Action Model Learning for Socio-Communicative Human Robot Interaction

Ankuj Arora

Communauté UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTÉ UNIVERSITÉ GRENOBLE ALPES

Spécialité : Mathématiques et Informatique

Arrêté ministériel : 25 mai 2016

Présentée par

Ankuj ARORA

Thèse dirigée par **Sylvie PESTY**, Professeur, UGA

préparée au sein du **Laboratoire Laboratoire d'Informatique de Grenoble**
dans **l'École Doctorale Mathématiques, Sciences et technologies de l'information, Informatique**

Apprentissage du modèle d'action pour une interaction socio-communicative des hommes-robots

Action Model Learning for Socio-Communicative Human Robot Interaction

Thèse soutenue publiquement le **8 décembre 2017**,
devant le jury composé de :

Madame SYLVIE PESTY
PROFESSEUR, UNIVERSITE GRENOBLE ALPES, Directeur de thèse
Monsieur CEDRIC BUCHE
MAITRE DE CONFERENCES, ECOLE NATIONALE D'INGENIEURS DE BREST, Rapporteur
Monsieur ALEXANDRE PAUCHET
MAITRE DE CONFERENCES, INSA ROUEN, Rapporteur
Monsieur DOMINIQUE DUHAUT
PROFESSEUR, UNIVERSITE BRETAGNE-SUD, Président
Monsieur SAMIR AKNINE
PROFESSEUR, UNIVERSITE LYON 1, Examinateur
Madame SOPHIE DUPUY-CHESSA
PROFESSEUR, UNIVERSITE GRENOBLE ALPES, Examinateur
Monsieur DAMIEN PELLIER
MAITRE DE CONFERENCES, UNIVERSITE GRENOBLE ALPES, Examinateur
Monsieur HUMBERT FIORINO
MAITRE DE CONFERENCES, UNIVERSITE GRENOBLE ALPES, Examinateur

University of Grenoble-Alpes

Abstract

Faculty Name
Informatics Laboratory of Grenboble

Doctor of Philosophy

Action Model Learning for Socio-Communicative Human Robot Interaction

by Ankuj Arora

Driven with the objective of rendering robots as socio-communicative, there has been a heightened interest towards researching techniques to endow robots with social skills and "commonsense" to render them acceptable. This "commonsense" is not so common, as even a standard dialogue exchange integrates behavioral subtleties that are difficult to codify. In such a scenario, learning the behavioral model of the robot is a promising approach. This thesis tries to solve the problem of learning robot behavioral models in the Automated Planning and Scheduling (APS) paradigm of AI. During the course of this thesis, we introduce various symbolic and deep learning systems, by the names *SPMSAT* and *PDeepLearn* respectively, which facilitate the learning of action models, and extend the scope of these new techniques to learn robot behavioral models. The long term objective is to empower robots to communicate autonomously with humans without the need of "wizard" intervention.

Résumé en Français Conduite dans le but de rendre les robots comme socio-communicatifs, il y a eu un intérêt accru pour les techniques de recherche pour doter les robots de compétences sociales et de sens commun pour les rendre acceptables. Ce «sens commun» n'est pas si commun, car même un échange de dialogue standard intègre des subtilités comportementales difficiles à codifier. Dans un tel scénario, l'apprentissage du modèle comportemental du robot est une approche prometteuse. Cette thèse tente de résoudre le problème de l'apprentissage des modèles comportementaux du robot dans le paradigme de planification automatisée et d'ordonnancement (APS) de l'IA. Au cours de cette thèse, nous introduisons divers systèmes d'apprentissage symbolique et approfondi, par les noms SPMSAT et PDeepLearn, respectivement, qui facilitent l'apprentissage des modèles d'action et étendent la portée de ces nouvelles techniques pour apprendre les modèles de comportement du robot. L'objectif à long terme d'habiliter les robots à communiquer de manière autonome avec les humains sans avoir besoin d'une intervention "wizard".

Acknowledgements

Firstly, I would like to thank my supervisor Prof. Sylvie Pesty for granting me the opportunity to work on this topic. Further, I am grateful to the MAGMA team, especially Dr. Julie Dugdale, for their continued support during the course of my thesis. I extend my gratitude towards all the members of the ANR-SOMBRERO project as well. I would also like to thank Prof. James Crowley, Prof. Gaelle Calvary and Prof. Denis Trystram for all the love, guidance and support they have shown me during the course of my studies in Grenoble, France.

Last but not the least, I would like to thank the love of my life Dr. Elisa Vitiello, mother Prabha and sister Geetika for standing by my side through all the logical and illogical decisions I have taken throughout my life. Especially to my mother, who has always been my pillar of strength through thick and thin.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

With the near simultaneous advances in mechatronics on the engineering side and ergonomics on the human factors side, the field of social robotics has seen a significant spike in interest in the recent years. Driven with the objective of rendering robots as socio-communicative, there has been an equally heightened interest towards researching techniques to endow robots with cognitive, emotional and social skills. The strategy to do so draws inspiration from study of human behaviors. For robots, social and emotive qualities not only lubricate the interface between humans and robots, but also promote learning, decision making and so on. These qualities strengthen the possibility of acceptability and emotional attachment to the robot (Breazeal, 2003; Breazeal, 2004). This acceptance is only likely if the robot fulfils a fundamental expectation that one living being has of the other: not only to do the right thing, but also at the right time and in the right manner (Breazeal, 2003). This acceptability is seen to play a pivotal role in domains centered on human-robot interaction, such as Socially Assistive Robotics (SAR).

SAR is also becoming indispensable owing to a shortfall in skilled human labor. In order to render robots as assistive, they need to be "programmed" with socially acceptable behaviours. For social robots to be brought more into widespread use in the fields of companionship, care taking and domestic help, they must be capable of demonstrating social intelligence. This social intelligence or "commonsense" of the robot is what eventually determines its social acceptability in the long run.

"Commonsense", however, is not that common. Robots can, thus, only learn to be acceptable with experience. However, teaching a humanoid the subtleties of a social interaction is not evident. Even a standard dialogue exchange integrates the widest possible panel of signs which intervene in the communication and are difficult to codify (synchronization between the expression of the body, the face, the tone of the voice, etc.). In such a scenario, learning the behavioral model of the robot in place of encoding these behaviors is a promising approach. This learning can be performed with the help of AI techniques. This study tries to solve the problem of learning robot behavioral models in the Automated Planning (AP) paradigm of AI. In the domain of Automated Planning (AP), intelligent agents by virtue require an action model (blueprints of action definitions and descriptions whose interleaved executions effectuates transitions of the system state) in order to plan and solve real world problems. During the course of this thesis, we introduce two new learning systems which facilitate the learning of action models, and extend the scope of these new systems to learn robot behavioral models. This is in the interest of a long term objective to introduce behavioral autonomy in robots, such that they can communicate autonomously with humans without the need of "wizard" intervention. This corresponds to one of the milestones set by our funding project, *ANR-SOMBRERO*.

In response to the growing need of rendering robots as socially intelligent, the project

ANR-SOMBRERO[1] proposes to endow humanoid robots with cognitive, emotional and social skills by immersive teleoperation i.e."beaming" of human pilots, interfacing this teleoperated robot with human subjects with the intent of producing interaction data, and learning from this data the underlying behavioral model of the robot. The project has three core phases:

- Perform beaming experiments involving adults interacting with a humanoid robot in cooperative, situated and finalized tasks. The targeted task is a series of tests used in neuropsychological records. We focus on the socio-communicative behavior that should accompany the monitoring of the task, i.e. the correct comprehension of the instructions, the seamless execution of the task by the interlocutor, the positive feedback and encouragement given along the performance and correction for errors, misunderstandings or precisions,

- Develop and implement autonomous socio-communicative behaviors into the robot cognitive architecture via statistical modeling of the multimodal behaviors monitored during the prior robot-mediated interactions,

- Assess these behaviors and the achieved social embodiment with acceptance measures and analysis of user attitudes.

The key feature of this project is the monitoring, characterization and learning of robot-mediated sensory- motor loops via real-time beaming. By shaping pilots' perception and action skills through a robotic embodiment, SOMBRERO offers a unique way to study the social acceptance and usage profiles of robots that do not have yet full-fledged autonomous reasoning, scene understanding and action planning. We now explain the aforementioned phases in finer detail as follows:

- Phase 1: "Beaming" phase
  Our research strategy consists of endowing humanoid robots with cognitive, emotional and social skills via immersive teleoperation (also known as beaming) by human pilots (Argall et al., 2009; Verstaevel et al., 2015; Bailly, Elisei, and Sauze, 2015). Thanks to this technique, a human operator can perceive, analyze, act and interact with a remote person through robotic embodiment. A human operator will solve both the scaling and social problems by optimally exploiting the robots affordances: we acknowledge that humans are more experts in performing social interactions than engineering social models. The experimental scenario consists of a humanoid piloted by a psychologist interacting with patients to determine their level of memory loss. This interaction produces rich multimodal traces. During the beaming stage, the robot passively monitors its sensory-motor state and stores these multimodal traces into a behavioral memory. This process is also illustrated in figure 1.1. Once it has cumulated sufficient experience, semi-automatic machine learning techniques use this behavioral memory to compute behavioral (action) models, explained better in the following phase.

- Phase 2: Multimodal behavior modelling phase
  In this phase, the multimodal traces recorded in the previous phase are used to reconstruct the core behavioral model of a robot, in a bid to automate its future multimodal interactions with humans. The ability to learn the underlying action (behavioral) model which produces these traces could save the effort from having to code these operators from scratch, and promote re usability at the same time. It is very challenging to script

[1]The project context is a part of the SOMBRERO project financed by the French National Research Agency (ANR) with a grant ANR-14-CE27-0014

this behavioral model relying solely on the expertise of a domain expert who also, in his own right, is likely to commit errors while scripting. The effort required by the domain expert to script this subtle and delicate humanoid behavioral model can be diminished by ML. The work done in ML goes hand in hand with the long history of planning, as ML is viewed as a potentially powerful means of endowing an agent with greater autonomy and flexibility. It often compensates for the designer's incomplete knowledge of the world that the agent will face. The developed ML techniques could be applied across a wide variety of domains to speed up planning. The underlying behavioral model is learnt from the experience accumulated during the dialog exchanges between the robot and human. To explain this, it becomes necessary to ignore the definite uncertainty in communication, and view the human utterances as speech acts. By means of these acts, the speaker *plans* to influence the listeners' beliefs, goals and emotional states (Austin, 1975; Cohen and Perrault, 1979) with the intent of fulfilling his own goals. Prosody is heavily interlinked with body movements and gestures which complete the act. For example, the act of greeting somebody can be represented by the sentence "Hello!" followed by the waving of the hand. These interleaved physical and speech acts i.e. multi modal acts can be modeled as operators in a planning system, and frameworks can be developed for providing their semantics (Sadek, 1991; Perrault, 1990). This HRI sequence can be dubbed into the AP paradigm. In the field of AP, agents interact with the environment by executing actions which change the state of the environment, gradually propelling it from its initial state towards the agents' desired goal. The speech act exchange sequence between the robot and human can thus be treated as a sequence of actions which constitutes the plan execution traces (training data). Each multimodal act sequence can thus be viewed as a *plan execution trace*. Thus, the entire HRI can be viewed as plan execution traces in the AP paradigm, with the intent to reconstruct the underlying behavioral (or action) model which serves as the driving force of the interaction. This process is also illustrated in figure 1.2. This is essentially the core contribution of this thesis towards the larger objective of the project as a whole.

- Phase 3: Analysis of Achieved Autonomy and User Feedback
  Once the behavioral model of the robot has been learned, it can be re-used to generate a fresh dialog sequence between the robot and human, this time without the intervention of a skilled pilot to preside over the proceedings. The execution of this dialogue sequence is envisioned to be performed again with the patients of Phase 1. These patients would then be requested to provide their feedback on the quality and fluidity of the interaction. This feedback would help determine the quality or shortcomings of the learning approach, thus providing as a mechanism for continuous quality improvement of the learning algorithms. This process is also illustrated in figure 1.3.

1.2 Problem Statement

While the overall objective of ANR-SOMBRERO is to endow robots with socio-communicative autonomy, this particular study looks to tackle the reconstruction-specific challenges and solve the modelling-specific hindrances which appear in the pipeline towards autonomy. These challenges correspond principally to Phase 2 of the project. This study can be considered as a progressive step towards the attainment of socio-communicative autonomy in robots. Thus, the central aim of this study is: given a set of plan execution traces from an orchestrated HRI, we aim to propose novel learning systems capable of reconstructing the underlying action model. This model can then be used to auto-generate interaction sequences

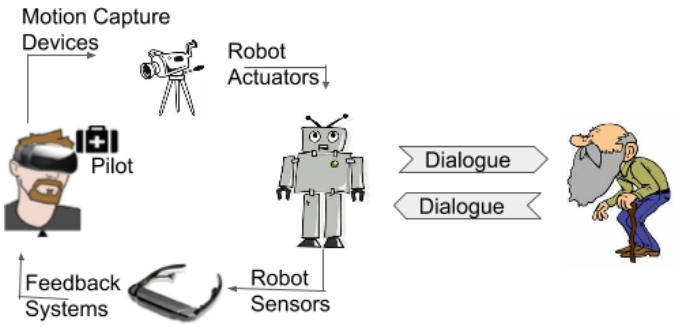

FIGURE 1.1: Step 1 - Initial Run of HRI experiment by beaming (Bailly, Elisei, and Sauze, 2015)

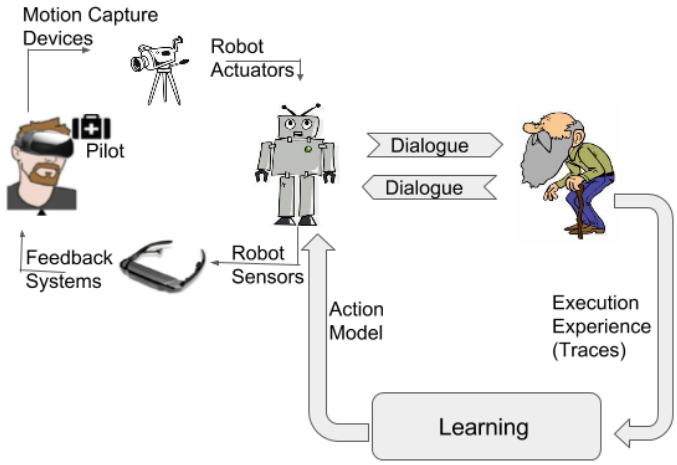

FIGURE 1.2: Step 2 - Learning the behavioral model from collected traces

to drive future autonomous interactions of the robot, subtracting the need for a skilled pilot. Our problem is better illustrated with the Figure 1.4. The left part of the figure represents the input to the reconstruction system, whereas the right part of the figure represents the behavioral model which is the output of the reconstruction system. The syntax and semantics of the behavioral model will be explained in chapter 2. This learnt model can then be fed to a planner which is capable of generating fresh interaction sequences to engineer future Human-Robot Interactions, without the need of a human operator to preside over the robots' vocal interactions (see Figure 1.3).

1.3 Contribution of this Study

This study proposes two novel learning systems which view the problem of reconstruction from two different perspectives and solves them accordingly. The input to both these learning systems are alternating sequences of symbolic representations of (i) executed actions and (ii) the state of the execution environment as a result of the action execution (called the planning state); constituting a *trace*. A *planning state* is the current state of the world represented in the form of ground atoms of predicate logic (called *predicates*) that evaluate to true in that state: all other atoms assumed to be false. Actions have a name and a signature, accompanied by *preconditions* and *effects*, both of which are represented in the form of conjunctions of predicates. An action is applied to a world state by binding all the parameters in the signature to the constants of the world state, creating an action instance. The precondition of

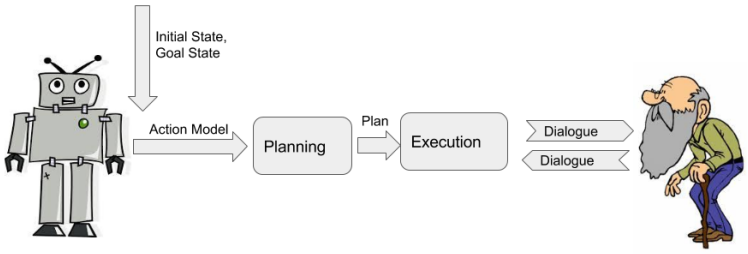

FIGURE 1.3: Step 3 - Autonomous re-run of HRI

this created action instance must be satisfied in the current state, allowing the effects of the action to take charge of changing the world state. The atoms, both positive and negative, comprise the *effects* and are responsible for changing the world state: positive effects make the predicate true in the new state, while negative effects make the predicate false. Each such sequence of executing actions catalyze the gradual progression of the world state from an initial one towards a desired goal state. Summarizing, each sequence of:

- Initial state
- Alternating sequence of action applications and resulting state representations, and
- the goal state

constitutes a *trace*. An *action model* m is the blueprint of all the domain-applicable actions. Each action is defined as an aggregation of:

- the action signature (consisting of the action name with zero or more arguments),
- three lists, namely (pre, add and del). These are the *pre* list (predicates whose satisfiability determines the applicability of the action), *add* list (predicates added to the current system state by the action execution) and the *del* list (predicates deleted from the current system state upon action execution), respectively.

. The goal of both of the learning systems can be illustrated with the following example. AP defines a number of synthetic planning domains (simplified domains in classical planning aimed to reduce the size of the problem space), out of which we use the *gripper* domain to formulate our learning problem (figure 1.5). In this domain, the task of the robot is to move an object from one room to the other. The principal actions in this domain (mentioned on the left of the image) are *(move, pick, drop)*. The principal predicates include: *at* (true if object *?obj* is present in room *?room*), *at-robby* (true if robot *?r* is present in room *?room*), *free* (true if gripper *?g* of robot *?r* is free), *carry* (true if object *?obj* is carried in gripper *?g* of robot *?r*). We represent a snippet of the learnt model with the actions *pick* (a robot picks up an object in a room with either its left or its right gripper) and *move* (a robot moves from one room to another) on the right.

The proposed systems are first tested on AI planning domains (*gripper* being one of them) to validate the conceptual strength of these systems. They are then tested with HRI traces in a bid to validate the applicability of the systems to a real-life scenario. This real-scenario testing comes with its share of issues. Dialogue planning for HRI is not straightforward as the interleaving between various body gestures and utterances which constitute multimodal dialogues is highly delicate (Arora et al., 2016).. In such cases, since the intricacies and subtleties are so barely identifiable, even expert intervention would not be sufficient to render the model reusable. The work plan followed in this study is better illustrated in Figure 1.6.

Our principal contributions in this study are outlined as follows:

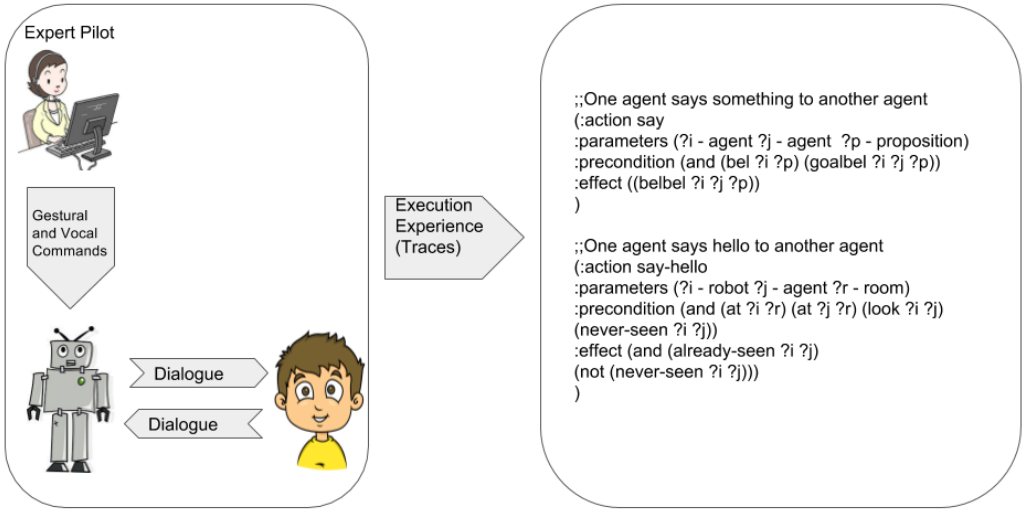

FIGURE 1.4: Objective of the two reconstruction systems proposed in this study. The left block symbolizes the input to the reconstruction system while the right block symbolizes the output of the reconstruction system. The syntax of the learnt model is as follows: this model consists of two actions, namely *sayHello* and *say*. Each action is broadly represented by its signature, preconditions (predicates or properties representing the world state which serve as prerequisites for the execution of the action), and effects (properties which are either added or deleted from the world state following the execution of the action). Each action is syntactically represented by its name (adjacent to the *:action* tag), parameters (adjacent to the *:parameters* tag), preconditions (adjacent to the *:preconditions* tag, consisting of predicate names and signatures), and effects (adjacent to the *:effects* tag). This syntax is written in the PDDL (Planning Domain Description Language) (McDermott et al., 1998) language.

- SAT-based Techniques: The *SRMLearn* System
  To learn the underlying action model, it is possible to exploit intra-action and inter-action dependencies and constraints. While the intra-action constraints deal with the syntax of the action, the inter-action ones exploit the semantic relationship between successive action in a plan sequence. We present an approach called SRMLearn (**S**equential **R**ules-based **M**odel **Learn**er), which encodes the aforementioned constraints in the form of a maximum satisfiability problem (MAX-SAT), and solves it with a MAX-SAT solver to learn the underlying action model. Unlike previous MAX-SAT driven approaches, our chosen constraints exploit the relationship between consecutive actions, rendering more accurately learnt models in the end. This approach is capable of learning purely from plan execution sequences, subtracting the need for domain knowledge. Experimental results highlight that the learnt model is syntactically divergent from the hand woven model (ground truth action model which representing the domains of the benchmarks), thus requiring further refining at the hands of a domain expert. This motivates our search for other techniques which go a step further in reducing the need for human intervention, resulting in the below mentioned learning system called *PDeepLearn*.

- Connectionist Techniques: The *PDeepLearn* System

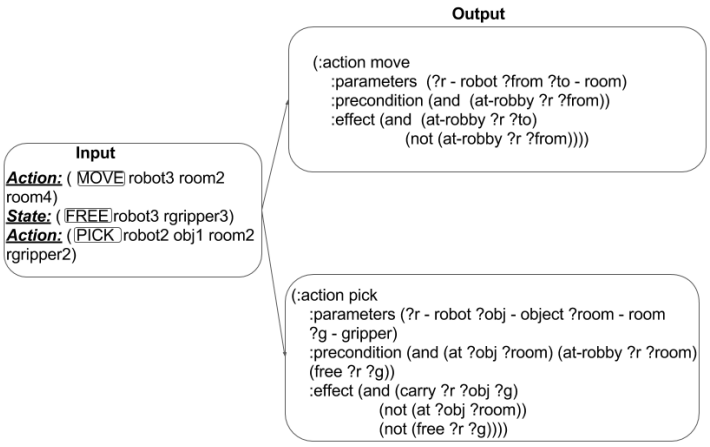

FIGURE 1.5: Illustration of our learning problem

We introduce our approach which is called PDeepLearn, which stands for **P**DDL domain **Deep Learn**er. It uses long short-term memory (LSTM) techniques for the acquisition of the underlying action model. We use the sequence labelling capabilities of LSTMs to predict the next action in a plan execution sequence based on previously observed actions. This is done by iteratively testing the sequence labelling capabilities of each of the models sampled from an exhaustively generated then trimmed set of all possible action models. Empirical results highlight that the sequence labelling achieved with an empirically isolated model speculated to be identical to the hand woven model attains the highest accuracy compared to other models in the set. The structural consistency of the acquired model with the hand woven model renders this LSTM-based sequence labelling approach to learn action models as an effective one.

1.4 Thesis Outline

This section provides a brief overview of the contents of the following chapters.

- **Chapter 2** provides an overview of some notions in Automated Planning (AP). It provides introductory definitions and descriptions of elemental concepts such as *actions, predicates* etc. which are building blocks to more complex concepts such as *action model, plan, trace* etc. It provides an account of the various languages used in AP to represent problem, domain and solution descriptions. These languages are listed with their syntax representations, advantages and shortcomings. Some applications of AP in the real world are highlighted to demonstrate the potential of the field in solving real world problems. We also provide a brief account of the usage of AP in HRI with the intent of planning of socio-communicative acts to endow the robot with interactional abilities in their bid to interface with humans.

- **Chapter 3** provides an account of the literature centered around the learning of action models. It identifies certain key characteristics of learning systems, such as the kind of input, kind of output, format of traces, stochastic nature of environment and actions, presence/absence of background knowledge, family of learning algorithms etc. It uses these characteristics to classify and categorize the algorithms to the best possible extent. The classification is primarily done on the basis of the family of learning algorithms the particular approach belongs to: like inductive learning, analytical learning, reinforcement learning, neural networks etc. These classifications are nourished by the

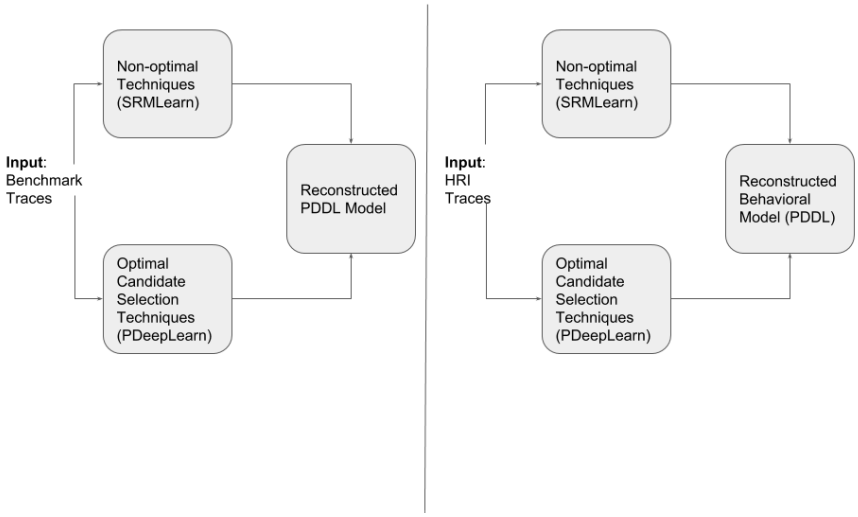

FIGURE 1.6: Work plan followed in this study. The blocks on the left and the right represent the two learning systems introduced during the course of this study, and their eventual application to different sets of data. The introduced systems (non-optimal SAT-based techniques and connectionist techniques) are first tested on benchmark domains available in the field of AP to validate their correctness. These approaches are then applied to HRI traces to reconstruct the underlying behavioral model in the PDDL format.

identified characteristics in order to present to the reader a concrete representation of the learning system, seconded by the characteristics of the individual approaches. We then provide for the ease of the reader: (i) a tabular representation of the learning systems characterized by the aforementioned criteria, and (ii) an algorithmic "cheat sheet" intended to help the reader choose an approach from the literature based on their individual envisioned output and provided inputs. We then provide a brief account of our chosen approaches, justifying our choice vis-a-vis the state of the art.

- **Chapter 4** is pivoted on more classical techniques to learn action models. Since these techniques render an incomplete model in the end, they are labelled as "Non-optimal learning techniques". In this chapter, our objective is to learn the underlying action model that resembles the ground truth model to the highest possible extent, thus facilitating the work of domain experts who would only have to do minimal additions to the learnt model in order to make it usable. To learn this model, it is possible to exploit intra-action and inter-action dependencies and constraints. While the intra-action constraints deal with the syntax of the action, the inter-action ones exploit the semantic relationship between successive actions in a plan sequence. We present an approach called SRMLearn (**S**equential **R**ules-based **M**odel **Learn**er), which encodes the aforementioned constraints in the form of a weighted maximum satisfiability problem (MAX-SAT), and solves it with a weighted MAX-SAT solver to learn the underlying action model. Unlike previous MAX-SAT driven approaches, our chosen constraints exploit the relationship between consecutive actions, rendering more accurately learnt models in the end. Our approach is capable of learning purely from plan execution sequences, subtracting the need for domain knowledge. SRMLearn uses a bare minimum of domain knowledge in the form of state-action interleaved traces in the input to learn an action model as the output. It proceeds in the following fashion: it defines a set of intra-action constraints, both syntactic and semantic in nature. It then defines certain inter-action constraints which exploit short and long-term dependencies between actions in the sequence by means of pattern mining techniques. Finally, it encodes

all the aforementioned constraints in the form of a satisfiability problem and solves the problem with the help of a MAX-SAT solver. The solved constraints are used to reconstruct the underlying action model. While there are other approaches in the literature that learn action models by encoding it in the form of a satisfiability problem (for example ARMS (Yang, Wu, and Jiang, 2007)), empirical results of our evaluation demonstrates a higher accuracy of the learnt model with SRMLearn. However, the results highlight a syntactic divergence of the learnt model from the hand woven model, eliciting the need of a domain expert to "fine tune" the model in order to make it usable. This motivates our search for other techniques which go a step further in reducing the gap between the learnt and ground truth model, resulting in the conception of the below mentioned learning system called *PDeepLearn*.

- **Chapter 5** Speaks about the use of optimal learning techniques to learn action models. Most of the approaches highlighted in the previous chapter start from scratch to build in a brick-by-brick fashion the eventual action model. These, however, are deemed insufficient for they stay shy of learning the entire model, resulting in an incapability to feed the learnt model to a planner to consequently plan again. There is an imperative need of a domain expert to "fine tune" the learnt model to make it ready to plan again. With the intent of taking the human out of the loop, we follow an exhaustive model generation technique, gradually filtering down to the most optimum model which best explains the traces. The exhaustive generation step, although cumbersome and computationally intensive, ensures that there is no lack of information, and there is atleast one model in the exhaustive set which contains all the required information. These exhaustively generated models are then filtered down by means of syntactic and semantic pruning techniques furnished by pattern mining algorithms such as TRuleGrowth (Fournier-Viger, Nkambou, and Tseng, 2011). Since a plan by virtue is a sequential execution of actions which bear dependencies, we use the sequence labelling capabilities of the LSTM (Long Short Term Memory) family of memory based recurrent neural networks (Hochreiter and Schmidhuber, 1997) to identify the most appropriate model which best explains the plan traces. The LSTM is extensively used in the fields of language processing and computer vision as it is ideal to cater to long-term dependencies among entities such as those among words of a sentence and images. Our approach is called PDeepLearn, which stands for **P**DDL domain **Deep Learn**er. Promising results with *PDeepLearn* indicate a definite potential of deep learning techniques to learn action models.

- **Chapter 6** employs the optimal and non-optimal learning techniques introduced in the aforementioned chapters to learn the behavioral model of the robot. This chapter introduces an approach to learn robot behavioral models from HRI traces in the form of PDDL action models using a MAX-SAT framework. This learning approach is very much on the lines of SPMSAT, using intra-action constraints and inter-action ones which have been isolated with the help of pattern mining techniques to learn patterns in temporal time sequences. The Temporal Interval Tree Association Rule Learning (Titarl) algorithm (Guillame-Bert and Crowley, 2012) allows the representation of imprecise and inaccurate temporal information between speech acts considered as symbolic events. It also helps to mine temporal association rules which are used to form soft constraints. All the constraints are encoded in the form of a satisfiability problem and solves the problem with the help of a MAX-SAT solver. The solved constraints are used to reconstruct the underlying action model. The learnt model is used to label speech acts in the HRI exchanges using the sequence labeling capabilities of the LSTM network. The relatively high error rates of the reconstructed model are attributed to the

linear structure of the predicates relative to the actions of the hand woven model, introducing discrepancies in the form of falsified predicates in the output. We conclude that a reworking of the semantics of the hand woven model in terms of the utilized predicates is required to boost the learning rate and further reduce the error. The sequence labeling achieved with the corrected behavioral model is however encouraging, demonstrating the fact that the learnt model represents the most appropriate feature set for the given problem.

- **Chapter 7** provides a closing perspective to the work done in this study. It highlights some of the shortcomings of AP in learning action models and using the learnt models to solve real world problems. It also highlights the shortcomings of our learning approaches when applied to the HRI traces to learn behavioral models. We finish by outlining some future perspectives from the current state of the study.

Chapter 2

Automated Planning- Theory, Language and Applications

2.1 Introduction

Automated planning has been a continuous field of study since the 1960s, since the notion of accomplishing a task using an ordered set of actions resonates with almost every known activity domain. However, as we move from synthetic domains closer to the complex real world, these actions become increasingly difficult to codify. The reasons range from intense laborious effort, to intricacies so barely identifiable, that programming them is a challenge that presents itself much later in the process. In such domains, planners now leverage recent advancements in machine learning to learn action models i.e. blueprints of all the actions whose execution effectuates transitions in the system. This learning provides an opportunity for the evolution of the model towards a version more consistent and adapted to its environment, augmenting the probability of success of the plans. It is also a conscious effort to decrease laborious manual coding and increase quality. This chapter introduces the domain of Automated Planning (AP), the problem of learning action models in AP and characteristics and features of learning systems in the field. These characteristics and criteria serve as a means to classify the learning systems introduced in the next chapter.

2.2 AP Formulation and Representation

Automated Planning (AP) is a means-end reasoning i.e. the process of deciding how to achieve a goal given a set of available and applicable actions. We begin with some definitions of fundamental concepts. The following definitions have been derived from (Ghallab, Nau, and Traverso, 2004).

Definition 1 *(Predicates and Actions).* *Predicates* are properties that constitute the world state and actions. Here, each action a∈A where A = $\{a_1, a_2, \ldots a_n\}$, n being the maximum number of actions in the domain. We use actions and operators interchangably in our context. *States* are defined to be sets of ground (positive) predicates.

Definition 2 *(Classical Planning).* Classical planning is the branch of planning in which predicates are propositional: they do not change unless acted upon by the planning agent. Moreover, all relevant attributes can be observed at any time, the impact of action execution on the environment is known and deterministic, the effects of action execution occur instantly and so on (Zimmerman and Kambhampati, 2003).

Definition 3 *(Action Model).* An *action model* m is the blueprint of all the domain-applicable actions belonging to the set A = $\{a_1, a_2, \ldots a_n\}$. Each action is defined as an aggregation of:

- the action signature (consisting of the action name with zero or more arguments),

- three lists, namely (pre, add and del). These are the *pre* list (predicates whose satisfiability determines the applicability of the action), *add* list (predicates added to the current system state by the action execution) and the *del* list (predicates deleted from the current system state upon action execution), respectively.

Definition 4 *(Planning Problem).* A planning problem is a triplet P = (STS, s_0, g) composed of:

- the initial state s_0 of the world,
- the goal g, (set of propositions representing the goal to achieve), belonging to the set of goal states S_g, and
- the state transition system (STS).

Definition 5 *(STS).* The STS in the previous definition is formally a 3-tuple (S, A, δ) further composed of

- S = $\{s_0, s_1, s_2, \ldots, s_n\}$ as the set of states,
- A = $\{a_0, a_1, a_2, \ldots, a_n\}$ as the set of actions,
- $\delta : (S \times A \rightarrow S)$ is the state transition function (Ghallab, Nau, and Traverso, 2004)).

All the aforementioned elements in this section contribute to the formulation of a plan.

Definition 6 *(Plan).* A plan, given the initial state of the system, goal, and an action model, is a sequence of actions $\pi = [a_1, a_3, a_2, \ldots, a_n]$ that drives the system from the initial state to the goal. This plan is usually generated by a *planner*.

The transition from the initial state to the goal is driven by the previously mentioned transition function in Definition 5 (Ghallab, Nau, and Traverso, 2004) as:

$$\delta(s, \pi) = \begin{cases} s \text{ if } |\pi| = 0 \\ \delta(\delta(s, a_1), [a_2, \ldots a_k]) \text{ if } precond(a_1) \subseteq s \end{cases}$$

This transition function, when applied to the set of the current state and the applicable action, produces the next state as:

$$s_{i+1} = \delta(s_i, a) = (s_i - del(a)) \cup add(a)$$

This transition function, when successively applied to the resulting intermediate states at each step, leads to the goal.

We explain the aforementioned definitions in the form of a concrete example. AP defines a certain number of artificial domains, one of the most famous ones being *Blocksworld*. It consists of a set of blocks resting on a table, the goal being to build one or more piles of blocks. However, only one block may be moved at a time: either placed on top of another block or on the table. A sample problem in the *Blocksworld* domain with its solution sequence is represented in the figures 2.1 and 2.2.

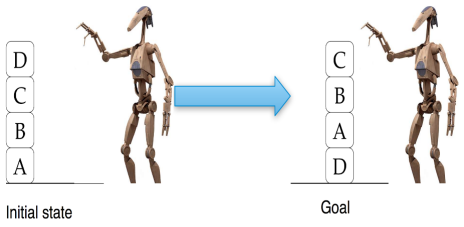

FIGURE 2.1: Problem definition. The objective of the planner is to generate a plan which will allow the robot to re-order the blocks constituting the stack in such a way that the block D from the top of the stack (initial state of system) is placed at the bottom of the stack (goal of the system)

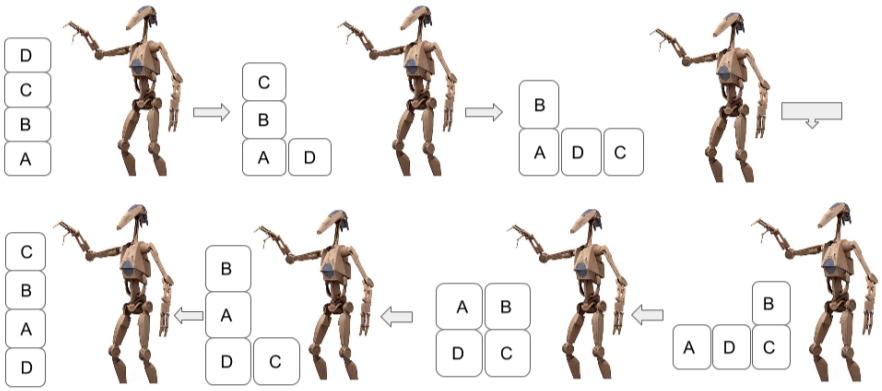

FIGURE 2.2: Solution steps. In this sequence of steps, the robot successively unstacks all the blocks and places them on the ground, after which it proceeds to restack them, keeping block D at the bottom. Each step is representative of a system state as successive stacking and unstacking actions propel the system to the goal.

2.3 Languages in Automated Planning

AP problems and domains are typically represented in a standard language called the Planning Domain Definition Language (PDDL). It has been the official language for the representation of the problems and solutions in all of the International Planning Competitions (IPC) which have been held from the year 1998 onwards.

The PDDL representation of the actions and the sequence of steps leading to the final block configuration can be summarized in the figure 2.3. The left of the figure represents the sequence of actions to achieve the goal mentioned in Figure 2.1: *pick-up, putdown, stack* and *unstack*. The right of the figure represents a magnified view into two of the four actions, namely $stack$ and $unstack$. These actions are constituted by the following predicates: *holding* (true if the block is being held), *clear* (true if the block does not have another block sitting on top of it), *arm-empty* (true if the arm stacking and un-stacking the blocks is empty) and *on* (true if one block is on top of another). In the PDDL representation of these actions, the name of the action follows the : $action$ tag. Similarly, the parameters constituting the action signature follow the : $parameters$ tag. The preconditions and effects of the action are represented as a conjunction of predicates follow the *:precondition* (equivalent to the *pre* list in Definition 5) and *:effect* (equivalent to the *add* and *del* lists in Definition 5) tags respectively.

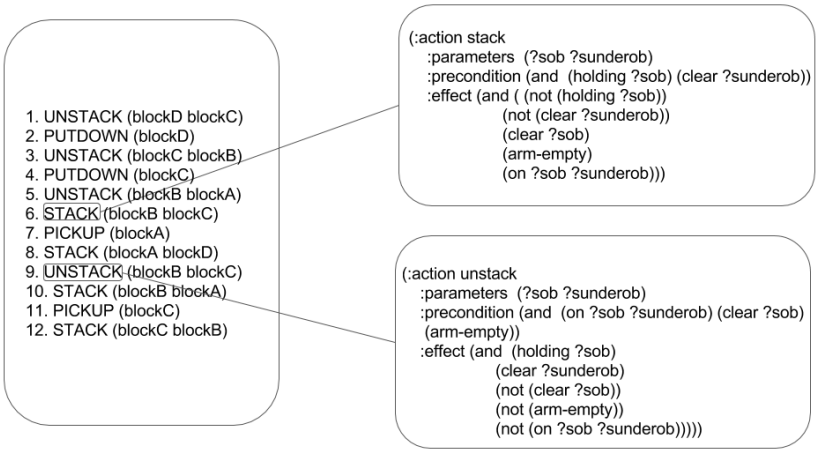

FIGURE 2.3: PDDL Representation Of the actions *stack* and *unstack* in the *Blocksworld* domain

Choosing a good representation language which accurately models every action effect the system might encounter and no others is essential. Any extra modeling capacity is wasted and complicates learning, since the system will have to consider a larger number of potential models, and be more likely to overfit (Pasula, Zettlemoyer, and Kaelbling, 2007). Some languages in the domain of AP and their features are summarized in the table 2.1. We return to our example of the *Blocksworld* domain, and refer to the action *pick-up* which represents the picking up of a block from the table or from another block. A syntactic representation of the action *pick-up* of the *Blocksworld* domain in six prominent languages (described in table 2.1) can be found in table 2.2. An analysis of the language formats facilitates our choice of the representation language. From the descriptions, characteristics as well as the limitations of the various languages, we choose PDDL as our choice of representation language. Since we intend to learn deterministic models with classical representations of actions, predicates, types and constants; a language on the lines of STRIPS only flexible enough to include typing is sufficient for our learning problem. This is our key reason behind our choice of PDDL as a representation language.

TABLE 2.1: Representation Languages in AP (PDDL = Planning Domain Definition Language, PPDDL = Probabilistic PDDL, OCL = Object Centered Language, RDDL = Relational Dynamic Diagram Language, ADL = Action Description Language)

Language	Features	Limitations
PDDL (McDermott et al., 1998)	(i) Machine-readable, standardized syntax for representing STRIPS and other languages. (ii) Has types, constants, predicates and actions	Goal cannot be included into action models
PPDDL (Younes et al., 2005)	(i) Extension of PDDL2.1 as the standard high level planning language for probabilistic planners (ii) Supports actions with probabilistic effects	Goal cannot be included into action models
STRIPS (Fikes and Nilsson, 1971)	Sublanguage of PDDL	(i) Accounts for deterministic not probabilistic actions (ii) Requires exponential number of samples to learn
OCL (McCluskey, Richardson, and Simpson, 2002)	High level language with representation centered around objects instead of states	Cannot be input into standard planners
RDDL (Sanner, 2010)	STRIPS + functional terms, leading to higher expressiveness	Does not cater to non-determinism
ADL (Pednault, 1989)	STRIPS operators augmented with quantifiers and conditional effects, resulting in situational calculus-like expressiveness	Computational efficiency proportional to knowledge of initial state

Language	Domain	Action Definition
PDDL	*%Definition of the blocksworld domain* (define (domain BLOCKSWORLD) *%Declaration of used packages (strips, typing)* (:requirements :strips :typing)	*% Action definition* (:action pick-up :parameters (?x - block) *% Preconditions for action applicability* :precondition (and (clear ?x) (ontable ?x) (handempty)) *% Action-induced object transitions* :effect (and (not (ontable ?x)) (not (clear ?x)) (not (handempty)) (holding ?x)))
PPDDL	*%Declaration of constant objects* (:types block) *% Declaration of symbolic facts* (:predicates (on ?x - block ?y - block) (ontable ?x - block) (clear ?x - block) (handempty) (holding ?x - block))	*% Action definition* (:action pick-up :parameters (?x - block) *% Preconditions for operator applicability* :precondition (and (clear ?x) (ontable ?x)) *% Action-induced object transitions* :effect (and (when (not (handempty)) (probabilistic 0.00) (and (clear ?x) (ontable ?x)) (when (handempty) (probabilistic 1.00) (and (not (ontable ?x)) (not (clear ?x)) (not (handempty)) (holding ?x)))
STRIPS	*%Definition of the blocksworld domain* (define (domain BLOCKSWORLD) *%Declaration of used packages (strips)* (:requirements :strips) *% Declaration of symbolic facts* (:predicates (on ?x ?y) (ontable ?x) (clear ?x) (handempty) (holding ?x)) *% Action definition* (:action pick-up :parameters (?x) *% Preconditions to apply operator* :precondition (and (clear ?x) (ontable ?x)(handempty)) *% Action-induced object transitions* :effect (and (not (ontable ?x)) (not (clear ?x)) (not (handempty)) (holding ?x)))	*Action representation in STRIPS* :action (pick-up (b) $PRECOND$: $clear(b) \wedge ontable(b) \wedge handempty$ $EFFECT$: $\neg ontable(b) \wedge \neg clear(b)$ $\wedge \neg handempty \wedge holding(x)$

OCL	*% objects (Type and instances of each object)* objects (block, [block1]) objects(gripper, [tom]) *% All predicates* predicates([on_block (block,block), on_table (block), clear (block), gripped (block,gripper), busy (gripper), free (gripper)])	*%Substrate Class (sc): "typical situations" an object of a particular sort may exist in due to planning process* operator (grip_from_table (B,G) *% prevail (preconditions for operator applicability)* [] *% necessary (transitions for objects changed by action)* [sc (block, B,[on_table (B),clear (B)] =>[gripped (B,G)]), sc (gripper,G,[free (G)] =>[busy (G)])] *% conditional (optional conditions)* [])

TABLE 2.2: Representation of action "pick-up" of the *Blocksworld* domain in various languages

2.4 Applications of AP

This section highlights some of the feats of AP in terms of deployments in real-world applications and serving actual needs.

Planning has found its place into an array of real world applications which range from human multi-drones search-and-rescue missions (Bevacqua et al., 2015), single-operator multi-aircraft guidance (Strenzke and Schulte, 2011) to military evacuation operations (Muñoz-Avila et al., 1999).

Robotics is one of the most appealing application areas for AP. The fast-developing world of social robotics has cited planning and its ability to scale to real-world problems posed by robotics as a key issue to resolve. This is because most work done in AP is focused on action planning, whereas actions are not the only aspects that a robot reasons upon. Planning also needs to account for temporal, causal, resource and information dependencies between computational, perceptual, and actuation components (Di Rocco, Pecora, and Saffiotti, 2013). This is increasingly falling into place. For example, the ROBIL team in the DARPA robotics challenge (DRC) (www.darpa.mil/NewsEvents/Releases/2012/10/24.aspx) used hierarchical plans for runtime monitoring of resources (Cohen, Shimony, and Weiss, 2015). The DARPA Grand Challenge is pivoted on path planning for autonomous vehicles, thus is beyond the scope of this work. All in all, these works are gradual steps towards conceptualizing autonomous systems.

Achieving autonomous systems is one of the most prominent challenges of AP. AI planners have been successfully used in a wide range of applications with the intent of contributing towards the creation of autonomous systems. However AP alone is not sufficient to achieve autonomy, and requires the amalgamation of perception, comprehension and projection capabilities in order to achieve contextual awareness leading to fully autonomous behavior (Endsley and Garland, 2000). From the perspective of AI planning and within the scope of this work, we present a list of non-exhaustive guidelines that a robotic system can follow in order to be proclaimed autonomous:

- Capability of Autonomous Exploration and Conservative Planning: a robot must be capable of observing its surroundings using onboard sensors, selecting from a set of candidate locations, planning a trajectory and autonomously navigating to the selected location (Gregory and Lindsay, 2016). Conservative planning is a method that seeks to exploit the classical planning framework, while simultaneously recognizing the opportunities (unexpected events). These opportunities are then used to increase the plan utility during its execution (Cashmore et al., 2016a).
- Persistent Autonomy: a robot must be capable of planning long-term behaviour and performing activity over extended periods of time without human intervention (Cashmore et al., 2016b).
- Interleaving with other tasks: This entails the combination of two types of planners: AI task planners that handle the high-level symbolic reasoning part and motion planners that plan the movements in space and check the geometrical feasibility of the plans output by the task planners (Ferrer-Mestres, Frances, and Geffner, 2015). It also entails concurrent planning and execution abilities so that the robot can achieve its goal as quickly as possible.

AP techniques have also been used extensively in the field of space exploration, notably in satellite, rover and spacecraft missions. For example, the Deep Space One employed a constraint-based integrated temporal planner and resource scheduler which performed periodic planning to manage resources and develop plans to achieve goals in a timely manner (Muscettola et al., 1998), (Pell et al., 1997).

2.5 Automated Planning in HRI

On the robotics side, there has been considerable work done on usage of state-of-the-art supervised (Young et al., 2013) and non-supervised (Fung and Liu, 2003) machine learning techniques for continuous sensorimotor spaces, task-oriented robot behavior learning (Fung and Liu, 2003) and so on. Our work, however, is concentrated on the learning of PDDL (Planning Domain Description Language (McDermott et al., 1998), syntax for planning domains) based behavioral models for robots. In other words, it is the extension of the notion of learning action models in AP and applying it to the HRI domain.

In general, AP methods can help an AI system to reason about how to interact. By equating the system's communicative goals to goals of planning problems, appropriate plans that solve these problems can be created. Because effects of both linguistic (e.g., factual description of a location) and physical (e.g., locomotive acts enabling an addressee to arrive at that location) nature could result from an utterance, planning in HRI involves a mix of linguistic and non-linguistic elements. This fundamental form of reasoning, centric to both human and artificial intelligence, has been employed in computational models of natural language in different ways. On the linguistic end, AP has principally been used in the case of natural language generation and understanding. Early work has shown how speakers' acts of producing certain types of utterances can be modeled in terms of planning actions, and how such acts can be automatically generated. Later work enriched this kind of planning in two principal ways: with physical acts of some sort, with the intent of capturing the speakers' behaviour; and with a grammar, with the intent of constructing, word by word, full sentences that obey the grammar rules. Another line of research has employed planning for natural language understanding instead of generation, inferring a speaker's plans (thus their desired communicative goals) by observing their actions (Garoufi, 2014).

Sticking to the case of natural language generation which is closer to our cause, the generation of a meaningful utterance is what termed a speech act (Austin, 1975). Since producing

utterances is much like performing speech acts, and AP can be used to figure out which sequences will achieve a desired goal, the question arises whether planning methods, given the action model and a planning problem as input, can be applied to the automatic generation of utterances. (Perrault and Allen, 1980) were among the first to explore this question, arguing that the same processes used to construct plans of physical actions could also be used to construct interleaved sequences of speech acts. They showed how techniques from classical planning could be employed to the generation of speech acts for the satisfaction of a speaker's communicative goals. The authors focused on requesting and furnishing information in a cooperative task-based dialog setting. Modeling speech acts as planning actions based on the agents' mental states has been the topic of a considerable amount of later work (e.g., (Cohen and Levesque, 1988) refine the semantics of speech acts using a modal temporal logic). This expressive formalism, which is based on philosophical foundations laid out by (Bratman, 1987), formalizes the principles of rational action using reasoning about the beliefs, desires, and intentions (BDI) of agents participating in dialog, thus referred to as BDI-based framework of communicative planning.

Bratman principally defines *intention*, an attitude whose key role is to influence an agent to performing rational action. Intention instigates action rather than merely influencing it. It is"a distinctive attitude, not to be conflated with or reduced to ordinary desires and beliefs" (Bratman, 1987). Desires merely influence behavior because they than contribute to a sum of competing as well as cooperating forces, whereas intentions do not have to deal with competing intentions, thus retaining control. The functional role of intention is more comprehensible in case of plans for the future rather than in the case of current-day or immediate ones. The role of future-directed plans is to help an agent to solve a practical problem incrementally and in advance, rather than at the moment when an action is indispensable. A downside of the BDI approach is that it typically does not address uncertainty, which is a key component of real-world interactions. Among other works, (Steedman and Petrick, 2007) generate speech acts using (Petrick and Bacchus, 2002) contingent planning with knowledge and sensing (PKS) system, aimed at constructing conditional plans to cover all possible surprises. AP-based methods in the BDI paradigm have also been applied (Benotti and Blackburn, 2011) use the classical FF planner (Hoffmann and Nebel, 2001) to construct plans from which conversational implicatures can be inferred. The literature also speaks of other approaches which have treated dialogues as planning operators. In (Brenner, 2003) the authors introduce a new language called MAPL (Multi Agent Programming Language) which represents speech acts as operators with qualitative and quantitative temporal relations between them. Other works also use AP to generate natural language sentences for communication (García-Martínez and Borrajo, 2000; Garoufi and Koller, 2010; Marques and Rovatsos, 2016).

AP has been used to reinforce the communion between communicative and physical act planning in the case of natural language generation, where language unfolds in the context of a physical environment that the communicating agents share. In such an environment, non-linguistic aspects of context like the agents' spatial position can have a direct impact on the type and form of language they choose to produce; spoken language can in turn have its own impact on the agents' future physical actions. Situated language planning thus becomes part of a more general architecture for an agent's behavior planning, integrating both speech and physical acts. Ideally, depending on their levels of embodiment, such agents must be able to switch seamlessly among the planning of speech acts, the planning of physical acts, the execution of these plans, and the observation of their environment. (Brenner and Kruijff-Korbayová, 2008) approach this problem with an algorithm for continual multi-agent planning (Brenner and Nebel 2009). Functioning along the lines of continual planning, agents do not only execute plans but they also monitor the validity of their current plans, revising parts that are no longer executable. Along similar lines, (Briggs and Scheutz, 2013) extend (Perrault and Allen, 1980) approach to the generation and understanding of indirect

speech acts (in addition to direct speech acts and physical acts) in accordance with social norms of conduct and politeness. In conclusion, AP has extensively been studied as a means of lubricating the synergy between physical and speech acts in both dialog and task planning.

2.6 Conclusion

This chapter provides an overview of some notions in Automated Planning (AP). It provides an account of the various languages used in AP to represent problem, domain and solution descriptions. These languages are listed with their syntax representations, advantages and shortcomings. Some applications of AP in the real world are highlighted to demonstrate the potential of the field in solving real world problems. We also provide a brief account of the usage of AP in HRI with the intent of planning of socio-communicative acts to endow the robot with interactional abilities in their bid to interface with humans. The next chapter uses these aforementioned criteria to characterize and label the various learning approaches proposed in the literature.

Chapter 3

Learning Systems in Automated Planning

3.1 Introduction

As mentioned in our problem statement in section 1.2, we aim to learn a domain model encompassing all the domain-applicable actions which best explains the observed plan traces. This chapter introduces the prominent learning techniques that have been employed by learning systems over the years. It then proceeds to classify the learning systems based on these techniques, highlighting in brief the operational core of the classified systems.

3.2 Machine Learning (ML) in AP

Whilst good in theory, the creation of models in the case of complex domains is a non trivial task. While it is possible to codify, debug and maintain action models pertaining to simple artificial domains, it remains laborious, unfeasible and sometimes impractical to do so for some complex real-world domains. The amount of effort needed to encode and maintain accurate action models is significant. There is also no “one size fits all" strategy. Thus, we seek help from various ML techniques which allow learning of the underlying action model from traces produced as a result of plan execution. This learnt model can ideally be re-injected into the planner for further planning purposes.

3.3 Characteristics of Learning Systems

During the course of this bibliographical review, we have been able to identify certain key criteria which help us characterize the works in the literature. These criteria are described in the following subsections. The ML techniques to learn from traces broadly fall into one of the following categories:

- **Online**: Learning occurs during the execution phase (Zimmerman and Kambhampati, 2003). The system can start learning as soon as the generation phase is complete. This kind of learning never stops, thus allowing the continuous correction and improvement of an incorrect model and adaptation to changing environment characteristics. On the flip side, the cost of learning is added to the cost of planning, augmenting the time window in which execution occurs.

- **Offline**: One-shot learning exercise from traces. It allows a decoupling of the learning and planning phases, ensuring that the cost of learning is not added to that of planning. However, this one-time learning also means that in case of the injection of a faulty domain, the planner may never be able to recover before the end of the planning and

the beginning of the learning phase, thus staying blocked. Offline learning is the more popular learning mechanism.

The characteristics of the learning systems in AP are detailed in the subsections below.

3.3.1 Representation Mechanism

This section tries to classify approaches based on the viewpoint the learning system is observed from. The learning system can be seen as:

- **Object-centered representation**: While the onus is on learning action models from action execution sequences, the focus is on the objects manipulated by an action during the course of its execution. Each such operator is represented by a single parameterized state machine. The learning systems which use this representation describe how object descriptions change during the execution of the operator. The action model is generally represented in OCL (Object Centered Language).
- **Action-centered representation**: A plan is viewed as an interleaved sequence of states and actions representing the transition of the system from the initial state to the goal. The action model in this case is generally represented in the form of PDDL and its sub-languages or variants.

3.3.2 Inputs To The Learning System

The inputs to the learning system as well as some other characteristics (quality, quantity etc.) go a long way in determining the output and its quality. The inputs to the system may include: the model, the background knowledge and the traces; all three in varying capacities.

Model

Before the learning phase begins, the action model may exist in one of the following capacities:

- **No Model:** This refers to the fact that no information on the actions that constitute the model is available in the beginning, and the entire model must be learnt from scratch.
- **Partial Model:** Some elements of the model are available to the learner in the beginning, and the model is enriched with more knowledge at the end of the learning phase.

Background Knowledge

Background knowledge (BK) is mostly required only in the absence of complete and correct domain knowledge. The availability of this BK is inversely proportional to the planning effort required. For example in the blocksworld domain, “only one block can be on top of some other block" or “a block being held means that the robotic arm is not empty" is an illustration of BK. To compensate for the absence of domain knowledge, this BK may also comprise of object type definitions as well as predicate and intermediate state information (Jilani *et al.* (2014)).

In most cases, BK restricts the possible action choices at every system state, thus speeding up planning. This BK, however, may also be difficult to acquire. Depending on the presence or absence of background knowledge, the associated learning techniques may either be classified as analytical or inductive respectively. These techniques are discussed in detail in section 3.4.1.

Traces

Traces are the residue or the proof of the execution of a series of actions in a domain. They represent a summary of the executed action and the resulting state of the world. These plan execution traces may be classified into pure or adulterated as follows:

- **Noisy:** The traces can be adulterated because of sensor miscalibration or faulty annotation by a domain expert. For example, AMAN (Action-Model Acquisition from Noisy plan traces) (Zhuo and Kambhampati, 2013) falls into this category.
- **Ideal**: There is no discrepancy between the ideal action and the recorded action. For example, OBSERVER (Wang, 1996).

3.3.3 System Outputs

Action Granularity

Irrespective of whether the trace is represented in form of action sequences or state-action interleavings, the actions can be classified as atomic or grouped. Atomic actions are unitary actions applied to a system state to effectuate a transition. Macro actions, on the other hand, are a group of actions applied at a time point like a single action, representing high level tasks while encompassing low level details (Newton et al., 2007; Newton et al., 2008; Newton and Levine, 2010). From a broader perspective, macros are like procedures in the programming realm. They are promising because they are capable of aggregating several steps in the state space, and providing extended search space visibility to the planner. Some merits of using macros include:

- Macros generated from plans of smaller problems are scalable and can be evaluated against larger problems.
- Macros allow planning of several steps at a time and allow for re-usability in plan snippets. Knowledge acquired from macros can be integrated into the planner or encoded into the domain (Newton et al., 2007; Newton et al., 2008; Newton and Levine, 2010).

Some limitations are as follows:

- They increase the processing time of the planner, adding more branches in the search tree at every point in time. However, the problem is significantly reduced by search tree pruning used by many recent planners (Newton et al., 2007).
- Macros often result in longer plans than when no macro is used (Newton and Levine, 2010).

State Observability and Action Effects

The observability of the current state of the system after the action execution may either be perfectly certain or flawed because of faulty sensor calibration. For illustration, we take the case of the SOMBRERO project. In the beaming experiments, the doctor manipulating the robot has an eye tracking device mounted on their head, connected to the cameras in the eyesockets of the robot. This helps the doctor remotely "see" the experimental setup and steer the visual focus of attention of the robot in a more context-aware fashion. This device is also equipped with a pupil tracker, tracking the movement of the pupil and helping to ascertain the direction of focus of attention of the doctor. This tracker is inherently imprecise in its pupil tracking, often wrongly estimating the position of the pupil thus the visual focus of attention. In such cases of faulty calibration, the state of the system is only partially observable at run

time, thus observations return a set of possible states (called as “belief states") instead of a single state (Ghallab, Nau, and Traverso, 2004).

Similarly, actions are not necessarily deterministic (single outcome on execution) but may be stochastic, that is: capable of producing multiple outcomes, each of them equipped with a different execution probability.

Keeping these variations of action effects and state observability in mind, we define four categories of learning systems:

- Deterministic effects, Full state observability: For example, the EXPO (Gil, 1992) system.
- Deterministic Effects, Partial State Observability: In this family, the system may be in one of a set of “belief states” after the execution of each action (Ghallab, Nau, and Traverso, 2004). For example, the ARPlaces system (Stulp et al., 2012).
- Probabilistic effects, Full State Observability: for example, the PELA (Planning, Execution and Learning Architecture) system (Jiménez, Fernández, and Borrajo, 2008).
- Probabilistic Effects, Partial State Observability: Barring a few initial works in this area (Yoon and Kambhampati, 2007), this classification remains as the most understudied one till date.

3.3.4 Learnt Model Granularity

Learning systems can be also be characterized according to the sheer amount of detail represented in the output representation of the learnt model. This can range from a bare-bones STRIPS model produced in the output (Zhuo, Nguyen, and Kambhampati, 2013), to PDDL models with quantifiers and logical implications (Zhuo et al., 2010), to PDDL models with static predicates (predicates existing implicitly in operator preconditions and never appearing in plans e.g. (Jilani et al., 2014)). Granularity is an important classification criterion because of the fact that the choice of a certain algorithm would depend on the level of granularity with which the environment needs to be modeled, thus determining the level of detail to be represented in the learnt model. Granularity is used a criteria to classify the works summarized in the table 3.1 based on the richness of the information learnt vis-a-vis the amount of input provided.

3.4 Type of Learning

This section sheds light on some classical learning techniques which have been around and in use for learning action models characterised in Chapter 2, as well as newer and interesting techniques which have emerged with recent advancements and the broadening spectrum of machine learning techniques. The algorithms described in this chapter are further summarized in the form of table 3.1. These tables provide a bird’s eye view of the key characteristics of each algorithm in terms of input provided, output produced, application environment etc. These tables are described with the intent of serving someone who is new to the domain and is looking to situate their problem in the context of a previously done work.

3.4.1 Learning Techniques based on availability of background knowledge

As discussed in section 3.3.2, the learning techniques based on the absence or presence of background knowledge are classified into inductive and analytical learning techniques respectively.

Inductive Learning

The learning system is fed with a hypothesis space H of possible rules and a set of traces T (defined in section 1.3). The desired output is a hypothesis from the space H that is consistent with these traces (Zimmerman and Kambhampati, 2003). Inductive techniques are useful because they can identify patterns and generalize over many examples in the absence of a domain model. One prominent inductive learning technique is that of regression tree learning. Regression trees are capable of predicting continuous variables and modeling noise in the data. In comparison to a decision tree which predicts along a category (i.e. class), a regression tree performs value prediction along the dependent dimension for all observations (Balac, Gaines, and Fisher, 2000).
The PELA (Planning, Execution and Learning Architecture) system (Jiménez, Fernández, and Borrajo, 2008) performs the three functions suggested in its name to generate probabilistic rules from plan executions and compile them to refine and upgrade its planning model. The upgraded planning model inculcates in the existing deterministic model two kinds of information: state-dependent probabilities of action success and state-dependent predictions of execution dead-ends. This model may be represented in PDDL as well as PPDDL formats. This is done by learning a relational decision tree for every action in the planning domain using the TDIDT (Top-Down Induction of Decision Trees) (Quinlan, 1986) algorithm. Each branch of the learned decision tree represents the following about the pattern of performance of the action in question: (i) the *internal nodes* of the branch represent conditions under which the pattern of performance is true, and (ii) the *leaf nodes* contain the corresponding conditions representing the performance of the action (*success/failure/dead-end*) and the number of examples covered by the pattern. Once done, all the trees of all the actions are combined to produce an action model with conditional or probabilistic effects. In ERA (Exploration, Regression tree induction and Action models) (Balac, Gaines, and Fisher, 2000), a robot learns action models for its navigation under various terrain conditions. With each exploration step, the robot records the executed action, the current terrain conditions and positions before and after the action execution. When exploration is finished, the data is divided into groups based on action type and a regression tree induction algorithm is applied to the data for each action type. This algorithm then builds a regression tree representing the model and expected outcome of that particular action.
LOCM (Cresswell, McCluskey, and West, 2009) uses an object-centered representation and learns a set of parameterized Finite State Machines (FSM). These FSMs are then exploited to identify pre- and post-conditions of the domain operators (Jilani et al., 2014).

LOCM2 (Cresswell and Gregory, 2011) is generalized and heuristic in nature to allow multiple state machines to represent a single object. This extends the coverage of domains for which a domain model can be learned. Unlike LOCM, LOCM2 constructs many possible solutions to the same problem. The heuristic is used to select a single solution from these possible solutions. The limitation of both LOCM and LOCM2 is that they learn only the dynamic properties of the system and not the static ones. This means that the learned models produce plans which are shorter than optimal solutions. This is a drawback as many systems use static predicates (predicates that never change) to contain the set of possible actions. This limitation is addressed by the NLOCM system.

NLOCM (Gregory and Lindsay, 2016) extends classical planning to numerical planning i.e. learns action costs in planning domains from state transition sequences using a constraint programming approach. It uses the LOCM2 and LOP systems as pre-processing steps. While the LOCM and LOCM2 algorithms only learn the dynamic aspects of the domain, the LOP system (Gregory and Cresswell, 2015) learns static relations. NLOCM extends the FSM of LOCM to an automata with numeric weights on the important features. A set of templates is defined for each operator which contributes to the action cost of that operator in the domain.

The algorithm successively solves more and more complex template sets (all possible subsets of operator parameters) until a satisfying assignment which minimizes the complexity cost is found. These templates are used to find the valid cost models.

Opmaker (McCluskey, Richardson, and Simpson, 2002) is a mixed initiative (planning methodology where both the human and the machine take initiative) tool to induce operators from action sequences, domain knowledge and user interaction. These operators are parameterized and hierarchical in nature. OCL is used to model the domain. For each action in the training solution sequence, it asks the user to input, if needed, the target state that each object would occupy after the action execution, thus creating its operator schema step-by-step. It is implemented inside a graphic tool called GIPO (Graphical Interface for Planning with Objects), which facilitates user interaction, domain modeling and operator induction (McCluskey, Richardson, and Simpson, 2002; Jilani et al., 2014).

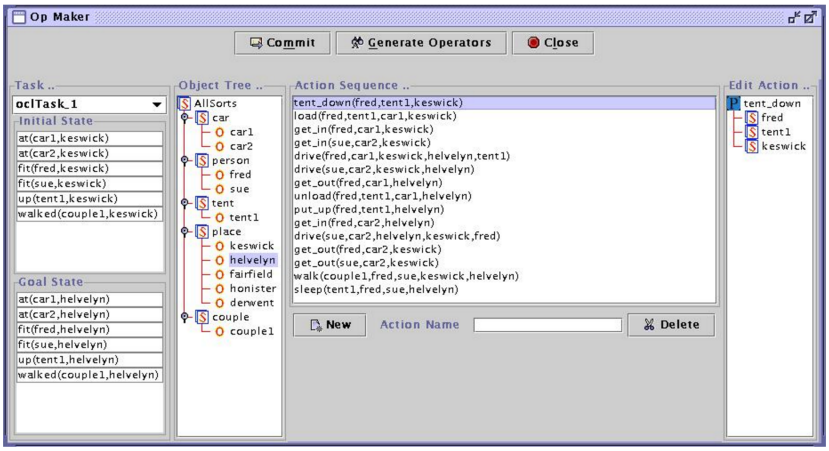

FIGURE 3.1: Opmaker Screen

Opmaker2 (McCluskey et al., 2009) is an improvement over Opmaker such that it eliminates the dependency on the user for intermediate state information. After Opmaker2 automatically infers this intermediate state information, it proceeds in the same fashion as Opmaker and induces the same operators. The main advantage of Opmaker2 is that it computes its own intermediate states using a combination of heuristics and inference from the Partial Domain Model (PDM), example sequences and solutions. In Opmaker2, the $DetermineStates$ procedure performs this function by tracking the changing states of each object referred to in the training example, taking advantage of the static and other information in the PDM. The output from $DetermineStates$ is, for each object, a map associating each object to a unique state value at each point in the training sequence. Once the map has been generated, the techniques of the original Opmaker algorithm are used to create an operator schema.

In (Martınez et al., 2016), the authors learn a probabilistic model including exogenous effects (predicates not related to any action execution) from a set of state transitions. They use the LFIT (Learning From Interpretation Transitions) framework (Inoue, Ribeiro, and Sakama, 2014) which uses Inductive Logic Programming (ILP) to induce a set of propositional rules that orchestrate the given input transitions. This is done by first transforming grounded to relational transitions, then relational transitions to probabilistic rules. Planning operators can be reconstructed from probabilistic rules. A score function evaluates the operators, and a heuristic search algorithm selects the set of operators that maximize the score.

Analytical Learning

The learning system has at its disposal the same hypothesis space H and traces as in the case of inductive learning but with an additional input: Background Knowledge (BK) that can explain traces. The desired output is a hypothesis from the space H that is consistent with both the traces and BK (Zimmerman and Kambhampati, 2003). Analytic learning relies on BK to analyze a given trace and identify its relevant features. Any system which relies on domain knowledge, predicate information or intermediate state information can be classified under this category. More details about analytic learning techniques can be found in (Zimmerman and Kambhampati, 2003).

3.4.2 Genetic and Evolutionary algorithm based approaches

Owing to advantages like scalability and re-usability, it may be a viable proposition to learn action models consisting of macro actions in place of unitary actions. A series of approaches which use evolutionary algorithms to learn macro actions in the absence of any kind of additional knowledge may be classified under this category. An approach by (Newton et al., 2007) learns macro actions given a domain and action sequences. It is generic in the sense that it does not require structural knowledge about the domain or the planner, but learns from plans generated for random planners and domains using a genetic algorithm .

Another approach by the same author (Newton and Levine, 2010) learns macros as well as macro-sets (collection of macros which collectively perform well). In this work, once a macro is learned, it is tinkered with by adding, deleting and modifying existing and new actions in a bid to learn how the new macro performs. In the first phase, only individual macros are learned using actions from generalized plans produced by solving input problems. In the second phase, macro-sets are learned using the macro pool obtained in the first phase, including only those that have fitness values greater than a certain minimum level.

In a third approach by the same author (Newton et al., 2008), non-observable macros i.e. macros that are not observable from traces are learned. This is done with the intent of constructing a comprehensive macro-library for future reuse. Learning is based on an evolutionary algorithm. Evolutionary algorithms repeatedly generate new macros (in every epoch) from current individuals by using given genetic operators: only the individuals with the best fitness values survive through successive epochs. The fitness function is based on three measures: $Cover, Score, Point$. $Cover$ evaluates the percentage of problems solved when the macro is used. $Score$ measures the time gained or lost over all the problems solved with the augmented domain compared to when they are solved using the original domain. $Point$ measures the percentage of problems solved with the augmented domain which take less or equal time compared to when they are solved using the original domain.

3.4.3 Reinforcement Learning

Reinforcement learning (RL) is a special case of inductive learning, and more specifically characterizes a learning problem instead of a learning technique (Sutton and Barto, 1998). An RL problem is composed of three principal elements: (i) the goal an agent must achieve, (ii) a stochastic environment, and (iii) actions an agent takes to modify the state of the environment (Sutton and Barto, 1998). RL takes place during the process of plan building. Through trial-and-error online traversal of states, an optimal policy (mapping from perceived states of the environment to actions to be taken when in those states) is found for achieving the agent's goals (Zimmerman and Kambhampati, 2003). The strength of RL lies in its ability to handle dynamic and stochastic environments where the domain model is either nonexistent or incomplete. The benefits of an RL system are two-fold: on one hand, it improves plan quality

(the system progressively moves towards finding an optimal policy) and, on the other hand, it can learn the domain model. One of the major drawbacks of RL is that of generalization: in its bid to achieve domain-specific goals, it cannot gather general knowledge of the system dynamics, leading to a problem of generalization. Despite its drawbacks, RL played a major role in the success of real life systems such as the recent AlphaGo (Silver et al., 2016). AlphaGo used policy gradient RL (policy represented by own function approximator) to optimize a deep convolutional neural network by simulating games of the network playing against previous versions of itself and rewarding itself for taking moves that gave it wins.

The LOPE (Learning by Observation in Planning Environments) (García-Martínez and Borrajo, 2000) system learned planning operators by observing the consequences of the execution of planned actions in the environment. In the beginning, the system is a blank slate and has no knowledge, it perceives the initial situation, and selects a random action to execute in the environment. It then loops by (1) executing an action, (2) perceiving the resulting situation of the action execution and its utility, (3) learning a model from this perception and (4) planning for further interaction with the environment. A global reinforcement mechanism either rewards or punishes operators based on their success in predicting the environment behavior. This is done by a virtual generalized Q table (associates each (s, a) pair with an estimate of $Q(s, a)$, where s and a are state and action respectively, and $Q(s, a)$ is the expected reward for taking action a in state s).

In an approach by (Safaei and Ghassem-Sani, 2007), probabilistic planning operators are learned in an incremental fashion. It has an integrated learning-planning system consisting of (i) an observation module for operator learning, (ii) an acting module for planning, and (iii) a control center for generating goals to further test the algorithm. This control center also dynamically assigns the reward function.

Relational Reinforcement Learning

Relational reinforcement learning is the combination between reinforcement learning and inductive logic programming (Džeroski *et al.* (2001)). Given a (i) possible state set S, (ii) possible action set A, (iii) state transition function $\delta : S \times A \rightarrow S$, (iv) reward function $r : S \times A \rightarrow S$, (v) background knowledge and (vi) declarative bias for learning policy representations; the task is finding a policy π for selecting actions $\pi : S \rightarrow A$ that maximizes the expected reward. RRL employs the Q-learning method (learns policies which are represented as value-assigned state-action pairs) using a relational regression tree. Due to the use of a more expressive relational language, RRL can be applied to a wider array of problems. It also allows the reuse of results of previous learning phases and their application to novel and more challenging situations.

MARLIE (Model-Assisted Reinforcement Learning in Expressive languages) is the first RRL system that learns a transition and reward function online (Croonenborghs *et al.* (2007)). MARLIE builds probabilistic models accounting for the uncertainty of world dynamics. It uses the TG algorithm (Driessens *et al.* (2001)) to learn relational decision trees. The decision trees are not restructured on the appearance of new examples. However, their fully relational aspect allows them to be applicable to a wider range of problems.
IRALe (Rodrigues *et al.* (2011)) produces a STRIPS-like (see table 2.1) action model consisting of a set of rules by combining RRL and active learning. At first, the action model is an empty rule-set. The interactions between the agent and the environment produce examples. In the event of a counter example (an example that contradicts the model), the example has to be revised by modifying or adding one or more rules. This contradiction is determined by matching operations performed between the rules and examples, and each identified counter-example leads to either a specialization or a generalization. The agent (independent entity

which perceives the world and performs actions accordingly) also performs active learning, that is, in every state it chooses an action that it expects to lead to a generalization of the model. With IRALe, new opportunities for generalizing the model will decrease the number of examples required to converge to the correct model. It uses a formalism called Extended Deterministic STRIPS (EDS), which is more expressive than traditional STRIPS.

3.4.4 Uncertainty-based techniques

Markov Logic Networks

A Markov Logic Network (Richardson and Domingos, 2006) is a combination of first order logic and probabilistic graphical models. It is a first-order knowledge base consisting of weighted formulas, and can be equated to a template used to construct Markov networks. They have numerous advantages. From the probability point of view, MLNs provide a language scalable enough to specify very large Markov networks, and the ability to inject a comprehensive range of domain knowledge into them. From the first-order logic point of view, MLNs are equipped with the ability to soundly handle imperfect knowledge and uncertainty. For example, the LAMP (Learning Action Models from Plan traces) system (Zhuo et al., 2010) learns action models with quantifiers and logical implications using MLNs. Firstly, the input plan traces (state-action interleavings) are encoded into propositional formulas (conjunction of ground literals) to store into a database as a collection of facts. These formulas are very close to PDDL (refer to table 2.1) in terms of representation. Secondly, candidate formulas are generated according to the predicate lists and domain constraints. Thirdly, a MLN uses the formulas generated in the above two steps to learn the corresponding weights of formulas and select the most likely subset from the candidate formula set. Finally, this subset is transformed into the final action models (Zhuo et al., 2010).

Noisy Trace Treatment Approaches

As explained in section 3.3.2, sensor miscalibration may cause introduce noise, giving birth to uncertainty. For example, AMAN (Action-Model Acquisition from Noisy plan traces) handles noise (Zhuo and Kambhampati, 2013) by finding a domain model that best explains the observed noisy plan traces using a probabilistic graphical model based approach. The only input to the learning phase are traces in the form of noisy action sequences. A probabilistic graphical model (PGM, graphical representation of dependence among random variables) is used to capture the relationship between the current state, correct action, observed action and the domain model. First, a set of all possible candidate models is generated. Then the observed noisy plan traces are used to predict the correct plan traces based on the PGM. Then, the correct plan traces are executed to calculate the reward of the predicted correct plan traces according to a predefined reward function. This reward function output is used to update the weights of the candidate models. The model with the highest weight is rendered as the model being searched for (Jilani et al., 2014).

In a series of works by (Pasula, Zettlemoyer, and Kaelbling, 2004; Zettlemoyer, Pasula, and Kaelbling, 2005), the goal is to learn action models in noisy and stochastic domains. This is done by: (i) allowing rules to refer to objects not mentioned in the signature of the action, (ii) relaxing the frame assumption (unspecified atoms in the operator's effects remain unchanged) and allowing the existence of "noise-induced" changes in the world, and (iii) extending the language: introducing newer and more complex concepts like existential quantification, universal quantification, counting and transitive closure. Transitive closure is represented in the language via the Kleene star operator. It allows to learn probabilistic rules which are affected by a factor of noise and include deictic references, thus are called Noisy Deictic Rules (NDR). Deictic references (first order description of the world in terms

of arguments of currently executing action (Mourao et al., 2012)) are used to identify objects, even those which are not directly affected by the mentioned predicates. The works conclude that the addition of noise and deictic references increases the quality of the learned rules.

3.4.5 Surprise Based Learning (SBL)

In a learning system, a surprise is produced if the latest prediction and the latest observation are considerably different (Ranasinghe and Shen, 2008). Once an action is performed, the world state is sensed by a perceptor module which extracts sensor information. If the algorithm had made a prediction, it is verified by the surprise analyzer. If the prediction turns out incorrect, the model modifier adjusts the world model accordingly. Based on the updated model, the action selector will perform the next action so as to repeat the learning cycle. This is illustrated in Figure 3.2.

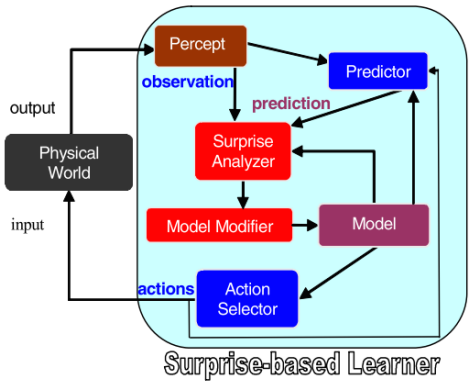

FIGURE 3.2: Overview of surprise based learning (taken from Ranasinghe and Shen, 2008)

A series of approaches based on SBL have used Goal Driven Autonomy (GDA). GDA (Weber, Mateas, and Jhala, 2012) is a conceptual model for creating an autonomous agent that (i) monitors expectations during plan execution (ii) detects the occurrence of discrepancies, (iii) builds explanations and new goals to pursue in the case of failures. In order to identify when planning failures occur, a GDA agent requires the planning component to generate an expectation of world state after executing each action in the execution environment. The GDA model thus provides a framework for creating agents capable of responding to unforeseen failures during plan execution in complex environments. Early implementations of GDA were focused both on operator learning and refining. Among approaches that refine operators, the EXPO (Gil, 1992) system refines incomplete planning operators by a method called the ORM (operator refinement method). EXPO does this by generating plans and monitoring their execution to detect the difference between the predicted and observed state. EXPO then constructs a set of specific hypotheses to fix the detected differences. After heuristic filtration, each hypothesis is experimentally tested and a plan is constructed to achieve the required situation (Jiménez et al., 2012). Among GDA approaches that learn operators, LIVE (Shen, 1993) is a discrimination-based learning method that creates general rules by noticing the changes in the environment, then specifying the rules by explaining their failures in prediction. When a rule executes and the predicted outcome does not match the observation, LIVE explores and creates new sibling rules. These rules are usually created by comparing the difference between a successful previous execution and the current failed one. New rules can accordingly be made more specific/generic. OBSERVER (Wang (1996)) is another operator learning system that integrates learning, planning and execution into a

system which incrementally learns planning operators. The learning module learns operator preconditions and effects by observing experts executing sample problems, then refining the operators collected from these observations. The operator preconditions are learned by creating and updating both the most specific and most general representations of the preconditions based on trace executions. Operator effects are learned by generalizing the delta-state (the difference between pre-state and post-state) from multiple observations. Later approaches have used GDA in conjunction with a relational or reinforcement learning component. For example: the FOOLMETWICE and LGDA systems. FOOLMETWICE (Molineaux and Aha, 2014) is goal-oriented (GDA-Goal Driven Agent) and not reward-driven. It identifies unknown events in a model based on surprise and explanation generation, and uses a relational learning method to update environment models.

LGDA (Jaidee, Muñoz-Avila, and Aha, 2011) is a GDA algorithm that uses case-based reasoning to map state-action pairs to a distribution over expected states, and map goal-discrepancy pairs to a value distribution over discrepancy-resolution goals. It also uses reinforcement learning to learn the expected values of the goals. It models goal formulation as an RL problem in which the value of a goal is estimated based on the expected future reward obtained on achieving it. More recent approaches have extended the reach of GDA to the territory of HTN-based planners as well. For example, the ARTUE (Autonomous Response To Unexpected Events) system (Molineaux, Klenk, and Aha, 2010) dynamically reasons about the goals to pursue when encountered with unexpected environmental situations. It consists of four principal components. A Hierarchical Task Network (HTN) planner reasons about exogenous events. A goal generation component reasons about and generates new goals. An explanation component reasons about hidden information in the environment. Finally, a goal management component manages and communicates the goals to the planner. ARTUE handles unexpected environmental changes by first explaining those changes, then generating new goals incorporating the explained knowledge about hidden environmental aspects.

3.4.6 Transfer Learning

In the case of some domains with limited knowledge and training examples at disposal, it is hard to learn the underlying domain model. In such cases, knowledge drawn from other domains may be plugged into the problem domain to achieve the objective of learning the underlying model of the problem domain. This can be made possible with the help of a technique called *Transfer Learning*. It is a technique in which, given a source task (in the context of task learning) and target task in the source and target domains respectively, aims to extract the knowledge from the source task and apply it to the target task in a situation where the latter has fewer training data (Pan and Yang, 2010). Most machine learning methods work well only under a common assumption: the training and test data are drawn from the same probability distribution. However, when the distribution changes, most models need to be reconstructed, trained and tested from scratch using data generated from the new probability distribution. In many real world applications, it is expensive, if not impossible, to regenerate the needed training data for model reconstruction. In order to reduce the effort to regenerate the training data, the possibility of knowledge transfer between domains would be ideal. Transfer learning (Pan and Yang, 2010) allows the domains, tasks, and probability distributions used in training and testing to be different. For example, we may find that learning to recognize apples might help to recognize oranges. Some characteristics of transfer learning systems include: the type of knowledge to be transferred, situation of transfer and technique of transfer. As far as type of knowledge goes, some knowledge is specific to individual domains, while other may be common between different domains such that it may help improve performance for the target domain. After discovering which knowledge can be transferred, learning algorithms need to be developed to perform the knowledge transfer.

The situation of transfer is an important characteristic as while it is important to know the situation in which a transfer must occur, it is equally important to know the situations in which transfer must not occur. In some cases, when the source domain and target domain are not related to each other, thus a brute-force transfer may be highly unsuccessful. In the worst case, it may even degrade the performance of the target domain, a situation referred to as *negative transfer* (for example, when an Indian or British tourist in Europe learns to drive on the left side of the road).

The advantages of using transfer learning are clear: a change of features, domains, tasks, and probability distributions from the training to the testing phase does not require the underlying learning model to be rebuilt. The disadvantages are listed as follows:

- Most existing transfer learning algorithms assume that the source and target domains are related to each other in some sense. However, if the assumption does not hold, negative transfer may happen. In order to avoid negative transfer learning, measures to ascertain transferability between source domains and target domains needs to be studied. Based on this study of possible transferability, we can then select relevant source domains/tasks to extract knowledge for learning the target domains/tasks.

- Most existing transfer learning algorithms so far have focused on improving generalization across different distributions between source and target domains or tasks. In doing so, they assumed that the feature spaces between the source and target domains are the same. However, in many applications, we may wish to transfer knowledge across domains or tasks that have different feature spaces. This type of transfer learning is referred to as *heterogeneous transfer learning*, which remains a challenge yet to be overcome (Pan and Yang, 2010).

A perfect example of transfer learning is the TRAMP (Transfer learning Action Models for Planning) system. TRAMP, given a set of traces in the target domain and expert-created domain models in the source domain, learns action models in the target domain by transferring knowledge from the source domain. TRAMP first encodes the input plan traces, which are state-action interleavings, into propositional formulas then as databases. Second, TRAMP encodes the action models from the source domains to a propositional formula set. Thirdly, TRAMP generates a candidate formula set to structure the target action models, building mappings between the source and target domains by searching keywords of predicates and actions from the Web, then calculating similarities of Web pages to bridge the gap between the two and transfer knowledge. Finally, TRAMP learns action models from the transferred knowledge with the help of Markov Logic Networks (Zhuo and Yang, 2014).

3.4.7 Deep Learning

Many AI problems can be solved by extracting the appropriate features that problem, then providing these features to a simple machine learning algorithm. In cases where it is difficult to extract high-level and abstract features from raw data (for example, to identify a breed of dog in millions of images from its intricate features such as tail length, body outline etc.), one solution is to allow computers to learn from experience and understand the world in terms of a hierarchy of concepts, with each concept defined in terms of its relation to simpler concepts. By gathering knowledge from experience, this approach avoids the need for human operators to formally specify all domain knowledge needed by the computer. The concept hierarchy allows the computer to learn complicated concepts by building them out of simpler ones (Goodfellow, Bengio, and Courville, 2016). If a graph is drawn showing how these concepts are built on top of each other, the graph is deep, with many layers. For this reason, this

approach is called *deep learning*. Deep learning thus by virtue offers multiple layers of representation wherein the features are not designed by human engineers but learn from data using a general-purpose learning procedure (LeCun, Bengio, and Hinton, 2015). It is an interesting field which, barring a few initial works on perceptron-based approaches (Mourao, Petrick, and Steedman, 2008; Mourao, Petrick, and Steedman, 2010; Mourao et al., 2012), remains a possibility to be put to use for the learning of domain models. In some recent approaches, deep learning has come into use in fields which are closely associated with AP, if not considered its sub-domains, such as plan recognition and goal recognition. Deep learning has however been used in the fields of plan recognition. For example, a recursive neural network is used in (Bisson, Larochelle, and Kabanza, 2015) to learn the HTN decision-making model of the observee agent (by the observer agent) for plan recognition. Assuming a given HTN plan library (drawn from the *Monroe* plan corpus (Blaylock and Allen, 2005), StarCraft Navigation and the *kitchen* domain), the problem of learning the decision-making model for an agent behaving based upon the library is cast as a multinomial logistic regression. A recursive neural network is trained to learn the vector representation of plan hypotheses, and compute a score for each one of them. A softmax classifier is then used to train the model to yield high scores for correct hypotheses and a low score for incorrect ones to allow score-based ranking of hypotheses.

Deep learning has also been used in the field of goal recognition (restricted form of plan recognition focused on identifying a player's high-level and end-level objectives in a game, given a sequence of low-level gaming actions). For example, in (Min et al., 2016a), players' goals' are recognized based on long short-term memory (LSTM) networks. With reference to data from the game *CRYSTAL ISLAND* in the form of logs of player actions, players' current actions are hypothesized to influence their upcoming goals. Thus extracting patterns from player gameplaying sequences is likely to provide inference to predict the players' next goal. For this, the next action is predicted based on a prior action sequence, and the model is trained using only player action data without goal labels. The output vector of the final memory cell in the LSTM is used to predict the most likely goal for the sequence of actions in a softmax layer (Min et al., 2016a).

3.4.8 MAX-SAT based approaches

The action model may be learnt by representing the inter- and intra-action dependencies in the form of a weighted MAX-SAT (maximum satisfiability) problem, and solving them with the help of a MAX-SAT solver to reconstruct the underlying action model. A weighted MAX-SAT problem can be stated as: given a collection C of m clauses $(C_1, ..., C_m)$ involving the disjunction of n logical variables, with clause weights w_i, find a truth assignment of the logical variables that maximizes the total weight of the satisfied clauses in C (Zhuo and Yang, 2014).

Among works that use MAX-SAT, the LAWS (Learn Action models with transferring knowledge from a related source domain via Web search) system (Zhuo et al., 2011) makes use of action models already created beforehand in other source domains, to help learn models in the target domain. The target domain and a related source domain are bridged by searching web pages related to the two domains, and then building a mapping between them by means of a similarity function which calculates the similarity between their corresponding web pages. The similarity is calculated using the Kullback-Leibler (KL) divergence. Based on the calculated similarity, a set of weighted constraints, called web constraints, are built. Other constraints such as state, action and plan constraints, are also built from traces in the target domain. All the above constraints are solved using a weighted MAX-SAT solver, and target domain action models are reconstructed from the satisfied clauses.

The ARMS (Action-Relation Modelling System) (Yang, Wu, and Jiang, 2007) system learns action models from traces which consist of action sequences occasionally interleaved with intermediate states. The parsed predicates are encoded in the form of intra-operator constraints called *information* and *action* constraints respected by these predicates. Frequent action pairs in the traces are identified with the help of the apriori algorithm (Agrawal and Srikant, 1994), which are then used to obtain a set of intra-operator plan constraints. These constraints are then solved with the help of a MAX-SAT solver. The constraints which amount to true are used to reconstruct the actions of the domain model. The process iterates until all actions are modeled.

The RIM (Refining Incomplete planning domain Models through plan traces) system (Zhuo, Nguyen, and Kambhampati, 2013) enriches its partial model during the process of learning. It constructs sets of soft and hard constraints from incomplete models and plan traces. These constraints are again intra-operator and inter-operator ones that must be satisfied by atomic actions as well as macro-actions. The novelty of this approach is that it learns macro-actions as well. Macro-actions increase the accuracy of the incomplete model. These constraints are solved with the help of a SAT solver. The constraints which amount to true are used to obtain sets of macro-actions and refined action models.

The Lammas (Learning Models for Multi-Agent Systems) system (Zhuo, Muñoz-Avila, and Yang, 2011) learns action models from input plan traces in the same lines of ARMS, however extended to a multi-agent setting. Lammas also encodes the input plan traces as constraints which fall into one of the following three categories: (1) *correctness constraints* imposed on the relationship between actions in plan traces to ensure that causal links in the plan traces is not broken, (2) *action constraints* encoding the actions based on STRIPS semantics and (3) *coordination constraints* encoded from an Agent Interaction Graph (AIG) describing inter-agent interactions. The learnt actions are represented in the MA-STRIPS language which is more adapted for a multi-agent setting (Brafman and Domshlak, 2008).

3.4.9 Supervised Learning Based Approaches

In (Mourao, Petrick, and Steedman, 2008), the authors learn the effects of an agent's actions given a set of actions and their preconditions, in a partially observable and noisy environment. 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. The final input vector representing a particular action applied to a certain state has the form: $(actions,\ object - independent\ properties,\ object - dependent\ properties)$. The output vector has the form: $(object - independent\ properties,\ object - dependent\ properties)$. The task is to learn the associations between action-precondition pairs and their effects. The learning problem has a set of binary classification problems, and the perceptron is used as a classifier. The prediction consists of calculating the kernel matrix, and the kernel trick is used in the case of non-linearly separable data (Mourao, Petrick, and Steedman, 2008).

Another work by the same authors (Mourao, Petrick, and Steedman, 2010) is based on the same lines, except that a voted kernel perceptron learning model with a DNF kernel, allowing the perceptron to run over the feature space of all possible conjunctions of bits in the input space i.e. the entire space of possible rules (Sadohara, 2001). Action and state information are encoded in a vector representation as input to the learning model, and resulting state changes are produced in the form of an output vector.

A very recent work by the same authors (Mourao et al., 2012) learns probabilistic planning operators from noisy traces. While the precedent approaches find the most probable outcome, this approach emphasizes on generating alternately less probable outcomes from noisy traces. It decomposes the learning problem into two stages: learning implicit action models and progressively deriving explicit rules from the implicit models. Rules are extracted

from classifiers based on the notion that more discriminative features will contribute more to the classifier's objective function. The classifiers use the k-DNF kernel with $k = 2, 3$ and 5. The rule fragments (partial precondition and single effect predicates) derived from classifiers that only contributed to the preconditions provide for a source of alternative effects for the planning operator. An alternative rule is created by reusing the precondition from the planning operator, and by one-by-one combining the effects of the rule fragments. Each new effect is accepted if the resulting rule gives better F_1-scores than the previous rule on the remaining training examples. The F_1 measure is a standard metric in information retrieval F_1, numerically the harmonic mean between precision and recall. Precision is the percentage of predicted classes that are correct, while recall is the percentage of correct classes that were correctly predicted. Precision is therefore the quality of the predictions. Recall measures how much of the correct classes are predicted.

TABLE 3.1: State of the art planning algorithms (PDM = Partial Domain Model, OCL = Object Centered Language, D = Deterministic, P = Probabilistic, FO = Fully Observable, PO = Partially Observable, PDL = Prodigy Description Language, MAXSAT = Maximum Satisfiability, PGM = Probabilistic Graphical Model, GDA = Goal Driven Autonomy, FOL = First Order Logic), Granularity = {$+$ (high), i (intermediate), $-$ (low)}

Algorithm	Input	Output/Language	Granularity	Technique	Merits	Limitations	Environment	Robust to Noise
Inductive Learning Based Approaches								
ERA	Map of area and error models that determine how robot's actions will behave in different terrain conditions	Action Models/ Simulation based	$-$	Regression tree induction	Ability to model noise and predict continuous variables in data	Cannot make multi-variate predictions	D, FO	Y
LOCM	Action Sequences	Action Models/ PDDL	i	Inductive learning on state machine representation of objects.	Does not require background information	(i) Induced models may contain detectable flaws (ii) Static background information not analysed by system	D, FO	N
LOCM2	Action Sequences/ Action Schema	Action Models/ PDDL	i	Allows many possible solutions to same problem, further using heuristic criteria to select a single solution.	Does not require background information	Does not work in case of domains with: (i) Dynamic many-many relationships (ii) Same object more than once in action arguments	D, FO	N
NLOCM	Action Sequences/ Action Schema	Action Models/PDDL	+	Inductive Logic Programming	Requires only final plan cost as additional information	Not equipped to handle complex domains	D, FO	N
OpMaker	Partial Model, Action sequences	Operators/OCL	$-$	Operator induction by mixed initiative	(i) Ease of operator encoding (ii) Useful for non-experts	Needs user input for intermediate state information	D, FO	Y
OPMaker2	PDM, Action sequences/OCL	Operators	$+$	Computes intermediate states using combination of heuristics, inference from PDM, training tasks and solutions	(i) Overcomes drawback of Opmaker by negating need for user input for intermediate state information (ii) Does not require too many examples	Expert required to transfer heuristic knowledge	D, FO	Y

PELA	Planning problem, PDM and action sequences	Enriched action model/PDDL	i	TDIDT	Based on off-the-shelf planning and learning components	Assumes correct initial action model	P, FO	N
Genetic Algorithm Based Approaches								
Newton *et al.* (2007)	Domain and example problems	Macros/PDDL	+	Genetic Algorithms	(i) Allow planning of several steps at a time (ii) Capture action sequences that help avoid troublesome regions	Macros, when replaced by constituent actions, often result in longer plans	D, FO	N
Reinforcement Learning Based Approaches								
LOPE	Number of execution cycles, possible action set	Operators/ Propositional Logic	+	Reinforcement Learning	Allows knowledge sharing among agents, increasing percentage of successful plans	Sensor differences among agents causes different ways of perception thus different biases towards operator generation	D, FO	N
Safaei & Ghassem-Sani (2007)	Action sequences	Operators/FOL without functions	+	Incremental Learning	(i) Removes redundant predicates in precondition (ii) Doesn't need prior knowledge	Conditional effects not supported in learned operators	P, FO	N
Relational Reinforcement Learning (RRL) Based Approaches								
MARLIE	Episodes (Action Sequences)	Transition and reward function/ Expressive relational language	+	Learning Relational Decision Trees	Relational thus widely applicable	Decision tree not restructured when new examples appear	P, FO	N
Uncertainty Based Approaches								
AMAN	Action Sequence	Operators/STRIPS	+	PGM, Reinforcement Learning	No background knowledge needed, can learn directly from traces	Model sampling mechanism unclear	D, PO	Y
LAMP	State-Action Interleavings	Action Models/PDDL	+	Markov Logic Network (MLN)	More expressive models with quantifiers and logical implications	Looses efficiency with increasing domain size	D, FO	N
Pasula *et al.* (2007)	State-Action Interleavings	Probabilistic STRIPS-like rules/ STRIPS-like	i	Learn action structure, action outcomes and parameter estimation	Allows learning of models of more realistic worlds	Rules cannot be applied in parallel	P, FO	N
Surprise Based Learning (SBL) Approaches								

ARTUE	Initial state, goal state, model	Expectations, discrepancies and goals/PDDL+	i	GDA	Integrates AI research in planning, environment monitoring, explanation, goal generation, and goal management.	Cannot be applied to complex environments	D, FO	N
EXPO	PDM, traces of state sequences	New preconditions, effects, conditional effects, operators, attribute values/PDL	–	Learning-by-experimentation for operator refinement	(i) Learns conditional effects (ii) Methods are goal-directed and learning is incremental	Rules learnt from general to specific	D, FO	N
LGDA	Initial state, dummy goal, dummy discrepancy, policy	Expectations, discrepancies and goals	i	Uses case-based reasoning and intent recognition in order to build GDA agents that learn from demonstrations	Case-based reasoning (CBR) and Reinforcement Learning (RL)	Lacks capabilities to learn explanations of discrepancies	D, FO	N
Surprise Based Learning (SBL) Approaches								
LIVE	Sequence of states, actions and predictions	Prediction rule (condition, action and precondition)/ CNF of condition, action and precondition	i	Surprise and explanation generation. Uses a set of domain-dependent search heuristics during planning	Hidden predicates can also be determined.	Exploration method is brute force	D, FO	N
OBSERVER	Action Sequences, practice problems	Operators/ STRIPS-like	i	Conservative specific-to-general inductive generalization process	(i) Can find out negated preconditions (ii) Does not require strong background knowledge.	May suffer from incomplete or incorrect domain knowledge	D, FO	N
Transfer Learning Based Approaches								
TRAMP	Action schemas, predicate set, small set of plan traces T from the target domain, set of action models from source domain.	Action models in target domain D_t/ STRIPS	i	Transfer Learning	Minimum training examples needed to learn target action model	Possibility of negative transfer	D, FO	N
MAX-SAT Based Approaches								

ARMS	Action sequence with partial/no state information	Operators/STRIPS	i	Builds weighted propositional satisfiability problem and solves it using weighted MAX-SAT solver	Can handle cases when intermediate state observations are difficult to acquire	(i) Cannot learn action models with quantifiers or implications (ii) Cannot learn complex action models	D, FO	N
LAWS	Action models, predicate set, small set of plan traces T from the target domain, set of action models from source domain.	Action models in target domain D_t/ STRIPS	i	Transfer Learning+ KL divergence	Web can be exploited as knowledge source	Possibility of negative transfer	D, FO	N
Lammas	Action sequences	Operators/ MA-STRIPS	i	Builds inter-agent and intra-agent constraints and solves it using weighted MAX-SAT solver	Can capture interactions between agents	Not tested exhaustively	D, FO	N
Supervised Learning Based Approaches								
Mourao *et al.* (2008)	Actions and states	Operators/PPDDL	$+$	Kernel perceptrons+ sequential covering	Can handle noisy domains	Cannot handle incomplete observations	P, FO	Y

3.5 Algorithmic "Cheat Sheet"

To choose a learning approach closer to a requirement set, this study proposes a "cheat sheet" with the intent of helping researchers starting out in the domain to choose a specific learning technique based on the characteristics defined in this section. This "cheat sheet" (see figure 3.3) is not exhaustive, and conceptualized bearing in mind the methodology adopted by the paper and the learning techniques discussed in section 3.4. The branch in this figure dedicated to the learning of heuristics is not detailed as it is beyond the scope of this study.

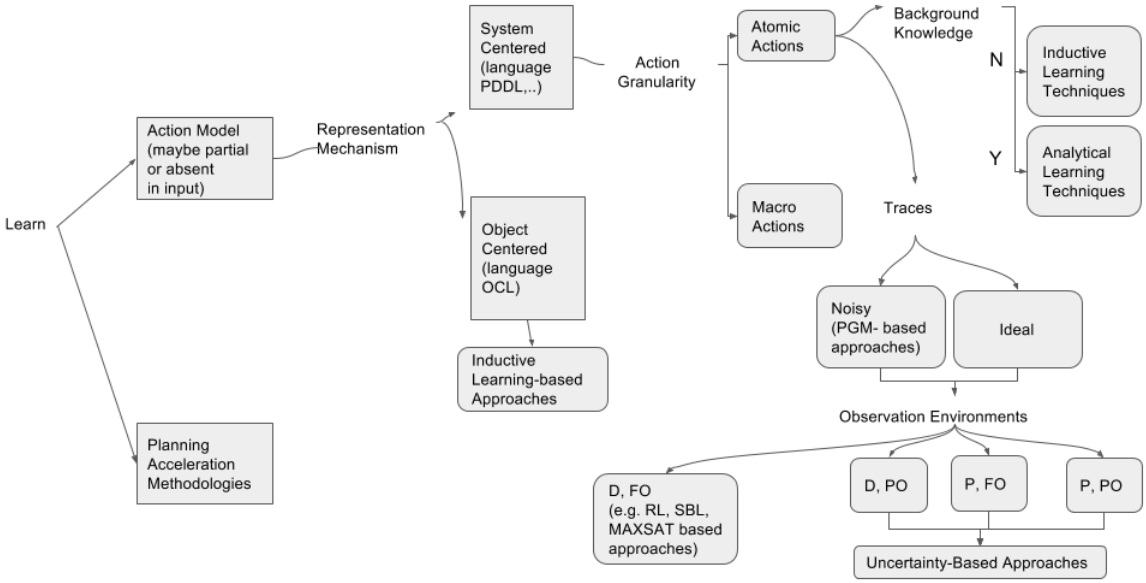

FIGURE 3.3: Guideline for choosing appropriate learning approach based on input requirements (OCL = Object Centered Language, D = Deterministic, P = Probabilistic, FO = Fully Observable, PO = Partially Observable, MAXSAT = Maximum Satisfiability, PGM = Probabilistic Graphical Model, RL = Reinforcement Learning, SBL = Surprise Based Learning)

3.6 Our Approach vis-a-vis the Literature

Vis-a-vis the literature described in this chapter, we introduce two approaches which are inspired by various learning systems and approaches. PDDL has been chosen as the representation language it is by far the most comprehensive representation language in the planning domain and offers the possibility to integrate temporal aspects of the context as well. These approaches are introduced as:

- SAT-based Learning Techniques: the *SRMLearn* system
  This approach is explored as it is one of the more classical approaches in the literature with a fair enough possibility of generalization. Various approaches in the literature are based on SAT based approaches to learn the underlying action model (Yang, Wu, and Jiang, 2007; Zhuo et al., 2010; Zhuo, Yang, and Kambhampati, 2012; Zhuo and Yang, 2014). We exploit the hypothesis that it is possible to equate intra-operator and inter-operator dependencies to constraints and represent them in the form of a SAT problem. While the intra-operator constraints deal with the syntax of the operator, the inter-operator ones exploit the semantic relationship between successive operators

in a plan sequence. We present an approach called SRMLearn (**S**equential **R**ules-based **M**odel **Learn**er), which encodes the aforementioned constraints in the form of a maximum satisfiability problem (MAX-SAT), and solves it with a MAX-SAT solver to learn the underlying action model. Unlike previous MAX-SAT driven approaches, our chosen constraints exploit the relationship between consecutive actions, rendering more accurately learnt models in the end. This approach is capable of learning purely from plan execution sequences, subtracting the need for domain knowledge.

- Connectionist Techniques: the *PDeepLearn* system
  Most approaches mentioned in the literature fall short of being used in the case of a real life scenario as anything less than a flawless reconstruction of the underlying model renders it unusable in certain domains. This can clearly be seen in the case of HRI, as any domain expert intervention to complete a partially learnt model will always fall short owing to the sheer subtlety of interleavings between speech, body movements and facial expressions. Also, learning approaches that fail to generalize to new contexts (e.g. Q-learning) do not evoke our interest as one of the underlying requirements of our problem is to find a learning algorithm with the capability to generalize (since the algorithm will be developed with benchmark traces and tested with real HRI traces).Keeping this in mind, we propose a system called PDeepLearn, which stands for **P**DDL domain **Deep Learn**er, and is capable of flawless reconstruction of the underlying action model. It is inspired in part by the kernel perceptron approaches introduced in (Mourao, Petrick, and Steedman, 2008; Mourao, Petrick, and Steedman, 2010; Mourao et al., 2012), however extended to a memory-based neural network setting. It uses long short-term memory (LSTM) techniques for the acquisition of the underlying action model. We use the sequence labelling capabilities of LSTMs to predict the next action in a plan execution sequence based on previously observed actions. This is done by iteratively testing the sequence labelling capabilities of each of the models sampled from an exhaustively generated then trimmed set of all possible action models. Empirical results highlight that the sequence labelling achieved with an empirically isolated model speculated to be identical to the hand woven model attains the highest accuracy compared to other models in the set. The structural consistency of the acquired model with the hand woven model renders this LSTM-based sequence labelling approach to learn action models as an effective one.

3.7 Conclusion

This chapter provides a brief account of the learning approaches used in the learning of action models in the literature, classifying them together and characterising them wherever appropriately possible. It can be seen that the approaches range from all walks of possible machine learning approaches, chosen broadly based on certain characteristics discussed in the Section 3.3. Furthermore, we justify the choice of our learning approaches vis-a-vis the literature and characteristics of our learning problem.

Chapter 4

SAT-based Techniques: SRMLearn

4.1 Introduction

Constraint satisfaction techniques can be employed to learn an action model. In this first part of the study, we follow a more classical approach to learn action models which is closer to the literature in terms of underlying principle and methodology. It looks to reconstruct from traces a model ground upwards. Since this model runs the possibility of being incomplete hence inherently not reusable for subsequent planning, services of a domain expert are employed to minimally fill in the gaps with this reconstructed model in a bid to render it complete. To learn this model, it is possible to exploit intra-operator and inter-operator dependencies and constraints. While the intra-operator constraints deal with the syntax of the operator, the inter-operator ones exploit the semantic relationship between successive operators in a plan sequence. We present an approach called SRMLearn (**S**equential **R**ules-based **M**odel **Learn**er), which encodes the aforementioned constraints in the form of a maximum satisfiability problem (MAX-SAT), and solves it with a MAX-SAT solver to learn the underlying action model. Unlike previous MAX-SAT driven approaches, our chosen constraints exploit the relationship between consecutive actions, rendering more accurately learnt models in the end. Our approach is capable of learning purely from plan execution sequences, subtracting the need for domain knowledge. SRMLearn uses a bare minimum of domain knowledge in the form of state-action interleaved traces in the input to learn an action model as the output. It proceeds in the following fashion: it defines a set of intra-operator constraints and inter-operator constraints. The intra-operator constraints are required to be satisfied in order to adhere to the semantics of STRIPS (Fikes and Nilsson, 1971). It then defines certain inter-operator constraints which exploit short and long-term dependencies between operators in the sequence by means of pattern mining techniques. Finally, it encodes all the aforementioned constraints in the form of a satisfiability problem and solves the problem with the help of a MAX-SAT solver. The solved constraints are used to reconstruct the underlying action model. While there are other approaches in the literature that learn action models by encoding it in the form of a satisfiability problem (for example ARMS (Yang, Wu, and Jiang, 2007)), empirical results of our evaluation demonstrates a higher accuracy of the learnt model with SRMLearn. This suggests that our approach is an effective approach to learn action models. This chapter is divided into the following sections: we detail the functioning of the SRMLearn system in Section 4.2, and present our empirical evaluations over 5 domains in section 4.3. We include some preliminary work done in the direction of learning temporal models using the SAT approach in Section 4.4, and conclude the chapter in section 4.5.

4.2 The SRMLearn Algorithm

The approach followed in this learning system is divided into three phases, which are summarized in figure 4.1. In the first phase, we generalize the actions and predicates in the trace set by replacing the variables by their types to obtain a trace set of generalized actions and

predicates respectively. Phase 2 is dedicated to constraint generation. These constraints are constituted by:

- Intra-operator constraints: primarily hard constraints i.e. are required to be solved by the MAX-SAT solver without fail, and
- Inter-operator constraints: these are primarily soft (no strict obligation on the solver to solve them) and short-range constraints on frequent action pairs and long-range constraints on the action chain as a whole.

In phase 3, these constraints are supplied to a MAX-SAT solver and the constraints that evaluate to true are used to reconstruct the underlying domain model. These phases are further elaborated in the forthcoming sections. They can also be understood in the form of pseudo code demonstrated in the algorithm 1.

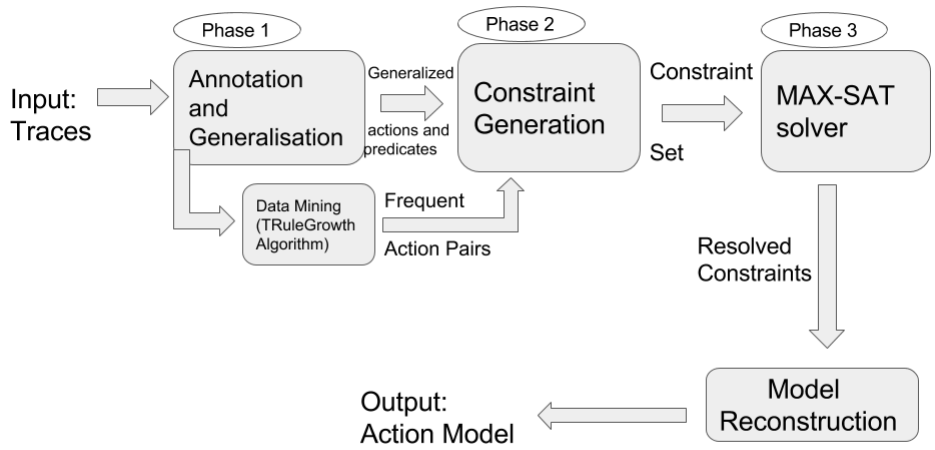

FIGURE 4.1: Approach phases of SRMLearn

Algorithm 1 SRMLearn

1: **Input**: set of plan traces *T*, dictionary VT associating objects with their types *(type, object)*
2: **Output**: domain model *m*
3: Relevant Predicate Dictionary RPD = *generalization (T, VT)*
4: Constraint Adherence: Identify frequent action pairs from TRulesGrowth Algorithm and proceed with pruning phase to eliminate improbable models from *M*;
5: Build *intra-operator constraints* using RPD:
6: Build *inter-operator constraints* like short-range constraints and long range constraints using RPD.
7: Solve all constraints, build *m*
8: Return *m*

All the phases in SRMLearn are illustrated with an example in Figure 4.5.

4.2.1 Generation of Traces

The traces used in the course of our experimentation are solutions to plans. These plans have been generated with the help of problem generators which are publicly available in

Algorithm 2 Procedure *generalization (T)*

Input: Set of plan traces *T*, dictionary VT associating objects with their types as *(type, object)*
Output: Relevant Predicate Dictionary RPD
for `j = 1 to T` **do**
    Replace all objects in signatures of action and predicates in T with their types referenced from the *(type, object)* pairs to build generalized action schema A_s and predicate schema P_s respectively
for a_i `in` A_s **do**
    for p_i `in` P_s **do**
        If p_i relevant to a_i, then add to list $relPre_{a_i}$ containing relevant predicates for a_i
    Add to relevant predicate dictionary RPD an entry with (key, value) as (a_i, $relPre_{a_i}$)
Return RPD

the planning domain. These generators are usually scripts custom built for every planning domain which accept a certain set of user defined parameters (such as the number of problems to generate among others) and generate problems in accordance to the parameters defined. [1]. These problems were then fed to the PDDL4J planner which is customized to produce the plans in the format used in this study (i.e. alternating state-action pairs) [2]. The planner then resolves the problems fed to it one-by-one and adds the solutions of each plan to our data corpus (for more implementation details see Appendix). Since the problems generated by the generator are unique, it is ensured that each generated plan that is added to the corpus is unique as well.

4.2.2 Annotation and Generalization

In this subsection, each trace is taken one by one, and each action as well as each predicate in the initial, goal and intermediate states of the plan trace is scanned to substitute the instantiated objects in the signatures with their corresponding variable types. This substitution is done by referencing the dictionary VT that associates instantiated objects with their types. These associations are drawn from the problem definition. This builds the corresponding generalized action schema A_s and predicate schema P_s respectively. Each action in the action schema is associated with its relevant predicates from the predicate schema, where a predicate $p_{a_i} \in P_s$ is said to be *relevant* to an action $a_i \in A_s$ if they share the same variable types. The objective is to find for every action, the predicates that can be applied to it and thus constitute the *(pre, add, del)* lists of that specific action. For example, the predicate *at-robby (?robot, ?room)* is relevant to the action *find (?robot, ?object, ?room, ?gripper)* as both of them contain the parameter types *(robot, room)* in their signatures and the types $(robot, room) \subset (robot, object, room, gripper)$. The set of relevant predicates to an action a_i can be denoted as $relPre_{a_i}$. With schemas A_s and P_s, a relevant predicate dictionary is built, where the key is the name of the action and the value is a list of all relevant predicates to that action. This procedure is detailed in Algorithm 2.

For example, let us perform the above mentioned steps in the case of the *move* action of the *gripper* domain. In this action a robot moves an object from one room to the other. The

[1] The problem generators used for this study have been sourced from http://www.plg.inf.uc3m.es/ipc2011-learning/Domains.html

[2] This planner is available for download at https://github.com/pellierd/pddl4j

generalization procedure for the *move* action is represented in Listing 4.1 and in Figure 4.2.

```
trace snippet before generalization phase::
[(action) (move robot3 room2 room4)
(state) (at-robby robot3 room4)]
trace snippet after generalization phase::
[(action) (move robot3 - robot room2 - room room4 - room)
(state) (at-robby robot3 - robot room4 - room)]
Relevant predicates for action move:
(at-robby)
```

LISTING 4.1: Generalization procedure (the predicate *at-robby* signifies the room that the robot is present in)

4.2.3 Constraint Generation

In this phase, we account for certain intra-operator hard constraints that individual actions must follow and inter-operator soft constraints that actions observe with other actions. The constraint generation procedure is illustrated in the Figure 4.3.

Intra-operator constraints

The *intra-operator constraints* are accounted for during the course of creating elements of the relevant predicate dictionary RPD. In order to qualify as legitimate, each predicate of the set RPD_{a_i} of action a_i must satisfy some *intra-operator constraints* proposed by STRIPS (Fikes and Nilsson, 1971). Thus, for an action $a_i \in [a_1, a_2, \ldots, a_n]$ and relevant predicate $p \in relPre_{a_i}$:

- p cannot be in the *add* list and the *del* list for the same action.
- p cannot be in the *add* list and the *pre* list for the same action.

These constraints can be summarized in the Listing 4.2.

```
Relevant predicates for action move:
(at, at-robby)
Semantic constraints for action move:
```
$at \in del_{move} \Rightarrow at \in pre_{move}$
$at \in add_{move} \Rightarrow at \notin pre_{move}$
$at - robby \in del_{move} \Rightarrow at - robby \in pre_{move}$
$at - robby \in add_{move} \Rightarrow at - robby \notin pre_{move}$

LISTING 4.2: intra-operator constraints for action *move*

Soft Constraints

Next, we generate a set of soft constraints which direct the MAX-SAT algorithm to learn the most approximate representation of the action models. These constraints explore the inter-dependencies between the actions which constitute the traces. These dependencies may be short-term and long-term, and are detailed in the section below.

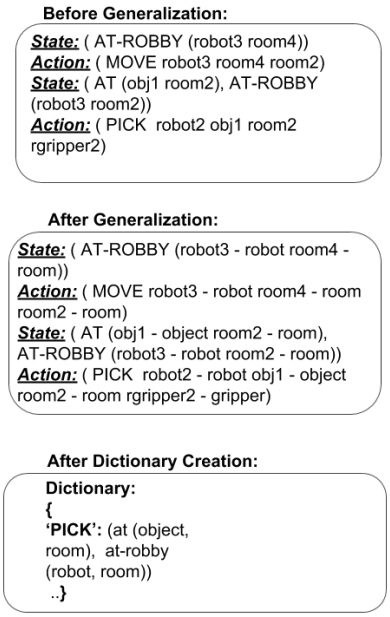

FIGURE 4.2: Diagram representing the first phase of the *SRMLearn* algorithm consisting of annotation and generalization. The generalization tags the objects in the signatures of the actions and predicates (i.e. robot2, robot3, obj1, room2, room3, room4) with their types as referenced from the dictionary ("robot": robot2, robot3, robot4), ("room": room2, room3, room4), ("object": obj1)

Short-Term Constraints: We hypothesize that if a sequential pair of actions appears frequently in the traces, there must be a reason for their frequent co-existence. We are thus interested in the branch of pattern mining algorithms which treat frequent sequential action pairs. Other algorithms in the pattern mining literature, like the apriori algorithm (Agrawal and Srikant, 1994) (used in ARMS (Yang, Wu, and Jiang, 2007)) are not equipped to satisfy this requirement. Bearing this requirement in mind, we parse the input traces and feed them to an algorithm called TRuleGrowth (Fournier-Viger, Nkambou, and Tseng, 2011), an algorithm for mining sequential rules common to several sequences that appear in a sequence database. It belongs to a family of algorithms constituting a data mining library called SPMF (Fournier-Viger et al., 2014), a comprehensive offering of implementations of data mining algorithms for association rule mining, sequential rule mining and sequential pattern mining among others. TRuleGrowth is a variation of the RuleGrowth algorithm (Fournier-Viger, Nkambou, and Tseng, 2011) with a window size constraint (representing the number of consecutive actions to consider). The input to TRuleGrowth is (1) a sequence database *T*, (2) three user-specified thresholds, namely $support$ (a value in [0, 1]) and $confidence$ (a value in [0, 1]) and (3) a parameter named $window\_size$ (an integer ≥ 0 representing the number of successive actions to consider, which in our case is 1 since we consider action pairs). Given a sequence database *T*, and the parameters $support$, $confidence$ and $window\_size$, TRuleGrowth outputs all sequential rules having a support and confidence higher than $support$ and $confidence$ respectively and which contain $window\_size$ number of consecutive actions. We test our system with a trace set consisting of 200 traces. Starting with 20 traces, we consistently double the number of traces till we reach 200. In the process, we identify frequent rules (action pairs) which consistently maintain the *confidence* and *support* over an increasing number of traces. These frequent pairs can be suspected to share a "semantic" relationship among themselves, in terms of a correlation between their pre, add and del lists. These constraints have been proposed by the ARMS (Yang, Wu, and Jiang, 2007) system,

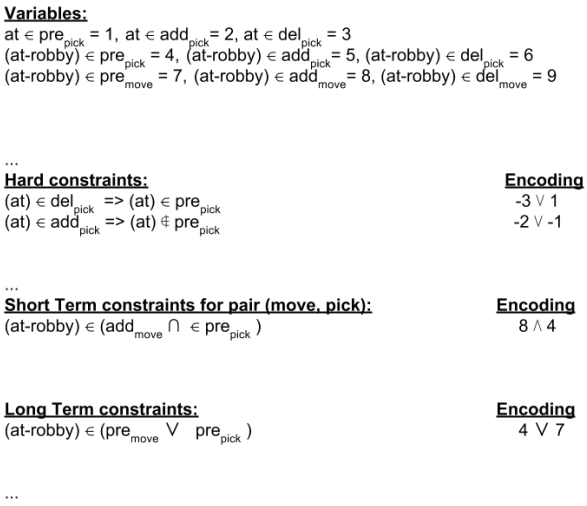

FIGURE 4.3: Diagram representing the second phase of the *SRMLearn* algorithm consisting of constraint generation. The relevant predicates of each of the actions are represented in the form of variables. The association of the relevant predicate $relPre_{a_i}$ in the (pre, add, del) list of the action is labelled in the form of positive integers. The representation of these variables in the form of intra-operator and inter-operator constraints is encoded with the help of De-Morgan's laws. These encodings are then converted into CNF form and passed to a weighted MAX-SAT solver which the objective of constraint satisfaction.

and serve as heuristics to explain the frequent co-existence of these actions. These heuristics are the only ones available in the literature which fit into our scheme of things, also producing good results in the case of the ARMS system. This serves as incentive to us for re-using the same. More precisely, if there is an action pair $(a_i, a_j), 0 \leq i < j \leq (n-1)$ where n is the total number of actions in the plan; and pre_i, add_i and del_i represent a_i's *pre*, *add* and *del* list, respectively:

- A predicate p such that ($p \in relPre_{a_i}$, $p \in relPre_{a_j}$) added by the first action ($p \in add_i$), which serves as a prerequisite for the second action a_j ($p \in pre_j$), cannot be deleted by the first action a_i. That is, a predicate which serves as a pre-requisite for two successive actions cannot be deleted by the first action.

- A relevant predicate p ($p \in relPre_{a_i}$, $p \in relPre_{a_j}$) is added by the first action a_i thus also appears in the *pre* list of the second action a_j. That is, a predicate that is added by the first action appears as a prerequsite for the second action in an action pair.

- A predicate p that is deleted by the first action a_i is added by a_j. In other words, an action re-establishes a predicate that is deleted by a previous action.

- The above plan constraints can be combined into one constraint and restated as: $\exists p \in (relPre_{a_i}, relPre_{a_j}) : ((p \in (pre_i \cap pre_j) \wedge p \notin (del_i)) \vee (p \in (add_i \cap pre_j)) \vee (p \in (del_i \cap add_j))$

The aforementioned constraints can be summarized in the listing 4.10.

Variables Resolved by MAX-SAT: (1, 7, 12, ...)
Learnt Model:

```
(:action pick
   :parameters (?r - robot ?obj - object ?room - room ?g - gripper)
   :precondition (and (at ?obj ?room) (at-robby ?r ?room) (free ?r ?g))
   :effect (and (carry ?r ?obj ?g) (not (at ?obj ?room)) (not (free ?r ?g)))
)
```

FIGURE 4.4: Diagram representing the third phase of the *SRMLearn* algorithm consisting of model reconstruction. The variables that evaluate to true by the weighted MAX-SAT solver are translated back to the predicates they represent, thus reconstructing the entire model back piece by piece.

Frequent action pairs for *gripper* domain: (pick, move)
Common relevant predicate for actions
move **and** *pick*: (at-robby)
constraint:: $\exists(at-robby)((at-robby \in (pre_{pick} \cap pre_{move})) \wedge at-robby \notin (del_{pick})) \vee (at-robby \in (add_{pick} \cap pre_{move})) \vee (at-robby \in (del_{pick} \cap add_{move}))$

LISTING 4.3: Short term constraints for action pair *(move, pick)*

Long-Term Constraints: We introduce this set of soft constraints to explore the relationships between a chain of actions constituting a plan. While the first two are proposed by the ARMS implementation (Yang, Wu, and Jiang, 2007), the third one is our contribution.

- If a predicate p is observed to be true for an action a_n of a plan sequence and p is a relevant predicate to $a_1, \ldots, a_i, \ldots, a_n$ where $0 \leq i < n$, then the predicate p must be generated, that is, exist in the *add* list of a_i. This can be expressed as $p \in add_{a_1} \vee add_{a_2} \vee \ldots \vee add_{a_{n-1}}$. This constraint, however, ceases to be true in the case that the predicate in question pre-exists in the initial state (Yang, Wu, and Jiang, 2007).
- The initial states of a plan contains many predicates, some of which are often applicable to the first action of a plan. In this case, we hypothesize that these predicates are preconditions of the first executed action in the trace (Yang, Wu, and Jiang, 2007).
- If a predicate p is observed to be true in the intermediate states right before an action a_k of a plan sequence and p is a relevant predicate to $a_k, a_{k+1}, \ldots, a_n$, then the predicate p must serve as a precursor to these following actions, that is, exist in the pre-list of a_k. This can be expressed as $p \in (pre_{a_k} \vee pre_{a_{k+1}} \vee pre_{a_{k+2}} \vee \ldots \vee pre_{a_n})$ (Arora et al., 2017).

The aforementioned constraints can be summarized in the listing 4.4.

trace Snippet::
[**(state)** (at-robby (..)) **(action)** (pick) **(action)**
(pick (..)) **(action)** (move (..)) **(action)** (drop (..))][**(state)**
(at (..)) **(action)** (pick (..))][**(action)** (drop (..)) **(action)**
(move (..))][**(action)**(drop (..))]

Long Term Constraints for predicate *at-robby*::
$(at-robby) \in (pre_{pick} \vee pre_{drop})$
$(at-robby) \in (add_{move} \vee add_{pick} \vee add_{drop})$

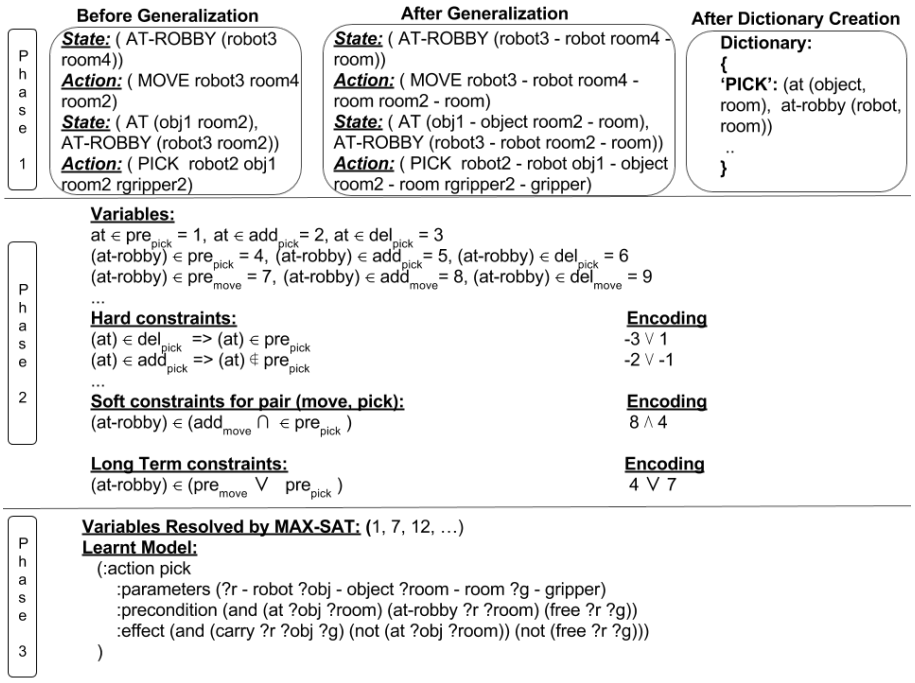

FIGURE 4.5: Illustration of our learning problem. The learnt action model is written in PDDL (Planning Domain Description Language) (McDermott et al., 1998) and conforms to the semantics of STRIPS (Fikes and Nilsson, 1971).

$$(at - robby) \in (pre_{pick})$$

LISTING 4.4: Long term constraints for *gripper* domain

4.2.4 Constraint Resolution and Model Reconstruction

As seen in the previous subsections, the relevant predicates to each action can be encoded in the form of constraints. For each relevant predicate $relPre_{a_i}$ to an action a_i, we create three constraints: one each for the *pre, add* and *del* list. Thus the number of relevant constraints $relCons_{a_i}$ for an action a_i is thrice the number of relevant predicates for the same action. These relevant constraints are then used to generate inter and intra-operator constraints. This process is better illustrated in Figure 4.3. Each of these constraints has a specific weight. While the weights of the intra-operator constraints and the long term constraints is varied between 50-100, the weight of the short term constraints is equivalent to the support of the rules which are associated with the frequent action pairs obtained with the TRuleGrowth algorithm. A CNF formula which is formed by the conjunction of these weighted constraints is then fed to a weighted MAX-SAT solver (Borchers and Furman, 1998). This solver finds a truth assignment that maximizes the total weight of the satisfied constraints, thus producing as output the constraints which evaluate to true. The constraints which evaluate to true are used for the reconstruction of the entire model.

4.3 Evaluation

The objective of our evaluation is to convert the hard and soft constraints generated from the traces into clauses, feed them to a MAX-SAT solver, record the resolved constraints, and use these resolved constraints to reconstruct the action model. We measure the accuracy of our

learning algorithm by performing a syntactic comparison between the reconstructed model and the ground truth action model for each of our evaluated domains. The details of this evaluation are listed in the following subsections.

The traces for the domains must be parsed and converted into a specific format which can serve as an input to the TRulesGrowth algorithm. The emerging rules emerging from the TRulesGrowth algorithm along with their *confidence* and *support* are recorded. This is iteratively done while doubling the number of traces from 20, 50, 100, 200. The rules maintaining a consistently high value of *confidence* and *support* with increasing number of traces for domains are recorded since they represent a highly frequent and sequential pair of actions. The confidence and support are varied between the values of 0.8 and 1.0 for all the tests.

The relevant predicates to each action are encoded in the form of intra and inter-operator constraints. A CNF formula which is formed by the conjunction of these weighted constraints (weighted with the scheme mentioned in Section 4.2.4) is then fed to a weighted MAX-SAT solver. This solver finds a truth assignment that maximizes the total weight of the satisfied constraints, thus producing as output the constraints which evaluate to true. The constraints which evaluate to true are used for the reconstruction of the entire model. This reconstructed model is termed as the empirical model. This model is compared with artificial models which are considered as the ground truth. The difference between the ground truth model and the empirically determined model is represented in the form of a *reconstruction error*. This error is based on the similarity among the predicates between the empirical model and ground truth model. Let $diffpre_{a_i}$ represent the syntactic difference in *pre* lists of action a_i in the ground truth (or ideal) model and the empirical model. Each time the *pre* list of the ideal model presents a predicate which is not in the *pre* list of the empirical model, the count $diffpre_{a_i}$ is incremented by one. Similarly, each time the *pre* list of the empirical model presents a predicate which is not in the *pre* list of the ideal model, the count $diffpre_{a_i}$ is incremented by one. Similar counts are estimated for the *add* and *del* lists as $diffadd_{a_i}$ and $diffdel_{a_i}$ respectively. This total count is then divided by the number of relevant constraints for that particular action $relCons_{a_i}$ to obtain the cumulative error per action. This error is summed up for every action and averaged over the number of actions of the model to obtain an average error E for the entire model.

The cumulative error for the model is thus represented by:

$$E = \frac{1}{n}\sum_{i=1}^{n}\frac{diffPre_{a_i} + diffAdd_{a_i} + diffDel_{a_i}}{relCons_{a_i}} \tag{4.1}$$

During the course of our experiments, we evaluate the performance of SRMLearn with our implementation of ARMS over five domains as follows: for each of the domains, we set the number of traces as (20, 50, 100, 200), solve them with the help of two SAT solvers ((Borchers and Furman, 1998; Kautz, Selman, and Jiang, 1996)) and calculate the cumulative error for SRMLearn and our implementation of ARMS. It should be noted that the key difference between SRMLearn and ARMS lies in the constraint set and chosen data mining algorithm. The error is calculated whilst augmenting the number of traces till 200, after which the obtained error percentages are plotted against each other. The evaluation domains include *depots, parking, mprime, gripper* and *satellite* and are represented in the figures 4.6, 4.7, 4.8, 4.9 and 4.10. In the first four domains, the error percentages with SRMLearn are lower than with ARMS, which can be attributed to the injection of short term constraints. As these constraints are directly obtained from the traces and signal the presence of correlated action pairs, they guide the SAT solver accordingly by reinforcing the constraints represented by the action pairs. This produces a more accurate model. The higher accuracy of SRMLearn in these domains can be attributed to the fact that sequential frequent action pairs with the

TRuleGrowth algorithm demonstrate a stronger correlation than frequent action pairs in the case of the apriori algorithm. The near equal error percentage in the case of the *satellite* domain is assumed to be due to a reduced influence of short term constraints with 200 traces, resulting in near identical errors produced by the two algorithms. A summary of the number of traces per domain, average number of actions per trace, the number of constraints evaluated per domain, and the execution times for a range of traces are summarized in Table 4.1.

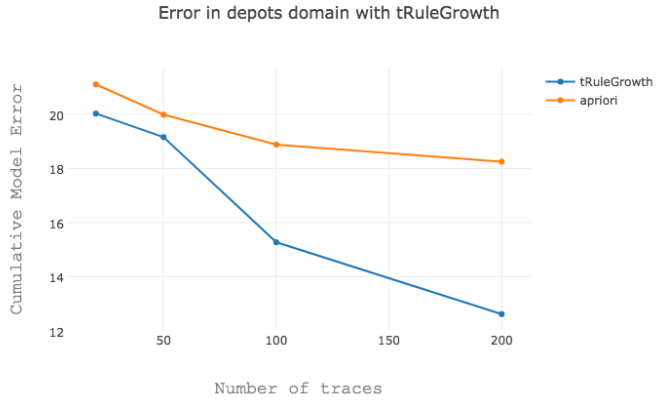

FIGURE 4.6: Plot for error comparison in depots domain (number of traces v/s error)

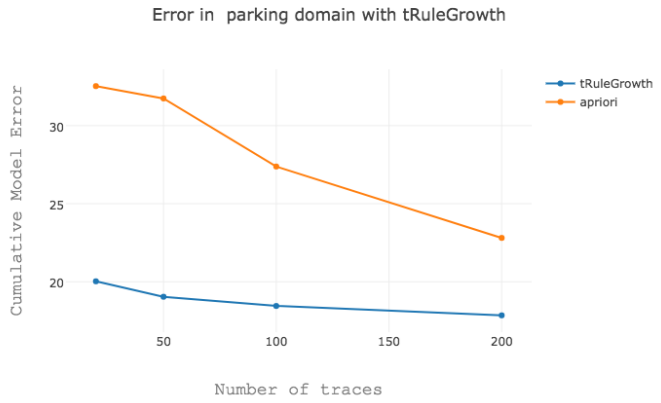

FIGURE 4.7: Plot for error comparison in parking domain (number of traces v/s error)

4.4 Future Perspectives: Learning temporal models

The planning community has increasingly oriented its research towards solving real life problems. For example, interest in planning demonstrated by the space research community principally in observation scheduling and planetary rover exploration, logistics planning, and manufacturing processes among others. The International Planning Competitions have acted as an important motivating force behind the progress that has been made in planning since 1998. This re-orientation, however, inevitably brings problems involving time into the scheme of things. In the third competition the authors therefore took the initiative in defining an expressive language capable of modeling temporal domains. This language was called *PDDL 2.1*, and has been designed to be backward compatible with the version of PDDL

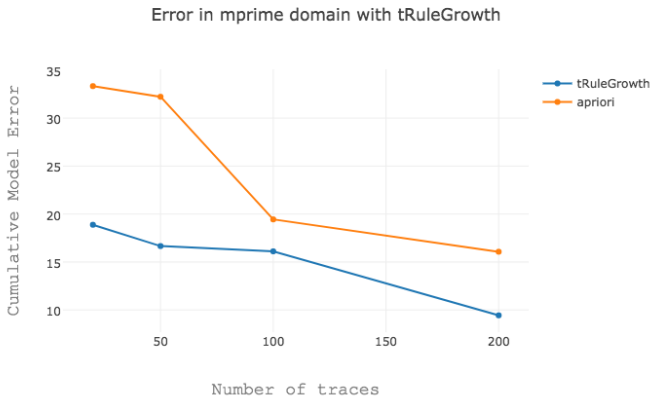

FIGURE 4.8: Plot for error comparison in mprime domain (number of traces v/s error)

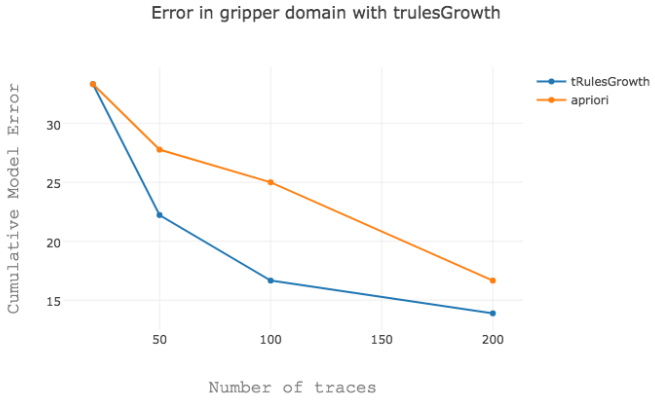

FIGURE 4.9: Plot for error comparison in gripper domain (number of traces v/s error)

prevalent since 1998 (Fox and Long, 2003). This version of the language re-introduces actions laced with temporal properties as *durative actions*. There may be two forms of *durative actions*, namely *discretised durative actions* and *continuous durative actions*. State changes induced by action application are considered to be instantaneous, thus the continuous aspects of a continuous durative action refer only to how numeric values change over the action interval of the action.

The preconditions and effects of actions are also annotated with various durative terminologies. The duration of the action is represented by the *:duration* attribute. The annotation of a precondition signifies whether the associated predicate must hold at the start of the interval (the instant of action application, illustrated with the *at start* construct), the end of the interval (the instant at which the final effects of the action are visible, illustrated with the *at end* construct) or over the interval from the start to the end (invariant over the duration of the application of the action, illustrated with the *overall* construct). The annotation of an effect signifies whether the effect is instantaneous (occurring at the start of the interval) or delayed (occurring at the end of the interval). Invariant predicates that hold over an interval that is open at both ends (starting and ending at the end points of the action) are expressed using the *over all* construct. If it is required to specify that predicate p holds in the closed interval over the duration of a durative action, then three conditions are required: (*at start* p), (*over all* p) and (*at end* p). If a condition is required as a precondition as well as an invariant condition,

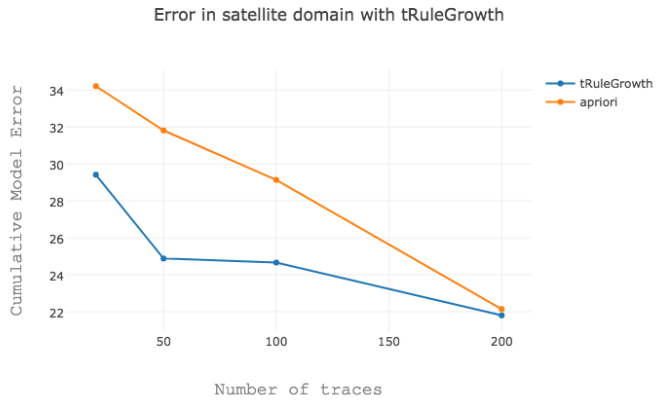

FIGURE 4.10: Plot for error comparison in satellite domain (number of traces v/s error)

it means that any action that affects the invariant must start after the end of the action requiring that invariant. Prior to the introduction of time into PDDL, all plans were interpreted sequentially. However, durative actions incentivize the exploitation of concurrency. That is, actions can have pre- and postconditions that are local to the two end-points of the action, and a planner can choose to exploit a durative action for effects it has at its start or at its end. An example of a durative action for the *parking* domain is mentioned in the Listing 4.5 below.

```
(:durative-action move-curb-to-curb
        :parameters (?car - car ?curbsrc ?curbdest - curb)
        :duration (= ?duration 1)
        :condition (and
                (at start (car-clear ?car))
                (at start (curb-clear ?curbdest))
                (at start (at-curb-num ?car ?curbsrc))
        )
        :effect (and
                (at start (not (curb-clear ?curbdest)))
                (at end (curb-clear ?curbsrc))
                (at end (at-curb-num ?car ?curbdest))
                (at start (not (at-curb-num ?car ?curbsrc)))
        )
)
```

LISTING 4.5: Durative PDDL domain

For an action with precondition p to hold at time t,there must be a half open interval immediately preceding t in which p holds. To tackle concurrency, an approach which is considered to be on the lines of the mutual exclusion principle in the case of shared memory is implemented. It is identical to the idea of prohibiting the simultaneous read and write of a variable in the case of POSIX threads abiding by the law of *mutual exclusion*. Precisely, no two actions can simultaneously access value if one of the two is accessing the value to update it. To implement the mutual exclusion principle, we require non-zero-separation between mutually exclusive action end points. Planners can thus exploit concurrency by ensuring only that there is non-zero separation between conflicting start and end points of actions.

TABLE 4.1: Summary of characteristics of traces and their execution time

Domain	Number of Constraints	Number of Traces	Average number of actions per trace	Execution time (seconds)
Mprime	42	20	4	4.68
	42	50		7.02
	42	100		17.87
	42	200		75.67
Depots	129	20	21	41.32
	132	50		233.24
	129	100		1134.22
	129	200		5420.12
Parking	69	20	6	5.39
	69	50		10.83
	69	100		34.37
	68	200		194.33
Gripper	30	20	13	13.39
	30	50		63.10
	30	100		223.92
	30	200		1018.52
Satellite	45	20	7	12.01
	54	50		79.19
	53	100		303.48
	54	200		1327.04

4.4.1 Temporal intra-operator and inter-operator constraints

The temporal intra-operator constraints are identical to the constraints presented in Section 4.2, with the addition of a temporal perspective. In this perspective, all the predicates potentially figuring in the pre list are augmented with conditions *start, overall* and *end*. This gives rise to three possible constraints for each predicate in place of one possible constraint (as in the case of intra-operator constraints in Section 4.2). For example, if these three temporal labels are associated to each predicate p in the precondition list of action a, it gives rise to three representations, namely: $(p) \in (pre, start)_a, (p) \in (pre, overall)_a, (p) \in (pre, end)_a$ representing that the predicate p which is relevant to the action a may be associated to any one of the *start, overall* or *end* labels respectively. Similarly in the case of the effects, the predicate may either be associated with the *start* or with the *end* temporal label. This association may be represented as follows, for the predicate p and action a: $(p) \in (add, start)_a, (p) \in (add, end)_a, (p) \in (del, start)_a, (p) \in (del, end)_a$.

The modified temporal intra-operator constraints are represented in the Listing 4.6:

intra-operator temporal constraints for predicate *at-robby*::

$(at-robby) \in (add, start)_{move} \Rightarrow (at-robby) \notin (pre, start)_{move}$
$(at-robby) \in (add, overall)_{move} \Rightarrow (at-robby) \notin (pre, start)_{move}$
$(at-robby) \in (add, end)_{move} \Rightarrow (at-robby) \notin (pre, start)_{move}$
$(at-robby) \in (add, start)_{move} \Rightarrow (at-robby) \notin (pre, overall)_{move}$
$(at-robby) \in (add, overall)_{move} \Rightarrow (at-robby) \notin (pre, overall)_{move}$
$(at-robby) \in (add, end)_{move} \Rightarrow (at-robby) \notin (pre, overall)_{move}$
$(at-robby) \in (add, start)_{move} \Rightarrow (at-robby) \notin (pre, end)_{move}$
$(at-robby) \in (add, overall)_{move} \Rightarrow (at-robby) \notin (pre, end)_{move}$
$(at-robby) \in (add, end)_{move} \Rightarrow (at-robby) \notin (pre, end)_{move}$
$(at-robby) \in (del, start)_{move} \Rightarrow (at-robby) \notin (pre, start)_{move}$

$$(at-robby) \in (del, end)_{move} \Rightarrow (at-robby) \notin (pre, end)_{move}$$

LISTING 4.6: Intra-operator temporal constraints for *gripper* domain

Similarly, the short-term constraints mentioned in Section 4.4 are modified and presented as follows, for a common relevant predicate *at-robby* for two successive actions *pick* and *move*:

- A predicate which serves as a pre-requisite for two successive actions cannot be deleted by the first action. That is:

$$\exists(at-robby)(at-robby \in (((pre, beg)_{pick} \cap (pre, beg)_{move})) \wedge at-robby \notin ((del, beg)_{pick}) \vee (((pre, overall)_{pick} \cap (pre, overall)_{move})) \wedge at-robby \notin ((del, beg)_{pick}) \vee (((pre, end)_{pick} \cap (pre, end)_{move})) \wedge at-robby \notin ((del, beg)_{pick}) \vee (((pre, beg)_{pick} \cap (pre, beg)_{move})) \wedge at-robby \notin ((del, end)_{pick}) \vee (((pre, overall)_{pick} \cap (pre, overall)_{move})) \wedge at-robby \notin ((del, end)_{pick}) \vee (((pre, end)_{pick} \cap (pre, end)_{move})) \wedge at-robby \notin ((del, end)_{pick})))$$

LISTING 4.7: First subset of short term temporal constraints for action pair *(move, pick)*

- A relevant predicate p ($p \in relPre_{a_i}$, $p \in relPre_{a_j}$) is added by the first action a_i thus also appears in the *pre* list of the second action a_j. That is, a predicate that is added by the first action appears as a prerequsite for the second action in an action pair. That is:

$$\exists(at-robby)(at-robby \in (((add, beg)_{pick} \cap (pre, beg)_{move}) \vee ((add, beg)_{pick} \cap (pre, overall)_{move}) \vee ((add, beg)_{pick} \cap (pre, end)_{move}) \vee ((add, overall)_{pick} \cap (pre, beg)_{move}) \vee ((add, overall)_{pick} \cap (pre, overall)_{move}) \vee ((add, overall)_{pick} \cap (pre, end)_{move}) \vee ((add, end)_{pick} \cap (pre, beg)_{move}) \vee ((add, beg)_{pick} \cap (pre, overall)_{move}) \vee ((add, end)_{pick} \cap (pre, end)_{move})))$$

LISTING 4.8: Second subset of short term temporal constraints for action pair *(move, pick)*

- A predicate p that is deleted by the first action a_i is added by a_j. In other words, an action re-establishes a predicate that is deleted by a previous action. That is:

$$\exists(at-robby)(at-robby \in ((((del, beg)_{pick} \cap (add, beg)_{move}) \vee ((del, beg)_{pick} \cap (add, overall)_{move}) \vee ((del, beg)_{pick} \cap (add, end)_{move}) \vee ((del, end)_{pick} \cap (add, beg)_{move}) \vee ((del, beg)_{pick} \cap (add, overall)_{move}) \vee ((del, end)_{pick} \cap (add, end)_{move}))))$$

LISTING 4.9: Third subset of short term temporal constraints for action pair *(move, pick)*

- The above plan constraints can be combined into one constraint and restated as:

Frequent action pairs for *gripper* domain: (pick, move)
Common relevant predicate for actions
move **and** *pick*: (at-robby)
$\exists(at-robby)(at-robby \in (((add, beg)_{pick} \cap (pre, beg)_{move}) \vee ((add, beg)_{pick} \cap (pre, overall)_{move}) \vee ((add, beg)_{pick} \cap (pre, end)_{move}) \vee ((add, overall)_{pick} \cap (pre, beg)_{move}) \vee ((add, overall)_{pick} \cap (pre, overall)_{move}) \vee ((add, overall)_{pick} \cap (pre, end)_{move}) \vee ((add, end)_{pick} \cap (pre, beg)_{move}) \vee ((add, beg)_{pick} \cap (pre, overall)_{move}) \vee ((add, end)_{pick} \cap (pre, end)_{move}) \vee ((del, beg)_{pick} \cap (add, beg)_{move}) \vee ((del, beg)_{pick} \cap (add, overall)_{move}) \vee ((del, beg)_{pick} \cap (add, end)_{move}) \vee ((del, end)_{pick} \cap (add, beg)_{move}) \vee ((del, beg)_{pick} \cap (add, overall)_{move}) \vee ((del, end)_{pick} \cap (add, end)_{move}) \vee (((pre, beg)_{pick} \cap (pre, beg)_{move})) \wedge$

$$at - robby \notin ((del, beg)_{pick}) \vee (((pre, overall)_{pick} \cap (pre, overall)_{move})) \wedge at - robby \notin ((del, beg)_{pick}) \vee (((pre, end)_{pick} \cap (pre, end)_{move})) \wedge at - robby \notin ((del, beg)_{pick}) \vee (((pre, beg)_{pick} \cap (pre, beg)_{move})) \wedge at - robby \notin ((del, end)_{pick}) \vee (((pre, overall)_{pick} \cap (pre, overall)_{move})) \wedge at - robby \notin ((del, end)_{pick}) \vee (((pre, end)_{pick} \cap (pre, end)_{move})) \wedge at - robby \notin ((del, end)_{pick}))))$$

LISTING 4.10: Combined set of short term temporal constraints for action pair *(move, pick)*

While the theoretical foundations of this approach have been put into place during the course of this study, the implementation and evaluation of the results is left as a scope for future work.

4.5 Conclusions

To learn operators in APS, it is possible to exploit intra-operator and inter-operator constraints. While the intra-operator constraints deal with the structure of the operator as a whole, the inter-operator ones exploit the semantic relationship between successive operators. We present an approach called SRMLearn, which encodes the aforementioned constraints in the form of a maximum satisfiability problem (MAX-SAT) and solves it with a weighted MAX-SAT solver to obtain the underlying operator blueprint (action model). Unlike previous MAX-SAT driven approaches, our chosen constraints exploit the relationship between consecutive actions, rendering more accurately learnt actions in the end. Our approach is capable of learning purely from traces, subtracting the need for domain knowledge. Empirical results demonstrate that our approach outperforms the ARMS implementation (Yang, Wu, and Jiang, 2007) due to its choice of constraints and pattern mining algorithm. As the planning community strives to tackle problems which are closer to the real world, there is a growing need to include the notion of time in the domain representations. To cater to this need, version 2.1 of PDDL is equipped with *durative actions* equipped with an *execution duration* and various temporal conditions to label the execution instant of a predicate during the course of action execution. The intra and inter-operator constraints developed in the *SRMLearn* system are extended to integrate a temporal aspect. While the theoretical foundations of this extension have been laid out in this study, the implementation and evaluation of the results is left as a scope for future work.

Chapter 5

Connectionist Techniques: The PDeepLearn System

5.1 Introduction

In a bid to reconstruct the underlying action model with the intent of using it to subsequently plan again, the margin of error is very small. The learnt model may be missing predicates or these predicates may be misplaced in the wrong lists. The displacement of a single predicate can render the model to be unusable to plan again without the intervention of a human domain expert to fill in the blanks. Most approaches in literature have used upward model building approaches to learn action models, gradually building the model component by component. These approaches either begin with a partial model or no model at all, which is eventually completed after the application of the learning algorithm. These approaches, however, fall short in providing a complete reconstruction of the domain model. To sum up, most approaches fall short of using a learned model to plan again as anything less than a flawless reconstruction of the underlying model renders it unusable.

We turn the problem on its head and look at it from a different perspective: given a set of all possible and exhaustively generated candidate models specific to a domain, are we able to devise a method to identify the speculated ideal model which best explains the traces? By means of this exhaustive generation, we ensure that we do not skip any predicate or sensitive information, and are eventually able to isolate a model speculated to be identical to the hand woven model. This hand woven model is necessary to serve as a benchmark to help evaluate the accuracy of our learning algorithm. The stronger the match between the learnt and the hand woven model, the more precise our learning algorithm is. This isolated model is thus called the speculated ideal model, and can directly be plugged into a planner to generate a fresh sequence of plans.

To do this, we explore the short-term and long-term dependencies between actions to converge to the speculated ideal model. Given the candidate model set, we explore short-term dependencies by pattern mining techniques and long-term dependencies based on sequence labeling techniques by using long short-term memory (LSTM) techniques for the acquisition of the underlying domain model in the output. Empirical results highlight that the sequence labelling achieved with an empirically acquired model proven to be syntactically identical to the hand woven model (ground truth model used to generate training data) attains the highest accuracy compared to other models in the set. This structural consistency of the acquired model with the hand woven model renders this LSTM-based sequence labelling approach to learn action models as an effective one. The biggest advantage of our approach can be seen from the fact that given a set of state-action interleaved traces, our approach is capable of learning a domain model which is structurally consistent to the hand woven model. Connectionist models in general are known to provide good performance, both in terms of training time as well as the quality of the learnt models (Mourão et al., 2012). Since deep learning by virtue is capable of learning intricate structures in high-dimensional data (LeCun, Bengio,

and Hinton, 2015), it is a viable option to explore in the case of complex domains where learners tend to struggle (Yang, Wu, and Jiang, 2007). Our approach is called PDeepLearn, which stands for **P**DDL domain **Deep Learn**er. As the name suggests, the approach learns PDDL (McDermott et al., 1998) domain models with the help of deep learning techniques. In particular, the noteworthy successes of LSTM (Hochreiter and Schmidhuber, 1997), a deep learning technique being increasingly used in learning long range dependencies in the fields of speech and handwriting recognition (LeCun, Bengio, and Hinton, 2015; Goodfellow, Bengio, and Courville, 2016), are exploited. A plan is an orchestrated execution of a series of actions which are by virtue interdependent in order to execute. Since a plan exhibits long-term dependencies and the LSTM by virtue exploits the same; there is a direct link between the principle of operation of the LSTM and plan execution, which this chapter seeks to explore. This chapter details out the specifics of our approach in Section 5.2, as well as the obtained results in Section 5.3. It concludes with Section 5.4.

5.2 The PDeepLearn Algorithm

The approach followed in PDeepLearn is divided into three phases, which are summarized in Figure 5.1. In the first phase, we enumerate all possible candidate action models. In phase two, we identify frequent action pairs using pattern mining algorithms. *Action pair constraints* are applied to the frequent action pairs to eliminate improbable candidate models. In the final phase, LSTM based techniques are used to label action sequences with the intent of identifying the speculated ideal model which produces the best labelling accuracy. This model is structurally consistent with the hand woven model. The algorithm for the PDeepLearn approach is specified in figure 5.1.

5.2.1 Overview and Definitions

Any sequence of actions are likely to exhibit implicit patterns and regularities within them. These patterns may be of the form of frequently co-occurring actions, which indicate the possible presence of inter-action dependencies and relationships. These relationships can be uncovered by means of pattern mining techniques to facilitate the process of learning. Each action name (signature omitted) in plan traces is viewed as an *itemset*, which has only one element, and a plan trace as a sequence. Whilst uncovering inter-action relationships with the aid of pattern mining, we lay emphasis on the actions constituting the traces and not the states. The set of traces T can now be treated as a *Sequence Database* (SD) in the pattern mining domain. An SD is a set of sequences of itemsets $T = [s_1, s_2, \ldots, s_s]$ where each sequence represents a trace. Each sequence is an ordered list of items (in our context, each item in the sequence is equivalent to an action) $s_i = [a_1, a_2, \ldots, a_n]$. A sequential rule $a_x \rightarrow a_y$ is a relationship between two actions a_x, $a_y \in A$ such that if a_x occurs in a sequence, then a_y will occur successively in the same sequence. Two measures are defined for sequential rules. The first one is the *sequential support*. For a rule $a_x \rightarrow a_y$, it is defined as $sup(a_x \rightarrow a_y) = |a_x \rightarrow a_y|/|T|$. The second one is the *sequential confidence* and it is defined as $conf(a_x \rightarrow a_y) = |a_x \rightarrow a_y|/|a_x|$, where $|a_x \rightarrow a_y|$ represents the number of times the a_x and a_y appear in succession in T, and $|a_x|$ represents the number of times a_x appears in T. The values of the confidence and support of a_x and a_y is directly proportional to the frequency of the sequential co-occurrence of a_x and a_y in the SD (Fournier-Viger et al., 2012).

The phases of the algorithm can be better understood from the pseudo code outlined in Algorithms 3,4 and 5.

Algorithm 3 Overview of the PDeepLearn Algorithm

Input: set of plan traces *T*, dictionary VT associating objects with their types *(type, object)*
Output: speculated ideal model $spec_m$
Generate candidate model set *M* = *generateAllPossiblemodels(T, VT)*
Semantic Constraint Adherence: eliminate models from *M* which do not satisfy semantic constraints;
Action Pair Constraint Adherence: Identify frequent action pairs from TRuleGrowth Algorithm and proceed with pruning phase to eliminate improbable models from *M*;
Identify the speculated ideal model $spec_m = encodeLSTM(T, M)$
Return $spec_m$

Algorithm 4 Procedure *generateAllPossible models (T, VT)*

Input: Set of plan traces *T*, dictionary VT associating objects with their types *(type, object)*
Output: Candidate model set *M* consisting of all possible combinations of predicates giving rise to all possible candidate action models
for `j = 1 to T` **do**
    Replace all objects in signatures of action and predicates in T with their types referenced from the *(type, object)* pairs to build generalized action schema A_s and predicate schema P_s respectively
    for a_i `in` A_s **do**
        for p_i `in` P_s **do**
            If p_i relevant to a_i, then add to list $relPre_{a_i}$ containing relevant predicates for a_i
        Find all possible combinations of $relPre_{a_i}$ to generate candidate action set CAS_{a_i} for a_i
        Add to candidate action dictionary an entry with (key, value) as (a_i, $relPre_{a_i}$)
    for a_i `in` A_s **do**
        Append to M the CAS_{a_i}
    Return M

Algorithm 5 Procedure *encodeLSTM (T, M)*

Input: Set of plan traces *T*, reduced set of possible candidate action models *M*
Calculate the maximum length among all the traces and store in $batchLen$
for `j = 1 to T` **do**
    Encoding for Training Phase: Encode each action in the form of input and output vector with encoding scheme mentioned in Section 5.2.6
    Pad length of trace with vectors containing zeros so that each trace has uniform length $batchLen$
Sample set m' from *M*
for `i = 1 to` m' **do**
    Encoding for Validation Phase: Encode each action in the form of input and output vector with encoding scheme mentioned in Section 5.2.6 with model i;
Run the computational graph and identify speculated ideal model $spec_m$ from m' as one with highest prediction accuracy
Return $spec_m$

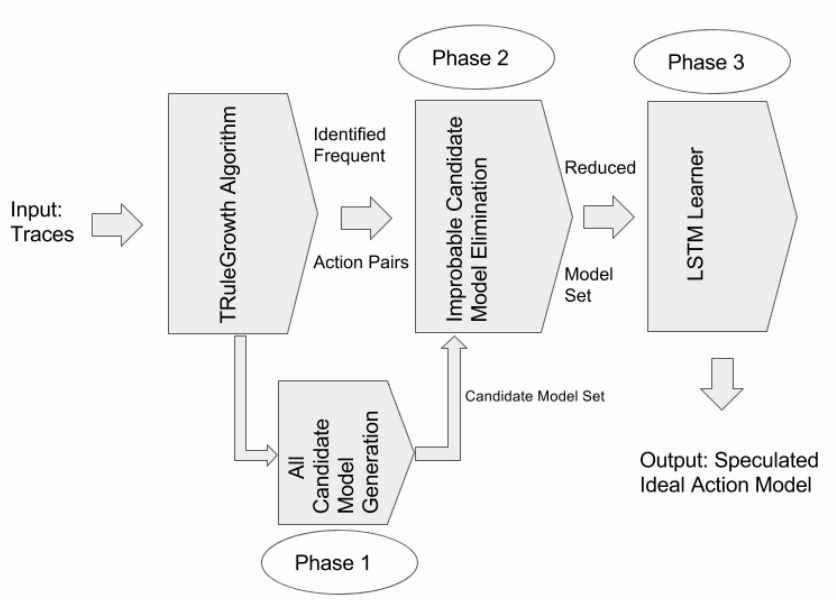

FIGURE 5.1: PDeepLearn approach phases

5.2.2 Candidate model Generation

In this subsection, all possible candidate action models are generated from the input plan traces. Firstly, each trace is taken one by one, and each action as well as each predicate in the initial, goal and intermediate states of the plan trace is scanned to substitute the instantiated objects in the signatures with their corresponding variable types. This substitution is done by referencing the dictionary VT that associates instantiated objects with their types. These associations are drawn from the problem definition. This builds the corresponding generalized action schema A_s and predicate schema P_s respectively. Each action in the action schema is associated with its relevant predicates from the predicate schema, where a predicate $p_{a_i} \in P_s$ is said to be *relevant* to an action $a_i \in A_s$ if they share the same variable types. The objective is to find for every action, the predicates that can be applied to it and thus constitute the *(pre, add, del)* lists of that specific action. For example, the predicate *at-robby (?robot, ?room)* is relevant to the action *find (?robot, ?object, ?room, ?gripper)* as both of them contain the parameter types *(robot, room)* in their signatures and the types $(robot, room) \subset (robot, object, room, gripper)$. The set of relevant predicates to an action a_i can be denoted as $relPre_{a_i}$. With schemas A_s and P_s, a candidate action dictionary is built. An illustration of this generalization and dictionary construction step is given in Figure 5.2. Each *key* in the dictionary is the name of the action ($a_i \in A_s$) and the *value* is a list of all predicates relevant to that particular action ($relPre_{a_i} \in P_s$). All possible combinations of relevant predicates per key are represented as elements of a candidate action set CAS. We represent the elements of the candidate action set of each action $a_i \in A_s$ (CAS_{a_i}) in the format (pre, add, del). For example, the number of elements of the candidate action set for action a_i containing m relevant predicates is represented as $(\prod_{i=1}^{n} \binom{m}{i})^3$. At this stage, the *semantic constraints* mentioned in the fourth coming Section 5.2.3 are applied to the elements of the the candidate action sets of each of the actions. This step eliminates various semantically incorrect models from propagating further into the chain. The cartesian product of all these remaining candidate action sets (after the elimination step with the semantic constraints) per action constitute the set of all possible candidate models M for the domain. The size of M is directly proportional to the complexity of the domain i.e. number of actions and predicates constituting the domain. We use the *gripper* domain to illustrate the

aforementioned steps, taking the example of the *pick* action. The candidate model generation procedure for the *pick* action is represented in Listing 5.1. It is also detailed in the algorithm represented by the Figure 4.

```
trace snippet before generation phase:: [
(state) ((at ball5 room1) (at-robby robot3 room1))
(action) (pick robot3 ball5 room1 rgripper3)
(state) ((carry robot3 ball5 room1))]
trace snippet after generation phase::
[(state) ((at ball room) (at-robby robot room))
(action) (pick robot ball room rgripper)
(state) ((carry robot ball room))]
Relevant predicates for action pick: (at, at-robby)
Candidate action set for action pick: {(at (obj0 - object, roo1 - room),
(), ¬at-robby (rob0 - robot, roo1 - room)), ((at-robby (rob0 -
robot, roo1 - room), (), ¬at (obj0 - object, roo1 - room)),...}
```

LISTING 5.1: Candidate model generation procedure for the *pick* action of the *gripper* domain

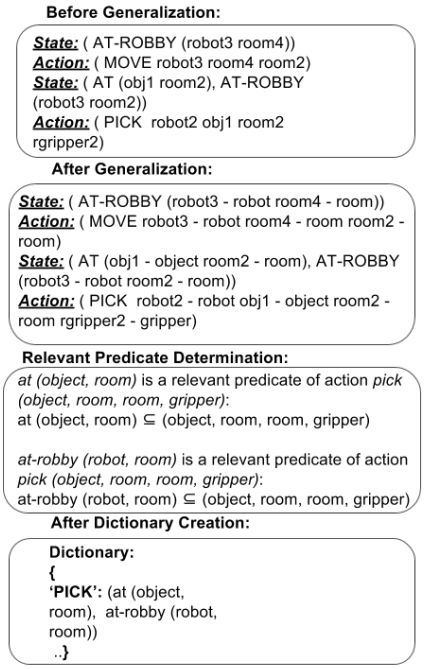

FIGURE 5.2: Candidate model generation procedure for the *pick* action of the *gripper* domain

5.2.3 Semantic Constraint Adherence

The *semantic constraints* are accounted for during the course of creating elements of the candidate action set CAS. In order to qualify as legitimate, each element of the candidate action set CAS_i of action a_i must satisfy some *semantic constraints* proposed by STRIPS (Fikes and Nilsson, 1971). Thus, for each action $a_i \in [a_1, a_2, \ldots, a_n]$ and relevant predicate $p \in relPre_{a_i}$:

- p cannot be in the *add* list and the *del* list for the same action.

- *p* cannot be in the *add* list and the *pre* list for the same action.

Elements which do not adhere to these constraints are deleted from CAS_i. This step is quintessential in performing a first round of filtration, thus weeding out a large number of improbable elements, leaving us with more accurate ones which are cross multiplied in order to obtain the set of all possible candidate models M. These constraints can be summarized in the listing 5.2.

Relevant predicates for action *move*:
`(at, at-robby)`
Semantic constraints for action *move*:
$at \in del_{move} \Rightarrow at \in pre_{move}$
$at \in add_{move} \Rightarrow at \notin pre_{move}$
$at - robby \in del_{move} \Rightarrow at - robby \in pre_{move}$
$at - robby \in add_{move} \Rightarrow at - robby \notin pre_{move}$

LISTING 5.2: Semantic constraints for action *move*

5.2.4 Sequence Pattern Mining

We hypothesize that if a sequential pair of actions appears frequently in the traces, there must be a reason for their frequent co-existence. We are thus interested in the branch of pattern mining algorithms which treat frequent sequential action pairs. Other algorithms in the pattern mining literature, like the apriori algorithm (Agrawal and Srikant, 1994) (used in ARMS (Yang, Wu, and Jiang, 2007)) are not equipped to satisfy this requirement. This is because of the fact that the apriori algorithm identifies the itemsets which are frequent but not necessarily sequential in a SD. This does not correspond to our requirement, which is to isolate frequent sequential itemsets (actions in our case). Bearing this requirement in mind, the input traces are parsed and fed to an algorithm called TRuleGrowth (Fournier-Viger et al., 2012), an algorithm used for mining sequential rules common to several sequences that appear in a sequence database. It belongs to a family of algorithms constituting a data mining library called SPMF (Fournier-Viger et al., 2014), a comprehensive offering of implementations of data mining algorithms. The inputs to TRuleGrowth are (1) a sequence database T, (2) *support* (a value in [0, 1]) and *confidence* (a value in [0, 1]) and (3) a parameter named $window\_size$ (an integer ≥ 0, set to 1 in our case since we consider action pairs). Given a sequence database T, and the parameters $support$, $confidence$ and $window\_size$, TRuleGrowth outputs all sequential rules having a support and confidence higher than $support$ and $confidence$ respectively containing $window\_size$ number of consecutive actions. Starting with 10 traces, we consistently double the number of traces till we reach 1000. In the process, we identify frequent rules (action pairs) which consistently maintain the values of *confidence* and *support* (empirically determined) over an increasing number of traces.

5.2.5 Action Pair Constraint Adherence

After identifying frequent action pairs in the previous step, this step filters elements of the candidate action set for the actions constituting the frequent action pairs. These frequent pairs are suspected to share a "semantic" relationship among themselves, in terms of a correlation between their pre, add and del lists. These relationships serve as *action pair constraints*, and have been proposed by the ARMS (Yang, Wu, and Jiang, 2007) system, serving as heuristics to explain the frequent co-existence of these actions. These heuristics produce good results in the case of the ARMS system, which serves as incentive for re-using the same. More precisely, if there is an action pair $(a_i, a_j), 0 \leq i < j \leq (n-1)$ where n is the total number of actions in the plan; and pre_i, add_i and del_i represent a_i's *pre*, *add* and *del* list, respectively:

- A predicate p which serves as a precondition in the *pre* lists of both a_i and a_j cannot be deleted by the first action. Formally:
  $\exists p(p \in (pre_i \cap pre_j) \wedge p \notin (del_i))$.
- A predicate p added by the first action which serves as a prerequisite for the second action a_j, cannot not be deleted by the first action a_i. That is:
  $\exists p(p \in (add_i \cap pre_j))$.
- A predicate p that is deleted by the first action a_i is added by a_j. In other words, an action re-establishes a predicate that is deleted by a previous action. That is:
  $\exists p(p \in (del_i \cap add_j))$
- The above plan constraints can be combined into one constraint and restated as: $\exists p((p \in (pre_i \cap pre_j) \wedge p \notin (del_i)) \vee (p(add_i \cap pre_j)) \vee (p \in (del_i \cap add_j))$

All pairs of elements in the candidate action set which satisfy one or more of the aforementioned constraints will be retained for the following learning step. For example, if *(move,pick)* are identified to be frequent action pairs and $((at(obj0{-}object, roo1{-}room), \neg(at{-}robby(rob0 - robot, roo1 - room)))) \in CAS_{move}$; and $((carry(rob0 - robot, obj0 - object, gr0{-}gripper), (at{-}robby(rob0{-}robot, roo0{-}room)), (\neg at(obj0{-}object, roo1{-}room)))) \in CAS_{pick}$ are two elements; then these two elements are retained in their respective action sets as they satisfy the third action pair constraint. Thanks to this heuristic, all actions which occur in frequent pairs will have pruned candidate action sets for the learning phase; thus speeding up learning by narrowing down the initial candidate set. This phase reduces the size of the candidate set M due to elimination. It is to be noted that all the actions which not figure in frequent sequential pairs will not undergo this pruning phase, and will directly pass to the following learning phase.

5.2.6 LSTM-based Classification

As previously mentioned, a plan is a chained series of actions orchestrating the accomplishment of a goal. Each action in this chain is influenced by previously executed actions, and will influence future actions further down the sequence. Thus, extracting patterns from sequences of previously executed actions is likely to provide strong evidence to predict the label of the next action in the chain. Through the medium of sequence labelling, we can narrow down and isolate the speculated ideal model among the reduced candidate set, as it would hypothetically be the one delivering the highest prediction rates amongst all models in the set. These dependency-driven and chained action executions inspire our investigation of long short-term memory networks (LSTMs) (Hochreiter and Schmidhuber, 1997) for action sequence labelling, with the intent of identifying the speculated ideal model. In the following subsections, we present our data encoding method for the input and output vector of the LSTM, and an overview of the training and validation phases of our framework.

LSTM Background

LSTM (Hochreiter and Schmidhuber, 1997) is a special type of gated recurrent neural network (RNN) which overcomes the limitation of traditional RNNs by demonstrating a capability of learning long-term dependencies. An LSTM network is formed exactly like a simple RNN, except that the nonlinear units in the hidden layer are replaced by recurrently connected subnets called *memory blocks*. These blocks can be thought of as a differentiable version of the memory chips in a digital computer, thus serving as a form of "explicit memory" (LeCun, Bengio, and Hinton, 2015). Each block contains one or more self-connected memory cells and three multiplicative units — the input, output and forget gates that provide

continuous analogues of write, read and reset operations respectively for the cells. The input and output gates scale the input and output of the cell, while the forget gate scales the internal state, controlling whether the previous state of the memory cell is remembered or forgotten (Min et al., 2016b). These multiplicative gates allow LSTM memory cells to store and access information over long periods of time.

In this implementation of LSTM, three principal decisions are made by three distinct layers. The decision as to what to throw away from a cell state is made by a sigmoid layer called the *forget gate* f_t. The second step is deciding what is to be stored in the cell state. This is divided into 2 phases: a first sigmoid layer called the 'input gate layer' decides what values will be updated. Next, a tanh layer creates a vector of new candidate values $\tilde{C}_t$ that could be added to the state. These two steps are summarized in the Equations 5.1- 5.3.

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f) \tag{5.1}$$

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i) \tag{5.2}$$

$$\tilde{C}_t = \tanh(W_c.[h_{t-1}, x_t] + b_c) \tag{5.3}$$

Now the old cell state C_{t-1} must be updated to the new cell state C_t. The old cell state is multiplied with f_t, forgetting what we decided to forget earlier. Then $i_t * \tilde{C}_t$ is added to obtain the new candidate values (demonstrated in equation 5.4):

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{5.4}$$

Finally the output must be decided. Firstly, a sigmoid layer decides what part of the cell state will propagate to the output. Then, the cell state is put through a tanh layer and multiplied with the output of the sigmoid gate, so as to output only the decided parts. This is summarized in Equations 5.5 and 5.6.

$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o) \tag{5.5}$$

$$h_t = o_t * \tanh(C_t) \tag{5.6}$$

Once the cell output h_t of the LSTM memory block is calculated, the next step is to use the computed cell output vectors to predict the next action in the plan sequence. This is done with the help of h_t, assuming that it is capable of capturing long-term dependencies from previous time steps.

Data Encoding for Labelling of Action Sequences

We use the sequence labelling capabilities of LSTM to identify the speculated ideal model from the reduced candidate set M. The sequence labelling in this case is in fact a classification of the most likely action that succeeds a given one. The input to our LSTM system is a large corpus comprising vector representations of each action of each trace. Each trace is taken one by one, and the comprising actions are sequentially encoded into input and output vectors; thus producing a large corpus of vectors. Each action of each trace is represented by two distinct vectors: an input vector which encodes the action, and an output vector which classifies the next action succeeding the action currently being encoded (we refer to this action as the current action for the sake of simplicity). These vectors serve as the input and output respectively to the LSTM cells, the encoding of which represents the heart of this section. This corpus of vector representations is divided into a training and validation set aimed at training the LSTM on the training set and gauging its performance on the validation set. At the output of this learning system, we obtain an accuracy of prediction on the folds of

validation data. The encoding of the input and output vectors is represented in the following paragraphs.

The input vector representing an action in a trace is encoded in the following fashion. It is divided into two sections: one section which labels the entire set of actions in the domain, and the other which labels the relevant predicates for the actions in the domain. In the first section, there is a slot for each action in the domain. The slot for the action currently being encoded is labeled as 1, and the slots for the remaining actions in the domains are labeled as 0. Thus if $(a_1, a_2, \ldots, a_n) \in A$ is the set of domain-applicable actions, the first *n* elements of the vector will be representing this first section, with the entry for the action currently being encoded being switched to 1, the other $(n-1)$ slots for the remaining $(n-1)$ actions being kept at 0. Once this first section has been assigned, we dedicate in the second section blocks of elements in the vector specific to each action in the domain. Thus for n actions in the domain, there are n different blocks (plus the one block for all the domain applicable actions as explained above). Each action-specific block contains one entry for each predicate relevant to that particular action. Revisiting the example in section 5.2.2 for the *gripper* domain, if *[at-robby (rob0 - robot, roo0 - room), at (obj0 - object, roo0 - room)]* are two predicates relevant to the action $pick$, they will constitute two entries in the $pick$ action block. We thus create action-specific blocks and for each action, assign entries for all the predicates relevant to that action, replicating this scheme for every action in the domain. Thus, the number of blocks for the input vector stands at $(n+1)$. The dimension d of this input vector is directly proportional to the number of actions in a domain, as well as the number of predicates relevant to each action. The dimension d of a vector for a specific domain will always remain the same, with the switching of a slot from 0 to 1 in the vector signalling the execution of a particular action. If $(a_1, a_2, \ldots, a_n) \epsilon A_s$ represents the action schema, the dimension of the input vector is given as:

$$d = n + \sum_{i=1}^{n} relPre_{a_i} \tag{5.7}$$

Here $relPre_{a_i}$ are the number of relevant predicates for the action a_i.
The output vector predicts the label of the action that follows the action currently being encoded in the trace. Very much like the input, the output is encoded as a binary vector. It consists of a single block which has as many slots as the number of actions in the domain, one for each action. The slot representing the succeeding action to the action being currently encoded is set to 1, the others being set to 0. Revisiting the example in section 5.2.2 for the *gripper* domain, the action currently being encoded is *pick* and the next action in the trace is *move*. While the number of actions in a trace, and thus the number of vectors representing all the actions in a trace may vary, the LSTM requires a fixed sized input. This is ensured by calculating the maximum trace length *batchLen* (maximum number of actions per trace for all the traces) for all the traces, and padding the shorter lists with *d*-dimensional vectors filled with zeros. This padding is done for all the traces till all the traces have the same *batchLen* number of actions. The same padding procedure is adopted for the output vectors.

Training and Validation Phase

The training and validation phases vary in the fashion that the slots of the input vector are labelled. This is because of the fact that the objective of the validation phase is to test the individual models to isolate the one with the highest labelling accuracy. In the training phase, two sections of the vector are filled out to represent the action currently being encoded. In the first section, the label of the action currently being encoded is set to 1, the rest being kept at 0. In the second section, slots of the predicates relevant to the currently encoded action

which occur in the current state (that is, are potential preconditions for the action about to be executed or about to be encoded) is set to 1. Also, all slots the newly introduced predicates as a result of the execution of the current action (i.e. the difference between the current state and the next state) which are relevant to the current action are set to 1, the other slots being kept at 0. The other blocks are kept switched off at 0 as well. This mechanism of encoding the training vector is elaborated as follows. Revisiting the example from Listing 5.1, if the action currently being encoded is *pick*, the state prior to the application of the action has the following predicates: *at, at-robby*. Both these predicates are relevant to the current action *pick*. These predicates serve as potential precursors to the application of the action *pick*, and are suspected to be the preconditions for the action. Thus, in the training vector, for the block specific to the action *pick*, the slots representing these two predicates are set to 1. The execution of the action introduces new predicates into the state represented after the action execution, namely the predicate *carry*. This predicate is relevant to the action *pick*, and can be suspected to constitute the effects of this executed action. Thus, in the case of the training vector, the slot representing the *carry* predicate of the *pick* action is set to 1. In conclusion, all the predicates slots relevant to the action being executed which are suspected to constitute the preconditions and effects of the model are switched to 1, while the blocks of the other actions in the training vector are kept at zero. This vector is similarly constructed with every successive action being encoded. This encoding scheme is illustrated in Figure 5.3.

In the validation phase, the objective is to test models sampled from the candidate set M in order to zero down to the speculated ideal model. Promising models are sampled from M by passing each model one by one though a planner and tested for their capability to solve a unitary problem. Models that fail to find a solution are discarded. Models that find a solution are retained in m'. The encoding for each of these models for validation with the LSTM is done in the following fashion. The first section is represented in the same way, with the label of the current action set to 1. The sections are, however, labelled differently. In this case, the slots in the vector which correspond to the relevant predicates present in the current action of the current model being evaluated are set to 1. For example, as illustrated in the Figure 5.4, if one of the candidate models are represented by ($move$: (at-robby, ¬(at-robby)), $pick$: (free, carry), $drop$: (carry, ¬(not-free))), then the slots in the vector for the $move$ action which represent the predicates $(at, \neg(at-robby))$ are switched to 1, the rest of the predicates being kept at 0. This validation is done for each of the models of the set m'.

5.3 Evaluation

The objective of the evaluation is to identify the speculated ideal model which can then be syntactically compared with the hand woven model for its identicalness. This section presents the evaluation results for the PDeepLearn system tested on four domains, namely: *satellite, mprime, gripper* and *depots*. The section is divided into three subsections which correspond to the three phases of the PDeepLearn algorithm.

5.3.1 All Candidate Model Generation

The cumulative total of all candidate actions for each domain are represented in the second column of Table 5.1. The size is proportional to the complexity of the domain i.e. the number of predicates relevant to each action. Although the complexity of the generation phase is proportional to the domain complexity, the generation time for each of the domains, as shown in column 3 of Table 5.2, is fairly negligible.

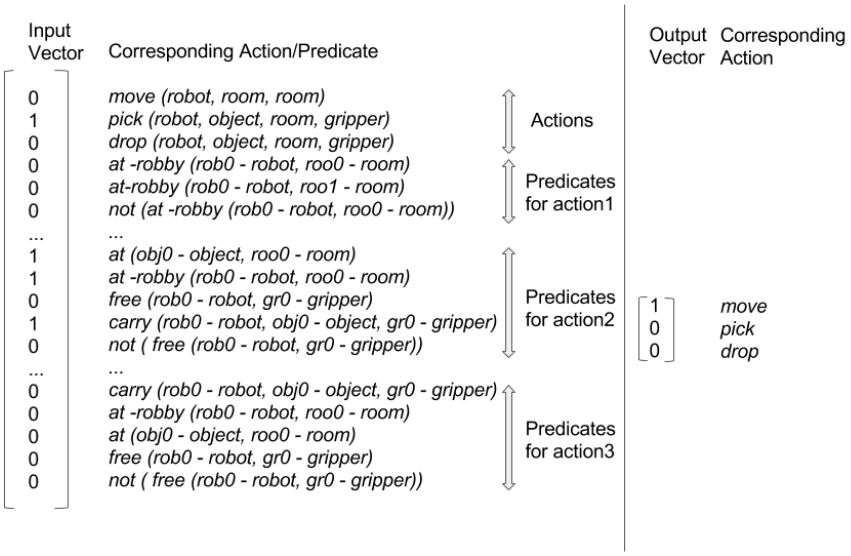

FIGURE 5.3: Input and output vectors to learning system for the training phase for currently encoded action "pick" and successive action "move". As illustrated in Listing 5.1, the predicates *at, at-robby* constitutes the state representation before the application of the action *pick*, and predicate *carry* constitutes the state after the action application. Thus, the slots representing the predicates *at, at-robby, carry* in the block designated to *pick* are set to 1, the other slots and blocks being labelled with 0.

5.3.2 Sequence Pattern Mining

The traces must be parsed and converted into a specific format which can serve as an input to the TRuleGrowth algorithm Fournier-Viger et al., 2012. The emerging rules with their confidence and support are recorded. The rules maintaining a consistently high value of confidence and support (empirically determined) with increasing number of traces for domains are recorded, since they represent a high-frequency sequential pair of actions, and are further pruned with *action pair constraints* detailed in Section 5.2.5.

5.3.3 Candidate Model Elimination

In accordance to the *action pair constraints* mentioned in Section 5.2.5, elements of the candidate action sets of the frequent actions which satisfy atleast one of the three constraints are retained, while the others are deleted. This elimination produces a reduced candidate set of models. The resulting number of candidate actions and the eventual reduction in the number of candidate actions of this phase are represented in the second and third column of the Table 5.1. The highest reduction can be seen in the *depots* domain, while the least reduction can be seen in the *mprime* domain. This can be explained by a stronger correlation among actions of the *depots* domain than among the actions of the *mprime* domain. The reduced candidate set is then passed on to the LSTM.

5.3.4 LSTM Based Speculated Ideal Model Identification

This subsection is used to isolate the speculated ideal model from the sampled candidate model set m' (sampling time recorded in Table 5.2). In this work, we explore two hyperparameters: the number of hidden units (set between (100, 200)), and the dropout rate (LeCun, Bengio, and Hinton, 2015) (set between (0.5, 0.75)); both of which have significant potential to influence the proposed LSTM-based labelling predictions. Other than these, we use a

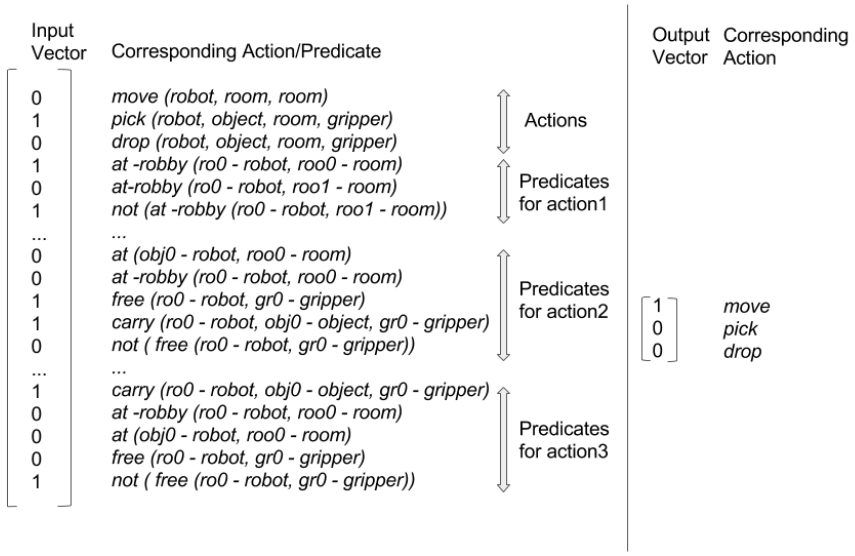

FIGURE 5.4: Input and output vectors to learning system for the validation phase with a candidate model for currently encoded action "pick" and successive action "move"

softmax layer for classifying given sequences of actions. The batch size is set to $batchLen$. We also use categorical cross entropy for the loss function and an adam optimizer (gradient descent optimizer). Each dataset for each of the 4 evaluated domains consists of 700 examples each, which is divided using five-fold cross validation. Thus, the number of data points available for cross validation is 700 multiplied by $batchLen$. For example, if $batchLen$ is 20, the number of data points are 700*20 = 14000. Every training example is presented to the network 10 times i.e. the network is trained for 10 *epochs*. The results are summarized in the Table 5.3 and are the obtained with the speculated ideal model, which produces the highest accuracy as compared to other models in the sampled set. The accuracy represented here is the *validation accuracy* (accuracy measured on the validation set), which is the proportion of examples for which the LSTM performs the correct classification. It is represented as the fraction of examples classified correctly. The speculated ideal model demonstrates the highest accuracy amongst all the models of the sampled candidate set which are tested. We compare in Table 5.2 the performance of the *PDeepLearn* system with the ARMS system, both in terms of the running time of the algorithm, and the syntactic similarity of the models learnt respectively by the systems with the hand woven model used to generate the traces. The results demonstrate that the execution time of PDeepLearn is close to that of ARMS for 700 traces.

The difference between the hand woven model and the empirically determined model is represented in the form of a *reconstruction error*. This error is based on the similarity among the predicates between the empirical model and hand woven (ideal) truth model. Let $diffpre_{a_i}$ represent the syntactic difference in *pre* lists of action a_i in the hand woven model and the empirical model. Each time the *pre* list of the ideal model presents a predicate which is not in the *pre* list of the empirical model, the count $diffpre_{a_i}$ is incremented by one. Similarly, each time the *pre* list of the empirical model presents a predicate which is not in the *pre* list of the ideal model, the count $diffpre_{a_i}$ is incremented by one. Similar counts are estimated for the *add* and *del* lists as $diffadd_{a_i}$ and $diffdel_{a_i}$ respectively. This total count is then divided by the number of relevant constraints for that particular action $relCons_{a_i}$ to obtain the cumulative error per action. This error is summed up for every action and averaged over the number of actions of the model to obtain an average error E for the entire model. The reconstruction error for the model is thus represented by:

TABLE 5.1: Model Pruning Results for PDeepLearn

Domain	Initial Number of Candidate Actions	Final Number of Candidate Actions post Pruning	Percentage Reduction (%)
Satellite	1633	155	90.6
Mprime	2834	466	83.55
Gripper	292	6	97.94
Depots	2913364	10240	99.65

TABLE 5.2: Comparison of running time for ARMS and PDeepLearn in seconds (PDL = PDeepLearn).

Domain	Running time for ARMS	Generation time for PDL	Sampling time for PDL	Running time for PDL
Satellite	29781.24	102.61	3.10	47857.19
Mprime	1614.09	16.52	618.80	2167.07
Gripper	36873.72	110.03	0.08	17253.49
Depots	110415.61	153.88	4080.17	98878.02

$$E = \frac{1}{n}\sum_{i=1}^{n}\frac{diffPre_{a_i} + diffAdd_{a_i} + diffDel_{a_i}}{relCons_{a_i}} \tag{5.8}$$

The reconstruction errors are summarized in the Table 5.3. The empirically obtained model is identical to the hand woven one, as exhaustive generation then filtration ensures that this identical model is always part of the set and eventually narrowed down upon by the LSTM. The error E of the model produced by ARMS fluctuates between 15-30 percent.

5.4 Conclusion

In this chapter, we have presented an approach called *PDeepLearn*, which uses state-action interleaved traces to learn an action model (termed as speculated ideal model) syntactically identical to the hand woven model used to generate the traces (training data). First, *PDeepLearn* generates a set of all possible action models from traces. This exhaustive generation ensures that we always generate the speculated ideal model in the set. Then, with the help of intra-action semantic constraints and inter-action constraints on sequential action pairs, we significantly prune improbable models to produce a reduced model set. Finally, we use the sequence labelling capabilities of LSTMs to iteratively test the capability of each

TABLE 5.3: Action Labelling Results for PDeepLearn (700 traces, 128 hidden units, 0.8 dropout rate, 0.001 learning rate) vis-a-vis the reconstruction error for ARMS

Domain	Accuracy Rate (%)	ARMS Reconstruction Error
Satellite	0.8571	28.88
Mprime	0.75	16.66
Gripper	1.00	22.22
Depots	0.72	24.07

of the models of a sampled set of this reduced set to predict the next action based on the previously observed ones. The highest prediction accuracy is obtained with the speculated ideal model. While it is the first approach in our knowledge that makes use of deep learning techniques to learn domain models, it has limitations worth highlighting. Since it is a deep learning technique, it fundamentally thrives on more traces to learn better. While it is also more accurate than *SPMSAT* and successfully reconstructs the speculated ideal model, its execution time owing to the complexity of the process is subsequently higher. Future proposed work would include integrating the possibility of *PDeepLearn* learning action models from noisy traces, in an effort to bring it closer to a real world scenario.

Chapter 6

Towards application of learning techniques to HRI scenario

In the two previous chapters, we have introduced machine learning techniques which provide optimal and non-optimal means of learning action models. The optimality in this definition refers to the sheer comprehensiveness of the learnt model. The reader may recall that the broad objective of this study is to develop learning techniques which are conceptualized and validated with planning benchmark domains to then on apply to an HRI scenario and learn the underlying behavioral model of the robot, which is a key to its autonomy during future interactions. This objective in itself spells a stark difference between the state-of-the-art approaches outlined in Chapter 2 and the objective of this study, as not too many of them have been applied to a real life scenario. This chapter proposes to amalgamate the previously introduced techniques with the intent of learning robot behavioral models from HRI dialog sequences.

To refresh the nature of our problem in the mind of the user: given a set of human-robot dialog exchange sequences, we aim to learn the robot's behavioral model comprising of the speech acts encoded in the form of operators alongwith their signatures, preconditions and effects. The body of HRI dialog exchanges is drawn from the Vernissage corpus (Jayagopi et al., 2012) and consists of exchanges between a Nao robot (operated by a skilled operator in a Wizard-of-Oz setup) posing as a museum guide and two human visitors to the museum. An example of the dialog exchange can be seen in the Listing 6.1.

LISTING 6.1: Example of dialog exchange between NAO as a museum guide and museum visitors

```
Nao : My name is Nao, what is yours?
Human : My name is Alan
Nao : It's a pleasure to meet you Alan, can I
tell you something about these paintings?
Human : Yes please
```

In the AP context, each utterance is considered ideologically equivalent to an action (represented in the corpus in the form of an action name and signature). Thus, each dialog sequence is represented as a sequential and orchestrated action name sequence which effectuates transitions in the world state, gradually propelling it towards a predetermined goal. By means of these utterances, the speaker *plans* to influence the listeners' beliefs, goals and emotional states (Breazeal, 2003; Breazeal, 2004); with the intent of fulfilling their own goals. These utterances, interleaved with head and body movements (i.e. multi modal acts) are then modeled as sequences of actions in an AP system, and frameworks can be developed for providing their semantics (Perrault, 1990; Sadek, 1991). However, for the sake of simplicity, in this work we ignore the definite uncertainty in communication, as well as the head and body movements comprising the acts, and concentrate solely on utterances. The

objective is then to leverage the advancements in Machine Learning (ML) to learn the underlying action model from the data i.e. learning the preconditions and effects of each of the constituent actions from their signatures. Learning the underlying model comprising action descriptions from action name-signature sequences could save the effort from having to code these action descriptions from scratch, thus promoting re usability. This model can further be fed to an automated planner to generate fresh dialogs, thus allowing the robot to communicate autonomously in future scenarios. We use the MAX-SAT framework for learning the behavioral model. The constraints used as the input for this framework are constituted by intra-operator and inter-operator specific constraints. These constraints are derived from the intra-operator semantic relationships and inter-operator temporal relationships and dependencies established during concurrent and near-concurrent execution of actions during the exchanges. We evaluate the accuracy of our learning framework by comparing the syntactic difference between the empirically obtained and ground truth behavioral model. This ground truth model has been hand crafted, with actions and predicates are derived from the encoding of speech acts in (Riviere et al., 2011). This hand woven model serves as a benchmark for the comparison of the model learnt with the help of the learning techniques proposed in this study.

Once the behavioral model of the robot is learnt, it can be fed again to a planner to plan fresh interaction sequences . Thus, the learnt model can be re-usable to plan future dialogue sequences between the robot and the human, in such a way that the need of a "teacher" to govern the robot behavior is suppressed, and the robot can interact autonomously. Feeding the learnt model to a planner to generate fresh interactions and testing them in a real scenario is in the scope of future work. We also do not use the LSTM to learn action models, the reasons for which are explained further down in this chapter. The framework is, however, used to fulfill other objectives. The learnt model is used to label speech acts occurring in the HRI exchanges using the sequence labeling capabilities of the LSTM (Long Short Term Memory) recurrent networks. For example, in the traces we observe that each time the robot encounters a human subject for the first time, it has a tendency to greet the individual followed by an act of self-introduction. We use an LSTM trained on the features of the behavioral model as a multi-class classifier, the classes in this case being the speech acts exchanged during the course of the exchanges. With reference to our example, the accuracy of the LSTM to predict the occurrence of the introduction act following the greet act is calculated. This is performed with the aid of sequence labelling capabilities of LSTMs similar to the implementation used in Chapter 5.

The listing 6.2 represents our hand crafted domain model which we call *sombrero*. It is crafted with the intent of orchestrating the interaction between a robot and a human in a room and geared towards a common objective. The *types* mentioned in the file and those specific to the interaction are: *robot, human, room* and *proposition*. The *room* is the enclosure in which the interaction occurs, and the proposition is the condition whose fulfillment is required at each exchange. The predicates are derived from a simplification logic called MLC (Modal Logic of Communication) (Guiraud et al., 2011) which is an extension of the BDI logic (Belief, Desire, Intention) (Cohen and Levesque, 1990; Rao and Georgeff, 1991). Both the logics are based on the ideology that beliefs and desires cause actions. MLC extends the BDI logic with a the notion of responsibility and complex emotions (stemming from the realization of responsibility). This logic is materialized in the form of a speech acts intended to be the communication protocol between agents. This terminology which serves as the communication protocol are termed as *Expressive Multimodal Conversational Acts* (EMCA). For example, the first predicate *bel (?r, ?p)* represents the notion that agent $?r$ believes in the proposition $?p$. The description of the predicates and their corresponding implications with reference to the MLC logic are detailed in the Table 6.1. In the listing, the representation of the predicates in the PDDL domain is followed by the *definition* (name-signature representations) and

description (representations of preconditions and effects) of each action. These actions are representations of the speech acts used during the dialog exchanges between a human and the robot. Following are the speech acts encoded in the domain: *(sayHello, farewell, inform, say, ask, claim, deny, advise, thank, autoFeedback (feedback on last heard dialogue e.g. "oh", "great" etc.))*. For example, the *sayHello* speech act is employed when the robot encounters the human subject for the first time, and proceeds to greet the subject. The preconditions and effects of this act represent this notion in logical form. These actions and predicates are derived from the encoding of speech acts in (Riviere et al., 2011). The speech acts in this work are represented in MLC logic, which we translate into PDDL operators (owing to the logical equivalence of the speech acts in (Riviere et al., 2011) to PDDL operators) for our study and represent in the listing below. The reader is invited to read the paper by (Riviere et al., 2011) for more detailed information on the encoding scheme.

LISTING 6.2: Domain Description and Schema for an HRI domain labelled 'sombrero'

```
(define (domain sombrero)
(:requirements :strips :typing)
(:types
   agent - object
   robot human - agent
   proposition - object
   room - object
)
(:predicates
   (bel ?i - agent ?p - proposition)
   (goal ?i - agent ?p - proposition)
   (ideal ?i - agent ?p - proposition)
   (approval ?i - agent ?p - proposition)
   (belbelapproval ?i - agent ?j - agent ?p - proposition)
   (belideal ?i - agent ?j - agent ?p - proposition)
   (notgoal ?i - agent ?p - proposition)
   (notbel ?i - agent ?p - proposition)
   (belbel ?i - agent ?j - agent ?p - proposition)
   (belgoal ?i - agent ?j - agent ?p - proposition)
   (notbelbel ?i - agent ?j - agent ?p - proposition)
   (goalbel ?i - agent ?j - agent ?p - proposition)
   (goalresp ?i - agent ?j - agent ?p - proposition)
   (notgoalresp ?i - agent ?j - agent ?p - proposition)
   (belbelgoalresp ?i - agent ?j - agent ?p - proposition)
   (belbelnotgoalresp ?i - agent ?j - agent ?p - proposition)
   (goalbelgoalresp ?i - agent ?j - agent ?p - proposition)
   (goalbelstr ?i - agent ?j - agent ?p - proposition)
   (goalbelgoalrespWeak ?i - agent ?j - agent ?p - proposition)
   (belbelgratitude ?i - agent ?j - agent ?p - proposition)
   (belresp ?i - agent ?j - agent ?p - proposition)
   (belbelmoralsatisfaction ?i - agent ?j - agent ?p - proposition)
   (look ?a - agent ?b - agent)
   (at ?a - agent ?l - room)
   (already-seen ?i - agent ?j - agent)
   (never-seen ?i - agent ?j - agent)
)
```

```
;; One agent greets another agent
(:action sayHello
  :parameters (?i - agent ?j - agent ?r - room)
  :precondition (and (at ?i ?r) (at ?j ?r) (look ?i ?j)
   (never-seen ?i ?j))
  :effect (and (already-seen ?i ?j) (not (never-seen ?i ?j)))
)

;; One agent bids farewell to another agent
(:action farewell
  :parameters (?i - agent ?j - agent ?r - room)
  :precondition (and (at ?i ?r) (at ?j ?r) (look ?i ?j)
   (already-seen ?i ?j))
  :effect ((not(look ?i ?j))
)

;;One agent informs another agent
(:action inform
        :parameters (?i - agent ?j - agent  ?p - proposition)
        :precondition (and (bel ?i ?p) (goalbel ?i ?j ?p)
         (notbelbel ?i ?j ?p) (already-seen ?i ?j))
        :effect (belbel ?i ?j ?p)
)

;;One agent says something to another agent
(:action say
        :parameters (?i - agent ?j - agent  ?p - proposition)
        :precondition (and (bel ?i ?p) (goalbel ?i ?j ?p)
         (already-seen ?i ?j))
        :effect (belbel ?i ?j ?p)
)

;;One agent asks/tells/suggests to another agent
(:action ask
        :parameters (?i - agent ?j - agent  ?p - proposition)
        :precondition (and (goalresp ?i ?j ?p)
        (goalbelgoalresp ?i ?j ?p) (already-seen ?i ?j))
        :effect (and (belbelgoalresp ?i ?j ?p))
)

;;One agent claims something to another agent
(:action claim
        :parameters (?i - agent ?j - agent  ?p - proposition)
        :precondition (and (bel ?i ?p) (goalbelstr ?i ?j ?p)
         (already-seen ?i ?j))
        :effect (belbel ?i ?j ?p)
)

;;One agent denies something to another agent
```

```
(:action deny
        :parameters (?i - agent ?j - agent  ?p - proposition)
        :precondition (and (bel ?i ?p) (goalbel ?i ?j ?p)
         (already-seen ?i ?j))
        :effect (belbel ?i ?j ?p)
)

;;One agent advises something to another agent
(:action advise
        :parameters (?i - agent ?j - agent  ?p - proposition)
        :precondition (and (goalresp ?i ?j ?p)
        (goalbelgoalrespWeak ?i ?j ?p) (already-seen ?i ?j))
        :effect (belbelgoalresp ?i ?j ?p)
)

;;One agent thanks another agent
(:action thank
        :parameters (?i - agent ?j - agent  ?p - proposition)
        :precondition (and (goal ?i ?p) (belresp ?i ?j ?p)
         (already-seen ?i ?j))
        :effect (belbelgratitude ?i ?j ?p)
)

;;One agent thanks another agent
(:action autoFeedback
        :parameters (?i - agent ?j - agent  ?p - proposition)
        :precondition (and (goal ?i ?p) (belresp ?i ?j ?p)
         (already-seen ?i ?j))
        :effect (belbelmoralsatisfaction ?i ?j ?p)
)
)
```

This chapter unfolds as follows: we detail the functioning of our learning system in Section 6.1, and present our empirical evaluations in Section 6.2. We conclude this chapter of our work in the Section 6.3.

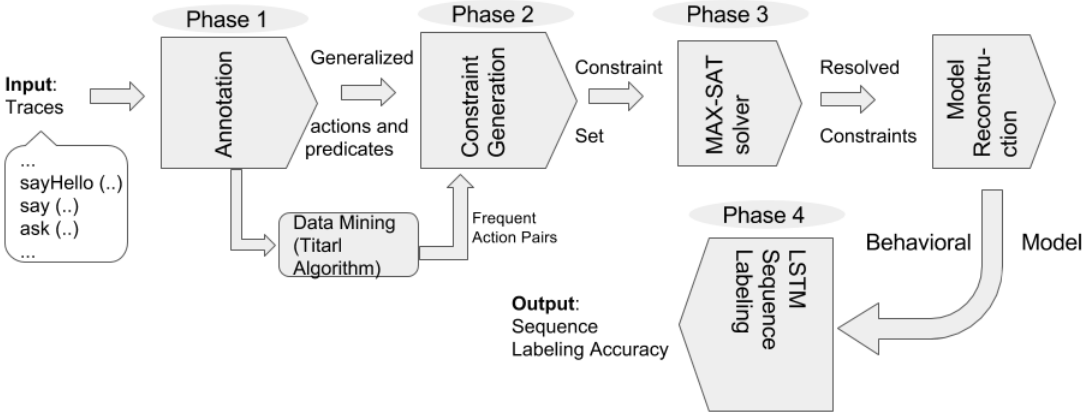

FIGURE 6.1: Our Approach Phases

TABLE 6.1: Predicates in the domain *sombrero* and their implications in speech acts

Predicate	Interpretation
bel (?i ?p)	agent ?i believes in the proposition ?p
goal (?i ?p)	proposition ?p is the goal of agent ?i
ideal (?i ?p)	proposition ?p should ideally be true for agent ?i
belbelapproval (?i ?j ?p)	agent ?i believes that agent ?j believes that ?i gives its approval for proposition ?p
belideal (?i ?j ?p)	agent ?i believes that proposition ?p should be the ideal for agent ?j
goal (?i ?p)	proposition ?p is not the goal of agent ?i
notbel (?i ?p)	agent ?i does not believe in the proposition ?p
belbel (?i ?j ?p)	agent ?i believes that agent ?j believes in the proposition ?p
belgoal (?i ?j ?p)	agent ?i believes that the goal of agent ?j is proposition ?p
notbelbel (?i ?j ?p)	agent ?i believes that agent ?j does not believe in the proposition ?p
goalbel (?i ?j ?p)	the goal of agent ?i is that agent ?j believes in proposition ?p
goalresp (?i ?j ?p)	the goal of agent ?i is that agent ?j assumes responsibility of proposition ?p
notgoalresp (?i ?j ?p)	the goal of agent ?i is not that the agent ?j assumes responsibility of proposition ?p
belbelgoalresp (?i ?j ?p)	agent ?i believes that agent ?j believes that the goal of ?i is ?j assumes the responsibility of proposition ?p
belbelnotgoalresp (?i ?j ?p)	agent ?i believes that agent ?j believes that the goal of ?i is not that ?j assumes the responsibility of proposition ?p
goalbelgoalresp (?i ?j ?p)	goal of agent ?i is that agent ?j believes that the goal of ?i is that ?j assumes the responsibility of proposition ?p
goalbelgoalrespWeak (?i ?j ?p)	goal of agent ?i is that agent ?j believes that the goal of ?i is that ?j assumes the responsibility of proposition ?p to a lesser magnitude
goalbelstr (?i ?j ?p)	goal of agent ?i is that agent ?j believes strongly in proposition ?p
belbelgratitude (?i ?j ?p)	goal of agent ?i is to express to agent ?j its gratitude for proposition ?p
belresp (?i ?j ?p)	goal of agent ?i is that agent ?j assumes responsibility for proposition ?p
belbelmoralsatisfaction (?i ?j ?p)	agent ?i believes that agent ?j believes that agent ?i assumes moral satisfaction with proposition ?p
look (?i ?j)	agent ?i looks at agent ?j
at (?i ?r)	agent ?i is at room ?r
already-seen (?i ?j)	agent ?i has already seen agent ?j
never-seen (?i ?j)	agent ?i has never seen agent ?j

6.1 Approach

Our problem remains the one mentioned in the two preceding chapters, however in an HRI paradigm. It can be formulated as follows: given (i) a set of HRI traces T, each trace consisting of a dialogue sequence with individual dialogs encoded in the form of signatures of the speech acts represented in the Listing 6.2; our approach produces a complete domain model m encompassing all the domain-applicable operators which best explain the observed traces. The model comprises of the applicable speech acts which are encoded in the form of operators alongwith their signatures, preconditions and effects. This is done by encoding the inter-operator and intra-operator dependencies in the set T in the form of constraints and solving them as a satisfiability problem, followed by reconstructing the domain model from the satisfied constraints. It then proceeds to use the refined form of the learnt model to predict the label of the acts in the traces using LSTM techniques. Details of this approach are explained in the following paragraphs.

The approach can be divided into four phases, a snapshot of which is illustrated in Figure 6.1. The dialogs of the HRI traces are first annotated in Phase 1 by replacing each dialog with the most appropriate speech act definitions mentioned in Listing 6.2. This is done with the intent of getting closer to the standard action sequence representation constituting a plan. We then generalize the grounded actions in the trace set by replacing the variables by their types to obtain a trace set of operators. Phases 2 and 3 constituting the model learning are identical to those represented in the *SPMSAT* learning system. The second phase is dedicated to constraint generation; namely intra-operator and inter-operator ones. In the third phase, these constraints are supplied to a MAX-SAT solver and the satisfied ones are used to reconstruct the action model. This model consists of operator signatures, preconditions (alias for *pre* list), and effects (alias for *add* and *del* lists). It conforms to the semantics of STRIPS (Fikes and Nilsson, 1971). In the fourth phase, using the operators learnt in the previous phase, we classify the labels of the operators likely to be chosen by the robot during the dialog exchange highlighted in the corpus. This classification is done by encoding the operator sequences of the corpus in the form of input and output vectors to an LSTM. While the learnt model can serve many needs, in this work we demonstrate the utility of our learnt model to classify speech acts. We elaborate these phases in the forthcoming subsections.

6.1.1 Annotation and Generalization

In this phase, dialogues in the dialogue corpus are first annotated to actions by replacing each dialog with the most appropriate speech act definitions mentioned in Listing 6.2, then generalized to operators. These dialogues are taken from the Vernissage corpus which is a multimodal HRI dataset (Jayagopi et al., 2012), and has a total of 10 conversation instances between Nao and human participants. The dialogues and gestures of Nao are controlled in a "Wizard-of-Oz" fashion. It has been recorded and annotated to benchmark many relevant perceptual tasks such as: speaker localization, key word spotting, speech recognition, tracking, pose estimation, nodding, visual focus of attention estimation, and addressee detection. In the scenario, the robot explains paintings in a room and then performs a quiz with the participants, allowing an analysis of the quality of the interaction and the behavior of the human interaction partners. We annotate each robot dialogue within each trace as an action drawn from the speech acts mentioned in Listing 6.2: *(sayHello, farewell, inform, say, ask, claim, deny, advise, thank, autoFeedback (feedback on last heard dialogue e.g. "oh", "great" etc.))*. The participant responses are encoded in the form of *(speech, silence, laughter)*. An example of the annotation process can be seen in Table 6.3. In this example, the dialog sequence in the second column (represented by the header "Utterance") is uttered at time mentioned in the column "Timestamp", measured in seconds from the beginning of the interaction. The

TABLE 6.2: Sample annotation of traces. The "Utterance" in the second column at the timestamp from the beginning of the interaction mentioned in the column "Timestamp" is annotated with the speech act in the column "Annotation". The speech act chosen for the annotation (from the Listing 6.2) is one that is deemed most suitable to characterize the utterance. The (..) represents the operator parameters.

Timestamp (sec)	**Utterance**	**Annotation**
35.803	Hello	sayHello (..)
35.803	Should I tell you something about these paintings?	ask (..)
395.619	So, I am starting the quiz!	claim (..)
556.885	Great, your answer is perfectly right.	autoFeedback (..)
629.532	So. It's the end of this quiz.	say (..)

annotation associated with a certain utterance can be seen in the third column labeled "Annotation". As can be seen from the rightmost column, the annotation to a each dialog consists of the name of the act followed by deictic references (a deictic reference is a pointer to objects which have a particular role in the world, with object roles coded relative to the agent or current action (Agre and Chapman, 1987)) in parenthesis (abstracted from the reader view in this table).

The next step is generalization, which is done by scanning each action in each of the traces and substituting the deictic references with their corresponding types. This produces a trace set of operator sequences, with the generalized actions constituting an operator schema O_s. A snapshot of a trace produced from one of the dialog sequences of the Vernissage corpus is illustrated in Listing 6.3. It can be seen that the interaction kicks off with the "sayHello" act, followed by acts of "ask", "say" used by Nao. The responses by the human subjects are represented in the form of the "speech" and "laughter" acts. The dialog terminates with the "thank" act wherein the robot thanks the subjects for their participation, and terminates on a note of farewell depicted by the "farewell" act. We then create a dictionary of all possible relevant predicates to each operator, the keys of the dictionary identified by the operator names. Each operator in the operator schema is associated with its relevant predicates, where a predicate p is said to be relevant to an operator $o \in O_s$ if they share the same variable types. We denote the relevant predicate dictionary as $relPre$, with the set of relevant predicates to an operator o_i (represented as key) can be denoted as $relPre_{o_i}$ (represented as value of key o_i). The generalization procedure is represented in Figure 6.2.

```
sayHello (..) say (..) offer (..) speech (..) laughter (..)
ask (..) say (..) inform (..) thank (..) farewell (..)
```

LISTING 6.3: Annotated trace produced for one of the conversation sequences featuring in the Vernissage corpus

6.1.2 Constraint Generation

In this phase, we detail the intra-operator hard constraints for individual operators and inter-operator soft temporal constraints among operators.

Hard constraints

In order to satisfy the semantics of STRIPS (Fikes and Nilsson, 1971), each operator in O_s must satisfy certain *intra-operator constraints*. Thus, for each operator $o_i \in O_s$ and relevant predicate $p \in relPre_{o_i}$:

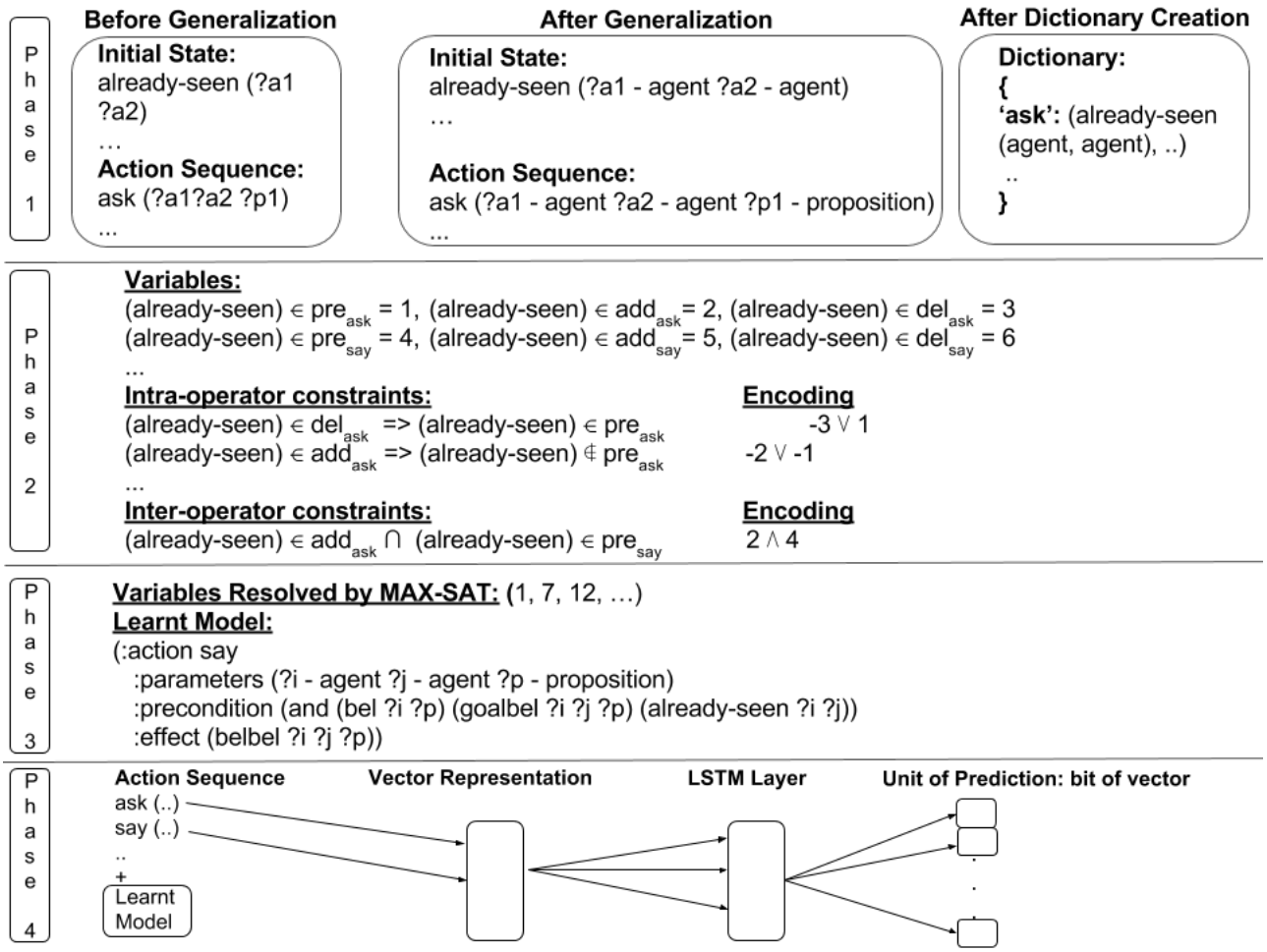

FIGURE 6.2: Example of our approach

- p cannot be in the *add* list and the *del* list at the same time, and
- p cannot be in the *add* list and the *pre* list at the same time.

The relevant predicates of each operator are encoded to generate variables. Each association of a relevant predicate with one of a *(pre, add, del)* list can be encoded as a variable. These variables and constraints are illustrated in Figure 6.2. These constraints are also illustrated in the listing 6.4.

Relevant predicates for action *inform*:
`(at, look, belbel, know-if)`
Hard constraints for action *query-if*:
$bel \in del_{inform} \Rightarrow bel \in pre_{inform}$
$bel \in add_{inform} \Rightarrow bel \notin pre_{inform}$
$bel \in del_{inform} \Rightarrow bel \in pre_{inform}$
$bel \in add_{inform} \Rightarrow bel \notin pre_{inform}$

LISTING 6.4: Intra-operator constraints for action *inform*

Soft Constraints

Next we generate a set of inter-operator constraints by exploring the inter-dependencies between the operators which constitute the traces. Any sequence of operators exhibiting implicit patterns of the form of frequently co-occurring operators indicates the possible presence of inter-operator dependencies and relationships. Any operator sequence exhibits inter-operator dependencies which can be uncovered by means of data mining techniques to facilitate the process of learning. In the case of an HRI scenario, operators may be temporally concurrent: for example, two agents may be speaking at the same time. These temporal dependencies can be explored with the help of pattern mining techniques. The aforementioned trace set *T* can thus be treated as a set S of transactions in the pattern mining domain. S can be written as $S = [s_1, s_2, \ldots, s_n]$ where each transaction s_n represents a trace. A transaction is a set of symbols, in our case representing operators. A *time series* is a set of unique time points. A *symbolic time series* is generally used to represent series of ordered (but non time-sampled) events (e.g. $< a, b, d, a, c >$). A *symbolic time sequence* is a multi-set of time points (i.e. elements may repeat). Contrary to time series, *symbolic time sequences* can deal with several events defined at the same temporal location. This difference is also illustrated in the Figure 6.3. Thus, relations between the events of a pattern is not limited to be "before" or "unconstrained" i.e. the events may be represented without order. In this problem, the challenge is not only to select the events of the patterns, but also to extract the relations (or constraints) between them (Guillame-Bert and Crowley, 2012). Since we deal with HRI interactions composed of an interplay between utterances in a way that they may overlap, we symbolize these interactions in the form of operators chaining together in a symbolic time sequence (Guillame-Bert and Crowley, 2012). We symbolize the concurrent utterances in the Vernissage corpus in the form of operators chaining together in a symbolic time sequence (Guillame-Bert and Crowley, 2012). This symbolic time sequence is the most appropriate approximation of the dialog corpus in the pattern mining domain with the intent of extracting frequent association rules between the operators.

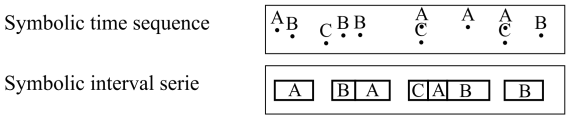

FIGURE 6.3: Difference between symbolic time series and symbolic time sequences. While items in a symbolic time series are partially ordered and do not repeat, items in a symbolic time sequence may be co-occurring and repeat as well. (Guillame-Bert and Crowley, 2012).

Various approaches to mine temporal time sequences are present in the literature. Winepi (Mannila, Toivonen, and Inkeri Verkamo, 1997) is a well known algorithm which learns episodes and association rules based on the episodes. The *face* algorithm allows for mining of chronicles from symbolic time sequences (Dousson and Duong, 1999). These approaches are, however, not equipped to deal with temporal inaccuracies in the temporal events. We choose the Temporal Interval Tree Association Rule Learning (Titarl) algorithm (Guillame-Bert and Crowley, 2012) because it allows the representation of imprecise (non-deterministic) and inaccurate temporal information between speech acts considered as symbolic events. Following is an example of a rule mined with Titarl: "If there is an event A at time t, then an event of type B will occur between times t+5 and t+10 with a 95% chance". The temporal relationships between operators can be uncovered by means of *association rule learning*, which is the search for association rules. An association rule is a $conditions \rightarrow implications$ pattern. We hypothesize that if an association rule frequently correlates two operators, there must be a reason for their frequent co-existence. We are thus interested in the branch of

TABLE 6.3: Sample annotation of traces. The "Utterance" in the second column at the timestamp from the beginning of the interaction mentioned in the column "Timestamp" is annotated with the speech act in the column "Annotation". The (..) represents the operator parameters.

Timestamp (sec)	Utterance	Annotation
35.803	Hello	sayHello (..)
395.619	So, I am starting the quiz!	claim (..)
556.885	Great, your answer is perfectly right.	autoFeedback (..)
629.532	So. It's the end of this quiz.	say (..)

pattern mining algorithms which treat frequent sequential action pairs. Other algorithms in the pattern mining literature, like the Apriori algorithm (Agrawal and Srikant, 1994)(used in ARMS (Yang, Wu, and Jiang, 2007)) are not equipped to satisfy this requirement. The input traces are parsed and fed to the Titarl algorithm, which produces temporal association rules (Guillame-Bert and Crowley, 2012). The operators featuring in the frequent temporal rules are suspected to share a "semantic" relationship among themselves, which can be represented in the form of *inter-operator constraints*. These constraints are identical to the constraints mentioned in the Section 5.2.5. The reader is referred to Section 5.2.5 for more detail.

A snippet of the aforementioned constraints is illustrated for the action pair *(ask, say)* and relevant predicate *already-seen* in the Figure 6.2 and summarized in the listing 6.5.

Frequent action pair for the domain: `(ask, say)`
Common relevant predicate for actions
ask **and** *say*: `already-seen`
$\exists(already-seen) : (((already-seen) \in (pre_{ask} \cap pre_{say}))) \vee (already-seen) \notin (del_{ask}) \vee ((already-seen) \in (add_{ask} \cap pre_{say})) \vee ((already-seen) \in (del_{ask} \cap add_{say}))$

LISTING 6.5: Inter operator constraints for operators *ask* and *say*

The details pertaining to the resolution of the intra and inter-operator constraints with the help of a SAT solver to reconstruct the underlying action model is detailed in the Section 6.2.

Data Encoding for Labelling of Operator Sequences

As previously mentioned, a plan is a chained series of interdependent operators directed towards the accomplishment of a goal. Thus, extracting patterns from sequences of previously executed operators is likely to provide strong evidence to predict the label of the next operator in the chain; inspiring our investigation of long short-term memory networks (LSTMs) (Hochreiter and Schmidhuber, 1997) for action sequence labelling. In the following subsections we present our data encoding method for the input and output vector of the LSTM.

We use the sequence labelling capabilities of LSTM to identify the most likely operator that succeeds a given one. The input to our LSTM system is a large corpus comprising vector representations of each operator of each trace. Each trace is taken one by one, and the comprising operators are sequentially encoded into input and output vectors; thus producing a large corpus of vectors. Each operator of each trace is represented by two distinct vectors: an input vector which encodes the operator, and an output vector which classifies the successive operator. These vectors serve as the input and output respectively to the LSTM cells, the encoding of which represents the core of this section. This corpus of vector representations is divided into a training and validation set to eventually train the LSTM on the training set and gauge its performance on the validation set. At the output of this learning system, we obtain an accuracy of prediction on the folds of validation data. The encoding of the input and output vectors is presented in the following paragraphs.

The input vector representing an operator in a trace is encoded in the following fashion. It is divided into two sections: one section which labels the entire set of operators in the domain, and the other which labels the relevant predicates for the operators in the domain. In the first block, there is a slot for each operator in the domain. The slot for the operator currently being encoded is labeled as 1, and the slots for the remaining operators in the domain are labeled as 0. Thus if $(o_1, o_2, \ldots, o_n) \in O_s$ is the set of domain-applicable operators, the first *n* elements of the vector will be representing this first block, with the entry for the operator currently being encoded being switched to 1, the other $(n-1)$ slots for the remaining $(n-1)$ operators being kept at 0. Once this first block has been assigned, we dedicate blocks of elements in the vector specific to each operator in the domain. Thus for n operators in the domain, there are n different blocks (plus the one block for all the domain applicable operators as explained above). Each operator-specific block contains one entry for each predicate relevant to that particular operator. For example, if *[goalresp (Ag0 - Agent, Pr0 - Proposition), already-seen (Ag0 - Agent, Ag1 - Agent)]* are two predicates relevant to the operator ask, they will constitute two entries in the ask operator block. We thus create operator-specific blocks and for each operator, the number of blocks for the input vector standing at $(n+1)$. The dimension d of this input vector is directly proportional to the number of operators in a domain, as well as the number of predicates relevant to each operator. The dimension d of a vector for a specific domain will always remain the same, with the switching of a slot from 0 to 1 in the vector signalling the execution of a particular operator. If $(o_1, o_2, \ldots, o_n) \epsilon O_s$ represents the operator schema, the dimension of the input vector is given as:

$$d = n + \sum_{i=1}^{n} relPre_{o_i} \tag{6.1}$$

Here $relPre_{o_i}$ are the number of relevant predicates for the operator o_i. The output vector predicts the label of the operator that follows the operator currently being encoded in the trace. Very much like the input, the output is encoded as a binary vector. It consists of a single block which has as many slots as the number of operators in the domain, one for each operator. The slot representing the succeeding operator to the operator being currently encoded is set to 1, the others being set to 0. For example, let us assume that the operator currently being encoded is *ask* and the next operator in the trace is *say*. The input and output vectors for the *ask* operator are represented in Figure 6.4. While the number of operators in a trace thus the number of vectors representing all the operators in a trace may vary, the LSTM requires a fixed sized input. This is ensured by calculating the maximum trace length *batchLen* (maximum number of operators per trace) for all the traces, and padding the shorter lists with *d*-dimensional vectors filled with zeros. This padding is done for all the traces till all the traces have the same *batchLen* number of operators. The same padding procedure is adopted for the output vectors.

The input vectors are identical in the way they are labelled for the training and validation set. The first section is represented in the same way, with the label of the currently encoded operator set to 1. In this case, the slots in the vector which correspond to the relevant predicates in each operator of the empirical model are set to 1. For example, as illustrated in the Figure 6.4, if the empirical model is represented by ($operator_1$: $bel(Ag0 - Agent, Pr0 - Proposition), not(already - seen(Ag0 - Agent, Ag1 - Agent))$, $operator_2$: $(already - seen(Ag0 - Agent, Ag1 - Agent), belbelgoalresp(Ag0 - Agent, Ag1 - Agent, Pr0 - Proposition))$, $operator_3$: $(bel(Ag0 - Agent, Pr0 - Proposition), already - seen(Ag0 - Agent, Ag1 - Agent))$, then the slots in the vector for the $operator_1$ action which represent the predicates $(bel, \neg(already - seen))$ are switched to 1, the rest of the predicates being kept at 0. This scheme is replicated for the other two operators as well.

In the evaluation phase, the aforementioned scheme is compared with an encoding scheme

sans the presence of the speech model. In this alternative scheme, the first section is represented in the same way, with the label of the currently encoded operator set to 1. For the second section, the slots in the vector which correspond to the relevant predicates in each operator are set to 1. For example, as illustrated in the figure 6.5, if the action currently being encoded is *ask*, the predicates relevant to the action *ask* i.e. the first block of the second section corresponding to the first operator are labelled to 1, the rest of the blocks of this section being kept at zero.

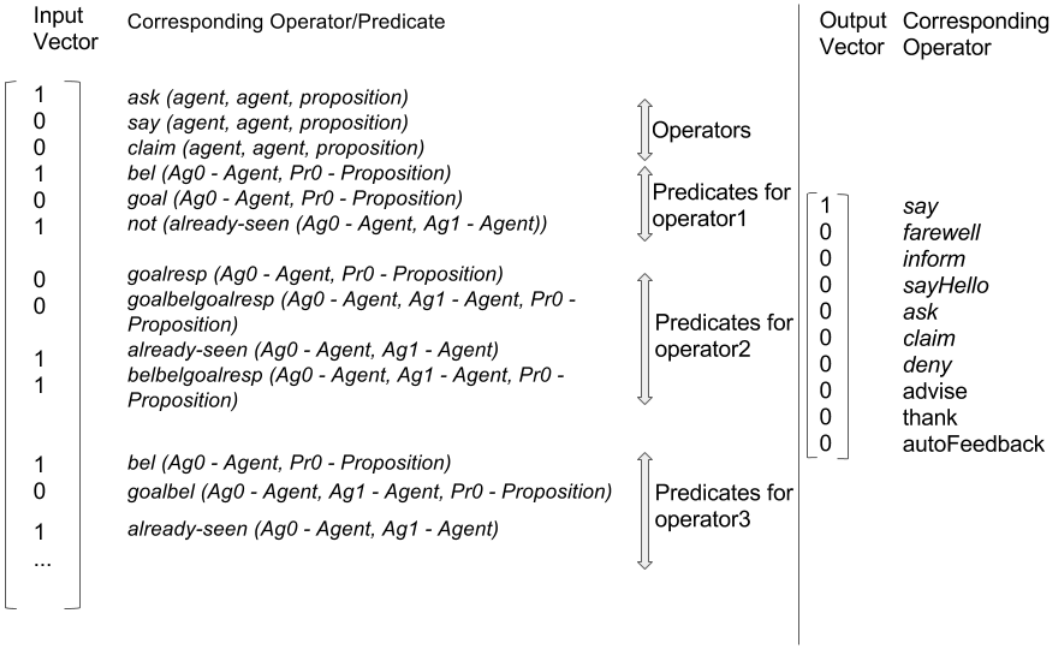

FIGURE 6.4: Vector representations for the operator "ask" and successive operator "say" for the learnt behavioral model

6.2 Evaluation

The objective of our evaluation is to obtain the best possible accuracy of: (i) the learnt model vis-a-vis the ground truth model and (ii) sequence labeling obtained with the learnt speech model. Concerning inter-operator constraint representation, two measures are defined for association rules of the form $a \Rightarrow b$, b being the head and a being the body of the rule. The confidence of a rule is the percentage of occurrences of the rule's body which also matches the body and the head of the rule i.e. $support(body)/support(head + body)$. The support of a rule is the percentage of occurrences of the rule's head which also matches the body and the head of the rule i.e. $support(head)/support(head + body)$ (Guillame-Bert and Crowley, 2012). The Titarl-mined association rules along with their confidence and support are recorded. The rules with a highest value of confidence and support (empirically determined) are retained for inter-operator constraint generation (see Table 6.4).

Finally we encode all the intra and inter-operator constraints in the form of a weighted MAX-SAT problem. The weights of the CNF clauses representing the constraints are determined differently for the inter and intra-operator cases. While the weights of the intra-operator clauses are empirically determined, the weights of the inter-operator clauses are equal to the support of the association rules. This problem can be stated as: Given a collection C of m clauses, $(C_1, \ldots, C_m)$ involving n logical variables with clause weights w_i , find a truth assignment that maximizes the total weight of the satisfied clauses in C (Yang, Wu,

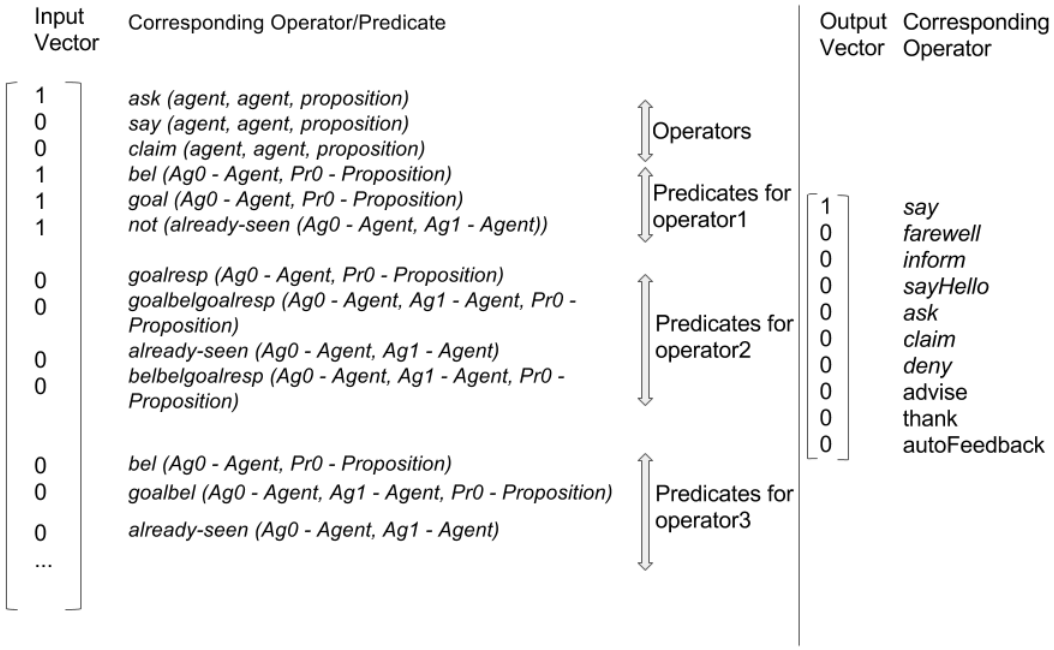

FIGURE 6.5: Vector representations for the operator "ask" and successive operator "say" for the features respective to the relevant predicates in the encoding scheme sans speech model

```
 (:action  say-hello
:parameters( ?Ag0 - Agent ?Ag1 - Agent
?Roo0 - Room)
:precondition (and  (already-seen  ?Ag0 ?Ag1) (at  ?Ag0 ?Ro0)
(at  ?Ag1 ?Ro0))
:effect (and  (not (already-seen ?Ag0 ?Ag1)) (not (at ?Ag0 ?Ro0))))
```

FIGURE 6.6: Snapshot of PDDL representation of the learnt action *sayHello*

and Jiang, 2007). We use 2 SAT solvers: the MaxSatSolver (Borchers and Furman, 1998) and the MaxWalkSat (Kautz, Selman, and Jiang, 1996). The solution produced by either solver contains all the variables which evaluate to true, which are then used to reconstruct the empirical model.

The difference between the ground truth model and the empirically determined model is represented in the form of a *reconstruction error*. This error is based on the similarity between the relevant predicates and the empirically determined predicates per operator. Let $diffPre_{o_i}$ represent the difference in *pre* lists of operator o_i in the ground truth model and the empirical model. Each time the *pre* list of the ground truth model presents a predicate which is not in the *pre* list of the empirical model, the count $diffPre_{o_i}$ is incremented by one. Similarly, each time the *pre* list of the empirical model presents a predicate which is not in the *pre* list of the ground truth model, the count $diffPre_{o_i}$ is incremented by one. Similar counts are computed for the *add* and *del* lists as $diffAdd_{o_i}$ and $diffDel_{o_i}$ respectively. This total count is then divided by the number of relevant constraints for that particular operator $relCons_{o_i}$ to obtain the cumulative error per operator. This error is summed up for every operator and averaged to obtain an average error E for the entire model. This cumulative error for the model is represented by:

TABLE 6.4: Representation of the temporal rules mined with Titarl algorithm

Rule	Confidence (%)	Support (%)
ask → say	73	96
inform → deny	69	100
autoFeedback → ask	100	60

TABLE 6.5: Cumulative error in model learning (* = with MaxWalkSat, ** = with MaxSatSolver)

Number of variables	Number of clauses	Model Error (E)	Execution Time (secs)
759	3606	41.78* 39.78**	36.44* 36.25**

$$E = \frac{1}{n}\sum_{i=1}^{n}\frac{diffPre_{o_i} + diffAdd_{o_i} + diffDel_{o_i}}{relCons_{o_i}} \tag{6.2}$$

The obtained cumulative error is mentioned in the third column of Table 6.5. The relatively high error rates can be attributed to the fact that owing to the linear structure of the relevant predicate dictionary, there is little variation among the constraints produced specific to each operator, as well as their weights. A great deal of constraints are thus solved, thus introducing a great deal of noise in the reconstructed model. We conclude from these results that the ground truth model needs to be fine tuned and reworked upon to ensure that the operators being learnt are not semantically as close to ensure a better learning rate.

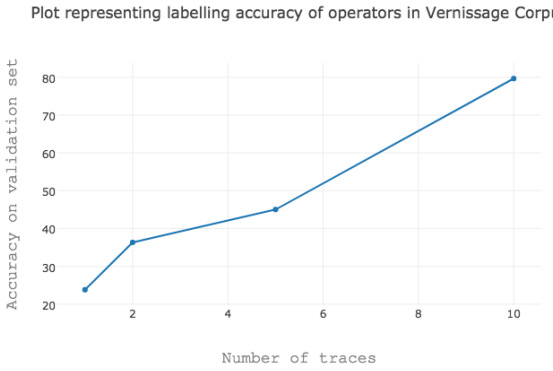

FIGURE 6.7: LSTM operator labelling accuracy (128 hidden units, 0.8 dropout rate)

6.2.1 LSTM Based Sequence Labeling

In this work, we explore two hyperparameters: the number of hidden units (set between (100, 200)), and the dropout rate (LeCun, Bengio, and Hinton, 2015) (set between (0.5, 0.75)); both of which have significant potential to influence the predictions. We use a softmax layer for classifying given operator sequences. The batch size is set to $batchLen$. We also use categorical cross entropy for the loss function and an adam optimizer (gradient descent optimizer). Each of the 10 sequences consists of about of 400 dialogues each, which is divided using

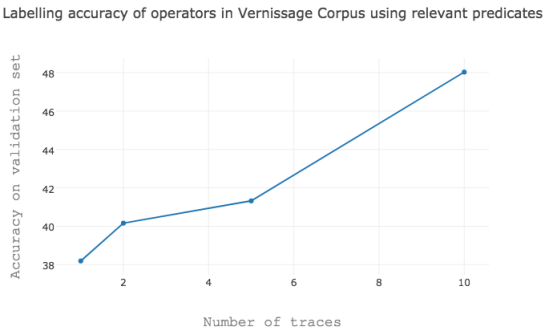

FIGURE 6.8: LSTM operator labeling accuracy with relevant predicates as constituents of the feature vector (128 hidden units, 0.8 dropout rate)

five-fold cross validation. Every training example is presented to the network 10 times i.e. the network is trained for 10 *epochs*. The results are summarized in the Figure 6.7 and are the obtained with the encoded behavioral model. The accuracy represented in the figure is the *validation accuracy* (accuracy measured on the validation set), which is the proportion of examples for which the model produces the correct output. It is represented as the fraction of examples classified correctly. The accuracy is recorded for 1, 2, 5 and 10 traces. As can be seen, the accuracy improves with the number of traces. If we consider the learning performance with the encoding scheme sans model in Figure 6.8, this prediction accuracy is not so impressive. The highest accuracy obtained is in the range of the upper forties. This demonstrates the fact that the presence of the behavioral model in the feature vector boosts the sequence labeling capacity of the learning system.

6.2.2 Towards Subsequent Planning and Robot Animation

In this subsection, we highlight some of the early work being done with the intent of translating the learnt behavioral model to animate speech and gestures in the NAO robot. The BEAT toolkit is used to map utterances produced from mapping of certain elementary propositions observed in the Vernissage corpus directly into dialogues, and feeding those dialogues to the BEAT toolkit. BEAT is the Behavior Expression Animation Toolkit that takes text as input and generates a interleaved set of gestures, with the intent of animating a graphical or tangible agent (Aly and Tapus, 2013; Cassell, Vilhjálmsson, and Bickmore, 2001). It employs the contextual and linguistic information of the text, supplemented by information from a knowledge base, with the intent to control body and face gestures alongside voice intonation. This mapping from text to gesture is implemented through a set of rules derived from extensive research on the nonverbal conversational behaviors. The modularity of the system allows to add new rules or modify existing ones. The BEAT pipeline is composed of different XML-based modules supplemented with knowledge bases and user processes, as illustrated in Figure 6.9. Each of the modules act take as input an XML object tree and produce a modified XML tree as output. The language tagging module receives text from the aforementioned corpus, and converts it into a parse tree with different discourse annotations (e.g., theme and rheme). The behavior generation module uses the output tags of the language module and suggests all possible gestures, inserting these gestural descriptions at appropriate places in the tree. The types of nonverbal behaviors are specified in the *NVBTypes.xml* file, and currently include the following:

1. Gaze: either towards or away from the hearer,

2. Eyebrows,

3. Intonation Accent,
4. Intonation Tone,
5. Gesture Right,
6. Head nod,
7. Posture Shift and
8. Pause

This tree is then passed to the behavior filtering module which filters out conflicting and undesired behaviors from the XML. The user-defined data structures, like: the generator and filter sets, provide the rules for the generation and filtration modules respectively. This is supplemented with contextual information from the knowledge base for generating relevant and precise nonverbal behaviors. The behavior scheduling module converts the surviving behaviors in the input XML tree into a set of interleaved speech and gestures. It includes a TTS (text-to-speech) engine that calculates the timings of words and phonemes, which helps in constructing an animation script for the interleaved gestures with words. The script compilation module compiles the animation script into execution commands that can be executed by a virtual agent or a humanoid robot.

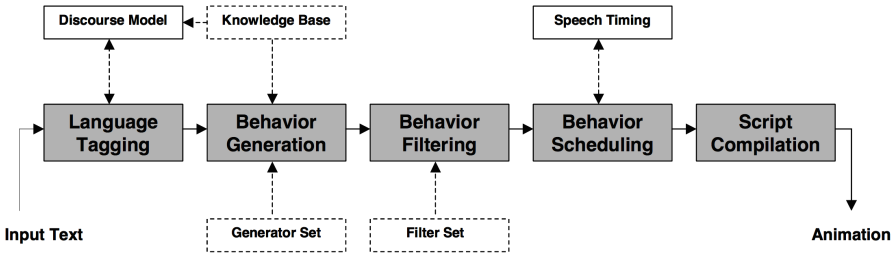

FIGURE 6.9: Architecture of the BEAT toolkit (taken from Cassell, Vilhjálmsson, and Bickmore, 2001)

Some of the dialogues and their Behaviour Modelling Language (BML) (Kopp et al., 2006) outputs are mentioned in the Figures 6.10, 6.11 and 6.12.

In our future course of work, we intend to develop a module which converts the BML markup into animations in the NAO. These animations will then be executed on the NAO and be used to interface with human subjects to evaluate their feedback.

6.3 Conclusion and Perspectives

This chapter introduces an approach to learn robot behavioral models from HRI traces in the form of PDDL action models using a MAX-SAT framework. This learning approach is very much on the lines of SPMSAT, using the hard constraints and a portion of the soft constraints which are mined with an algorithm used to mine temporal time sequences. The utility of the learnt model is demonstrated by using it to label speech acts in the HRI exchanges using the sequence labeling capabilities of the LSTM network. The relatively high error rates of the reconstructed model are attributed to the linear structure of the predicates relative to the operators of the hand woven model, introducing discrepancies in the form of falsified predicates in the output. Some conclusions and perspectives are as follows:

- A reworking of the semantics of the hand woven model in terms of the utilized predicates is required to boost the learning rate and reduce the error.

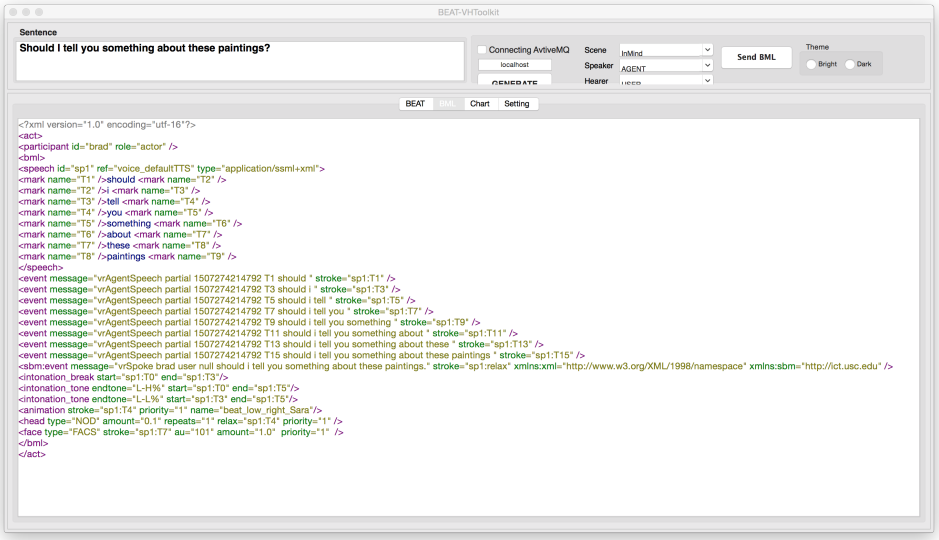

FIGURE 6.10: BML representation of the dialog "So should I tell you something about these paintings?"

- The usage of the Titarl algorithm to represent the trace set in the form of symbolic time sequences is done with the assumption that exchanges between humans are characterized by an interplay of gesture and speech, which have a tendency to overlap (i.e. overlaps between speech-gesture and speech-speech). The chosen algorithm is thus equipped to deal with such overlaps. We, however, do not observe such an overlap in the case of the Vernissage corpus. Thus, with respect to the dataset at hand, the chosen algorithm can be considered as somewhat of an overkill.

- In future versions of this work, we look to include predicates which symbolize the physical gestures accompanying the acts. For example, in the case of the "sayHello" act, the definition would include predicates which symbolize the hand waving gesture.

- In our future course of work, we intend to develop an animation module which converts the BML markup into animations in the NAO. These animations will then be executed on the robot and be used to interface with human subjects to evaluate their feedback and assess the quality and impact of our learning systems.

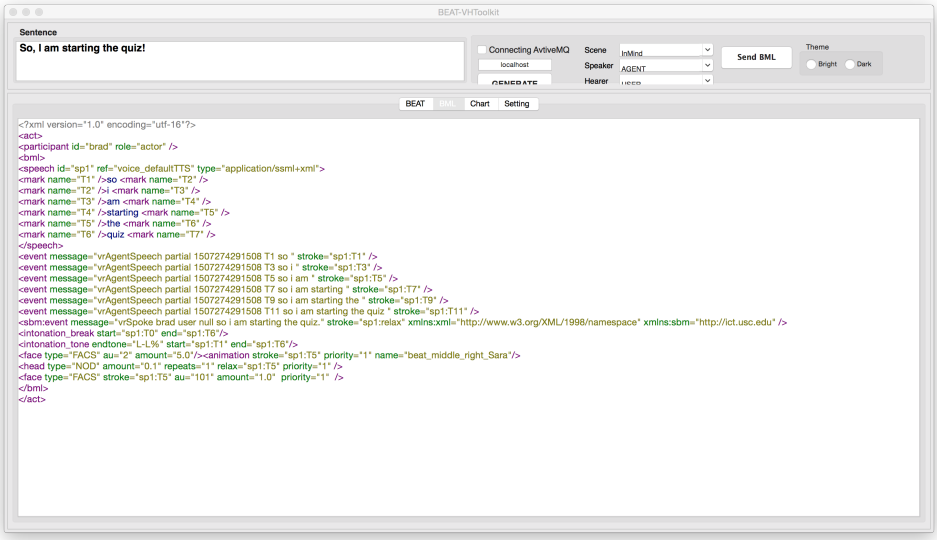

FIGURE 6.11: BML representation of the dialog “So I am starting the quiz!”

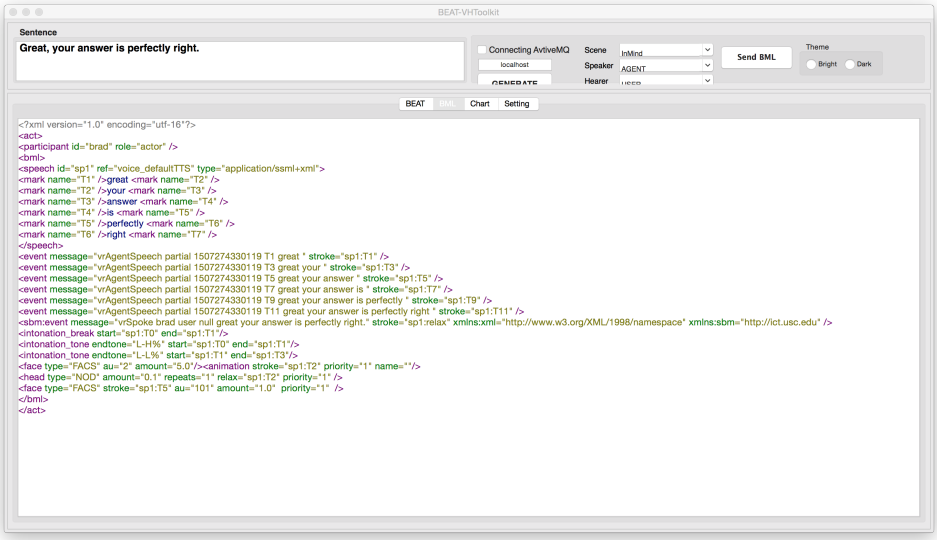

FIGURE 6.12: BML representation of the dialog “Great your answer is perfectly right”

Chapter 7

Conclusion

This chapter summmarizes the contributions of this study, the lessons learnt, the limitations of the proposed systems, and proceeds with some future perspectives of future directions of work.

7.0.1 Contribution

This study proposes two novel approaches which view the problem of reconstruction from two different perspectives and solves them accordingly. These approaches are first tested on AI planning domains to validate the conceptual strength of the approaches. They are then tested with HRI traces in a bid to validate the applicability of the approaches to a real-life scenario. A snapshot of these approaches is provided as follows:

- SAT-based Techniques: SRMLearn
  To learn the underlying action model, it is possible to exploit intra-operator and inter-operator dependencies and constraints. While the intra-operator constraints deal with the syntax of the operator, the inter-operator ones exploit the semantic relationship between successive operators in a plan sequence. We present an approach called SRMLearn (**S**equential **R**ules-based **M**odel **Learn**er), which encodes the aforementioned constraints in the form of a maximum satisfiability problem (MAX-SAT), and solves it with a MAX-SAT solver to learn the underlying action model. Unlike previous MAX-SAT driven approaches, our chosen constraints exploit the relationship between consecutive actions, rendering more accurately learnt models in the end. This approach is capable of learning purely from plan execution sequences, subtracting the need for domain knowledge.

- Connectionist Techniques: PDeepLearn
  We introduce our approach which is called PDeepLearn, which stands for **P**DDL domain **Deep Learn**er. It uses long short-term memory (LSTM) techniques for the acquisition of the underlying action model. We use the sequence labelling capabilities of LSTMs to predict the next action in a plan execution sequence based on previously observed actions. This is done by iteratively testing the sequence labelling capabilities of each of the models sampled from an exhaustively generated then trimmed set of all possible action models. Empirical results highlight that the sequence labelling achieved with an empirically isolated model speculated to be identical to the hand woven model attains the highest accuracy compared to other models in the set. Subsequent planning possibilities with this acquired model renders this LSTM-based sequence labelling approach to learn action models as an effective one.

- Application of Learning Systems to HRI scenario
  One key source of validation of the learning capacity of the aforementioned learning algorithms is to test them with real interaction traces and evaluate their capability to

learn the underlying behavioral model of the robot. We introduce an approach to learn robot behavioral models from HRI traces in the form of PDDL action models using a MAX-SAT framework. This learning approach is very much on the lines of SPMSAT, using the hard constraints and a portion of the soft constraints which are mined with the Titarl algorithm which is originally used to mine temporal time sequences (Guillame-Bert and Crowley, 2012). The utility of the learnt model is demonstrated by using it to label speech acts in the HRI exchanges using the sequence labeling capabilities of the LSTM network of memory-based recurrent neural networks. The results demonstrate the need to re-work the semantics of the ground truth behavioral model to boost the accuracy of the learnt model. The inclusion of symbolic representations of physical gestures which accompany speech acts in the PDDL model representation is in the scope of future work.

7.0.2 Open Issues

Despite the aforementioned joint advances in the fields of AP and ML, there remain some issues which continue to cloud the domain. Some of the key issues are highlighted and discussed below.

- **Fully Automated Planning-boon or bane?** The necessity of fully automated planning can be debated in a two-fold fashion:
  - **Is it rational to envision a fully automated learning-planning-execution system?** Despite the usage of the most comprehensive and state-of-the-art ML techniques, there is a certain amount of user-provided domain knowledge which cannot be discounted; especially bearing in mind the direct correlation between the speed of learning and the amount of domain knowledge furnished. Thus even the highest level of automation cannot weed out the need and advantages of human intervention.
  - **Is a fully automated system needed at all?** Even the most competent ML system is somewhat of questionable use if it is not designed to factor in human's desire of being able to control its surroundings and automated systems. Modern day users, aware of the fact that an ideal system is difficult to envision, prefer a system that can not only be useful in a real-life situation, but can also be at the mercy of the user, such that the user gets autonomy over the representation mechanism, language, kind of traces etc.
- **Ignored Aspects:**
  - **Re-usable knowledge:** The cross domain transfer of knowledge from an existing domain to enrich the model learning process in an alien domain (similar to the source domain) with few training examples, is an approach which has come into light recently with a series of works in transfer learning (Pan and Yang, 2010)).
  - **Learning during plan execution:** this refers to a situation where the expected and obtained system state after an action execution are not in accord with each other. It arises due to a flawed domain theory. It has been a major topic of neglect owing to the fact that a flawed domain theory is somewhat of an alien concept in classical planning. This has, however, been a topic of greater attention given a flurry of approaches which have been pivoted on Surprise Based Learning (SBL) (Ranasinghe and Shen, 2008)).
- **Learning with the time dimension:** Time plays an imperative role in most real life domains. For example, each dialogue in a Human-Robot Interaction (HRI) is composed

of an utterance further accompanied by gestural, body and eye movements; all of them interleaved in a narrow time frame. These interactions may thus be represented by a time sequence, with the intent of learning the underlying action model. Barring some initial works in this area, time remains an interesting dimension to explore (Zhang, Sreedharan, and Kambhampati, 2015; Guillame-Bert and Crowley, 2012).

- **Direct re-applicability of learned model:** Direct re-use of a learned model by a planner continues to remain a concern. A model that has been learned by applying ML techniques is more often than not incomplete, more concretely: inadept to be fed to a planner to directly generate plans. It needs to be retouched and fine tuned by a domain expert in order to be reusable. This marks a stark incapability of prominently used machine learning techniques to be comprehensive, leaving scope for more research.

- **Extension of classical planning to a full scope domain:** The applicability of the aforementioned approaches, most of which have been tested on highly simplified toy domains and not in real scenarios, remains an issue to be addressed. As mentioned in section 2.2, classical planning is founded on restrictive assumptions and dwells in determinism. However, the real world is laced with unpredictability: a predicate might switch its value spontaneously, the world may have hidden variables, the exact impact of actions may be unpredictable and so on (Zimmerman and Kambhampati, 2003). Thus, the application of a model learned on benchmark traces into the real world remains a point to ponder about. A fair share of algorithms that treat noisy traces are learning on benchmark traces which they have been "self-adulterated" to produce noisy ones. This can be seen as a means of a conscious steering of the learning algorithm towards a higher rate of learning, raising questions over its neutrality.

7.1 Overall Conclusion of Study

Automated Planning (AP) has been gaining steam ever since studies into human problem solving, operations research (primarily state space search) and theorem proving started gaining momentum. This is owing to the fact that the notion of a series of actions orchestrating the accomplishment of a goal is one that resonates amongst all these fields. However, depending on the domain characteristics, its constituent actions are difficult to quantify and hence codify. Planners are now equipped with the capability to reverse engineer the signatures, preconditions and effects of the domain-applicable actions. This reverse engineering is achieved by capitalizing on several state-of-the-art and classical machine learning (ML) techniques. In recent times, with the explosion in computing power and the influx of ML techniques into a considerable variety of computing fields, ML seems to have staged its comeback since the late 1990s.

This study has introduced two promising learning systems, the second of which is incremental to first. The study then proceeds to implement both systems in a chained fashion in the context of an HRI scenario. We can also conclude the following:

- The Direct re-applicability of learning systems drawn from AI domains to HRI scenario not as sound, and needs adaptation.

- Despite its fair share of drawbacks, study opens up the advantages of fusing symbolic and statistical machine learning techniques with the intention of driving forward autonomous HRI.

- While a significant number of prominent and interesting techniques have been applied to highly controlled experimental setups and simplified domains, their full-blown application to diverse and uncertain real world scenarios remains a topic of further research.

7.2 Future Work

- The learning of durative actions is a part of our future work. This is essential as the time driven interleaving of coverbal and verbal components of a speech act render it complete and "natural". The duration of each of these acts, as well as the instance at which they occur is essential.
- The behavioral model produced in this study is a starting point for more work in the direction of planning dialog acts from recycled models. In its current state, it is pretty naive to be used in a real life scenario. These speech acts are incomplete without corresponding physical acts which supplement these acts and dialogue in general. The introduction of physical acts in the speech act representations is core to the production of multimodal acts in the case of a subsequent planning scenario. These physical acts may be introduced as predicates or static predicates and must be conceptualized and tested by carving a plan out of a behavioral model containing multimodal acts.

Appendix A

Technical Aspects of Introduced Learning Systems

A.1 Technical Outline of the *SRMLearn system*

This section of the thesis introduces the technical architecture of the *SRMLearn* system which are spoken about during the course of this thesis.

A.1.1 Required Softwares

The softwares required are the following:

- Python (>version 3.0)
- SPMF Pattern mining library:
  http://www.philippe-fournier-viger.com/spmf/
- Maxsat solver (Borchers and Furman, 1998) and Maxwalksat solver (Kautz, Selman, and Jiang, 1996)

A.1.2 Implementation

The code specific to the *SPMSAT* implemetation can be found at the Github repository at the URL:
https://gitlab.com/ankuj/spmsat
The reader is invited to download the project in the form of a zip file. The reader may then scroll to the *src/* folder which houses the main file called *spmsat.py*. The following command may then be input into the command line to initiate the system:

bash: python3 spmsat.py [Domain Name] [Path to data file] [number of records] [path to problem file] [path to domain file] [weight of first information constraint] [weight of second information constraint] [weight of first action constraint] [weight of second action constraint] [choice of data mining algorithm] [confidence] [support] [satsolver]

A.1.3 Command Line Parameters

Each of the command line parameters are detailed as follows:

- *Domain Name*: This can be one of the five evaluated domains, namely: *parking, depots, mprime, satellite* and *gripper*
- *Name of the input data file*: This file normally carries a *.data* extension, with each line representing one trace. The data files usually consist of about 1000 traces each. The format of a single trace or solution can be divided into four parts:

- Action sequence
- Initial state of problem
- Goal of problem
- Alternating sequence of operator-state pairs: Each pair consists of the applied operator and the resulting world state. These pairs represent the eventual transition of the system from the initial state to the goal. These traces can be generated first by generating problems with the help of various generators available at (http://www.plg.inf.uc3m.es/ipc2011-learning/Domains.html). The generated problems can then be resolved with the help of a planner to generate the set of traces used in our algorithm. This work uses the PDDL4J planner (https://github.com/pellierd/pddl4j) to resolve problems and generate traces. A snippet of a trace is represented as:

```
[(move-car-to-curb car_2 car_0 curb_2),
(move-curb-to-car car_0 curb_1 car_2)];
[(at-curb-num car_0 curb_1),
(curb-clear curb_2),
(car-clear car_1)];
[(at-curb-num car_0 curb_0),
(at-curb-num car_1 curb_1)];
[(operator: move-car-to-curb car_2 car_0 curb_2),
(state: (behind-car car_1 car_3)),
(state: (at-curb car_3)),
(state: (at-curb car_2))];
```

Number of tested traces: This argument selects the number of traces out of a maximum of 1000 that will be used as input to SRMLearn. It is a positive integer between 1 and 1000.

Name of problem file: Subject to the domain chosen, this argument inputs the domain-specific problem file specific to SRMLearn. This problem file is used to isolate the variables in the traces and their corresponding types. This will then be used in the annotation and generalization phase to replace the variables in the traces with their corresponding types.

Name of domain file: Subject to the domain chosen, this argument inputs the domain file to SRMLearn. This domain file serves to compare the difference between the learnt empirical model and the ground truth action model, and thus calculate the accuracy of SRMLearn.

[Weight of first information constraint] [weight of second information constraint] [weight of first action constraint] [weight of second action constraint]: The following parameters help input the weights of the hard and soft constraints. These total to 4 parameters: 2 for the hard constraints, and 2 for the soft constraints (long term constraints to be specific) respectively. Each of these arguments are positive integers.

Choice of data mining algorithm: This parameter inputs the chosen data mining algorithm in order to find the frequent action pairs and use them as short term constraints. The available algorithms include the apriori algorithm 3 or the TRuleGrowth algorithm 5. These form part of the SPMF mining library (Fournier-Viger et al., 2014). This parameter can thus either be "apriori" or "trulegrowth" depending on the user's choice.

Confidence: In case the mining algorithm chosen happens to be "trulegrowth", this parameter allows to set the minimum confidence that the mined action pairs must satisfy. This parameter is used by the SRMLearn system as an input parameter to the SPMF API during an internal call to this API during SRMLearn's course of execution. In the case of ARMS, this parameter is not taken into account. This value is a decimal number between 0 and 1.

Support: Irrespective of the choice of mining algorithm, this parameter allows to set the minimum support that the mined action pairs must satisfy. This parameter is used by the SRMLearn system as an input parameter to the SPMF API during an internal call to this API during SRMLearn's course of execution. This value is a decimal number between 0 and 1.

Choice of MAX-SAT solver: This parameter helps choose the kind of MAX-SAT solver for solving the constraints which have been constructed by the original ARMS and our SRM-Learn systems. The first of these solvers has been developed by (Borchers and Furman, 1998). In the first phase, they use a GSAT heuristic to find a good solution to solving the problem. This solution serves as a seed in generating heuristics for the second phase. In the second phase, they use an enumeration procedure based on the Davis–Putnam–Loveland algorithm to find a provably optimal solution. The first heuristic stage improves the performance of the algorithm by obtaining an upper bound on the minimum number of unsatisfied clauses that can be used in pruning branches of the search tree, and use this upper bound to guide further searches. The second one is MaxWalkSat (Kautz, Selman, and Jiang, 1996), which maximizes the sum of weights of satisfied clauses, or conversely, minimizes the sum of weights of unsatisfied clauses. This parameter thus has a string value of either "maxsat" or "maxwalksat".

A.1.4 Illustration

An example of the system in action is as follows:

```
spmsat.py gripper data/gripperWithStates.data 2
problems/p01gripper.pddl
domains/gripper.pddl 100 100 50 100
apriori 1.0 0.8 maxsat
```

A.2 Technical Outline of the *PDeepLearn* system

This page represents the technical Outline of the PDeepLearn system.

A.2.1 Required softwares

- Python (>version 3.0)
- SPMF Pattern mining library (http://www.philippe-fournier-viger.com/spmf/)
- Tensorflow library (https://www.tensorflow.org/install/)
- TFLearn library (http://tflearn.org/installation/)
- Numpy library (https://docs.scipy.org/doc/numpy-1.13.0/user/install.html)

The source code pertaining to the implementation of the *PDeepLearn* system may be downloaded from the Github repository available at the link:
https://gitlab.com/ankuj/pdeeplearn
The reader may scroll to the *src/* folder which houses the main file called *pdeeplearn.py*. The following command may then be input into the command line to initiate the system:

```
bash: python3 pdeeplearn.py [Domain Name]
[Path to data file] [number of records]
[path to problem file]
```

A.2.2 Command Line Parameters

Each of these parameters of the command line are detailed as follows:

- *Domain Name*: This can be one of the four evaluated domains, namely: *depots, mprime, satellite* and *gripper*.
- *Name of the input data file*: This file normally carries a *.data* extension, with each line representing one trace. The data files usually consist of about 700 traces each. The format of a single trace or solution can be divided into four parts:
  - Action sequence
  - Initial state of problem
  - Goal of problem
  - Alternating sequence of operator-state pairs. Each pair consists of the applied operator and the resulting world state. These pairs represent the eventual transition of the system from the initial state to the goal. These traces can be generated first by generating problems with the help of various generators available at (http://www.plg.inf.uc3m.es/ipc2011-learning/Domains.html). The generated problems can then be resolved with the help of a planner to generate the set of traces used in our algorithm. This work uses the PDDL4J planner (https://github.com/pellierd/pddl4j) to resolve problems and generate traces. A snippet of a trace is represented as:

```
[(move-car-to-curb car_2 car_0 curb_2),
(move-curb-to-car car_0 curb_1 car_2)];
[(at-curb-num car_0 curb_1),(curb-clear curb_2),
(car-clear car_1)];
[(at-curb-num car_0 curb_0),
(at-curb-num car_1 curb_1)];
[(operator: move-car-to-curb car_2 car_0 curb_2),
(state: (behind-car car_1 car_3)),
(state: (at-curb car_3)),
(state: (at-curb car_2))];
```

Number of tested traces: This argument selects the number of traces out of a maximum of 700 that will be used as input to PDeepLearn. It is a positive integer between 1 and 700.

Name of problem file: Subject to the domain chosen, this argument inputs the domain-specific problem file specific to SRMLearn. This problem file is used to isolate the variables in the traces and their corresponding types. This will then be used in the annotation and generalization phase to replace the variables in the traces with their corresponding types.

A.2.3 Illustration

An example of the system in action is as follows:

```
python3 pdeeplearn.py gripper /data/gripperWithStates.data
10 /problems/p01gripper.pddl /depots/domains/gripper.pddl
```

Note: The reconstructed PDDL domain files can be found in the generatedFiles folder starting with the names $allModelFile_{(...)}.txt$. In order to test a model, the information adjacent to the text *';;model:*' in the first line of the generated model file must be copied into the command prompt when the algorithm displays the prompt 'add the model to be tested here'.

Bibliography

Agrawal, Rakesh, Ramakrishnan Srikant, et al. (1994). "Fast algorithms for mining association rules". In: *Proc. 20th int. conf. very large data bases, VLDB*. Vol. 1215, pp. 487–499.

Agre, Philip E and David Chapman (1987). "Pengi: An Implementation of a Theory of Activity." In: *AAAI*. Vol. 87. 4, pp. 286–272.

Aly, Amir and Adriana Tapus (2013). "A model for synthesizing a combined verbal and nonverbal behavior based on personality traits in human-robot interaction". In: *Proceedings of the 8th ACM/IEEE international conference on Human-robot interaction*. IEEE Press, pp. 325–332.

Argall, Brenna D et al. (2009). "A survey of robot learning from demonstration". In: *Robotics and autonomous systems* 57.5, pp. 469–483.

Arora, Ankuj et al. (2016). "A Review on Learning Planning Action Models for Socio-Communicative HRI". In: *Workshop on Affect, Compagnon Artificiel and Interaction*.

— (2017). "Action Model Acquisition using Sequential Pattern Mining". In: *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*. Springer, pp. 286–292.

Austin, John Langshaw (1975). *How to do things with words*. Oxford university press.

Bailly, Gérard, Frédéric Elisei, and Miquel Sauze (2015). "Beaming the Gaze of a Humanoid Robot". In: *Human-Robot Interaction*.

Balac, Natasha, Daniel M Gaines, and Doug Fisher (2000). "Learning action models for navigation in noisy environments". In: *ICML Workshop on Machine Learning of Spatial Knowledge, Stanford*.

Benotti, Luciana and Patrick Blackburn (2011). "Classical planning and causal implicatures". In: *International and Interdisciplinary Conference on Modeling and Using Context*. Springer, pp. 26–39.

Bevacqua, Giuseppe et al. (2015). "Mixed-Initiative Planning and Execution for Multiple Drones in Search and Rescue Missions." In: *ICAPS*, pp. 315–323.

Bisson, Francis, Hugo Larochelle, and Froduald Kabanza (2015). "Using a Recursive Neural Network to Learn an Agent's Decision Model for Plan Recognition." In: *IJCAI*, pp. 918–924.

Blaylock, Nate and James Allen (2005). "Generating artificial corpora for plan recognition". In: *User Modeling 2005*, pp. 151–151.

Borchers, Brian and Judith Furman (1998). "A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems". In: *Journal of Combinatorial Optimization* 2.4, pp. 299–306.

Brafman, Ronen I and Carmel Domshlak (2008). "From One to Many: Planning for Loosely Coupled Multi-Agent Systems." In: *ICAPS*, pp. 28–35.

Bratman, Michael (1987). "Intention, plans, and practical reason". In:

Breazeal, Cynthia (2003). "Emotion and sociable humanoid robots". In: *International Journal of Human-Computer Studies* 59.1, pp. 119–155.

— (2004). "Social interactions in HRI: the robot view". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 34.2, pp. 181–186.

Brenner, Michael (2003). "Multiagent planning with partially ordered temporal plans". In: *IJCAI*. Vol. 3, pp. 1513–1514.

Brenner, Michael and Ivana Kruijff-Korbayová (2008). "A continual multiagent planning approach to situated dialogue". In: *Proceedings of the Semantics and Pragmatics of Dialogue (LONDIAL)*, p. 61.

Briggs, Gordon Michael and Matthias Scheutz (2013). "A Hybrid Architectural Approach to Understanding and Appropriately Generating Indirect Speech Acts." In: *AAAI*.

Cashmore, Michael et al. (2016a). "Opportunistic Planning for Increased Plan Utility". In: *Proceedings of the 4th ICAPS Workshop on Planning and Robotics (PlanRob 2016)*.

— (2016b). "Strategic Planning for Autonomous Systems over Long Horizons". In: *Proceedings of the 4th ICAPS Workshop on Planning and Robotics (PlanRob 2016)*.

Cassell, Justine, Hannes Högni Vilhjálmsson, and Timothy Bickmore (2001). "Beat: the behavior expression animation toolkit". In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, pp. 477–486.

Cohen, Liat, Solomon Eyal Shimony, and Gera Weiss (2015). "Estimating the Probability of Meeting a Deadline in Hierarchical Plans." In: *IJCAI*, pp. 1551–1557.

Cohen, Philip R and Hector J Levesque (1988). *Rational interaction as the basis for communication*. Tech. rep. DTIC Document.

— (1990). "Intention is choice with commitment". In: *Artificial intelligence* 42.2-3, pp. 213–261.

Cohen, Philip R and C Raymond Perrault (1979). "Elements of a plan-based theory of speech acts". In: *Cognitive science* 3.3, pp. 177–212.

Cresswell, SN, Thomas Leo McCluskey, and Margaret M West (2009). "Acquisition of object-centred domain models from planning examples". In:

Cresswell, Stephen and Peter Gregory (2011). "Generalised Domain Model Acquisition from Action Traces." In: *ICAPS*.

Di Rocco, Maurizio, Federico Pecora, and Alessandro Saffiotti (2013). "Closed loop configuration planning with time and resources". In: *Planning and Robotics*, p. 36.

Dousson, Christophe and Thang Vu Duong (1999). "Discovering Chronicles with Numerical Time Constraints from Alarm Logs for Monitoring Dynamic Systems." In: *IJCAI*. Vol. 99, pp. 620–626.

Endsley, Mica R and DJ Garland (2000). "Theoretical underpinnings of situation awareness: A critical review". In: *Situation awareness analysis and measurement*, pp. 3–32.

Ferrer-Mestres, Jonathan, Guillem Frances, and Hector Geffner (2015). "Planning with state constraints and its application to combined task and motion planning". In: *Proc. of Workshop on Planning and Robotics (PLANROB)*, pp. 13–22.

Fikes, Richard E and Nils J Nilsson (1971). "STRIPS: A new approach to the application of theorem proving to problem solving". In: *Artificial intelligence* 2.3-4, pp. 189–208.

Fournier-Viger, Philippe, Roger Nkambou, and Vincent Shin-Mu Tseng (2011). "RuleGrowth: mining sequential rules common to several sequences by pattern-growth". In: *Proceedings of the 2011 ACM symposium on applied computing*. ACM, pp. 956–961.

Fournier-Viger, Philippe et al. (2012). "Mining sequential rules common to several sequences with the window size constraint". In: *Canadian Conference on Artificial Intelligence*. Springer, pp. 299–304.

Fournier-Viger, Philippe et al. (2014). "SPMF: a java open-source pattern mining library". In: *The Journal of Machine Learning Research* 15.1, pp. 3389–3393.

Fox, Maria and Derek Long (2003). "PDDL2. 1: An extension to PDDL for expressing temporal planning domains". In: *Journal of artificial intelligence research*.

Fung, Wai-keung and Yun-hui Liu (2003). "Adaptive categorization of ART networks in robot behavior learning using game-theoretic formulation". In: *Neural Networks* 16.10, pp. 1403–1420.

García-Martínez, Ramón and Daniel Borrajo (2000). "An integrated approach of learning, planning, and execution". In: *Journal of Intelligent and Robotic Systems* 29.1, pp. 47–78.

Garoufi, Konstantina (2014). "Planning-Based Models of Natural Language Generation". In: *Language and Linguistics Compass* 8.1, pp. 1–10.

Garoufi, Konstantina and Alexander Koller (2010). "Automated planning for situated natural language generation". In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, pp. 1573–1582.

Ghallab, Malik, Dana Nau, and Paolo Traverso (2004). *Automated planning: theory & practice*. Elsevier.

Gil, Yolanda (1992). *Acquiring domain knowledge for planning by experimentation*. Tech. rep. DTIC Document.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Gregory, Peter and Stephen Cresswell (2015). "Domain Model Acquisition in the Presence of Static Relations in the LOP System." In: *ICAPS*, pp. 97–105.

Gregory, Peter and Alan Lindsay (2016). "Domain Model Acquisition in Domains with Action Costs". In: *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016*. Pp. 149–157.

Guillame-Bert, Mathieu and James L Crowley (2012). "Learning Temporal Association Rules on Symbolic Time Sequences." In: *ACML*, pp. 159–174.

Guiraud, Nadine et al. (2011). "The face of emotions: a logical formalization of expressive speech acts". In: *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*. International Foundation for Autonomous Agents and Multiagent Systems, pp. 1031–1038.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Hoffmann, Jörg and Bernhard Nebel (2001). "The FF planning system: Fast plan generation through heuristic search". In: *Journal of Artificial Intelligence Research* 14, pp. 253–302.

Inoue, Katsumi, Tony Ribeiro, and Chiaki Sakama (2014). "Learning from interpretation transition". In: *Machine Learning* 94.1, pp. 51–79.

Jaidee, Ulit, Héctor Muñoz-Avila, and David W Aha (2011). "Integrated learning for goal-driven autonomy". In: *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*. Vol. 22. 3, p. 2450.

Jayagopi, Dinesh Babu et al. (2012). *The vernissage corpus: A multimodal human-robot-interaction dataset*. Tech. rep.

Jilani, Rabia et al. (2014). "Automated knowledge engineering tools in planning: state-of-the-art and future challenges". In:

Jiménez, Sergio, Fernando Fernández, and Daniel Borrajo (2008). "The PELA architecture: integrating planning and learning to improve execution". In: *National Conference on Artificial Intelligence (AAAI'2008)*.

Jiménez, Sergio et al. (2012). "A review of machine learning for automated planning". In: *The Knowledge Engineering Review* 27.04, pp. 433–467.

Kautz, Henry A, Bart Selman, and Yueyen Jiang (1996). "A general stochastic approach to solving problems with hard and soft constraints." In: *Satisfiability Problem: Theory and Applications* 35, pp. 573–586.

Kopp, Stefan et al. (2006). "Towards a common framework for multimodal generation: The behavior markup language". In: *International Workshop on Intelligent Virtual Agents*. Springer, pp. 205–217.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning". In: *Nature* 521.7553, pp. 436–444.

Mannila, Heikki, Hannu Toivonen, and A Inkeri Verkamo (1997). "Discovery of frequent episodes in event sequences". In: *Data mining and knowledge discovery* 1.3, pp. 259–289.

Marques, Tânia and Michael Rovatsos (2016). "Classical Planning with Communicative Actions." In: *ECAI*, pp. 1744–1745.

Martınez, David et al. (2016). "Learning relational dynamics of stochastic domains for planning". In: *Proceedings of the 26th International Conference on Automated Planning and Scheduling*.

McCluskey, T Lee, N Elisabeth Richardson, and Ron M Simpson (2002). "An Interactive Method for Inducing Operator Descriptions." In: *Artificial Intelligence Planning Systems*, pp. 121–130.

McCluskey, Thomas Leo et al. (2009). "Automated acquisition of action knowledge". In:

McDermott, Drew et al. (1998). "PDDL-the planning domain definition language". In:

Min, Wookhee et al. (2016a). "Player Goal Recognition in Open-World Digital Games with Long Short-Term Memory Networks." In: *IJCAI*, pp. 2590–2596.

Min, Wookhee et al. (2016b). "Predicting Dialogue Acts for Intelligent Virtual Agents with Multimodal Student Interaction Data." In: *EDM*, pp. 454–459.

Molineaux, M and David W Aha (2014). *Learning unknown event models*. Tech. rep. DTIC Document.

Molineaux, Matt, Matthew Klenk, and David W Aha (2010). *Goal-driven autonomy in a Navy strategy simulation*. Tech. rep. DTIC Document.

Mourao, Kira, Ronald PA Petrick, and Mark Steedman (2008). "Using kernel perceptrons to learn action effects for planning". In: *International Conference on Cognitive Systems (CogSys 2008)*, pp. 45–50.

— (2010). "Learning action effects in partially observable domains." In: *ECAI*, pp. 973–974.

Mourao, Kira et al. (2012). "Learning strips operators from noisy and incomplete observations". In: *arXiv preprint arXiv:1210.4889*.

Mourão, Kira et al. (2012). "Learning STRIPS Operators from Noisy and Incomplete Observations". In: *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, August 14-18, 2012*, pp. 614–623.

Muñoz-Avila, Héctor et al. (1999). "HICAP: An interactive case-based planning architecture and its application to noncombatant evacuation operations". In: *Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*. American Association for Artificial Intelligence, pp. 870–875.

Muscettola, Nicola et al. (1998). "Remote agent: To boldly go where no AI system has gone before". In: *Artificial Intelligence* 103.1-2, pp. 5–47.

Newton, MA Hakim et al. (2008). "Learning macros that are not captured by given example plans". In: *Poster Papers at the International Conference on Automated Planning and Scheduling (ICAPS 2008)*.

Newton, Muhammad Abdul Hakim and John Levine (2010). "Implicit Learning of Compiled Macro-Actions for Planning." In: *ECAI*, pp. 323–328.

Newton, Muhammad Abdul Hakim et al. (2007). "Learning Macro-Actions for Arbitrary Planners and Domains." In: *ICAPS*, pp. 256–263.

Pan, Sinno Jialin and Qiang Yang (2010). "A survey on transfer learning". In: *IEEE Transactions on knowledge and data engineering* 22.10, pp. 1345–1359.

Pasula, Hanna, Luke S Zettlemoyer, and Leslie Pack Kaelbling (2004). "Learning Probabilistic Relational Planning Rules." In: *International Conference on Automated Planning and Scheduling*, pp. 73–82.

Pasula, Hanna M, Luke S Zettlemoyer, and Leslie Pack Kaelbling (2007). "Learning symbolic models of stochastic domains". In: *Journal of Artificial Intelligence Research*, pp. 309–352.

Pednault, Edwin PD (1989). "ADL: Exploring the Middle Ground Between STRIPS and the Situation Calculus." In: *Kr* 89, pp. 324–332.

Pell, Barney et al. (1997). "Robust periodic planning and execution for autonomous spacecraft". In: *IJCAI*, pp. 1234–1239.

Perrault, C Raymond (1990). "An application of default logic to speech act theory". In: *Intentions in communication*, pp. 161–186.

Perrault, C Raymond and James F Allen (1980). "A plan-based analysis of indirect speech acts". In: *Computational Linguistics* 6.3-4, pp. 167–182.

Petrick, Ronald PA and Fahiem Bacchus (2002). "A Knowledge-Based Approach to Planning with Incomplete Information and Sensing." In: *AIPS*, pp. 212–222.

Quinlan, J. Ross (1986). "Induction of decision trees". In: *Machine learning* 1.1, pp. 81–106.

Ranasinghe, Nadeesha and WeiMin Shen (2008). "Surprise-based learning for developmental robotics". In: *Learning and Adaptive Behaviors for Robotic Systems, 2008. LAB-RS'08. ECSIS Symposium on*. IEEE, pp. 65–70.

Rao, Anand S and Michael P Georgeff (1991). "Modeling rational agents within a BDI-architecture." In: *KR* 91, pp. 473–484.

Richardson, Matthew and Pedro Domingos (2006). "Markov logic networks". In: *Machine learning* 62.1-2, pp. 107–136.

Riviere, Jeremy et al. (2011). "Expressive Multimodal Conversational Acts for SAIBA Agents." In: *IVA*. Springer, pp. 316–323.

Sadek, M David (1991). "Dialogue acts are rational plans". In: *The Structure of Multimodal Dialogue; Second VENACO Workshop*.

Sadohara, Ken (2001). "Learning of boolean functions using support vector machines". In: *International Conference on Algorithmic Learning Theory*. Springer, pp. 106–118.

Safaei, Javad and Gholamreza Ghassem-Sani (2007). "Incremental learning of planning operators in stochastic domains". In: *SOFSEM 2007: Theory and Practice of Computer Science*. Springer, pp. 644–655.

Sanner, Scott (2010). "Relational dynamic influence diagram language (rddl): Language description". In: *Unpublished ms. Australian National University*, p. 32.

Shen, Wei-Min (1993). "Discovery as autonomous learning from the environment". In: *Machine Learning* 12.1-3, pp. 143–165.

Silver, David et al. (2016). "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587, pp. 484–489.

Steedman, Mark and Ronald PA Petrick (2007). "Planning dialog actions". In: *Proceedings of the 8th SIGDIAL Workshop on Discourse and Dialogue (SIGdial 2007)*, pp. 265–272.

Strenzke, Ruben and Axel Schulte (2011). "The MMP: A Mixed-Initiative Mission Planning System for the Multi-Aircraft Domain". In: *ICAPS 2011*, p. 74.

Stulp, Freek et al. (2012). "Learning and reasoning with action-related places for robust mobile manipulation". In: *Journal of Artificial Intelligence Research*, pp. 1–42.

Sutton, Richard S and Andrew G Barto (1998). *Reinforcement learning: An introduction*. MIT press.

Verstaevel, Nicolas et al. (2015). "Principles and experimentations of self-organizing embedded agents allowing learning from demonstration in ambient robotic". In: *Procedia Computer Science* 52, pp. 194–201.

Wang, Xuemei (1996). "Planning While Learning Operators." In: *Artificial Intelligence Planning Systems*, pp. 229–236.

Weber, Ben George, Michael Mateas, and Arnav Jhala (2012). "Learning from Demonstration for Goal-Driven Autonomy." In: *Association for the Advancement of Artificial Intelligence*.

Yang, Qiang, Kangheng Wu, and Yunfei Jiang (2007). "Lear ning action models from plan examples using weighted MAX-SAT". In: *Artificial Intelligence* 171.2, pp. 107–143.

Yoon, Sungwook and Subbarao Kambhampati (2007). "Towards model-lite planning: A proposal for learning & planning with incomplete domain models". In: *ICAPS Workshop on AI Planning and Learning*.

Younes, Håkan LS et al. (2005). "The First Probabilistic Track of the International Planning Competition." In: *J. Artif. Intell. Res.(JAIR)* 24, pp. 851–887.

Young, Steve et al. (2013). "Pomdp-based statistical spoken dialog systems: A review". In: *Proceedings of the IEEE* 101.5, pp. 1160–1179.

Zettlemoyer, Luke S, Hanna Pasula, and Leslie Pack Kaelbling (2005). "Learning planning rules in noisy stochastic worlds". In: *AAAI*, pp. 911–918.

Zhang, Yu, Sarath Sreedharan, and Subbarao Kambhampati (2015). "Capability Models and Their Applications in Planning". In: *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, pp. 1151–1159.

Zhuo, Hankz H, Qiang Yang, and Subbarao Kambhampati (2012). "Action-model based multi-agent plan recognition". In: *Advances in Neural Information Processing Systems*, pp. 368–376.

Zhuo, Hankz Hankui and Subbarao Kambhampati (2013). "Action-Model Acquisition from Noisy Plan Traces." In: *IJCAI*.

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

Zhuo, Hankz Hankui, Tuan Anh Nguyen, and Subbarao Kambhampati (2013). "Refining Incomplete Planning Domain Models Through Plan Traces." In: *IJCAI*.

Zhuo, Hankz Hankui and Qiang Yang (2014). "Action-model acquisition for planning via transfer learning". In: *Artificial intelligence* 212, pp. 80–103.

Zhuo, Hankz Hankui et al. (2010). "Learning complex action models with quantifiers and logical implications". In: *Artificial Intelligence* 174.18, pp. 1540–1569.

Zhuo, Hankz Hankui et al. (2011). "Cross-Domain Action-Model Acquisition for Planning via Web Search." In: *ICAPS*.

Zimmerman, Terry and Subbarao Kambhampati (2003). "Learning-assisted automated planning: looking back, taking stock, going forward". In: *AI Magazine* 24.2, p. 73.