulated, and polynomial variants. *IPC2018–Classical Tracks*, pages 22–26, 2018.

[38] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, Joelle Pineau, et al. An introduction to deep reinforcement learning. volume 11, chapter 4, pages 242–253. Now Publishers, Inc., 2018.

[39] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, Joelle Pineau, et al. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11:219–354, 2018.

[40] Max Franz, Christian T Lopes, Gerardo Huck, Yue Dong, Onur Sumer, and Gary D Bader. Cytoscape. js: a graph theory library for visualisation and analysis. *Bioinformatics*, 32(2):309–311, 2015.

[41] Scott Fujimoto, Herke van Hoof, and Dave Meger. Addressing function approximation error in actor-critic methods. *CoRR*, abs/1802.09477, 2018.

[42] Héctor Geffner. Functional strips: a more flexible language for planning and problem solving. In *Logic-based artificial intelligence*, pages 187–209. Springer, 2000.

[43] Alfonso Gerevini and Derek Long. Plan constraints and preferences in pddl3. Technical report, Technical Report 2005-08-07, Department of Electronics for Automation ?, 2005.

[44] Alfonso Gerevini and Derek Long. Preferences and soft constraints in pddl3. In *ICAPS workshop on planning with preferences and soft constraints*, pages 46–53, 2006.

[45] Alfonso E Gerevini, Patrik Haslum, Derek Long, Alessandro Saetti, and Yannis Dimopoulos. Deterministic planning in the fifth international planning competition: Pddl3 and experimental evaluation of the planners. *Artificial Intelligence*, 173(5-6):619–668, 2009.

[46] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.

[47] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated planning and acting*. Cambridge University Press, 2016.

[48] Dan Goldwasser and Dan Roth. Learning from natural instructions. *Machine Learning*, 94(2):205–232, Feb 2014. ISSN 1573-0565. doi: 10.1007/s10994-013-5407-y.

[49] Arturo González-Ferrer, Juan Fernández-Olivares, Luis Castillo, et al. Jabbah: a java application framework for the translation between business process models and htn. 2009.

[50] Peter Gregory and Stephen Cresswell. Domain model acquisition in the presence of static relations in the lop system. In *Twenty-Fifth International Conference on Automated Planning and Scheduling*, 2015.

[51] Peter Gregory and Alan Lindsay. Domain model acquisition in domains with action costs. In *Twenty-Sixth International Conference on Automated Planning and Scheduling*, 2016.

[52] PJ Gregory, Alan Lindsay, and Julie Porteous. Domain model acquisition with missing information and noisy data. 2017.

[53] David Ha and Jürgen Schmidhuber. World models. *CoRR*, abs/1803.10122, 2018.

[54] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018.

[55] Thomas Hayton, Julie Porteous, Joao Ferreira, Alan Lindsay, and Jonathon Read. Storyframer: From input stories to output planning models. In *Workshop on Knowledge Engineering for Planning and Scheduling (KEPS). The 27th International Conference on Automated Planning and Scheduling (ICAPS)*, 2017.

[56] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[57] Malte Helmert, Gabriele Röger, and Erez Karpas. Fast downward stone soup: A baseline for building planner portfolios. In *ICAPS 2011 Workshop on Planning and Learning*, pages 28–35, 2011.

[58] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[59] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[60] Jörg Hoffmann and Bernhard Nebel. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[61] Matthew Honnibal and Ines Montani. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear, 2017.

[62] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification, 2018.

[63] Rabia Jilani, Andrew Crampton, Diane Kitchin, and Mauro Vallati. Ascol: A tool for improving automatic planning domain model acquisition. In *Congress of the Italian Association for Artificial Intelligence*, pages 438–451. Springer, 2015.

[64] Sergio Jiménez, Fernando Fernández, and Daniel Borrajo. The pela architecture: integrating planning and learning to improve execution. In *National Conference on Artificial Intelligence (AAAI?2008)*, 2008.

[65] Sergio Jiménez, Tomás De la Rosa, Susana Fernández, Fernando Fernández, and Daniel Borrajo. A review of machine learning for automated planning. *The Knowledge Engineering Review*, 27(4):433–467, 2012.

[66] Ari K Jonsson, Paul H Morris, Nicola Muscettola, Kanna Rajan, and Benjamin D Smith. Planning in interplanetary space: Theory and practice. 2000.

[67] Daniel Jurafsky and James H Martin. Speech and language processing: 3rd edition, draft. `https://web.stanford.edu/~jurafsky/slp3/`, 2019. Last accessed: 2019-09-14.

[68] Subbarao Kambhampati. Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain models. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 1601. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

[69] Yaser Keneshloo, Tian Shi, Naren Ramakrishnan, and Chandan K. Reddy. Deep reinforcement learning for sequence to sequence models, 2018.

[70] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[71] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[72] Jiří Kučera and Roman Barták. Louga: learning planning operators using genetic algorithms. In *Pacific Rim Knowledge Acquisition Workshop*, pages 124–138. Springer, 2018.

[73] Yuncong Li and Hankz Hankui Zhuo. Human-in-the-loop domain-model acquisition.

[74] Shiyu Liang and R. Srikant. Why deep neural networks? *CoRR*, abs/1610.04161, 2016.

[75] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2015.

[76] Alan Lindsay, Jonathon Read, Joao F Ferreira, Thomas Hayton, Julie Porteous, and Peter Gregory. Framer: Planning models from natural language action descriptions. In *Twenty-Seventh International Conference on Automated Planning and Scheduling*, 2017.

[77] Nir Lipovetzky and Hector Geffner. Best-first width search: Exploration and exploitation in classical planning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[78] Rui Liu, Junjie Hu, Wei Wei, Zi Yang, and Eric Nyberg. Structural embedding of syntactic trees for machine comprehension. *CoRR*, abs/1703.00572, 2017.

[79] Matt MacMahon, Brian Stankiewicz, and Benjamin Kuipers. Walk the talk: Connecting language, knowledge, and action in route instructions. *Def*, 2(6):4, 2006.

[80] Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60, 2014.

[81] Thomas Leo McCluskey, N Elisabeth Richardson, and Ron M Simpson. An interactive method for inducing operator descriptions. In *AIPS*, pages 121–130, 2002.

[82] Thomas Leo McCluskey, SN Cresswell, N Elisabeth Richardson, and Margaret Mary West. Action knowledge acquisition with opmaker2. In *International Conference on Agents and Artificial Intelligence*, pages 137–150. Springer, 2009.

[83] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. Pddl-the planning domain definition language, 1998.

[84] Hongyuan Mei, Mohit Bansal, and Matthew R Walter. Listen, attend, and walk: Neural mapping of navigational instructions to action sequences. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[85] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[86] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[87] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[88] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[89] Kira Mourao, Luke S Zettlemoyer, Ronald Petrick, and Mark Steedman. Learning strips operators from noisy and incomplete observations. *arXiv preprint arXiv:1210.4889*, 2012.

[90] Christian Muise. Planning. domains. *ICAPS system demonstration*, 2016.

[91] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

[92] Dana S Nau. Current trends in automated planning. *AI magazine*, 28(4):43–43, 2007.

[93] Muhammad Abdul Hakim Newton, John Levine, Maria Fox, and Derek Long. Learning macro-actions for arbitrary planners and domains. In *ICAPS*, volume 2007, pages 256–263, 2007.

[94] Nils J Nilsson. Shakey the robot. Technical report, SRI INTERNATIONAL MENLO PARK CA, 1984.

[95] Christopher Olah. Neural Networks, Types, and Functional Programming. `http://colah.github.io/posts/2015-09-NN-Types-FP/`, 2018. Last accessed: 2019-09-14.

[96] Hanna M Pasula, Luke S Zettlemoyer, and Leslie Pack Kaelbling. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research*, 29:309–352, 2007.

[97] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[98] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[99] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proc. of NAACL*, 2018.

[100] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf*, 2018.

[101] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8), 2019.

[102] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine learning*, 62(1-2):107–136, 2006.

[103] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified modeling language reference manual, the*. Pearson Higher Education, 2004.

[104] Scott Sanner. Relational dynamic influence diagram language (rddl): Language description. *Unpublished ms. Australian National University*, 32, 2010.

[105] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[106] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015.

[107] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[108] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[109] José Á Segura-Muros, Raúl Pérez, and Juan Fernández-Olivares. Learning numerical action models from noisy and partially observable states by means of inductive rule learning techniques. *KEPS 2018*, page 46, 2018.

[110] Jendrik Seipp. Fast downward remix. *Ninth International Planning Competition (IPC 2018)*, pages 67–69, 2018.

[111] Jendrik Seipp and Gabriele Röger. Fast downward stone soup 2018. *IPC2018–Classical Tracks*, pages 72–74, 2018.

[112] M Shah, Lukás Chrpa, Falilat Jimoh, D Kitchin, T McCluskey, Simon Parkinson, and Mauro Vallati. Knowledge engineering tools in planning: State-of-the-art and future challenges. *Knowledge engineering for planning and scheduling*, 53:53, 2013.

[113] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[114] Ron M Simpson, Diane E Kitchin, and Thomas Leo McCluskey. Planning domain definition using gipo. *The Knowledge Engineering Review*, 22(2):117–134, 2007.

[115] David E Smith, Jeremy Frank, and William Cushing. The anml language. In *The ICAPS-08 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*, 2008.

[116] Volker Strobel and Alexandra Kirsch. Planning in the wild: Modeling tools for pddl. In Carsten Lutz and Michael Thielscher, editors, *KI 2014: Advances in Artificial Intelligence*, pages 273–284, Cham, 2014. Springer International Publishing. ISBN 978-3-319-11206-0.

[117] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[118] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[119] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[120] Atif Talukdar. *Inference in Temporal Planning to Enhance Planning Performance for Problems with Required Concurrency*. Phd thesis, 2019.

[121] Andrew Trask, Phil Michalak, and John Liu. sense2vec-a fast and accurate method for word sense disambiguation in neural word embeddings. *arXiv preprint arXiv:1511.06388*, 2015.

[122] Mauro Vallati, Daniele Magazzeni, Bart De Schutter, Lukás Chrpa, and Thomas Leo McCluskey. Efficient macroscopic urban traffic models for reducing congestion: a pddl+ planning approach. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[123] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[124] Tiago S Vaquero, José R Silva, Flavio Tonidandel, and J Christopher Beck. itsimple: towards an integrated design system for real planning applications. *The Knowledge Engineering Review*, 28(2):215–230, 2013.

[125] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[126] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4): 279–292, 1992.

[127] Theophane Weber, Sébastien Racanière, David P. Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, Razvan Pascanu, Peter W. Battaglia, David Silver, and Daan Wierstra. Imagination-augmented agents for deep reinforcement learning. *CoRR*, abs/1707.06203, 2017.

[128] Lilian Weng. Generalized language models. `https://lilianweng.github.io/lil-log/2019/01/31/generalized-language-models.html`, 2018. Last accessed: 2019-09-14.

[129] Qiang Yang, Kangheng Wu, and Yunfei Jiang. Learning action models from plan examples using weighted max-sat. *Artificial Intelligence*, 171(2-3):107–143, 2007.

[130] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding, 2019.

[131] Sungwook Yoon and Subbarao Kambhampati. Towards model-lite planning: A proposal for learning & planning with incomplete domain models. In *ICAPS2007 Workshop on Artificial Intelligence Planning and Learning*, 2007.

[132] Håkan LS Younes and Michael L Littman. Ppddl1. 0: The language for the probabilistic part of ipc-4. In *Proc. International Planning Competition*, 2004.

[133] Ye Zhang, Stephen Roller, and Byron C Wallace. Mgnc-cnn: A simple approach to exploiting multiple word embeddings for sentence classification. In *Proceedings of NAACL-HLT*, pages 1522–1527, 2016.

[134] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015.

[135] Hankui Zhuo, Qiang Yang, Derek Hao Hu, and Lei Li. Transferring knowledge from another domain for learning action models. In *Pacific Rim International Conference on Artificial Intelligence*, pages 1110–1115. Springer, 2008.

[136] Hankui Zhuo, Qiang Yang, and Lei Li. Transfer learning action models by measuring the similarity of different domains. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 697–704. Springer, 2009.

[137] Hankz Hankui Zhuo and Subbarao Kambhampati. Action-model acquisition from noisy plan traces. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

[138] Hankz Hankui Zhuo and Subbarao Kambhampati. Model-lite planning: Case-based vs. model-based approaches. *Artificial Intelligence*, 246:1–21, 2017.

[139] Hankz Hankui Zhuo and Qiang Yang. Action-model acquisition for planning via transfer learning. *Artificial intelligence*, 212:80–103, 2014.

[140] Hankz Hankui Zhuo, Qiang Yang, Derek Hao Hu, and Lei Li. Learning complex action models with quantifiers and logical implications. *Artificial Intelligence*, 174(18): 1540–1569, 2010.

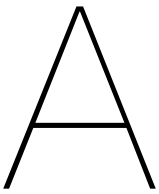
[141] Hankz Hankui Zhuo, Hector Muñoz-Avila, and Qiang Yang. Learning action models for multi-agent planning. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 217–224. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[142] Hankz Hankui Zhuo, Qiang Yang, Rong Pan, and Lei Li. Cross-domain action-model acquisition for planning via web search. In *Twenty-First International Conference on Automated Planning and Scheduling*, 2011.

[143] Hankz Hankui Zhuo, Tuan Nguyen, and Subbarao Kambhampati. Model-lite case-based planning. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[144] Hankz Hankui Zhuo, Tuan Nguyen, and Subbarao Kambhampati. Refining incomplete planning domain models through plan traces. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

[145] Terry Zimmerman and Subbarao Kambhampati. Learning-assisted automated planning: looking back, taking stock, going forward. *AI Magazine*, 24(2):73–73, 2003.



Reference PDDL Domains from IPC

A.1. Child Snack (Sequential, Optimal) - IPC 2014

```
;; Author: Raquel Fuentetaja and Tomas de la Rosa

(define (domain child-snack)
(:requirements :typing :equality)
(:types child bread-portion content-portion sandwich tray place)
(:constants kitchen - place)

(:predicates (at_kitchen_bread ?b - bread-portion)
             (at_kitchen_content ?c - content-portion)
             (at_kitchen_sandwich ?s - sandwich)
             (no_gluten_bread ?b - bread-portion)
             (no_gluten_content ?c - content-portion)
             (ontray ?s - sandwich ?t - tray)
             (no_gluten_sandwich ?s - sandwich)
             (allergic_gluten ?c - child)
             (not_allergic_gluten ?c - child)
             (served ?c - child)
             (waiting ?c - child ?p - place)
             (at ?t - tray ?p - place)
             (notexist ?s - sandwich)
  )

(:action make_sandwich_no_gluten
         :parameters (?s - sandwich ?b - bread-portion ?c - content-portion)
         :precondition (and (at_kitchen_bread ?b)
                            (at_kitchen_content ?c)
                            (no_gluten_bread ?b)
                            (no_gluten_content ?c)
                            (notexist ?s))
         :effect (and
                   (not (at_kitchen_bread ?b))
                   (not (at_kitchen_content ?c))
                   (at_kitchen_sandwich ?s)
                   (no_gluten_sandwich ?s)
                   (not (notexist ?s))
                   ))


(:action make_sandwich
         :parameters (?s - sandwich ?b - bread-portion ?c - content-portion)
```

```
        :precondition (and (at_kitchen_bread ?b)
                           (at_kitchen_content ?c)
                           (notexist ?s)
                           )
        :effect (and
                  (not (at_kitchen_bread ?b))
                  (not (at_kitchen_content ?c))
                  (at_kitchen_sandwich ?s)
                  (not (notexist ?s))
                  ))


(:action put_on_tray
        :parameters (?s - sandwich ?t - tray)
        :precondition (and  (at_kitchen_sandwich ?s)
                            (at ?t kitchen))
        :effect (and
                  (not (at_kitchen_sandwich ?s))
                  (ontray ?s ?t)))


(:action serve_sandwich_no_gluten
       :parameters (?s - sandwich ?c - child ?t - tray ?p - place)
       :precondition (and
                       (allergic_gluten ?c)
                       (ontray ?s ?t)
                       (waiting ?c ?p)
                       (no_gluten_sandwich ?s)
                       (at ?t ?p)
                       )
       :effect (and (not (ontray ?s ?t))
                    (served ?c)))

(:action serve_sandwich
       :parameters (?s - sandwich ?c - child ?t - tray ?p - place)
       :precondition (and (not_allergic_gluten ?c)
                          (waiting ?c ?p)
                          (ontray ?s ?t)
                          (at ?t ?p))
       :effect (and (not (ontray ?s ?t))
                    (served ?c)))

(:action move_tray
        :parameters (?t - tray ?p1 ?p2 - place)
        :precondition (and (at ?t ?p1))
        :effect (and (not (at ?t ?p1))
                     (at ?t ?p2)))


)
```

A.2. Woodworking Subset IPC 2008, Sequential-Optimal Track

```
;; Woodworking subset

(define (domain woodworking)
  (:requirements :typing :action-costs)
  (:types
      acolour awood woodobj machine
      surface treatmentstatus
      aboardsize apartsize - object
      highspeed-saw glazer grinder immersion-varnisher
      planer saw spray-varnisher - machine
      board part - woodobj)

  (:constants
             verysmooth smooth rough - surface
             varnished glazed untreated colourfragments - treatmentstatus
             natural - acolour
             small medium large - apartsize)

  (:predicates
           (unused ?obj - part)
           (available ?obj - woodobj)

           (surface-condition ?obj - woodobj ?surface - surface)
           (treatment ?obj - part ?treatment - treatmentstatus)
           (colour ?obj - part ?colour - acolour)
           (wood ?obj - woodobj ?wood - awood)

           ;...


           (is-smooth ?surface - surface))

  ; .... we only learn a subset

       (:action do-plane
    :parameters (?x - part ?m - planer ?oldsurface - surface
                 ?oldcolour - acolour ?oldtreatment - treatmentstatus)
    :precondition (and
           (available ?x)
           (surface-condition ?x ?oldsurface)
           (treatment ?x ?oldtreatment)
           (colour ?x ?oldcolour))
    :effect (and
           ;;;;;; we ignore costs  (increase (total-cost) (plane-cost ?x))
           (not (surface-condition ?x ?oldsurface))
           (surface-condition ?x smooth)
           (not (treatment ?x ?oldtreatment))
           (treatment ?x untreated)
           (not (colour ?x ?oldcolour))
           (colour ?x natural)))

         (:action do-glaze
    :parameters (?x - part ?m - glazer
                 ?newcolour - acolour)
    :precondition (and
           (available ?x)
           (has-colour ?m ?newcolour)
           (treatment ?x untreated))
    :effect (and
```

```
            ;;; (increase (total-cost) (glaze-cost ?x))
            (not (treatment ?x untreated))
            (treatment ?x glazed)
            (not (colour ?x natural))
            (colour ?x ?newcolour)))



)
```

B

Learned IPC Domain Models

This appendix contains the input (one process manual per domain) and extracted sequence, the learned PDDL model and their state dictionary for our evaluation domains. State dictionary represents the meaning of states in terms of start and end of transitions.

B.1. Child Snack (Sequential, Optimal) - IPC 2014

B.1.1. Input and Extracted Sequence

```
NO1: Jeff(1) Barbara(2) and(3) Nirmal(4) are(5) allergic(6) children(7)
NO2: Shivam(1) and(2) Kanav(3) are(4) non-allergic(5) children.(6)

NO3: Take(1) gluten-free(2) ingredients(3) and(4) gluten-free(5) bread(6)
and(7) make(8) a(9) gluten-free(10) sandwich.(11)
<1>  Take (gluten-free, ingredients)    <2>  make (gluten-free, sandwich.)

NO4: Put(1) gluten-free(2) sandwich(3) on(4) a(5) tray.(6)
<3>  Put (gluten-free, sandwich)

NO5: Move(1) the(2) tray(3) containing(4) sandwich(5) from(6) the(7)
kitchen(8) to(9) the(10) table-5.(11)
<4>  Move (tray)

NO6: Serve(1) the(2) gluten-free(3) sandwich(4) to(5) Jeff.(6)
<5>  Serve (gluten-free, sandwich)

NO7: Move(1) the(2) tray(3) back(4) from(5) the(6) table(7) to(8) the(9) kitchen.(10)
<6>  Move (tray)

NO8: Take(1) gluten(2) ingredients(3) and(4) gluten(5) bread(6)
and(7) make(8) a(9) gluten(10) sandwich.(11)
<7>  Take (gluten, ingredients)    <8>  make (gluten, sandwich.)

NO9: Put(1) gluten(2) sandwich(3) on(4) a(5) tray.(6)
<9>  Put (gluten, sandwich)

NO10: Move(1) the(2) tray(3) from(4) kitchen(5) to(6) the(7) table-11(8)
<10>  Move (tray)

NO11: Serve(1) the(2) gluten(3) sandwich(4) to(5) Shivam.(6)
<11>  Serve (gluten, sandwich)

NO12: Move(1) the(2) tray(3) back(4) from(5) table-11(6) to(7) the(8) kitchen(9)
<12>  Move (tray)
```

```
NO13: Take(1) gluten-free(2) ingredients(3) and(4) gluten-free(5)
bread(6) and(7) make(8) a(9) gluten-free(10) sandwich.(11)
<13>  Take (gluten-free, ingredients)    <14>  make (gluten-free, sandwich.)

NO14: Put(1) gluten-free(2) sandwich(3) on(4) a(5) tray.(6)
<15>  Put (gluten-free, sandwich)

NO15: Move(1) the(2) tray(3) containing(4) sandwich(5) from(6)
the(7) kitchen(8) to(9) the(10) table7.(11)
<16>  Move (tray)

NO16: Serve(1) the(2) gluten-free(3) sandwich(4) to(5) Barbara.(6)
<17>  Serve (gluten-free, sandwich)

NO17: Move(1) the(2) tray(3) back(4) from(5) the(6) table(7) to(8) the(9) kitchen.(10)
<18>  Move (tray)

NO18: Take(1) gluten-free(2) ingredients(3) and(4) gluten-free(5)
bread(6) and(7) make(8) a(9) gluten-free(10) sandwich(11)
<19>  Take (gluten-free, ingredients)    <20>  make (gluten-free, sandwich)

NO19: Put(1) gluten-free(2) sandwich(3) on(4) a(5) tray.(6)
<21>  Put (gluten-free, sandwich)

NO20: Move(1) the(2) tray(3) containing(4) sandwich(5) from(6)
the(7) kitchen(8) to(9) the(10) table-5.(11)
<22>  Move (tray)

NO21: Serve(1) the(2) gluten-free(3) sandwich(4) to(5) Nirmal.(6)
<23>  Serve (gluten-free, sandwich)

NO22: Move(1) the(2) tray(3) back(4) from(5) table-5(6) to(7) the(8) kitchen.(9)
<24>  Move (tray)

NO23: Take(1) gluten(2) ingredients(3) and(4) gluten(5)
bread(6) and(7) make(8) a(9) gluten(10) sandwich.(11)
<25>  Take (gluten, ingredients)    <26>  make (gluten, sandwich.)

NO24: Put(1) gluten(2) sandwich(3) on(4) a(5) tray.(6)
<27>  Put (gluten, sandwich)

NO25: Move(1) the(2) tray(3) from(4) kitchen(5) to(6) the(7) table-11(8)
<28>  Move (tray)

NO26: Serve(1) the(2) gluten(3) sandwich(4) to(5) Kanav.(6)
<29>  Serve (gluten, sandwich)

NO27: Move(1) the(2) tray(3) back(4) from(5) table-11(6) to(7) the(8) kitchen(9)
<30>  Move (tray)
```

B.1.2. Learned Domain Model

```
(define (domain childsnack4)
        (:requirements :typing)
        (:types sandwich-type ingredients tray kitchen sandwich)
        (:predicates
                (sandwich-type_fsm0_state0)
                (sandwich-type_fsm0_state1 ?v0 - sandwich ?v1 - sandwich)
                (sandwich-type_fsm0_state2)
                (ingredients_fsm0_state0)
                (tray_fsm0_state0)
                (kitchen_fsm0_state0)
                (kitchen_fsm0_state1)
                (sandwich_fsm0_state0)
                (sandwich_fsm0_state1)
                (sandwich_fsm1_state0)
        )
        (:action        move
        :parameters     (?tray - tray )
        :precondition   (and
                                (tray_fsm0_state0)
        )
        :effect (and
                                (tray_fsm0_state0)
        ))

        (:action        take
        :parameters     (?gluten-free - sandwich-type ?ingredients - ingredients )
        :precondition   (and
                                (sandwich-type_fsm0_state0)
                                (sandwich-type_fsm0_state1 ?v0 - sandwich ?v1 - sandwich)
                                (sandwich-type_fsm0_state2)
                                (ingredients_fsm0_state0)
        )
        :effect (and
                                (sandwich-type_fsm0_state0)
                                (sandwich-type_fsm0_state1 ?v0 - sandwich ?v1 - sandwich)
                                (sandwich-type_fsm0_state2)
                                (ingredients_fsm0_state0)
        ))

        (:action        serve
        :parameters     (?gluten-free - sandwich-type ?sandwich - sandwich )
        :precondition   (and
                                (sandwich-type_fsm0_state0)
                                (sandwich-type_fsm0_state1 ?v0 - sandwich ?v1 - sandwich)
                                (sandwich-type_fsm0_state2)
                                (sandwich_fsm0_state0)
                                (sandwich_fsm0_state1)
                                (sandwich_fsm1_state0)
        )
        :effect (and
                                (sandwich-type_fsm0_state0)
                                (sandwich-type_fsm0_state1 ?v0 - sandwich ?v1 - sandwich)
                                (sandwich-type_fsm0_state2)
                                (sandwich_fsm0_state0)
                                (sandwich_fsm0_state1)
                                (sandwich_fsm1_state0)
        ))

        (:action        make
```

```
        :parameters     (?gluten-free - sandwich-type ?sandwich - sandwich )
        :precondition   (and
                                (sandwich-type_fsm0_state0)
                                (sandwich-type_fsm0_state1 ?v0 - sandwich ?v1 - sandwich)
                                (sandwich-type_fsm0_state2)
                                (sandwich_fsm0_state0)
                                (sandwich_fsm0_state1)
                                (sandwich_fsm1_state0)
        )
        :effect (and
                                (sandwich-type_fsm0_state0)
                                (sandwich-type_fsm0_state1 ?v0 - sandwich ?v1 - sandwich)
                                (sandwich-type_fsm0_state2)
                                (sandwich_fsm0_state0)
                                (sandwich_fsm0_state1)
                                (sandwich_fsm1_state0)
        ))

        (:action        put
        :parameters     (?gluten-free - sandwich-type ?sandwich - sandwich )
        :precondition   (and
                                (sandwich-type_fsm0_state0)
                                (sandwich-type_fsm0_state1 ?v0 - sandwich ?v1 - sandwich)
                                (sandwich-type_fsm0_state2)
                                (sandwich_fsm0_state0)
                                (sandwich_fsm0_state1)
                                (sandwich_fsm1_state0)
        )
        :effect (and
                                (sandwich-type_fsm0_state0)
                                (sandwich-type_fsm0_state1 ?v0 - sandwich ?v1 - sandwich)
                                (sandwich-type_fsm0_state2)
                                (sandwich_fsm0_state0)
                                (sandwich_fsm0_state1)
                                (sandwich_fsm1_state0)
        ))

)
```

B.1.3. State Dictionary

```
sandwich-type_fsm0_state0:{'start(put.0)', 'end(make.0)'}
sandwich-type_fsm0_state1:{'end(serve.0)', 'start(take.0)', 'end(put.0)', 'start(serve.0)'}
sandwich-type_fsm0_state2:{'end(take.0)', 'start(make.0)'}
ingredients_fsm0_state0:{'start(take.1)', 'end(take.1)'}
tray_fsm0_state0:{'end(move.0)', 'start(move.0)'}
kitchen_fsm0_state0:{'end(move.1)'}
kitchen_fsm0_state1:{'start(move.1)'}
sandwich_fsm0_state0:{'start(make.1)', 'end(put.1)'}
sandwich_fsm0_state1:{'end(make.1)', 'start(put.1)'}
sandwich_fsm1_state0:{'end(serve.1)', 'end(put.1)', 'start(serve.1)',
 'end(make.1)', 'start(make.1)', 'start(put.1)'}
```

B.2. Woodworking Subset - IPC 2008

B.2.1. Input and Extracted Action Sequences

```
NO1: Take(1) very(2) smooth(3) piece(4) of(5) wood.(6)

NO2: Do(1) planing(2) on(3) that(4) wood(5) and(6) use(7) colour(8)
fragment(9) treatment(10) with(11) colour(12) of(13) blue.(14)
<1>  Do (planing)    <2>  use (colour)

NO3: Increase(1) the(2) cost(3) by(4) plane(5) cost.(6)
<3>  Increase (cost, plane)

NO4: Glaze(1) the(2) piece(3) of(4) wood(5) with(6) natural-colored(7)
glaze(8) and(9) treatment(10) of(11) glaze.(12)
<4>  Glaze (piece, wood)

NO5: Increase(1) the(2) cost(3) by(4) glaze(5) cost.(6)
<5>  Increase (cost, glaze)

NO6: Take(1) very(2) smooth(3) piece(4) of(5) wood.(6)
<6>  Take (piece)

NO7: Do(1) planing(2) on(3) that(4) wood(5) and(6) use(7)
colour(8) fragment(9) treatment(10) with(11) colour(12) of(13) mauve.(14)
<7>  Do (planing)    <8>  use (colour)

NO8: Increase(1) the(2) cost(3) by(4) plane(5) cost.(6)
<9>  Increase (cost, plane)

NO9: Glaze(1) the(2) piece(3) of(4) wood(5) with(6) natural-colored(7)
glaze(8) and(9) treatment(10) of(11) glaze.(12)
<10>  Glaze (piece, wood)

NO10: Increase(1) the(2) cost(3) by(4) glaze(5) cost(6)
<11>  Increase (cost, glaze)
```

B.2.2. Learned Domain Model

```
;; woodworking subset
(define (domain woodworking)
        (:requirements :typing)
        (:types planing colour cost plane piece wood)
        (:predicates
                (planing_fsm0_state0)
                (colour_fsm0_state0)
                (cost_fsm0_state0)
                (plane_fsm0_state0 ?v0 - cost)
                (piece_fsm0_state0)
                (piece_fsm0_state1)
                (wood_fsm0_state0 ?v0 - piece)
)
        (:action        take
        :parameters     (?piece - piece )
        :precondition   (and
                                (piece_fsm0_state0)
                                (piece_fsm0_state1)
        )
        :effect (and
                                (piece_fsm0_state0)
                                (piece_fsm0_state1)
        ))
```

```
        (:action        glaze
        :parameters     (?piece - piece ?wood - wood )
        :precondition   (and
                                (piece_fsm0_state0)
                                (piece_fsm0_state1)
                                (wood_fsm0_state0 ?v0 - piece)
        )
        :effect (and
                                (piece_fsm0_state0)
                                (piece_fsm0_state1)
                                (wood_fsm0_state0 ?v0 - piece)
        ))

        (:action        do
        :parameters     (?planing - planing )
        :precondition   (and
                                (planing_fsm0_state0)
        )
        :effect (and
                                (planing_fsm0_state0)
        ))

        (:action        increase
        :parameters     (?cost - cost ?plane - plane )
        :precondition   (and
                                (cost_fsm0_state0)
                                (plane_fsm0_state0 ?v0 - cost)
        )
        :effect (and
                                (cost_fsm0_state0)
                                (plane_fsm0_state0 ?v0 - cost)
        ))

        (:action        use
        :parameters     (?colour - colour )
        :precondition   (and
                                (colour_fsm0_state0)
        )
        :effect (and
                                (colour_fsm0_state0)
        ))

)
```

B.2.3. State Dictionary

```
planing_fsm0_state0:{'start(do.0)', 'end(do.0)'}
colour_fsm0_state0:{'end(use.0)', 'start(use.0)'}
cost_fsm0_state0:{'start(increase.0)', 'end(increase.0)'}
plane_fsm0_state0:{'start(increase.1)', 'end(increase.1)'}
piece_fsm0_state0:{'end(glaze.0)', 'start(take.0)'}
piece_fsm0_state1:{'start(glaze.0)', 'end(take.0)'}
wood_fsm0_state0:{'start(glaze.1)', 'end(glaze.1)'}
```

B.3. Driverlog - IPC2002

This model was induced from structured plans and not natural language data. This was learned to explain the NLtoPDDL in Chapter 4. However, we can see that this model is much less intuitive and understandable than the ones learned from natural language data.

B.3.1. Learned Domain Model

```
(define (domain driverlog2)
        (:requirements :typing)
        (:types driver truck package location)
        (:predicates
                (driver_fsm0_state0)
                (driver_fsm0_state1)
                (truck_fsm0_state0)
                (truck_fsm0_state1)
                (truck_fsm1_state0)
                (package_fsm0_state0)
                (package_fsm0_state1)
                (location_fsm0_state0)
                (location_fsm0_state1)
                (location_fsm0_state2)
                (location_fsm1_state0)
                (location_fsm1_state1)
                (location_fsm2_state0)
                (location_fsm2_state1)
                (location_fsm2_state2 ?v0 - driver)
                (location_fsm3_state0)
                (location_fsm3_state1)
                (location_fsm4_state0)
                (location_fsm4_state1)
                (location_fsm4_state2)
                (location_fsm5_state0)
                (location_fsm5_state1)
                (location_fsm6_state0)
                (location_fsm6_state1)
                (location_fsm7_state0)
                (location_fsm7_state1)
                (location_fsm7_state2)
                (location_fsm8_state0)
                (location_fsm8_state1 ?v0 - driver)
                (location_fsm8_state2)
                (location_fsm8_state3)
                (location_fsm8_state4)
                (location_fsm9_state0)
                (location_fsm9_state1)
                (location_fsm9_state2)
                (location_fsm9_state3)
                (location_fsm10_state0 ?v0 - driver ?v1 - driver)
)
        (:action        load-truck
        :parameters     (?package5 - package ?truck3 - truck ?aula - location )
        :precondition   (and
                                (package_fsm0_state0)
                                (package_fsm0_state1)
                                (truck_fsm0_state0)
                                (truck_fsm0_state1)
                                (truck_fsm1_state0)
                                (location_fsm0_state0)
                                (location_fsm0_state1)
                                (location_fsm1_state0)
                                (location_fsm1_state1)
```

```
                        (location_fsm2_state0)
                        (location_fsm2_state2 ?v0 - driver)
                        (location_fsm3_state0)
                        (location_fsm3_state1)
                        (location_fsm4_state0)
                        (location_fsm4_state1)
                        (location_fsm5_state0)
                        (location_fsm5_state1)
                        (location_fsm6_state0)
                        (location_fsm6_state1)
                        (location_fsm7_state1)
                        (location_fsm8_state1 ?v0 - driver)
                        (location_fsm9_state1)
                        (location_fsm9_state2)
                        (location_fsm10_state0 ?v0 - driver ?v1 - driver)
    )
    :effect (and
                        (package_fsm0_state0)
                        (package_fsm0_state1)
                        (truck_fsm0_state0)
                        (truck_fsm0_state1)
                        (truck_fsm1_state0)
                        (location_fsm0_state0)
                        (location_fsm0_state1)
                        (location_fsm1_state0)
                        (location_fsm1_state1)
                        (location_fsm2_state0)
                        (location_fsm2_state2 ?v0 - driver)
                        (location_fsm3_state0)
                        (location_fsm3_state1)
                        (location_fsm4_state0)
                        (location_fsm4_state1)
                        (location_fsm5_state0)
                        (location_fsm5_state1)
                        (location_fsm6_state0)
                        (location_fsm6_state1)
                        (location_fsm7_state1)
                        (location_fsm8_state1 ?v0 - driver)
                        (location_fsm9_state1)
                        (location_fsm9_state2)
                        (location_fsm10_state0 ?v0 - driver ?v1 - driver)
    ))

    (:action        walk
    :parameters     (?driver1 - driver ?ewi - location ?3me - location )
    :precondition   (and
                        (driver_fsm0_state0)
                        (driver_fsm0_state1)
                        (location_fsm0_state0)
                        (location_fsm0_state1)
                        (location_fsm1_state0)
                        (location_fsm1_state1)
                        (location_fsm2_state0)
                        (location_fsm2_state2 ?v0 - driver)
                        (location_fsm3_state0)
                        (location_fsm3_state1)
                        (location_fsm4_state0)
                        (location_fsm4_state1)
                        (location_fsm5_state0)
                        (location_fsm5_state1)
                        (location_fsm6_state0)
```

```
                        (location_fsm6_state1)
                        (location_fsm7_state1)
                        (location_fsm8_state1 ?v0 - driver)
                        (location_fsm9_state1)
                        (location_fsm9_state2)
                        (location_fsm10_state0 ?v0 - driver ?v1 - driver)
    )
    :effect (and
                        (driver_fsm0_state0)
                        (driver_fsm0_state1)
                        (location_fsm0_state0)
                        (location_fsm0_state1)
                        (location_fsm1_state0)
                        (location_fsm1_state1)
                        (location_fsm2_state0)
                        (location_fsm2_state2 ?v0 - driver)
                        (location_fsm3_state0)
                        (location_fsm3_state1)
                        (location_fsm4_state0)
                        (location_fsm4_state1)
                        (location_fsm5_state0)
                        (location_fsm5_state1)
                        (location_fsm6_state0)
                        (location_fsm6_state1)
                        (location_fsm7_state1)
                        (location_fsm8_state1 ?v0 - driver)
                        (location_fsm9_state1)
                        (location_fsm9_state2)
                        (location_fsm10_state0 ?v0 - driver ?v1 - driver)
    ))

    (:action        disembark-truck
    :parameters     (?driver1 - driver ?truck3 - truck ?sports-center - location )
    :precondition   (and
                        (driver_fsm0_state0)
                        (driver_fsm0_state1)
                        (truck_fsm0_state0)
                        (truck_fsm0_state1)
                        (truck_fsm1_state0)
                        (location_fsm0_state0)
                        (location_fsm0_state1)
                        (location_fsm1_state0)
                        (location_fsm1_state1)
                        (location_fsm2_state0)
                        (location_fsm2_state2 ?v0 - driver)
                        (location_fsm3_state0)
                        (location_fsm3_state1)
                        (location_fsm4_state0)
                        (location_fsm4_state1)
                        (location_fsm5_state0)
                        (location_fsm5_state1)
                        (location_fsm6_state0)
                        (location_fsm6_state1)
                        (location_fsm7_state1)
                        (location_fsm8_state1 ?v0 - driver)
                        (location_fsm9_state1)
                        (location_fsm9_state2)
                        (location_fsm10_state0 ?v0 - driver ?v1 - driver)
    )
    :effect (and
                        (driver_fsm0_state0)
```

```
                        (driver_fsm0_state1)
                        (truck_fsm0_state0)
                        (truck_fsm0_state1)
                        (truck_fsm1_state0)
                        (location_fsm0_state0)
                        (location_fsm0_state1)
                        (location_fsm1_state0)
                        (location_fsm1_state1)
                        (location_fsm2_state0)
                        (location_fsm2_state2 ?v0 - driver)
                        (location_fsm3_state0)
                        (location_fsm3_state1)
                        (location_fsm4_state0)
                        (location_fsm4_state1)
                        (location_fsm5_state0)
                        (location_fsm5_state1)
                        (location_fsm6_state0)
                        (location_fsm6_state1)
                        (location_fsm7_state1)
                        (location_fsm8_state1 ?v0 - driver)
                        (location_fsm9_state1)
                        (location_fsm9_state2)
                        (location_fsm10_state0 ?v0 - driver ?v1 - driver)
))

(:action        board-truck
:parameters     (?driver1 - driver ?truck3 - truck ?aula - location )
:precondition   (and
                        (driver_fsm0_state0)
                        (driver_fsm0_state1)
                        (truck_fsm0_state0)
                        (truck_fsm0_state1)
                        (truck_fsm1_state0)
                        (location_fsm0_state0)
                        (location_fsm0_state1)
                        (location_fsm1_state0)
                        (location_fsm1_state1)
                        (location_fsm2_state0)
                        (location_fsm2_state2 ?v0 - driver)
                        (location_fsm3_state0)
                        (location_fsm3_state1)
                        (location_fsm4_state0)
                        (location_fsm4_state1)
                        (location_fsm5_state0)
                        (location_fsm5_state1)
                        (location_fsm6_state0)
                        (location_fsm6_state1)
                        (location_fsm7_state1)
                        (location_fsm8_state1 ?v0 - driver)
                        (location_fsm9_state1)
                        (location_fsm9_state2)
                        (location_fsm10_state0 ?v0 - driver ?v1 - driver)
)
:effect (and
                        (driver_fsm0_state0)
                        (driver_fsm0_state1)
                        (truck_fsm0_state0)
                        (truck_fsm0_state1)
                        (truck_fsm1_state0)
                        (location_fsm0_state0)
                        (location_fsm0_state1)
```

```
                        (location_fsm1_state0)
                        (location_fsm1_state1)
                        (location_fsm2_state0)
                        (location_fsm2_state2 ?v0 - driver)
                        (location_fsm3_state0)
                        (location_fsm3_state1)
                        (location_fsm4_state0)
                        (location_fsm4_state1)
                        (location_fsm5_state0)
                        (location_fsm5_state1)
                        (location_fsm6_state0)
                        (location_fsm6_state1)
                        (location_fsm7_state1)
                        (location_fsm8_state1 ?v0 - driver)
                        (location_fsm9_state1)
                        (location_fsm9_state2)
                        (location_fsm10_state0 ?v0 - driver ?v1 - driver)
))

(:action        drive-truck
:parameters     (?truck3 - truck ?aula - location ?sports-center - location ?driver1 - drive
:precondition   (and
                        (truck_fsm0_state0)
                        (truck_fsm0_state1)
                        (truck_fsm1_state0)
                        (location_fsm0_state0)
                        (location_fsm0_state1)
                        (location_fsm1_state0)
                        (location_fsm1_state1)
                        (location_fsm2_state0)
                        (location_fsm2_state2 ?v0 - driver)
                        (location_fsm3_state0)
                        (location_fsm3_state1)
                        (location_fsm4_state0)
                        (location_fsm4_state1)
                        (location_fsm5_state0)
                        (location_fsm5_state1)
                        (location_fsm6_state0)
                        (location_fsm6_state1)
                        (location_fsm7_state1)
                        (location_fsm8_state1 ?v0 - driver)
                        (location_fsm9_state1)
                        (location_fsm9_state2)
                        (location_fsm10_state0 ?v0 - driver ?v1 - driver)
                        (driver_fsm0_state0)
                        (driver_fsm0_state1)
)
:effect (and
                        (truck_fsm0_state0)
                        (truck_fsm0_state1)
                        (truck_fsm1_state0)
                        (location_fsm0_state0)
                        (location_fsm0_state1)
                        (location_fsm1_state0)
                        (location_fsm1_state1)
                        (location_fsm2_state0)
                        (location_fsm2_state2 ?v0 - driver)
                        (location_fsm3_state0)
                        (location_fsm3_state1)
                        (location_fsm4_state0)
                        (location_fsm4_state1)
```

```
                        (location_fsm5_state0)
                        (location_fsm5_state1)
                        (location_fsm6_state0)
                        (location_fsm6_state1)
                        (location_fsm7_state1)
                        (location_fsm8_state1 ?v0 - driver)
                        (location_fsm9_state1)
                        (location_fsm9_state2)
                        (location_fsm10_state0 ?v0 - driver ?v1 - driver)
                        (driver_fsm0_state0)
                        (driver_fsm0_state1)
    ))

    (:action        unload-truck
    :parameters     (?package5 - package ?truck3 - truck ?sports-center - location )
    :precondition   (and
                        (package_fsm0_state0)
                        (package_fsm0_state1)
                        (truck_fsm0_state0)
                        (truck_fsm0_state1)
                        (truck_fsm1_state0)
                        (location_fsm0_state0)
                        (location_fsm0_state1)
                        (location_fsm1_state0)
                        (location_fsm1_state1)
                        (location_fsm2_state0)
                        (location_fsm2_state2 ?v0 - driver)
                        (location_fsm3_state0)
                        (location_fsm3_state1)
                        (location_fsm4_state0)
                        (location_fsm4_state1)
                        (location_fsm5_state0)
                        (location_fsm5_state1)
                        (location_fsm6_state0)
                        (location_fsm6_state1)
                        (location_fsm7_state1)
                        (location_fsm8_state1 ?v0 - driver)
                        (location_fsm9_state1)
                        (location_fsm9_state2)
                        (location_fsm10_state0 ?v0 - driver ?v1 - driver)
    )
    :effect (and
                        (package_fsm0_state0)
                        (package_fsm0_state1)
                        (truck_fsm0_state0)
                        (truck_fsm0_state1)
                        (truck_fsm1_state0)
                        (location_fsm0_state0)
                        (location_fsm0_state1)
                        (location_fsm1_state0)
                        (location_fsm1_state1)
                        (location_fsm2_state0)
                        (location_fsm2_state2 ?v0 - driver)
                        (location_fsm3_state0)
                        (location_fsm3_state1)
                        (location_fsm4_state0)
                        (location_fsm4_state1)
                        (location_fsm5_state0)
                        (location_fsm5_state1)
                        (location_fsm6_state0)
                        (location_fsm6_state1)
```

```
                                (location_fsm7_state1)
                                (location_fsm8_state1 ?v0 - driver)
                                (location_fsm9_state1)
                                (location_fsm9_state2)
                                (location_fsm10_state0 ?v0 - driver ?v1 - driver)
        ))

)
```

B.3.2. State Dictionary

```
driver_fsm0_state0:{'end(drive-truck.3)', 'start(disembark-truck.0)', 'start(drive-truck.3)', 'end(board-truck.0)'}
driver_fsm0_state1:{'start(walk.0)', 'start(board-truck.0)', 'end(disembark-truck.0)', 'end(walk.0)'}
truck_fsm0_state0:{'start(board-truck.1)', 'end(disembark-truck.1)'}
truck_fsm0_state1:{'end(board-truck.1)', 'end(drive-truck.0)', 'start(disembark-truck.1)', 'start(drive-truck.0)'}
truck_fsm1_state0:{'end(disembark-truck.1)', 'start(disembark-truck.1)', 'start(load-truck.1)','end(load-truck.1)',
 'start(board-truck.1)', 'end(unload-truck.1)', 'end(drive-truck.0)', 'end(board-truck.1)', 'start(unload-truck.1)',
 'start(drive-truck.0)'}
package_fsm0_state0:{'start(unload-truck.0)', 'end(load-truck.0)'}
package_fsm0_state1:{'start(load-truck.0)', 'end(unload-truck.0)'}
location_fsm0_state0:{'end(walk.1)', 'start(drive-truck.2)'}
location_fsm0_state1:{'start(load-truck.2)', 'end(drive-truck.2)', 'end(load-truck.2)'}
location_fsm0_state2:{'start(walk.1)'}
location_fsm1_state0:{'start(board-truck.2)', 'end(drive-truck.1)', 'start(drive-truck.2)'}
location_fsm1_state1:{'end(board-truck.2)', 'start(drive-truck.1)', 'end(drive-truck.2)'}
location_fsm2_state0:{'end(walk.2)', 'start(walk.1)'}
location_fsm2_state1:{'start(drive-truck.1)'}
location_fsm2_state2:{'end(walk.1)', 'start(walk.2)', 'end(drive-truck.1)'}
location_fsm3_state0:{'end(drive-truck.1)', 'start(drive-truck.2)'}
location_fsm3_state1:{'end(drive-truck.2)', 'end(unload-truck.2)', 'start(drive-truck.1)', 'start(unload-truck.2)'}
location_fsm4_state0:{'end(walk.1)', 'start(drive-truck.2)'}
location_fsm4_state1:{'end(drive-truck.2)', 'end(unload-truck.2)', 'start(unload-truck.2)'}
location_fsm4_state2:{'start(walk.1)'}
location_fsm5_state0:{'end(drive-truck.1)', 'start(drive-truck.2)'}
location_fsm5_state1:{'start(load-truck.2)', 'end(drive-truck.2)', 'start(drive-truck.1)', 'end(load-truck.2)'}
location_fsm6_state0:{'start(board-truck.2)', 'end(disembark-truck.2)', 'end(drive-truck.1)'}
location_fsm6_state1:{'end(board-truck.2)', 'start(disembark-truck.2)', 'start(drive-truck.1)'}
location_fsm7_state0:{'start(walk.2)'}
location_fsm7_state1:{'start(load-truck.2)', 'start(unload-truck.2)', 'end(unload-truck.2)', 'end(load-truck.2)'}
location_fsm7_state2:{'end(walk.2)'}
location_fsm8_state0:{'end(walk.2)'}
location_fsm8_state1:{'start(walk.2)', 'end(drive-truck.1)'}
location_fsm8_state2:{'start(drive-truck.1)'}
location_fsm8_state3:{'end(disembark-truck.2)'}
location_fsm8_state4:{'start(disembark-truck.2)'}
location_fsm9_state0:{'end(walk.1)'}
location_fsm9_state1:{'end(board-truck.2)', 'start(drive-truck.1)'}
location_fsm9_state2:{'start(board-truck.2)', 'end(drive-truck.1)'}
location_fsm9_state3:{'start(walk.1)'}
location_fsm10_state0:{'end(walk.1)', 'end(board-truck.2)', 'start(walk.1)', 'start(walk.2)', 'start(load-truck.2)',
'start(board-truck.2)', 'end(walk.2)', 'end(load-truck.2)', 'start(unload-truck.2)', 'start(drive-truck.1)',
 'end(disembark-truck.2)', 'start(drive-truck.2)', 'end(drive-truck.2)', 'end(unload-truck.2)',
 'start(disembark-truck.2)', 'end(drive-truck.1)'}
```

C

Learned Domain Model from Real-World Fire Safety Process Manual

C.1. Input and Extracted Sequence for One Shot Learning

```
NO1: Fire(1) Alarm(2) Instructions(3)

NO2: Turn-off(1) all(2) hazardous(3) experiments(4) or(5) procedures(6) before(7)
evacuating.(8)
<1> Turn-off (experiments)

NO3: If(1) possible(2) take(3) or(4) secure(5) all(6) valuables(7) wallets(8)
purses(9) keys(10) etc(11) as(12) quickly(13) as(14) possible.(15)
<2> take (valuables)    <3> secure (valuables, wallets)

NO4: Close(1) all(2) doors(3) behind(4) you(5) as(6) you(7) exit.(8)
<4> Close (doors)

NO5: Check(1) all(2) doors(3) for(4) heat(5) before(6) you(7) open(8) or(9) go(10)
through(11) them(12) to(13) avoid(14) walking(15) into(16) a(17) fire.(18)
<5> Check (doors, heat)    <6> avoid (walking)

NO6: Evacuate(1) the(2) building(3) using(4) the(5) nearest(6) exit(7) or(8) stairway(9)
<7> Evacuate (building)    <8> using (nearest, exit)

NO7: Do(1) not(2) use(3) the(4) elevators.(5)

NO8: Call(1) 911(2) from(3) a(4) safe(5) area(6) and(7) provide(8) name(9) location(10)
and(11) nature(12) of(13) emergency.(14)
<9> Call (911)    <10> provide (name, location)

NO9: Proceed(1) to(2) a(3) pre-determined(4) assembly(5) area(6) of(7) building(8)
and(9) remain(10) there(11) until(12) you(13) are(14) told(15) to(16)
re-enter(17) by(18) the(19) emergency(20) personnel(21) in(22) charge.(23)
<11> Proceed (pre-determined, assembly, area)    <12> remain (there)

NO10: Do(1) not(2) impede(3) access(4) of(5) emergency(6) personnel(7)
to(8) the(9) area.(10)

NO11: Inform(1) Building(2) Safety(3) Personnel(4) or(5) Emergency(6)
Personnel(7) of(8) the(9) event(10) conditions(11) and(12)
location(13) of(14) individuals(15) who(16) require(17) assistance(18)
and(19) have(20) not(21) been(22) evacuated(23)
<13> Inform (Safety, Personnel)
```

C.2. Learned Domain Model.

```
define  (domain fire_alarm)
        (:requirements :typing)
        (:types experiments valuables wallets doors heat walking building nearest exit 911 nam
        (:predicates
                (experiments_fsm0_state0)
                (experiments_fsm0_state1)
                (valuables_fsm0_state0)
                (valuables_fsm0_state1)
                (valuables_fsm0_state2)
                (wallets_fsm0_state0)
                (wallets_fsm0_state1)
                (doors_fsm0_state0)
                (doors_fsm0_state1)
                (doors_fsm0_state2)
                (heat_fsm0_state0)
                (heat_fsm0_state1)
                (walking_fsm0_state0)
                (walking_fsm0_state1)
                (building_fsm0_state0)
                (building_fsm0_state1)
                (nearest_fsm0_state0)
                (nearest_fsm0_state1)
                (exit_fsm0_state0)
                (exit_fsm0_state1)
                (911_fsm0_state0)
                (911_fsm0_state1)
                (name_fsm0_state0)
                (name_fsm0_state1)
                (location_fsm0_state0)
                (location_fsm0_state1)
                (pre-determined_fsm0_state0)
                (pre-determined_fsm0_state1)
                (assembly_fsm0_state0)
                (assembly_fsm0_state1)
                (area_fsm0_state0)
                (area_fsm0_state1)
                (there_fsm0_state0)
                (there_fsm0_state1)
                (safety_fsm0_state0)
                (safety_fsm0_state1)
                (personnel_fsm0_state0)
                (personnel_fsm0_state1)
        )
        (:action        provide
        :parameters     (?name - name ?location - location )
        :precondition   (and
        )
        :effect (and
        ))

        (:action        call
        :parameters     (?911 - 911 )
        :precondition   (and
        )
        :effect (and
        ))

        (:action        take
        :parameters     (?valuables - valuables )
```

```
:precondition   (and
                        (valuables_fsm0_state2)
)
:effect (and
                        (valuables_fsm0_state2)
))

(:action        turn-off
:parameters     (?experiments - experiments )
:precondition   (and
)
:effect (and
))

(:action        secure
:parameters     (?valuables - valuables ?wallets - wallets )
:precondition   (and
                        (valuables_fsm0_state2)
)
:effect (and
                        (valuables_fsm0_state2)
))

(:action        using
:parameters     (?nearest - nearest ?exit - exit )
:precondition   (and
)
:effect (and
))

(:action        proceed
:parameters     (?pre-determined - pre-determined ?assembly - assembly ?area - area )
:precondition   (and
)
:effect (and
))

(:action        inform
:parameters     (?safety - safety ?personnel - personnel )
:precondition   (and
)
:effect (and
))

(:action        check
:parameters     (?doors - doors ?heat - heat )
:precondition   (and
                        (doors_fsm0_state0)
)
:effect (and
                        (doors_fsm0_state0)
))

(:action        close
:parameters     (?doors - doors )
:precondition   (and
                        (doors_fsm0_state0)
)
:effect (and
                        (doors_fsm0_state0)
))
```

```
        (:action        avoid
        :parameters     (?walking - walking )
        :precondition   (and
        )
        :effect (and
        ))

        (:action        remain
        :parameters     (?there - there )
        :precondition   (and
        )
        :effect (and
        ))

        (:action        evacuate
        :parameters     (?building - building )
        :precondition   (and
        )
        :effect (and
        ))

)
```

C.3. State Dictionary

```
experiments_fsm0_state0:['end(turn-off.0)']
experiments_fsm0_state1:['start(turn-off.0)']
valuables_fsm0_state0:['start(take.0)']
valuables_fsm0_state1:['end(secure.0)']
valuables_fsm0_state2:['start(secure.0)', 'end(take.0)']
wallets_fsm0_state0:['start(secure.1)']
wallets_fsm0_state1:['end(secure.1)']
doors_fsm0_state0:['start(check.0)', 'end(close.0)']
doors_fsm0_state1:['end(check.0)']
doors_fsm0_state2:['start(close.0)']
heat_fsm0_state0:['end(check.1)']
heat_fsm0_state1:['start(check.1)']
walking_fsm0_state0:['start(avoid.0)']
walking_fsm0_state1:['end(avoid.0)']
building_fsm0_state0:['end(evacuate.0)']
building_fsm0_state1:['start(evacuate.0)']
nearest_fsm0_state0:['end(using.0)']
nearest_fsm0_state1:['start(using.0)']
exit_fsm0_state0:['end(using.1)']
exit_fsm0_state1:['start(using.1)']
911_fsm0_state0:['start(call.0)']
911_fsm0_state1:['end(call.0)']
name_fsm0_state0:['end(provide.0)']
name_fsm0_state1:['start(provide.0)']
location_fsm0_state0:['end(provide.1)']
location_fsm0_state1:['start(provide.1)']
pre-determined_fsm0_state0:['start(proceed.0)']
pre-determined_fsm0_state1:['end(proceed.0)']
assembly_fsm0_state0:['end(proceed.1)']
assembly_fsm0_state1:['start(proceed.1)']
area_fsm0_state0:['end(proceed.2)']
area_fsm0_state1:['start(proceed.2)']
there_fsm0_state0:['start(remain.0)']
there_fsm0_state1:['end(remain.0)']
safety_fsm0_state0:['end(inform.0)']
```

```
safety_fsm0_state1:['start(inform.0)']
personnel_fsm0_state0:['start(inform.1)']
personnel_fsm0_state1:['end(inform.1)']
```

D

Original and Learned Tea Domain with Durative Actions

D.1. Original Tea Domai

The Tea domain has been taken from [120].

```
(define (domain temporalTea)
        (:requirements :strips :typing :equality :durative-actions )

        (:types mug tea teaBag milk water )

        (:predicates
                (addedTo ?m - milk ?mu - mug)
                (atBottomOf ?t - teaBag ?m - mug)
                (containedIn ?w - water ?m - mug)
                (drinkMade ?t - tea)
                (haveDrink ?t - tea)
                (handempty)
                (athome)
                (atcafe)
                (handdirty)
                (atfrontdoorhome)
                (nothanddirty))


        (:durative-action visitcafe
                        :parameters (?t - tea)
                :duration (= ?duration 25)
                :condition (and (at start (athome)) (at end (haveDrink ?t)) )
                :effect (and (at start (not (athome))) (at start (atcafe))
                (at end(athome)) (at end(drinkMade ?t))))

        (:durative-action buytea
                :parameters (?t - tea)
                :duration (= ?duration 15)
                :condition (and (at start (atcafe)) )
                :effect (and (at end (haveDrink ?t)) ))

        (:durative-action getMilk
                :parameters(?m - milk ?mu - mug)
                :duration (= ?duration 2)
                :condition (and (at start (handempty)) (over all(athome)))
                :effect(and (at start (addedTo ?m ?mu)) (at start (not (handempty)))
                (at end (handdirty)) (at end (not (nothanddirty)))))
```

```
        (:durative-action addWater
                :parameters (?w - water ?m - mug)
                :duration (= ?duration 2)
                :condition (and (at start (handempty)) (over all(athome)))
                :effect(and (at start (containedIn ?w ?m)) (at start (not (handempty)))
                (at end (handdirty)) ))

        (:durative-action addTeaBag
                :parameters (?t - teaBag ?m - mug)
                :duration (= ?duration 2)
                :condition (and (at start (handempty)) (over all(athome)))
                :effect(and (at start(atBottomOf ?t ?m)) (at start (not (handempty)))
                (at end (handdirty)) (at end (not (nothanddirty)) )))

        (:durative-action clean
                :parameters ()
                :duration (= ?duration 1)
                :condition (and (at start (handdirty)) (over all(athome)))
                :effect (and (at start (nothanddirty)) (at start (not (handdirty)))
                (at end (handempty))))

        (:durative-action mix
                :parameters (?t - teaBag ?w - water ?m - milk ?mu - mug ?te -tea)
                :duration (= ?duration 3)
                :condition (and (at start (handempty))(at start(addedTo ?m ?mu))
                (at start(atBottomOf ?t ?mu)) (at start(containedIn ?w ?mu))
                (over all(athome)))
                :effect (and (at end(drinkMade ?te)) (at start (not (handempty)))
                (at end (handempty))))

)
```

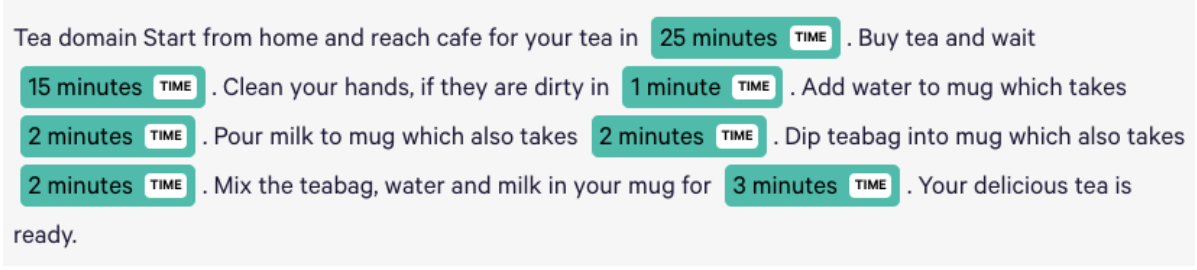
D.2. Learned Tea Domain

D.2.1. Input Sequence and Extracted Sequence for One-Shot Learning

```
NO1: Tea(1) domain(2)

NO2: Start(1) from(2) home(3) and(4) reach(5) cafe(6) for(7) your(8) tea(9) in(10)
25(11) minutes.(12)
<1> Start (home)    <2> reach (cafe)

NO3: Buy(1) tea(2) and(3) wait(4) 15(5) minutes.(6)
<3> Buy (tea)    <4> wait (minutes.)

NO4: Clean(1) your(2) hands(3) if(4) they(5) are(6) dirty(7) in(8) 1(9) minute.(10)
<5> Clean (hands)

NO5: Add(1) water(2) to(3) mug(4) which(5) takes(6) 2(7) minutes.(8)
<6> Add (water)

NO6: Pour(1) milk(2) to(3) mug(4) which(5) also(6) takes(7) 2(8) minutes.(9)
<7> Pour (milk)

NO7: Dip(1) teabag(2) into(3) mug(4) which(5) also(6) takes(7) 2(8) minutes.(9)
<8> Dip (teabag)

NO8: Mix(1) the(2) teabag(3) water(4) and(5) milk(6) in(7) your(8) mug(9) for(10)
3(11) minutes.(12)
<9> Mix (teabag, water, milk)

NO9: Your(1) delicious(2) tea(3) is(4) ready.(5)
```

D.2.2. NER extraction from SpaCy

The extracted time sequences were appended to most recent action. The following figure shows extracted time elements using SpaCy [61]:



D.2.3. Learned Domain Model

```
(define (domain tea_new)
        (:requirements :typing :durative-actions)
        (:types home cafe tea 15_minutes hands water milk teabag)
        (:predicates
                (home_fsm0_state0)
                (home_fsm0_state1)
                (cafe_fsm0_state0)
                (cafe_fsm0_state1)
                (tea_fsm0_state0)
                (tea_fsm0_state1)
                (15_minutes_fsm0_state0)
                (15_minutes_fsm0_state1)
                (hands_fsm0_state0)
                (hands_fsm0_state1)
                (water_fsm0_state0)
                (water_fsm0_state1)
                (water_fsm0_state2)
                (milk_fsm0_state0)
                (milk_fsm0_state1)
                (milk_fsm0_state2)
                (teabag_fsm0_state0)
                (teabag_fsm0_state1)
                (teabag_fsm0_state2)
        )
        (:action        dip
        :parameters     (?teabag - teabag )
        :duration (= ?duration 2)
        :precondition   (and
                                (teabag_fsm0_state1)
        )
        :effect (and
                                (teabag_fsm0_state1)
        ))

        (:action        mix
        :parameters     (?teabag - teabag ?water - water ?milk - milk )
        :duration (= ?duration 3)
        :precondition   (and
                                (teabag_fsm0_state1)
                                (water_fsm0_state1)
                                (milk_fsm0_state1)
        )
        :effect (and
                                (teabag_fsm0_state1)
                                (water_fsm0_state1)
                                (milk_fsm0_state1)
        ))

        (:action        add
        :parameters     (?water - water )
        :duration (= ?duration 2)
        :precondition   (and
                                (water_fsm0_state1)
        )
        :effect (and
                                (water_fsm0_state1)
        ))
```

```
        (:action         start
        :parameters      (?home - home )
        :precondition    (and
        )
        :effect (and
        ))

        (:action         wait
        :parameters      (?15_minutes - 15_minutes )
        :duration (= ?duration 15)
        :precondition    (and
        )
        :effect (and
        ))

        (:action         reach
        :parameters      (?cafe - cafe )
        :duration (= ?duration 25)
        :precondition    (and
        )
        :effect (and
        ))

        (:action         clean
        :parameters      (?hands - hands )
        :duration (= ?duration 1)
        :precondition    (and
        )
        :effect (and
        ))

        (:action         pour
        :parameters      (?milk - milk )
        :duration (= ?duration 1)
        :precondition    (and
                                (milk_fsm0_state1)
        )
        :effect (and
                                (milk_fsm0_state1)
        ))

        (:action         buy
        :parameters      (?tea - tea )
        :precondition    (and
        )
        :effect (and
        ))

)
```

D.2.4. State Dictionary

```
home_fsm0_state0:['end(start.0)']
home_fsm0_state1:['start(start.0)']
cafe_fsm0_state0:['end(reach.0)']
cafe_fsm0_state1:['start(reach.0)']
tea_fsm0_state0:['end(buy.0)']
tea_fsm0_state1:['start(buy.0)']
15_minutes_fsm0_state0:['start(wait.0)']
15_minutes_fsm0_state1:['end(wait.0)']
hands_fsm0_state0:['start(clean.0)']
```

```
hands_fsm0_state1:['end(clean.0)']
water_fsm0_state0:['end(mix.1)']
water_fsm0_state1:['end(add.0)', 'start(mix.1)']
water_fsm0_state2:['start(add.0)']
milk_fsm0_state0:['start(pour.0)']
milk_fsm0_state1:['start(mix.2)', 'end(pour.0)']
milk_fsm0_state2:['end(mix.2)']
teabag_fsm0_state0:['end(mix.0)']
teabag_fsm0_state1:['start(mix.0)', 'end(dip.0)']
teabag_fsm0_state2:['start(dip.0)']
```

E

Architecture of CNN used in cEASDRL

The architecture used CNN (the Q-value estimator of the DQN) in [33] was based on MGNC-CNN [133]. One such instance is for GloVe vectors with 100 dimensions for argument DQN is shown below. It considered 100 words in the first layer, each of which are represented by 300 dimensions due to repeat representation. Then convolutional filters of bigram, trigram, four-gram and five-gram are applied and the output is concatenated. We flatten the layer to 128 dimensions (Feng et al. [33]'s implementation took two convolutions per layer and had 256 dimensions at this layer), and penultimate dense layer of 256 dimensions. Finally, a 2-dimensional final layer expresses RL actions "accept" or "reject".

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	(None, 100, 300, 1)	0	
conv2d_5 (Conv2D)	(None, 99, 1, 32)	19232	input_2[0][0]
conv2d_6 (Conv2D)	(None, 98, 1, 32)	28832	input_2[0][0]
conv2d_7 (Conv2D)	(None, 97, 1, 32)	38432	input_2[0][0]
conv2d_8 (Conv2D)	(None, 96, 1, 32)	48032	input_2[0][0]
activation_5 (Activation)	(None, 99, 1, 32)	0	conv2d_5[0][0]
activation_6 (Activation)	(None, 98, 1, 32)	0	conv2d_6[0][0]
activation_7 (Activation)	(None, 97, 1, 32)	0	conv2d_7[0][0]
activation_8 (Activation)	(None, 96, 1, 32)	0	conv2d_8[0][0]
max_pooling2d_5 (MaxPooling2D)	(None, 1, 1, 32)	0	activation_5[0][0]
max_pooling2d_6 (MaxPooling2D)	(None, 1, 1, 32)	0	activation_6[0][0]
max_pooling2d_7 (MaxPooling2D)	(None, 1, 1, 32)	0	activation_7[0][0]
max_pooling2d_8 (MaxPooling2D)	(None, 1, 1, 32)	0	activation_8[0][0]
concatenate_2 (Concatenate)	(None, 1, 4, 32)	0	max_pooling2d_5[0][0] max_pooling2d_6[0][0] max_pooling2d_7[0][0] max_pooling2d_8[0][0]
flatten_2 (Flatten)	(None, 128)	0	concatenate_2[0][0]
dense_3 (Dense)	(None, 256)	33024	flatten_2[0][0]
dense_4 (Dense)	(None, 2)	514	dense_3[0][0]

Total params: 168,066
Trainable params: 168,066
Non-trainable params: 0