Grounding Language Models in World Models

Roma Patel

A dissertation submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in the Department of Computer Science at Brown University

Providence, Rhode Island
May 2023
This dissertation by Roma Patel is accepted in its present form by the Department of Computer Science as satisfying the dissertation requirement for the degree of Doctor of Philosophy.

Date

Ellie Pavlick

Recommended to the Graduate Council

Date

George Konidaris

Date

Stefanie Tellex

Date

Felix Hill

Approved by the Graduate Council

Date

Andrew G. Campbell
Dean of Graduate School
Abstract

Computational models of language that are grounded in some world can learn the meaning of words in the world they exist in. A typical goal in grounded language learning is to learn to map language to some component of an agent’s decision process, for example, goals or rewards. This thesis puts forth a number of advances in methods that take in natural language, and learn to ground it to components of decision processes in reinforcement learning tasks. Unlike previous work, the models we build ground natural language to symbolic representations that can integrate with all components of an agents model, exploring the benefits of different types of symbolic languages that serve as an intermediate representation. Further, we explore how we might transfer grounded knowledge to pretrained text-only models while keeping the previously learned textual knowledge and parameters intact. The primary contributions of this thesis are to develop models that 1) convert natural language instructions to symbolic task representations that allow planning with temporal constraints, 2) allow different parts of natural language to update different parts of a reinforcement learning system and 3) allow us to transfer ungrounded textual models to learn grounded concepts. This thesis concludes by discussing the steps forward which outline the goal of grounding language models in world models.
Contents

Abstract iii
List of Figures ................................................................. 6
List of Tables ................................................................. 9

Part I: Preliminaries ......................................................... 11
1 Introduction ................................................................. 12
  1.1 The Grounding Problem .............................................. 13
  1.2 Grounded vs. Ungrounded Models of Language ............... 14
  1.3 Thesis Statement: Grounding Language Models in World Models .............................................. 16
  1.4 Contributions .......................................................... 17
2 Background ................................................................. 20
  2.1 Language Models .................................................... 20
  2.2 Reinforcement Learning Models .................................. 23
  2.3 Summary of Background ............................................ 26

Part II: Grounding Language to Rewards .......................... 28
3 Grounding Language to Non-Markovian Tasks with No Supervision of Task Specifications ....................... 28
  3.1 Linear Temporal Logic ............................................... 32
  3.2 Weakly Supervised Semantic Parsing Model .................. 33
  3.3 Planning in an Environment ....................................... 37
  3.4 Environments and Data .............................................. 39
  3.5 Experimental Evaluation .......................................... 44
  3.6 Conclusion ............................................................ 50
4 Planning with State Abstractions for Non-Markovian Task
4.1 Problem Formulation for State Abstraction 56
4.2 State Abstractions and Temporal Logic 57
4.3 Algorithm for Abstract MDPs 61
4.4 Grounding language to LTL formulae 62
4.5 Robot Experiments 66
4.6 Conclusion 67

Part III: Grounding Language to Multiple MDP Components 69
5 On the Relationship Between Structure in Natural Language and Models of Sequential Decision Processes 70
5.1 Why Sequential Decision Processes? 71
5.2 Classes of Markov Decision Processes 72
5.3 Syntactic Categories in Language 74
5.4 Different Syntax for Different Languages 75
5.5 Language Elements for Describing Decision Processes 75
5.6 Discussion and Conclusions 78

6 RLang: A Declarative Language for Expressing Prior Knowledge for Reinforcement Learning 80
6.1 Program Guided Agents and DSLs 84
6.2 RLang: Expressing Prior Knowledge about RL Tasks 85
6.3 Demonstrations 91
6.4 Discussion 96

Part IV: Teaching Ungrounded Models to be Grounded 98
7 Mapping Language Models to Grounded Conceptual Spaces 99
7.1 Experimental Design 102
7.2 Isomorphism and Controls for Memorisation 104
7.3 Evaluations 106
7.4 Generalisation to Unseen Worlds 108
7.5 Generalization to Unseen Concepts 110
7.6 Discussion 113

Part VI: Conclusion 116
ACKNOWLEDGEMENTS

It has been an absolute privilege to be advised by, to embark on research adventures, and to form long-lasting relationships—both academic and otherwise—with all the individuals I have interacted with throughout the course of my PhD.

First and foremost, to my advisor, Ellie Pavlick, for being such an exemplary example of a wonderful person, researcher, advisor and teacher—and for giving me so much to aspire to. Through the years of being advised by Ellie, I have learnt through observation, what it means to be a great advisor—how to ask the right questions, be thorough in experimentation and writing, how to be a good collaborator and how to inspire others to produce great work. I cannot thank Ellie enough for the countless illuminating discussions, for always being excited about my research agenda and ideas I put forward, for nudging me towards applying to opportunities, for making me read the right literature, for giving me the freedom to come up with my own ideas, but also directing me towards others that might be more fruitful, and finally, for teaching me to recognise and ask the questions that will truly make a difference.

To the members of my committee; George Konidaris—in the early years of my PhD for patiently answering questions about the field of reinforcement learning, and in the later years, for always making the time to discuss all our varied projects on language and RL. George has been such a wonderful example of the kind of researcher I want to be—always inspiring his students to aim for the hard problems; the ones that will change the field and open up new avenues of research, and for always being willing to discuss new ideas and ways that we can make them happen. To Stefanie Tellex, for introducing me to a world of robots, and real use-cases of the models of language that we build. Stefanie is truly and exceptionally different from any researcher I know—the excitement and energy she brings to every research discussions have been one of the biggest sources of motivation on many hard, robotics projects, and I am forever thankful, and in awe of her ability to encourage us to make things work on robots. To Felix Hill, for always making the time to discuss the exciting intersection
of various fields, and the inspiring talks on all the potential of language models and the future of language grounding—it has been such a source of inspiration, and I look forward to many more discussions and collaborations in the future. To Michael Littman—for the clarity and engagement brought to every talk and discussion, and for all the feedback and endless knowledge about the field of RL. To Stephen Bach, James Tompkin, Amy Greenwald, Daniel Ritchie and Chen Sun—for the collaborations on various projects, REU programs, feedback on drafts and talks throughout my time at Brown; and for the discussions, both research, and otherwise—it has been a privilege to know you all and I look forward to many more discussions in the future.

To the members of Lunar Lab and the journey we’ve gone through together—Aaron Traylor, Jack Merullo, Albert Webson, Dylan Ebert, Charlie Lovering, Sydney Zink—thank you all for the discussions, paper reviews, lab hangouts and everything in between.

To my collaborators outside of Brown—Jason Baldridge, Alexander Ku, Peter Anderson, Eugene Ie—I learned so much that summer from your wonderful guidance, and I look forward to more exciting discussions and collaborations in the future. To Yoram Bachrach, Angeliki Lazaridou, Richard Everett, Laura Weidinger, Ian Gemp, Edward Hughes, Chris Dyer, Marta Garnelo—I cannot imagine being as excited about my future research if it had not been for the summer I spent working with you all; it has truly been a privilege and an inspiration to know all of you. To Daniel Fried, Aida Nematzadeh, Nicholas Tomlin, Jenn Hu—our reading group discussions and the depth of knowledge of linguistics and pragmatics that this group had was incredible, and I look forward to many more such collaborations. To Ani Nenkova, Byron Wallace, Jessy Li, Ben Nye, Yinfei Yang, Iain Marshall—thank you for taking a chance on me as an undergrad, and getting me started on this path; and for the patience and encouragement in teaching me how to get started in research. To Sam Bowman, Ellie Pavlick, Ian Tenney, Alex Wang, Tom McCoy, Alexis Ross, Najoung Kim, Tal Linzen, Patrick Xia—that summer we all spent at JHU, confined in one conference room, was equal parts fun, stressful, productive and enlightening (even if it came with sometimes disappointing results) and I’m so glad I know all
of you because of it.

To my friends from home—Anuj Shetty, Aishwarya Sariya, Yash Sawant, Mikail Khona—thank you for the years of friendship and the memories we’ve made growing up together, and for making me look forward to coming back home, even if it is never often enough. To the backpacking crew—Sam Thomas, Franco Solleza, and Ellie Eng (long live FRSE), thank you for the long car rides, truly awful jokes (Sam), climbing obsession (Franco), near-deathly hikes, breathtaking views, and above all, the hours spent together outdoors together taking a break from work. To Catherine Chen, William Rudman, Jack Merullo—my work days would be so much less productive (and less fun) if not for the strictly enforced lunch breaks in our good ol’ PhD lounge, and I’m so sad that I have to leave them behind; and also to Catherine Chen and Michal Golavanesky for the weekly drinks, movie nights that I wish we’d realised we needed to start earlier. To the climbing group—Max Merlin, Eric Rosen, Thao Nguyen, Nick Demarinis, Ifrah Idrees—I am so grateful to have known you through the years, and for all hours spent in the robotics lab, and also at the climbing gym. To the Alaska (and now, band) crew—Calvin Luo, Jonah Philion and Yi Jiang—thank you for the travels, the music, and the hours spent over video calls laughing over song lyrics. To Cole’s family—Dana, Tess and Riley Williams, John Fowler, Eimile Nolan (and often times, the other Nolans)—for the always fun-filled holidays, and the winters and summers we’ve spent in different places.

To the friends I’ve made in academia—Nicks’ (Tomlin and McKenna), Alexs’ (Ku and Tamkin), Sidd Karamcheti, John Hewitt, Lucy Li, Valts Blukis, Hao Tan, Tim Dozat, Alexis Ross, Sabrina Mielle, Arya McCarthy, Vanya Cohen, Shailee Jain, Elias Stengel-Eskin, Mitchell Gordon, and so many others—one of the best things about being in this research field has been the many encounters we’ve had in all the places our academic lives have taken us—none of the conferences, internships, or research journeys would have been half as exciting if not for the conversations (both research and otherwise), adventures and academic experiences we’ve embarked on through this PhD journey. To Lauren Clarke, Laura Dobler, Jesse Polhemus, Eugenia DeGouevia, and now, John Tracy-Ursprung—thank you for
making the administrative tasks we’ve undertaken as painless as they could possibly be. I am forever indebted to your competence, warmth and how much of a pleasure it has been interacting with all of you. To my friends at Brown over the years—Nihal Nayak, Michal Golovanesky, Thao Nguyen, Eric Rosen, Ifrah Idrees, Max Merlin, Nick DeMarinis, Deniz Bayazit, Kaiyu Zheng, Cam Allen, Ben Abbamatteo, Saket Tiwari, Akhil Bagaria, Rafael Rodriguez-Sanchez, Skye Thompson, Jessica Forde, Talie Massachi, Lucy Qin, Denizalp Goktas, Kweku Kwegyir-Aggrey, Omar Afzaal, Bridget Powell—the last few years at Brown would not have been as fulfilling without the many dinners, lunches, happy hours and grad events we’ve had over the years. And to the people I’ve encountered through various academic endeavours, even if briefly, during my PhD—Alex Warstadt, Ethan Perez, Sebastian Schuster, Daniel Fried, Kaylee Burns, Alberto Testoni, Isabel Papadimitrou, Kawin Ethyarajh, Dibya Ghosh, Nelson Liu, Will Merrill, Sonia Murthy, Eric Wang, Gabriel Ilharco, Alane Suhr, Yoav Artzi, Gabe Grand, Dan Garette, Dipanjan Das, Neil Rabinowitz, Satinder Singh Baveja, Ben Coppin, Andrea Tachetti, Edgar Duéñez-Guzmán, Thore Graepel, Kevin McKee, Robert Hawkins, Desmond Elliott, Yonatan Bisk, Jesse Thomason, Jacob Andreas—thank you for making this research journey so enlightening and giving me so much to aspire to.

And finally, more than anything else, to my parents, Eliza and Ashok; my sister, Shaina, and to Cole—nothing I have done, either academic, or otherwise, would have been possible without the years of love, support, encouragement, travels and the many discussions that made my life so much more lively. Thank you for sharing every triumph and loss and for filling my life with so much happiness, excitement, laughter and love.
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1:</td>
<td>A visual description of the chapters in this thesis.</td>
<td>18</td>
</tr>
<tr>
<td>Figure 2:</td>
<td>Navigational instructions are often <em>path-oriented</em> and depend on intermediate states rather than just dependent on the final goal state. While all paths in the figure reach the goal, only the blue correctly follows the specified temporal constraints. Our model can learn to map English to LTL expressions that satisfy the constraints with no LTL at training time.</td>
<td>29</td>
</tr>
<tr>
<td>Figure 3:</td>
<td>Using a stack for type-constrained decoding to ensure syntax of produced logical forms.</td>
<td>35</td>
</tr>
<tr>
<td>Figure 4:</td>
<td>Search algorithm to produce logical programs.</td>
<td>36</td>
</tr>
<tr>
<td>Figure 5:</td>
<td>Example of progression an ltl expression against a trajectory in the environment. Semantics are as defined in Table 1.</td>
<td>37</td>
</tr>
<tr>
<td>Figure 6:</td>
<td>Examples of 4 different OSM maps in our data, with their corresponding number of landmarks available from the API.</td>
<td>39</td>
</tr>
<tr>
<td>Figure 7:</td>
<td>Creating Voronoi diagrams and underlying connecting graphs of the environment.</td>
<td>42</td>
</tr>
<tr>
<td>Figure 8:</td>
<td>Algorithmic path simulation.</td>
<td>43</td>
</tr>
<tr>
<td>Figure 9:</td>
<td>Example still from our robot demonstration in simulation with a Skydio drone.</td>
<td>50</td>
</tr>
</tbody>
</table>
Figure 10: Our environment is a gridworld with three floors, each consisting of rooms that consist of grid cells. The white arrow shows an example path the drone can take in the environment. We also include sample natural language commands (and their LTL formulae) that the drone successfully executed.

Figure 11: Complete pipeline for the translation of a natural language instruction to an LTL formula, then to a Buchi automaton, and to a plan that gives us action sequences to correctly reach the goal location specified by the task.

Figure 12: Deterministic Buchi automaton. Atomic propositions in yellow circles correspond to those in level 0 and atomic propositions in green circles correspond to those in level 1. The transitions of the automaton refer to constraints over the propositions that are satisfied on taking that path.

Figure 13: Examples, left and right, tested in simulation. In each example, a natural language instruction is converted to an LTL expression, then to a corresponding AP-MDP to find a policy. An agent then executes the policy in the specified environment to reach the correct goal state through the desired path.

Figure 14: RLang programs are parsed by the interpreter to create a partial model of the MDP and its solution.

Figure 15: Figure shows the environments we use to demonstrate different types of RLang information.
Figure 16: Figure(a) shows the return for an RLang-informed DDQN agent (blue) that leverages hierarchical information vs. standard (flat) DDQN (red) in 2D-Minecraft. Figure(b) shows average return for Lunar Lander in which an RLang-informed PPO agent (blue) leverages an initial advice policy (with average performance shown in yellow). Figure(c) shows the average return curves for RLang-informed Q-Learning with information about the reward and dynamics in Lava-Gap.  

92

Figure 17: Figure illustrates the difference between (a) learning the meaning of a word through other word forms, that could potentially be self-referential within a symbol system and (b) learning word meaning by grounding to some external context outside of the symbol system.  

100

Figure 18: Figure shows example worlds and groundings for three concept categories: colours, cardinal directions, and spatial terms.  

103

Figure 19: Figure shows how colours and modifiers transform in rotated worlds. The leftmost figure shows a full 3-D colour space of 367 colours. The three figures on the right, show four sample colours in their original world, a world rotated by 90°, and a randomly rotated world, showing us how the structure of the space is preserved in an isomorphic rotation, but distorted in a random rotation.  

106
Figure 20: Figure shows example worlds and groundings for the spatial domain. The left panel shows example grid worlds where the location of the 1 denotes a point grounded in the world, with a corresponding text instantiation of that particular concept. The two panels on the right show example outputs from the smallest and largest models. We show the top three most probable words from each model along with the probability of that word. When computing Top-1 accuracy, we only consider the first, however, we consider all 3 when computing Top-3 accuracy.

Figure 21: Figure shows example worlds and groundings for the colour domain. The left panel shows example RGB codes and associated colour names, while the right shows example outputs from the smallest and largest models, with the top three most probable words from each model along with the probability of that word. When computing Top-1 vs. Top-3 accuracy, we consider only the first vs. all 3 outputs respectively.

List of Tables

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Semantics of LTL progression functions.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 2</td>
<td>Evaluation of systems on the SAIL dataset.</td>
</tr>
<tr>
<td>Table 3</td>
<td>For path evaluation, we evaluate precision, recall, accuracy (higher is better) and edit distance (lower is better).</td>
</tr>
</tbody>
</table>
Table 4: Evaluation on all environments. The first number shows goal-state accuracy (higher is better) while the second shows edit distance compared to the ground-truth trajectory (lower is better).

Table 5: Evaluation on all environments. The first number shows goal-state accuracy (higher is better) while the second shows edit distance compared to the ground-truth trajectory (lower is better).

Table 6: Parts of speech (as tagged by the PTB) for word classes important to decision making.

Table 7: Differences in complexities in different classes of MDPs. Each row shows a class of MDPs while columns show POS categories that can be used to ground to existing parts of each model, as enumerated in Section 3.

Table 8: Comparison of DSLs proposed for RL agents and the types of expressible MDP information.

Table 9: Table shows average distance (lower is better) between model-predicted groundings and true groundings in the world, averaged over all instances in the test set. We see that the largest model has an average distance of predictions significantly lower than random.

Table 10: Table shows example model predictions (from the GPT-3 model) and distances from the true groundings in RGB space. The first column shows the true concept while the second column shows model predictions and their distances from the true concept.
Part I: Preliminaries
INTRODUCTION

Imagine you have lived in an enclosed room your entire life—separate from all the richness of the world, or the entities and objects that exist in it, that you would normally interact with. Imagine, further, that you have only ever received information as symbols written down on a wall in front of you. If the symbols were to be tokens of a natural language, such as English or French, through this communication, you might begin to infer some rules that determine the usage of words and their co-occurrences, which might give you the ability to generate fluent language. Is this information enough to truly understand the meaning of the words being used? This question (first brought forth in the Chinese Room Argument [120] has been made many times in the study of language, however the currently prevalent language models [29, 41, 114] are trained in a fashion that is similar to what we described above. They receive large amounts of streams of text data, and are trained to learn to understand or generate language, based on statistical co-occurrences and patterns that can be extracted purely from text alone.

Now consider instead the reality of what led you, as a human, to learn language. The world we live in allows us to perceive, interact, and communicate about objects that exist and events that take place around us. A large component of the meaning we attribute to a certain word (e.g., the word *red*) is some combination of it’s visual aspects, it’s similarity to other related concepts (e.g., the word *orange*) and so on. Without being able to connect the name attributed to an object i.e., the word form *red* to it’s realisation in the world, there can be no external meaning attributed to that word form. In order to allow models to condition on more than just textual information, there has been a parallel body of work that focuses on models that take in language inputs but are grounded in the world [91]—instead of just receiving textual information, they also take in visual features or
elements in an environment that an agent is immersed in. This *grounding* in the world is what can allow models to learn correspondences between words and how they are realised in the decisions an agent makes in the world.

This dissertation attempts to build models that understand natural language models by conditioning on the decision-making processes of agents that receive information from environments in the world, to allow better learning. We introduce a family of methods that build models that train on language as well as external non-linguistic contexts i.e., the sequential decision making processes of agents’ acting in environments. This modelling allows agents to learn the meaning of the input language they receive in correspondence with elements in the world. We show that natural language can be grounded to several components of an agent’s world model, each allowing different gains in performance, as well as how different families of languages can ground to agents in reinforcement learning environments. Lastly, we introduce techniques that allow us to teach ungrounded textual language models to learn grounded word meanings for a set of concepts.

Before we illustrate the above methods in later chapters of this dissertation, we use this chapter to define the broader definition of grounding, in the context of current models of natural language, to allow us to situate ourselves for the work we present in this thesis.

1.1 THE GROUNDING PROBLEM

In order for someone to learn a language, they must solve the symbol grounding problem [62]—the problem of realising the meaning of the words in a language, outside of the textual boundaries of that language. As argued by Searle and Harnard [62, 121] it might be very impractical for a non-speaker of a language (for example, Chinese) to learn the language solely from a Chinese-Chinese dictionary of textual definitions. The referents in each definition would pass endlessly from one symbol to another, without having any of them truly defined or grounded in the world, making this an impossible task. In every language the symbols are named (i.e., their *form*) and have some realisation that is their grounding in the world (i.e.,
their meaning) in order for them to be semantically interpretable.

To tackle the grounding problem, meaning can be imparted to a system of symbols, by connecting the system to the world in the right way [49, 50]. One way of achieving this, that we focus on in this thesis, is to ground natural language instances—be it single words or entire sentences—in the actions, objects and dynamics of the world that an agent exists in. To attempt to build systems that learn language in this manner, we formulate our grounding tasks as Markov Decision Processes (MDPs) that solve reinforcement learning tasks [128] that are given as natural language instructions or hints about the task. This has been explored by several prior bodies of work [6, 26], however, there has been much recent progress resulting in more sophisticated language models [29, 41] since then, and we situate the work in this thesis amidst the newer innovations in language models and semantic parsing models, in order to build systems that ground natural language in components of the world.

1.2 Grounded vs. Ungrounded Models of Language

In recent years, the field of Natural Language Processing has seen large successes of neural language models on many language tasks, achieving significant gains in performance on tasks that previous models struggled to perform well on. There has been a large body of work that focuses on building better model architectures [136], attention mechanisms [12, 92] decoding methods [65, 114] and training procedures [29, 41]. Despite these gains in performance, there is one fundamental limitation of all existing textual language models. These models, by construction, have only ever received text input i.e., word form during training but have no way of grounding this to the word’s meaning, as defined in §1.1. Instead, they only rely on contextual cues and patterns in the language data—thus learning in a manner quite analogous to the Chinese room problem described in §1.1. Several works therefore, have argued that the language modeling task (that only uses form as training data) cannot in principle lead to the learning of meaning [16].
To circumvent this, it is common to use a smaller, restricted language (for e.g., a categorial grammar defined over the world) that is directly tied to the world an agent exists in [8, 35, 87]. Each element in this language can be directly realised in the world, and functional aspects of a language can also directly be mapped to functions that can be executed by an agent in the world. By virtue of this mapping, such languages are grounded in the world, and by virtue of their underlying grammar and syntax, they are typically understandable by humans as a formal, and more restricted language. This makes the translation from natural language to such formal languages a well-formed task, since the meaning or intent of a command in each language can be mapped from one to another. This translation task, known as semantic parsing, forms a significant portion of this thesis; where we show how natural language can be grounded to components of an agent’s world by first mapping it to a suitable formal language.

In this thesis, we contextualise the grounding problem with respect to current semantic parsing models and language modelling innovations. The grounding tasks we tackle are motivated by a variety of prediction problems—for example, learning to predict rewards or trajectories for input natural language instructions or to predict concept occurrences in simple worlds. Crucially, these tasks include some other modality, apart from language, that the language inputs can be grounded to. This could be mapping from natural language strings to images, environment states, rewards in an agent’s decision process and so on. Although such tasks are only a first step towards tackling the language grounding problem, they provide a useful starting point for the field to understand how natural language might tie into tasks that involve modalities other than text.
1.3 Thesis Statement: Grounding Language Models in World Models

**Thesis Statement**

By drawing on insights from computational models of language and computational models of decision processes, we can build systems that connect elements of language to elements of the world to perform tasks specified in natural language.

In light of the framework above, there are many questions to be asked to advance the field further. The main questions that we investigate in this thesis are as below.

**Question 1.** Can we exploit existing logical languages and build frameworks that correctly ground natural language to a formal language, to allow agents to understand goals and rewards? Moreover, can we do this without requiring explicit supervision of logical forms, using techniques from unsupervised machine learning, reinforcement learning and deep learning to build such models?

**Question 2.** Given the success of translating natural language to formal languages, can we build formal languages that can better express all of the information we would like to relay? Can we structure the language such that it can ground to every component of an agent’s Markov Decision Process?

**Question 3.** Considering all the advances in grounding one natural language (English) to instruct reinforcement learning agents, can we create a
dataset and framework that allows us to ground other natural languages (e.g., Hindi or Telugu) to components of a reinforcement learning model?

**Question 4.** Finally, separate from the work above that attempts to train models to be grounded, can we use ungrounded, pretrained models that have been trained only on text, and adapt them to be grounded with minimal training data?

In this thesis, we answer these questions by laying out the above thesis statement. Section 1.4 gives an overview of the parts of this thesis that answer the questions above.

### 1.4 Contributions

The remainder of this thesis is structured as follows.

**Part 1** We go over the preliminaries of grounded language work, the models we use, and work related to this thesis that is prevalent in the literature.

**Part 2** We show how we can build weakly supervised algorithms that can translate natural language instructions into temporal logic, thus allowing natural language information to be incorporated into an agent’s reward function.

**Part 3** We create a new formal language, one that grounds not only to rewards, but any component of an agent’s model i.e., a Markov Decision Process. We show the benefits of having such a framework, and what information might be lost when only grounding to one component of an MDP, as is prevalent in the literature.

**Part 4** We show how pre-trained ungrounded language models that only function over textual inputs and outputs can be taught how to ground
language in small, constrained text environments. We evaluate over a range of grounded concepts to show the extent to which such a methodology can work.

Part 5 This concludes with a discussion of all the contents in this thesis, limitations and reflections on chapters in this thesis, and future directions of the work presented here.

The figure below contains a visual overview of this dissertation.

**Figure 1.** A visual description of the chapters in this thesis.
2

BACKGROUND

In this chapter, we provide necessary background material on the concepts, formalisms, and models used in this thesis, as well as related work in the area. Specifically, we will go over language models that are predominantly used in the field of NLP, reinforcement learning models used in the fields of Robotics and reinforcement learning (RL), and semantic parsing methods for translating natural language to formal languages that can integrate with RL models.

2.1 LANGUAGE MODELS

Language modelling is the task of predicting the next word in a given text corpus, when conditioned on the previous words before that word in a sentence. For example, consider the sentence below.

\[ c = \text{I’m hungry so I think I will order Chinese.} \]  \( (1) \)

A good model of language is one that given the context \( c \), would predict word \( w_1 = \text{food} \), over another word \( w_2 = \text{computer} \). What we want language models to estimate therefore, is the probability \( P(w_i|c) \) for all words in it’s vocabulary, and for any sequence of words that form the context \( c \), even outside of the sequences of words seen during training. Probabilities to these sequences of words can be assigned using conditional probabilities of words, in combination with the chain rule, as below. In classical statistical NLP, one way of estimating the probabilities of words is from frequency counts of words in a training corpus. More generally, as seen in Equation 2, a probability score is estimated for each word that is being generated by a language model, and for models trained only on text, this score is computed by conditioning on the words generated at previous timesteps in that sequence, as well as other co-occurrence statistics from the training
data, or current context, that differ based on the algorithm of training regime in question.

\[
P(w_{1:n} = \prod_{i=1}^{n} P(w_i|w_{<i}))
\]  

(2)

When language models are trained in such a manner—that is, on data that is entirely textual, they are ungrounded, since the words that they are learning have no explicit link to the world.

**Semantic Parsing** When dealing with reinforcement learning agents that exist and interact with the world they live in, this sort of representation of language is not usable by an agent, if necessary. To alleviate this, formal languages that are directly tied to the world that an agent lives in, and are therefore grounded, are predominantly used in the language grounding community. Such languages are more restricted than natural language in terms of their complexity, syntax, vocabulary and so forth, however, elements of the language are directly tied to elements in the world, and therefore functions defined over the language can be directly executed in the world.

The task of translating natural language to such formal languages is called semantic parsing. The formal languages typically used in semantic parsing tasks are often executable by some component of a robot solving a task. The objective of this task is to take in a natural language instruction and convert it into a logical representation, typically tied to the environment the agent is in, and one that unambiguously represents the intended meaning of the natural language instruction. For example, consider the semantic parse below.

"Walk to the bedroom and carefully pick up a blue, not the green, pillow"

\[
\downarrow
\]

Parse 1: \( \text{at\_bedroom} \land \text{blue\_pillow} \)
As we can see, the translation from natural language to a logical language results in some amount of information that is lost. The original natural language sentence was structured so as to define the meaning of each word in its context, along with certain grammatical rules that define the language. The semantic parse, in whichever language, retains the components of the input that are important to the task (namely, the intended goal that the robot’s reward function might require), however, potentially losing the subtle nuances and framing effects that might occur when people speak language. For goal-oriented tasks however, these subtleties are often irrelevant to the end goal that an agent has to solve, thus making this translation sufficient to encapsulate the information needed by an agent in its environment. In the section below, we survey related work in the field that attempts to convert natural language into formal languages in order to more easily solve reinforcement learning tasks.

**Related Work in Semantic Parsing**

In the semantic parsing literature, ample work has focused on supervised methods that translate natural language into logical forms by training models on datasets that contain paired samples of such translations. Several works have explored weak supervision methods [8, 80] that execute translated logical forms to determine final goal locations. Most approaches ground to lambda calculus expressions and logical forms of varying degrees of complexity. In the RL literature, several works use such logical forms to formulate tasks, either by creating reward functions built around such forms, or ones that maximise the probability of satisfying these formulae or by guiding policy search with a measure of distance to satisfaction of the task [85]. To the best of our knowledge, there is currently no work that attempts to ground natural language to temporal logic without supervision of logical forms. Below, we enumerate the different avenues explored by previous instruction following methods.
Parsing Natural Language to Logical Forms  In order to have a language that is grounded in the world that agents exist in, several formalisms such as combinator logics [86], neo-Davidsonian categorial grammars [8], program languages [126], temporal logical forms [89, 140] and even graphical structures [131]. There is ample prior work on supervised semantic parsing to such logical forms [18, 130, 146, 148], where datasets of paired natural language instructions and their logical form counterparts are used to train machine learning models to learn such a translation. Additionally, there is work that learns semantic parsers without explicit annotation of logical forms, by allowing the execution of learned logical forms to act as supervision in varying domains e.g., conversational logs [7], system demonstrations [8, 36, 55, 141] and question-answer pairs [38, 87]. Once the input natural language has been parsed into such logical forms, these can be executed in the environment, or used as input to a planner, in order to solve tasks as specified by the language [8, 56].

Mapping Natural Language to Components of an Agent’s World
Also relevant is the body of work which seeks to map natural language directly to action sequences, transition dynamics, and policy structures [26, 51, 98, 101]. Such methods are typically trained end-to-end, and do not pass through an explicit intermediate logical form, as we do in this work. Having an intermediate logical form can help interpretability and ensure correctness, by first representing intended semantic meaning. It also helps generalisability in different domains as opposed to sequence-to-sequence models trained to produce paths in one environment that cannot generalise or exhibit compositionality [75, 84].

2.2  Reinforcement Learning Models
In order to model agents and their interactions in environments, Markov Decision Processes (MDP) [113]—a class of sequential decision-making systems were introduced. Specifically, an MDP defines worlds that an agent exists in, where the reward at the next state of the world, and the probability of arriving at this state can be predicted based on the current state
of the world, as well as the action taken by the agent. An MDP is formally defined as follows.

**Definition 1.** A Markov Decision Process, or an MDP, can be defined as a six tuple $(S, A, R, T, \gamma, \rho_0)$, where $S$ refers to the set of states in the world, $A$ refers to the set of possible actions that an agent can take, each associated with an effect in the world, $R : S \times A \times S$ refers to the reward function, $T : S \times A \rightarrow \Delta(S)$ refers to the transition function denoting the probability of arriving at the next state in the world on taking an action, $\gamma \in [0, 1)$ refers to the discount factor, indicating what an agent’s preference might be wrt. near and long-term rewards, and $\rho_0$ refers to the probability of starting in each state.

Because of it’s Markovian nature, the reward function $R$ and transition function $T$ depend only on the previous state of the world. An agent therefore repeatedly interacts with an environment, and the goal of the optimisation process of an MDP is to find a policy $\pi(a|s) = p(A = a|S = s)$ that maximises the expected discounted cumulative return $\sum_{k=0}^{\infty} \gamma^k r_{k+1}$.

**Definition 2.** A Partially Observable Markov Decision Process or a POMDP, can be defined by extending the MDP tuple to be $(S, A, T, R, \Omega, O, \gamma)$ where the set $\Omega$ represents the set of observations that the agent can get and $O : S \times A \rightarrow \Delta(\Omega)$ is the observation function such that $O(s, a, z) = p(z|s, a)$, i.e. when the agent in state $s \in S$ executes action $a \in A$, $O(s, a, z)$ is the probability of getting the observation $z \in \Omega$; the agent never has direct access to $S$. The remaining elements are defined as in the MDP case.

When using language in combination with either MDPs or POMDPs, there are several ways in which language can be incorporated into the decision making process. For one, language instructions can be taken in as part of the observation space of the model, along with any information about the environment they receive. The model can therefore combine observational information e.g., state and environment variables as well as the additional text knowledge about the world, that can allow it to build a model of the dynamics of the world. For another, language can be parsed into some component of the reward function, that allows agents to be
rewarded when correctly following the behaviour intended by the input language. In this thesis, we mostly focus on the latter approach, where natural language is mapped to some component of an agent’s MDP, which serves as the grounded realisation of the language. In the section below we survey relevant related work on the different ways of using language in combination with MDPs.

**Related Work on Language and MDPs**

There are many different ways in which language can be used to inform RL agents [91]. These include methods that first parse the natural language into some formal language that is executable and directly tied to the underlying MDP in the world; as well as methods that directly map the input language to some component of the world—for example, action sequences that can be taken, or rewards that can be obtained. We survey the different ways that formal languages have been incorporated into RL problems, as well as the methods that directly map language to components of the world.

**Formal Languages in Reinforcement Learning** In classical planning it is standard to use a Planning Domain Description Language (PDDL; [53]) or a probabilistic extension PPDDL (probabilistic PDDL; [145]) to specify the complete dynamics of a factored-state environment. Agents may therefore request advice provided through such a language, for example to provide policy hints [94]. Similarly, [126] propose to learn a policy conditioned on a program from such a language. [5] use a simple grammar to represent policies as a concatenation of primitives (sub-policies) to provide RL agents with knowledge about the hierarchical structure of the tasks.

**Mapping Language to Components of the Reinforcement Learning Problem** Instead of translating language to an intermediate formal language, there is a large body of work that seeks to map natural language directly to action sequences, e.g. [26, 51, 98, 101]. Such methods are typi-
cally trained end-to-end, and do not pass through an explicit intermediate logical form, as we do in this work. Having an intermediate logical form can help interpretability and ensure correctness, by first representing intended semantic meaning. It also helps generalisability in different domains as opposed to sequence-to-sequence models trained to produce paths in one environment that cannot generalise or exhibit compositionality [75, 84].

2.3 Summary of Background

In this chapter we have discussed language models, world models, reinforcement learning methods, and semantic parsing approaches that translate language into intermediate representations that are directly tied to the world model of an agent. With this, we now delve into the technical contributions of this thesis, that draw from all of the technical matter above, to build models that ground natural language into world models in different ways.
PART II
3

Grounding Language to Non-Markovian Tasks with No Supervision of Task Specifications

This chapter is based on the paper "Grounding Language to Non-Markovian Tasks with No Supervision of Task Specifications" [110] published at RSS 2020, with Stefanie Tellex and Ellie Pavlick.

As motivated in the previous chapter, there are lots of instances in which we want languages that are grounded in the world, even if they are more restricted in their syntax and semantics than natural language. Moreover, natural language instructions can be translated into these languages, thus allowing humans to instruct agents in the language they are accustomed to, instead of having to learn the rules of a new formal language. However, even when instructed in natural language, a typical instruction following task is challenging for two reasons. First, the complex (e.g., temporal, sequential, conditional) constraints have to be represented in the form of logical goal specifications that state-of-the-art planners can take in. Assuming that these logical forms accurately represent the meaning of the natural language instruction, the planner has to temporally keep track of intermediate constraints while also reaching the final goal. The first problem can be resolved if we learn to parse natural language to more complex logical forms like Linear Temporal Logic (LTL) [30, 89] thus allowing the second planning problem to be solved by building LTL planners [56, 57, 89] that can find a path in the environment to satisfy the required temporal constraints. However, the semantic parsing problem in itself is extremely challenging, typically requiring paired data of language and LTL to train models to learn this translation function. This is data that is expensive to collect (in terms of time and money spent per task), difficult to collect (in
Navigational instructions are often path-oriented and depend on intermediate states rather than just dependent on the final goal state. While all paths in the figure reach the goal, only the blue correctly follows the specified temporal constraints. Our model can learn to map English to LTL expressions that satisfy the constraints with no LTL at training time.

terms of worker inaccuracies for this complex task) and moreover, models trained in this way are extremely brittle and fail to generalise to samples different from those seen during training [56, 84].

To address this, we propose to learn a semantic parsing model that does not require paired data of language and LTL logical forms, but instead learns from trajectories as a proxy. To collect trajectories on a large scale over a range of environments, we use path-finding algorithms to simulate trajectories in each environment. We then ask humans to annotate these trajectories with natural language instructions, creating datasets of language paired with trajectories in each environment. To validate the
model-produced logical forms with trajectories, we use formal methods of LTL progression [30, 133], to check satisfiability of logical forms against ground-truth paths. Once the trained model can produce these grounded LTL representations, we then use a planner with LTL-based rewards to follow the instruction. Our framework enables a robot to learn to map between language and LTL expressions with no LTL annotations required during training; only a dataset of English and trajectories, as well as a model of the environment. We test our approach on a test set of unseen natural language instructions in each environment. We evaluate on our newly collected range of different environments with natural language annotations, as well as an existing benchmark dataset [95], using metrics for path and goal-state accuracy.

In this chapter we present three main contributions. First, we ground natural language instructions to temporal logical form with no supervision of ground-truth logical forms. Second, our use of LTL as a meaning representation not only allows handling of temporal ordering but also enables the use of formal methods of progression that allow a more efficient feedback mechanism than previous approaches. Third, we test the generalisability of this method in more than 10 different environments (both artificial and real-world) by annotating trajectories with natural language instructions in each environment. We release this data as well as the data-collection procedure to algorithmically simulate paths in large environments. We see the benefits of using a more expressive language such as LTL in instructions that require temporal ordering, and also see that the path taken with our approach more closely follows constraints specified in natural language. This dataset consists of 10 different environments, with up to 2,458 samples in each environment, giving us a total of 18,060 samples. To the best of our knowledge this is the largest dataset of temporal commands in existence.

**Related Work**

In the semantic parsing literature, previous work has explored weak supervision methods to ground natural language to lambda calculus expressions or SQL queries for retrieval tasks. Related to navigation, previous
work [8, 80] has explored weakly supervised semantic parsing models. However, these ground to lambda calculus expressions and logical forms that do not handle temporal order in the way that LTL does. To the best of our knowledge, there is currently no work that attempts to ground natural language to temporal logic without supervision of logical forms during training.

LTL has been explored in the RL literature, to formulate tasks, either by creating reward functions that maximise the probability of satisfying the LTL formula [89, 140] or by guiding policy search with a measure of distance to satisfaction of the task [85]. Other work exploits the structure of LTL to decompose tasks into subtasks [133] to deal with temporal abstraction. Below, we enumerate the different avenues explored by previous instruction following methods.

**Language → Logical Form** There is ample prior work on supervised semantic parsing to logical forms other than LTL [18, 130, 146, 148] as well as to LTL [56], however all of these require paired data of language and logical form to train models. Other weakly-supervised work learns semantic parsers without explicit annotation of logical forms, by allowing the execution of learned logical forms to act as supervision in varying domains e.g., conversational logs [7], system demonstrations [8, 36, 55, 141] and question-answer pairs [38, 87]. However, these are all unrelated to navigation and planning, and the weak supervision required is much simpler, usually requiring one execution (e.g., against a database) as opposed to a longer sequence of executions (e.g., a trajectory in an environment). Most relevant to our work is the model of artzi2013weakly, however our work is different in that it grounds to LTL (rather than CCG expressions) and also provides a more efficient feedback mechanism making use of LTL progression.

**Language → Plan** Also relevant is the body of work which seeks to map natural language directly to action sequences, e.g. [26, 51, 98, 101]. Such methods are typically trained end-to-end, and do not pass through an explicit intermediate logical form, as we do in this work. Having an in-
termediate logical form can help interpretability and ensure correctness, by first representing intended semantic meaning. It also helps generalisability in different domains as opposed to sequence-to-sequence models trained to produce paths in one environment that cannot generalise or exhibit compositionality [75, 84].

**Logical Form (LTL) → Plan** Assuming we have correct logical goal specifications, there is ample work that explores how to plan and solve tasks *temporally* in environments. Previous work [30, 56, 89] has used planners in combination with LTL rewards to keep track of previous states. Recent approaches have used Deep Q-Networks with LTL specifications [133], by making use of LTL based rewards [89] where the input to the Q-value function is both the state and progressed LTL task. Other work uses hierarchical RL methods [82, 129] in the options framework, by creating one option per proposition with terminal states defined by states in which the proposition is true; giving the state and progressed LTL task as input to a meta-controller. While this line of work highlights the benefits of LTL for temporal abstraction, it does not deal with natural language, but focuses on task-solving when given the logical representations. It is therefore separate from the language grounding task that first attempts to convert natural language into representations that these methods can take in.

### 3.1 Linear Temporal Logic

In this section we explain the preliminaries of Linear Temporal Logic, the logical formalism into which we parse each natural language instruction. We go over the syntax and the semantics of the language and how they can be used in correspondence with states in an environment. We show how these semantics can be used for *progression* to check satisfiability of an LTL expression against a sequence of states (i.e., a trajectory) in the environment. Given that our weakly-supervised translation model does not have ground-truth LTL data to train on, the LTL progression is what supervises each logical form with a trajectory.
**LTL Syntax**  LTL has the following grammatical syntax:

\[ \phi ::= \pi \mid \neg \phi \mid \phi \land \varphi \mid \phi \lor \varphi \mid \diamond \phi \mid \Box \phi \mid \circ \phi \mid \phi \mathcal{U} \varphi \]

where the operators \( \neg, \land, \lor \) are the logical connectives for *negation, and, or* and the temporal operators are \( \diamond \) for *eventually*, \( \Box \) for *globally*, \( \mathcal{U} \) for *until* and \( \circ \) for *next*. Our set of propositions \( P \) consists of observable elements in the environment e.g., \((\text{at}_\text{object}, \text{is}_\text{intersection}, \text{is}_\text{corridor})\). All LTL expressions are constructed from the set \( P \) and the extended set of operators defined above i.e., the Boolean operators \( \land, \lor, \neg \) and the temporal operators \( \circ, \mathcal{U} \). From these we can define \( \Box \) (*always*) and \( \diamond \) (*eventually*) for e.g., \( \diamond \phi = \text{true} \mathcal{U} \phi \).

**LTL Semantics**  Given the observable elements in the environment that form atomic propositions, the truth value of an LTL formula is determined relative to a sequence of truth assignments \( \sigma =< \sigma_1, \sigma_2, \sigma_3, ... > \) where each state \( \sigma_i \) assigns a value of true or false to each proposition.

A proposition \( \rho \in \sigma_i \) indicates that the proposition \( \rho \) is true in the state \( \sigma_i \), thus allowing us to check existence of propositions in states for progression functions in Table 1.

**LTL Progression**  Given a sequence of truth assignments and an LTL task specification, an LTL formula can be *progressed* along the sequence. For example, the task \( \diamond(p \land \circ \diamond q) \) (i.e., eventually \( p \) and eventually \( q \)) can be progressed to \( \diamond q \) (i.e., eventually \( q \)) once the agent reaches a state where \( p \) is true. Table 1 shows how we define our progression functions.

### 3.2 Weakly Supervised Semantic Parsing Model

In this section we describe the semantic parsing model that takes in natural language instructions and converts them to LTL expressions. Our model is based on previous work [54, 61] which trains semantic parsers from denotations by searching through a space of programs during training. However, applications of previous work were to ground language to SQL queries or
logical operators to be executed against databases. We formulate this as a sequence prediction problem by representing an LTL program as a sequence of atomic propositions and operators in postfix notation. Each produced LTL expression is given a binary reward by progressing it along a ground-truth path. We explain the components of the model below.

**Program Representation**  Every token in an expression is either an operator of fixed arity (i.e., one or two arguments) or an atomic proposition. For example, the LTL expression \((a \land (\Diamond b))\) converted to postfix notation \(a b \Diamond \land\) can be realised in an environment where the propositions correspond to locations. This linearised representation allows easier execution with a stack based on the semantics of operators and propositions as shown in Figure 3. The vocabulary of atomic propositions is the set of all observable elements. In the SAIL environment, the simulated environment we use in Section 3.4, this consists of e.g., *stool, brick corridor*,.., while in every real-world environment (see Section 3.4) this consists of actual landmarks at that location in the map e.g, *Harlow Square, MIT Museum* for *Cambridge, Massachusetts* or *The Varsity Plaza, Bass Concert Hall* for *Austin, Texas*.
Figure 3. Using a stack for type-constrained decoding to ensure syntax of produced logical forms.

Encoder and Decoder Our training samples are of the form \((x, t)\) where \(x\) is a natural language instruction and \(t\) is a trajectory in the environment. As in DBLP:journals/corr/GuuPLL17 our model generates program tokens \(z_1, z_2, \ldots\) from left to right using a neural encoder-decoder model \cite{127}. We encode every utterance \(x\) with a bidirectional LSTM \cite{64} to create a contextualised representation \(h_i\) for every input token \(x_i\). Our decoder is then a feed-forward network with attention \cite{11} over the output from the encoder, that takes as input the last \(K\) decoded tokens. Formally, the probability of a decoded LTL expression is the product of the probability of its tokens conditioned on the history i.e., \(p_\theta(z|x) = \prod_t p_\theta(z_t|x, z_{1:t-1})\) and the probability of a decoded token comes from the learned parameters and embedding matrices. We keep track of the execution history i.e., the \(k\) most recent tokens \(z_{t-k:t-1}\) and concatenate their embeddings. While encoding, the input vocabulary consists of all the words contained in the natural language utterances, that are embedded with word vectors. While decoding, the output vocabulary is composed of all the propositional symbols (i.e., landmarks) in the environment as well as the operators until, and, or, not, eventually, globally, next. As shown in Figure 3 we use type-constrained decoding to ensure that the decoded programs are syntactically correct, while the LTL progression ensures that they are semantically correct, in that they give a reward of 1 against the ground truth trajectory. Using type-constrained decoding for the neural model ensures that the space of logical forms explored are always syntactically correct. Typical supervised
methods would compare each produced token with the ground-truth logical form to ensure semantic correctness, however we do this in a weakly supervised way, by progressing the final logical form against the ground-truth trajectory in the environment.

![Search algorithm to produce logical programs.](image.png)

**Figure 4.** Search algorithm to produce logical programs.

**Supervision** In a standard supervised setting, each produced logical form could simply be compared to the ground truth logical form in the data. In our weakly supervised approach, we instead use *trajectories* to supervise the produced logical forms during training. Each logical form gets a binary (1 or 0) reward by *progressing* it along the ground-truth trajectory, using the progression functions defined in Table 1. Specifically, for an LTL task $\phi$ and state $\sigma_i$, we can update $\phi$ at each point in time $i$ to reflect the parts of $\phi$ that have been satisfied. We can do this because if a sequence of truth assignments (i.e., a trajectory) satisfies an LTL formula at time $i$, then the formula progressed through $\phi_i$ is satisfied at time $i + 1$. An exception to this is a global constraint e.g., *always avoid* $A$ that requires the condition to always be met and cannot be progressed. We handle this by simply ensuring that the constraint holds at every time step.
Training

To train the model, we randomly initialise model parameters and optimise the objective function via stochastic gradient ascent, following the same training procedure in [61]. The token embedding size is 12, the beam size is 20 and the BiLSTM state dimensions are 30. The hidden state dimension of the decoder is 50, and this takes in the encoder representation, as well as the last 4 decoded tokens as input. We use Adam [74] as an optimiser with a learning rate of $0.001$. We use a mini-batch size of 8 and train for 40k iterations to achieve the reported results. For a more detailed explanation of the search algorithm and policy gradient updates for the objective function, we refer readers to [61].

3.3 Planning in an Environment

Once we produce LTL interpretations of instructions, we use an LTL-based planner that takes in the LTL task specification to find a path in the environment. Crucially, this path not only reaches the final goal-state but
**Algorithm 1 Algorithm for Supervision with LTL Progression**

1: Create training samples \((u, t)\) of utterance and trajectory pairs
2: Create LSTM encoder, feed-forward decoder, search algorithm, reward function
3: Build environment representation with states composed of landmarks
4: for each sample \((u, t)\) do
5: \(e \leftarrow \text{encoder}(u)\) .. encode utterance with LSTM encoder
6: \(\text{program} \leftarrow \text{decoder}(e)\) .. decode top \(k\) programs with feed-forward decoder for beam size \(k\)
7: for each program \(p\) do
8: for each state \(s\) in \(t\) do
9: \(p', a \leftarrow \text{prog}(p)\) .. progress the LTL expression and return the updated expression \(p'\) and the current truth assignment \(a\)
10: end for
11: if \(a = \text{True}\) then
12: \(\text{reward} \leftarrow 1\)
13: else
14: \(\text{reward} \leftarrow 0\)
15: end if
16: Update \(\text{reward}(p)\) for programs
17: end for
18: end for

also ensures that sequential constraints in the LTL formula are satisfied (e.g., first going to a certain location before reaching the goal). We use an MDP planner, adapted for LTL, as described below.

**Markov Decision Processes** We refer the reader to Definition 1 for a detailed definition.

**Linear Temporal Markov Decision Processes** To plan with temporal constraints, previous work combines the LTL expression with the
environment MDP in order to make an expanded MDP that can keep track of the relevant parts of the LTL state. A labelling function therefore annotates the transitions with labels i.e., valid propositions for each state, thus allowing the checking of satisfiability of LTL expressions. These MDPs have previously been used for planning over an MDP to satisfy an LTL formula [117]. Abstract Product MDPs (AP-MDPs) [106] have been used to combine labelled MDPs that can handle temporal expressions along with ones that can solve tasks at different levels of state abstractions. This work does not deal with different levels of abstraction, therefore we only use a product MDP at the lowest level of abstraction i.e., the environment composed of individual grid cells.

![Figure 6. Examples of 4 different OSM maps in our data, with their corresponding number of landmarks available from the API.](image)

### 3.4 Environments and Data

To compare with state of the art and existing semantic parsing models, we evaluate on SAIL [95] — a benchmark artificial environment. SAIL is an existing dataset with 3,000 samples of instructions and trajectories. While the language used is considerably complex, SAIL contains very little temporal language (e.g., use of words like until or multiple sequential constraints or landmarks to be met). To properly test handling of temporal language, as well as generalisability to real-world, environments, we also test on language commands in 10 newly-collected real-word OSM environments in different cities in the US. In all of these, our model trains without supervision of ground truth logical forms, requiring instead only example trajectories in the environment.
Here, we explain both environments and their complexities. Section 3.4 describes the artificial environment while Sections 3.4, 3.4, 3.4 and 3.4 describe the real-world environments, as well as the methods we use to algorithmically generate thousands of trajectories in OSM environments to create large-scale datasets for training.

**SAIL: A BENCHMARK DATASET FOR INSTRUCTION FOLLOWING**

SAIL, introduced in [95], is a navigation dataset containing route instructions annotated with trajectories for three different environments of varying size. An environment is composed of connected hallways with different floor patterns (grass, brick, wood, gravel, blue, flower, or yellow octagons), wall paintings (butterfly, fish, or Eiffel Tower) and objects (hat rack, lamp, chair, sofa, barstool, and easel) at intersections. In SAIL, the challenge of learning to ground natural language stems from the fact that instructions given by humans are complex, free-form and of variable length (either 3,266 single sentences in isolation or 706 full paragraphs). While this task and dataset bear superficial similarity to others [2, 31], the language and paths required here are quite different — the proportion of instructions to actions is much higher, the interpretation of language is highly compositional and instruction length varies widely. SAIL has therefore been the subject of focused attention in semantic parsing, resulting in a range of different approaches that attempt to plan in such settings.

**Open Street Maps: Instruction following in real world environments**

**Open Street Maps (OSM)**

We use Open Street Maps (OSM), a global open-sourced map API where users can add landmarks, as well as information about the landmarks that are then verified. This therefore gives us access to real world maps, names of landmarks and cartesian coordinates of their locations. We query this
semantic database to get maps for 10 cities across the US.

We chose cities across the US, in areas that have open spaces and interesting landmarks or statues that can provide referring expressions, e.g., around universities and parks. We provide detailed examples, lists of cities, and landmarks within cities in the supplementary material. Figure 6 shows examples of pictures of the map in 4 different possible environments.

**Converting Real World Maps to Underlying Graph Structure**

In order to convert dense OSM maps composed of (latitude, longitude) pairs of points into a graphical structure in which we can simulate paths, we use Voronoi cells \[ \] to partition the map. Given an OSM map composed of landmarks in a 300m radius square, the map can be partitioned into Voronoi cells as shown in Figure 7. This is done (as in the original Voronoi implementation) by randomly generating points inside the bounded region. Triangulation is done by connecting each node to it’s nearest neighbours, forming a network of triangles. The boundaries of Voronoi cells are then formed by connecting perpendicular bisectors of triangles. \[ \] shows that Voronoi-based enables faster and more efficient planning over larger distances.

**Algorithmic Simulation of Trajectories in Environments**

Most previous work (that is not concerned with large-scale trajectory data collection) hand-annotates trajectories in graphs. This collection of trajectories is then given to a human for an annotation task, thus obtaining paired (language, trajectory) data in an environment. When dealing with multiple large environments, with thousands of trajectories per environment, manually annotating trajectories quickly becomes infeasible.

Therefore, we instead simulate trajectories in the different environments using path-sampling algorithms. Specifically, given our environment graph that connects landmark nodes to one another, we sample start and end nodes \((s, e)\) from a uniform distribution. We use a \(k\)-th shortest path sampling algorithm (where \(k\) is sampled from a geometric distribution)
to obtain a path between between nodes $s$ and $e$. It is crucial to note that the paths obtained are therefore not always the shortest path between two nodes — a bias that can be quickly learned by models, that has been shown [67] to make them ignore important components of the input (e.g., if agents are trained on paths that are always the direct shortest path between two nodes, there is no need for them to condition on the natural language instruction, environment features and so on). In this way, we obtain up to 3,000 algorithmically simulated trajectories for each environment, distributed over different start and end nodes. We ensure that the lengths of the paths vary (i.e., are not all the same length) and that all landmarks in the graph have been covered in a subset of the trajectories.
Collecting Natural Language Annotations for Environments

Once we can algorithmically simulate thousands of trajectories in different environments, we collect natural language instruction data for trajectories using Amazon Mechanical Turk (AMT). Each AMT task consists of a picture of the real-world OSM map, as well as two trajectories drawn in different colours (blue and orange) on the graph of connecting landmarks. The two different trajectories are obtained from sampling from the k-th shortest paths, and we instruct workers to give a natural language instruction that describes one of the paths while specifically excluding the other as a possibility. This therefore elicits language instructions specific to one of the trajectories, ensuring that they refer to elements along the target path. Moreover, sampling from the k-th shortest trajectories ensures that the paths in our data are not always shortest paths from the start to the end, a heuristic that neural models have been shown to easily game [84]. We also provide a map key with the list of landmarks in the map, to make the task easier for workers. For each trajectory, we collect 3 different natural language commands, thus giving us an average of 2,884 trajectories per environment, for 10 different environments. To the best of our knowledge, this is the largest collection of temporal data aimed at LTL, for real-world...
Algorithm 2 Algorithmically simulating trajectories

1: Initialise number of paths required
2: for each OSM map $M$ do
3:     $V \leftarrow \text{Vorono}$($M$) .. voronoi diagram of landmarks
4:     $G \leftarrow \text{Landmarks}(V)$ .. graph with nodes as landmark centers, edges as distances between landmarks
5:     for $i$ in $\text{num\_paths}$ do
6:         $s \leftarrow \text{random}(G.\text{nodes})$ .. randomly sample a start node for path
7:         $e \leftarrow \text{random}(G.\text{nodes})$ .. randomly sample an end node for path
8:         $k \leftarrow \text{random}($$\text{geometric}(p=0.2)$$)$ .. sample $k$ from geometric distribution
9:         $\text{paths} \leftarrow \text{dijkstra}(s, e)$ .. find paths between $s$ and $e$ from shortest to longest
10:        $\text{path\_1} \leftarrow \text{paths}[k]$ .. choose the $k^{th}$ shortest path as the ground truth path
11:        $\text{path\_2} \leftarrow \text{random}($$\text{paths}$$)$ .. randomly select an alternative path different from the above
12:     end for
13: end for

environments.

3.5 EXPERIMENTAL EVALUATION

In this section we explain the environments and evaluation metrics, that attempt to evaluate not only the final goal location, but the entire path. To compare to previous instruction-following models on the benchmark set, we report goal-state accuracy on SAIL, but also propose to evaluate path metrics, to ensure that the entire trajectories are correct. We compare to all past models that evaluated on SAIL, as reported for the test set. To evaluate the real-world OSM environments, we report goal-state accuracy as well as path accuracy, comparing to a baseline that does not use LTL.
**Evaluation Metrics**

To evaluate models on the SAIL dataset, previous works compare the agent’s end state to a labelled state $s'$ i.e., the end point of the ground-truth trajectory, for all (single and multi-sentence) instructions in the test set. As explained in this section, we propose additional evaluation metrics to compare the entire path rather than just the goal location.

To directly compare our approach with existing work, we measure the *goal-state* accuracy i.e., the ability of the model to reach the same location in the environment as the ground-truth trajectory. Unlike prior work, we also propose a more fine-grained analysis to evaluate *path accuracy*. Often times, the path taken to reach the final goal location is crucial – especially when the instruction specifies constraints on how to reach the goal. A more specific analysis of the entire path is even more important in complicated environments with several possible paths that reach the goal. Metrics that only measure success rate (i.e., reaching the final goal location) and disregard the path taken will therefore not distinguish between such paths. We therefore propose to evaluate correctness of the produced paths in comparison to the ground-truth path. To evaluate the sequences, we treat paths as vectors of indices in the grid world and compute precision, recall, accuracy and edit distance to the gold path. Paths that more closely follow the required constraints will therefore have higher precision, recall and accuracy, and a lower edit distance.

**Results**

**Comparison to prior language grounding work in SAIL**  We compare our model to existing approaches that report final goal-state accuracy on the SAIL dataset. Most similar to our approach is the model from [8] that trains a CCG semantic parser supervised by trajectories. Other methods include algorithms supervised with logical forms [35, 36] that learn semantic parsers for instructions as well as ones that involve strategies for online learning of lexicons [33] and ones that use contextual information [34] for better language understanding. We also compare to the supervised alignment-based models [4] that build grounding graph
representations to execute instructions and neural sequence-to-sequence models [98] that translate language to actions in the environment.

Table 2: Evaluation of systems on the SAIL dataset.

<table>
<thead>
<tr>
<th>System</th>
<th>Single</th>
<th>Multi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chen and Mooney (2011)</td>
<td>54.4</td>
<td>16.18</td>
</tr>
<tr>
<td>Chen (2012)</td>
<td>57.28</td>
<td>19.18</td>
</tr>
<tr>
<td>+ additional data</td>
<td>57.62</td>
<td>20.64</td>
</tr>
<tr>
<td>Kim and Mooney (2012)</td>
<td>57.22</td>
<td>20.17</td>
</tr>
<tr>
<td>Artzi and Zettlemoyer (2013)</td>
<td>65.28</td>
<td>31.93</td>
</tr>
<tr>
<td>Andreas and Klein (2015)</td>
<td>59.60</td>
<td>-</td>
</tr>
<tr>
<td>Mei et. al. (2017)</td>
<td>71.05</td>
<td>30.34</td>
</tr>
<tr>
<td>Ours</td>
<td>66.92</td>
<td>20.17</td>
</tr>
</tbody>
</table>

Performance in SAIL. Table 2 shows goal-state accuracy of systems on SAIL, while Table 3 shows evaluation metrics for paths produced, compared to ground-truth paths in the dataset. Table 2 compares to other work that is different in the form of supervision provided and model architecture used, but evaluates on the same end-task i.e., goal-state accuracy over the SAIL dataset. We see that our model has comparable performance to previous models in terms of navigating to the correct goal location. However, this metric can still reward incorrect paths (e.g., ones that violate specified constraints) as long as they end up in the correct final location. In Table 3 we compare to the best-performing model in terms of path accuracy. We see that our model outperforms the previous best-performing model when we evaluate the entire path taken. We furthermore analyse the type of natural language instructions that involve complex temporal constraints. We do this by taking all instructions in the dataset that contain natural language counterparts of temporal operators (e.g., until, eventually, finally, always..) and evaluate models specifically on this subset. These temporal sentences form 20% of the data (476) sentences. We see that our model
that grounds to temporal logical form outperforms previous models under this finer-grained evaluation.

Table 3: For path evaluation, we evaluate precision, recall, accuracy (higher is better) and edit distance (lower is better).

<table>
<thead>
<tr>
<th>Data System</th>
<th>Prec.</th>
<th>Recl.</th>
<th>Acc.</th>
<th>ED</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Seq2seq</td>
<td>31.6</td>
<td>30.33</td>
<td>91.5</td>
<td>3.25</td>
</tr>
<tr>
<td>(SAIL) Ours</td>
<td><strong>33.5</strong></td>
<td><strong>32.34</strong></td>
<td><strong>93.7</strong></td>
<td><strong>2.24</strong></td>
</tr>
<tr>
<td></td>
<td>+1.9</td>
<td>+2.01</td>
<td>+2.2</td>
<td>-1.01</td>
</tr>
<tr>
<td>Temporal Seq2seq</td>
<td>31.2</td>
<td>30.19</td>
<td>91.2</td>
<td>3.91</td>
</tr>
<tr>
<td>(SAIL) Ours</td>
<td><strong>34.3</strong></td>
<td><strong>35.2</strong></td>
<td><strong>94.5</strong></td>
<td><strong>1.27</strong></td>
</tr>
<tr>
<td></td>
<td>+3.1</td>
<td>+5.01</td>
<td>+3.3</td>
<td>-2.64</td>
</tr>
</tbody>
</table>

Table 4: Evaluation on all environments. The first number shows goal-state accuracy (higher is better) while the second shows edit distance compared to the ground-truth trajectory (lower is better).

<table>
<thead>
<tr>
<th></th>
<th>Austin, TX</th>
<th>Ann Arbor, MI</th>
<th>Atlanta, GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours</td>
<td>89.4 / 1.1</td>
<td>84.3 / 1.9</td>
<td>77.6 / 1.6</td>
</tr>
<tr>
<td>Baseline</td>
<td>66.7 / 3.4</td>
<td>78.4 / 3.0</td>
<td>74.9 / 3.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boston, MA</td>
<td></td>
<td>69.8 / 3.1</td>
<td>88.5 / 1.1</td>
</tr>
<tr>
<td>Cambridge, MA</td>
<td>60.3 / 3.3</td>
<td>52.3 / 3.4</td>
<td>76.5 / 2.6</td>
</tr>
<tr>
<td>New Haven, CT</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Comparison to baseline in real-world OSM environments  We now turn to the real-world environments with the newly-collected complex lan-
Table 5: Evaluation on all environments. The first number shows goal-state accuracy (higher is better) while the second shows edit distance compared to the ground-truth trajectory (lower is better).

<table>
<thead>
<tr>
<th></th>
<th>Baltimore, MD</th>
<th>Berkeley, CA</th>
<th>SAIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours</td>
<td>74.5 / 1.4</td>
<td>80.3 / 1.5</td>
<td>66.9 / 2.1</td>
</tr>
<tr>
<td>Baseline</td>
<td>68.3 / 3.5</td>
<td>70.3 / 4.3</td>
<td>74 / 2.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Philadelphia, PA</th>
<th>Providence, RI</th>
<th>SAIL (T)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours</td>
<td>89.9 / 1.8</td>
<td>76.6 / 1.2</td>
<td>74.3 / 1.1</td>
</tr>
<tr>
<td>Baseline</td>
<td>63.5 / 3.9</td>
<td>74 / 2.2</td>
<td>70.1 / 4.3</td>
</tr>
</tbody>
</table>

language commands. To evaluate the generalisability of our approach in multiple real-world environments with complex language, we evaluate performance in each of the new environments, after grounding natural language instructions to logical forms, and then planning with logical forms in each environment. We compare to a baseline that does not use LTL. This model takes in a natural language utterance and predicts a final goal location from the set of elements in the environment; therefore not considering all the sequential constraints. The planner then finds a path from the start location to the goal. This model is a strong competitor in instances that do not require temporal reasoning (e.g., to directly go to one goal location without needing to meet path constraints), however for complex, sequential tasks, this does not account for temporal ordering and intermediate tasks. We use a Multi-Layer Perceptron (MLP) classifier, that encodes the natural language instruction and predicts the final goal-location (i.e., landmark) from the set of all landmarks. Each training sample of language and trajectory pairs therefore gives us a training sample of language paired with the correct end-goal location, which is obtained by simply taking the last point of the ground-truth trajectory. We use a cross-entropy loss against the correct goal-state and train this model for each environment. A model that correctly predicts the goal can therefore achieve perfect goal-state accuracy when this is given to a planner in the environment. However, this does not give us any guarantees on the path taken. This baseline model
therefore has the same number of training samples as the weakly supervised LTL parser, but performs a simple prediction task, rather than a semantic parsing task composed of operators and operands.

**Performance in OSM**  Tables 4 and 5 show goal-state accuracy as well as path metrics in each OSM map. Each test set is composed of a 100 different unseen trajectories with natural language commands of varying length. We compute goal-state accuracy by evaluating whether or not the final location after planning is the correct end location of the trajectory. We compute path accuracy by computing the edit distance between the computed path and the ground-truth trajectory. We report the average over all samples in each test set. We see that our model outperforms the non-LTL baseline on the path metric of edit distance (lower is better), since reasoning temporally enforces meeting all the intermediate constraints, therefore allowing the planner to find a path that matches the ground-truth path. We also see that goal-state accuracy is higher. This is because there exist several cases where the ordering of referents in the natural language statements can be reversed (e.g., both sentences “go to B and then finally to A” “go to A after going to B” have the same meaning, but in the latter, the location A is referred to before the location B in the natural language sentence, thus causing the final goal location to be predicted incorrectly).

**Demonstration in Simulation**  To demonstrate the working of our entire pipeline in real time, we demonstrate the execution of natural language instructions in simulation. This can then be connected to the Skydio R1 drone, that navigates over actual landmark coordinates in any of the real-world environments. Due to several drone navigation restrictions, we demonstrate this only inside the simulator, however navigating over the actual real-world environment coordinates.

Specifically, for our demonstration video, we sample a different human-given natural language instruction (that was never used during training) in each of the 10 environments. In real-time, we parse these with the weakly-supervised model trained in the environment, plan with the produced LTL specification, and show the drone navigating over the planned path. A
Figure 9. Example still from our robot demonstration in simulation with a Skydio drone.

video of the demonstration attached with this paper, shows the Skydio drone in simulation following these commands in different environments, demonstrating the entire end-to-end process. All of these environments follow our weakly supervised approach described above. This means, that this demonstration and real-time usage can be applied to any new environment in which we collect natural language annotations for trajectories, therefore allowing the model to train and then produce LTL representations that the planner needs.

3.6 Conclusion

In this chapter we use a weakly supervised semantic parsing model to ground natural language to temporal logical form — a formal language that allows handling of complex, temporal events that are typically unable to be handled by most (traditionally Markovian) methods. We evaluate on both artificial (SAIL) and real-world (OSM) environments. While the SAIL navigation dataset was not specifically constructed with these temporal, sequential constraints in mind, the fine-grained evaluations show our
method allows dealing with these naturally occurring constraints better than previous state-of-the-art methods. Moreover, in our newly collected OSM dataset that contains more complex, temporal language, we show that our method that can deal with temporal order, has superior performance.
Planning with State Abstractions for Non-Markovian Task Specification

This chapter is based on the paper "Planning with State Abstractions for Non-Markovian Task Specification" [106] published at RSS 2021, led by Yoonseon Oh, and co-authored with Thao Nguyen, Baichuan Huang, Stefanie Tellex and Ellie Pavlick.

In the previous chapter we saw how we could translate natural language into LTL that allows agents to follow temporal constraints that other language cannot easily represent. Apart from temporal language, there are also instances where language is used at different levels of abstraction. For example, an instruction to a robot can be expressed in terms of high-level goals (such as "fly to the end of the first floor"), lower-level specifications (such as "fly east, go south, go south and fly east again"), or mixed-level (such as "go to the yellow room and the second floor"). Language can also express explicit constraints on the path taken to reach the goal (for example, "fly to the red room first, without going through the green room."). The former category of commands requires an agent to fluidly move within an abstraction hierarchy (that is, knowing that a floor is at a higher level than individual rooms and directions), while the latter command restricts the space of possible paths that can be taken and sometimes induces temporal constraints on the order in which goals can be visited. It is crucial for robot systems to portray an adequate understanding of such commands, coupled with the ability to efficiently execute the underlying task.

Given an environment, a goal condition and constraints, robots can use planning to reach goal conditions while satisfying constraints. Existing approaches interpret language by mapping to a reward function in a Markov Decision Process (MDP) [93]. However, these models very quickly become
intractable as the state space of the world grows larger [57, 77]. Planning with abstractions in a hierarchical structure [57, 77, 78, 129], either by using an Abstract Markov Decision Process (AMDP) [57] or with options [77, 78, 129] can allow reduction of the state space. There has been previous work in interpreting natural language task specifications at different levels of spatial abstraction and planning using AMDPs [10]. Separately, as shown in Fig. 10, non-Markovian natural language commands can be mapped to linear temporal logic (LTL) formulae [21, 48, 79, 88] to allow efficient planning with an MDP, given the corresponding LTL task specifications [42, 43, 52, 56, 70, 117, 144]. Combining the interpretation of language using a hierarchical structure and the mapping of commands to LTL expressions is non-trivial, as the non-Markovian constraints might span different levels of abstraction. Plans in a more abstract state space could therefore lead to failure of constraints specified in a less abstract space (that is, plans at a lower level in the abstraction hierarchy).

In this chapter, we introduce the Abstract Product MDP (AP-MDP) framework to combine the benefits of LTL and AMDP, thus enabling a
robot to interpret non-Markovian commands at different levels of abstraction. There is previous work in planning for LTL tasks using options [90]. However, the AMDP approach suits our task better, as its hierarchical structure closely resembles the hierarchies formed by humans when planning to solve complex tasks that can be decomposed into subtasks [57].

In our approach, task specifications are first given as natural language utterances that are then translated into LTL expressions by a supervised neural sequence-to-sequence model. This LTL expression $\phi$ is converted into a finite state representation that accepts infinite inputs, or a Buchi automaton [30]. This representation allows us to decompose the problem into sub-problems (organized around sub-parts of the input LTL expression). Edges of the Buchi automaton consist of atomic propositions in expression $\phi$ and a sub-problem induces a state transition of the automaton. To further deal with different levels of abstraction, if atomic propositions in the same edge are from different levels, we solve the sub-problem using the lower level AMDP. The robot must then forgo the computational benefits of the AMDP to guarantee that the policy satisfies all the constraints present in the LTL expression. This entire pipeline (shown in Fig. 11) therefore fluidly allows complex task specifications with non-Markovian constraints to be specified using natural language and solved at different levels of the goal hierarchy.

We evaluate our approach by reporting the performance of AP-MDP in simulation and on a drone platform. We also present a new corpus of non-Markovian natural language commands at different levels of abstraction, a neural sequence-to-sequence model that translates human-uttered natural language commands to their corresponding LTL counterparts, and demonstrates the solving of complex natural language task specifications using AP-MDP on a drone.

Related Work LTL has been used to model agent behavior in planning problems with non-Markovian task specifications. Consider a task that requires an agent to visit regions of interest in a specific order (for example, “visit the red room first, then the blue room, and the green room last”). These kinds of expressions have intrinsic temporal information that must be taken into account when determining the kind of path that has to be
Figure 11. Complete pipeline for the translation of a natural language instruction to an LTL formula, then to a Buchi automaton, and to a plan that gives us action sequences to correctly reach the goal location specified by the task.

taken to achieve the goal. LTL allows us to formally describe these kinds of task specifications as logical functions, thus allowing robots to then execute these behaviors.

Decision making with an MDP often becomes intractable as the size of the state space increases. In order to overcome intractability, hierarchical frameworks [57, 77, 78, 83] are commonly used. The options framework [77, 78], for example, models temporally abstract macro-actions as options that can be adopted to build abstraction hierarchies. Similarly, AMDPs [57] can be used for abstraction by decomposing tasks into series of subtasks, thus allowing planning to take place more efficiently. However, these methods do not address the problem of solving LTL specifications with abstractions.

Hierarchical frameworks are powerful when an agent is faced with the task of planning a sequence of actions for complex LTL tasks. Several works [37, 46, 97, 105] propose incorporating both the robot dynamics and the given LTL constraints in a continuous space. A continuous state space can be abstracted into a discrete state space and a continuous path is derived by sampling guided by the high-level discrete plan [37, 97, 105]. Other works have focused on grounding natural language to LTL expressions [21, 56, 88].
to further allow a robot to make use of these LTL specifications. Previous work in hierarchical planning using options can accelerate planning for LTL tasks \cite{90}. However, the AMDP framework \cite{57} is better suited for our task than options, by virtue of encoding a goal hierarchy rather than learning a policy over goals.

To the best of our knowledge, this work is the first to propose a hierarchical framework for planning for LTL tasks using the structure of an Abstract MDP. An AMDP provides abstract states, actions, and transition dynamics in multiple layers above a base-level MDP, thus decomposing problems into subtasks with local rewards and local transition functions for policy generation. Moreover, as shown in our robot demonstrations, we start from human input given in the form of speech that is then converted to text. This textual input of the natural language command is translated to its LTL representation, and atomic propositions are directly mapped into propositions in each layer of a multi-level AMDP. We can then plan at levels higher than the lowest level whenever possible, and find a policy in a more efficient way than previous approaches.

4.1 Problem Formulation for State Abstraction

We consider a planning problem for a robot, when the task that the robot is required to interpret and solve is given through a natural language command. Our environment is a 3D grid world consisting of three floors as shown in Fig. 10. Each floor is composed of colored rooms, a room is composed of a set of grid cells, a landmark (such as a charging station) indicates a cell at position \((x,y,z)\). Landmarks (or cells) are therefore the lowest level of abstraction, rooms are abstract expressions of landmarks, and floors are abstract expressions of rooms and form the highest level of abstraction. A natural language command (such as \textit{"first go to the red room through landmark 1 and then go to the blue room."}) is given to the robot by virtue of observable visual elements in our abstraction hierarchy (landmarks, rooms, and floors). This natural language utterance is grounded to its LTL counterpart \((F (landmark_1 \land F (red\_room \land F (blue\_room))))\) which forms the task specification. The agent is required to accomplish
the task by correctly finding a path to the correct location and following the determined path by executing a sequence of actions from the action set (north, south, east, west, up, down).

We formulate this problem as an MDP that gets a high reward when the task is accomplished. Crucially, we make use of abstractions over the MDP state space for more efficient planning in large environments, and for the robot to efficiently find policies for commands at different levels of abstraction. Consider the example task above of “first go to the red room through landmark 1 and then go to the blue room.” This is an expression that spans different levels in the abstraction hierarchy (that is, rooms and landmarks) and can be translated into its equivalent LTL formula $\phi$ over atomic proposition sets $AP^L$ for each level $L$ in the hierarchy. For example, “landmark 1” occupies one grid cell in the environment and corresponds to an atomic proposition (denoted by $\alpha^0_0$) in $AP^0$ and “red room” and “blue room” correspond to atomic propositions (denoted by $\alpha^1_0$ and $\alpha^1_1$, respectively) in $AP^1$. The expression can be translated into $\phi = F(\alpha^0_0 \land F(\alpha^1_0 \land F\alpha^1_1))$ using the LTL operator $F$ or “finally”, converted to a Buchi automaton [30, 48], and then an AMDP [57] to decompose the problem into a series of smaller, and hence easier to solve, subproblems. Section 4.2 defines LTL and the variants of MDPs that our model relies on, while section 4.2 goes over how they are composed together to produce a more efficient solution, while describing the end-to-end pipeline with the natural language grounding components.

4.2 State Abstractions and Temporal Logic

This section defines the components used in our formulation and how they are transformed into one another to form state abstractions for complex, non-Markovian task specifications uttered by humans through natural language. We briefly introduce LTL and its syntax, explain the transformation of an LTL expression to a Buchi automaton and further to an MDP.

**Linear temporal logic** We refer the reader to Definition 3.1 for a comprehensive definition of LTL. The grammatical syntax is as follows. $\phi ::= $
\[ \pi | \neg \phi | \phi \land \varphi | \phi \lor \varphi | \mathcal{G} \phi | \mathcal{F} \phi | \phi \mathcal{U} \varphi \],
where \( \phi \) is the task specification or path formula, \( \phi \) and \( \varphi \) are both LTL formulae, \( \pi \in \Pi \) is an atomic proposition, \( \mathcal{F} \) denotes "finally", \( \mathcal{G} \) denotes "globally" or "always", \( \mathcal{U} \) denotes "until", and \( \neg, \land, \lor \) denote logical "negation", "and" and "or".

**Linear temporal logic to Buchi automaton** An LTL formula intuitively expresses properties over trajectories or traces (a sequence of sets of atomic propositions) in the environment. This can be translated into an equivalent Buchi automaton [30] — a deterministic automaton, that differs from the general notion of automata in that it accepts infinite traces represented by the input LTL formula. This handling of infinite traces is specifically necessary in cases of complex non-Markovian task specifications that can map to potentially unbounded action sequences.

We refer the reader to full definitions of Buchi automaton, labeled MDPs, product MDPs, abstract MDPs as defined in the original paper [106].

**Technical Approach** At a high level, we use a neural sequence-to-sequence model to convert an English command to the corresponding LTL expression, which is then translated to a Buchi automaton and then levels of the component AMDP to enable the robot to infer a policy based on the expression. We run a simulation that shows the produced action sequence, executable by a drone in a 3D environment.

**Abstract Labeled Markov Decision Processes** We propose Abstract Labeled MDPs (AL-MDPs) that decomposes an MDP \( \mathcal{M} \) into multiple abstract labeled MDPs which are based on abstract states, actions, and transitions in multiple layers. The labeled MDPs in the lowest level, the \( i \)th level, and the highest level are denoted by \( \hat{\mathcal{M}}^0, \hat{\mathcal{M}}^i, \) and \( \hat{\mathcal{M}}^L \), respectively. The abstract labeled MDP \( \hat{\mathcal{M}}^i \) is defined below:

\[
(\text{Abstract Labeled MDP}): \hat{\mathcal{M}}^i = (\hat{S}^i, \hat{A}^i, \hat{T}^i, \hat{s}_0^i, AP, R^i), \]
\[ \text{where } \hat{S}^i, \hat{A}^i, \hat{T}^i \]
\[ \text{and } R^i \text{ are a set of states, a set of actions, a transition function, and a reward function, respectively. States in } \hat{\mathcal{M}}^i \text{ correspond to a combination of} \]

atomic propositions in $AP$ by the labeling functions $i : \hat{S}^i \rightarrow 2^{AP}$. The set of atomic propositions $AP$ is a union of $L$ disjoint sets $AP^i$s, where $AP^i = \{\alpha^i_0, \cdots, \alpha^i_n\}$ (that is, $AP = \bigcup_{i=1}^{L} AP^i$). The proposition $\alpha \in AP$ belongs to $AP^i$, where $i$ is the largest value which satisfies that there exists a state $s \in \hat{S}^i$ which can determine the truth value of $\alpha$.

**Abstract Product MDPs**

We propose *Abstract Product MDPs* (AP-MDPs) which combine AL-MDPs and DBAs to solve ordinary product MDPs efficiently. We furthermore show how our approach handles a combination of atomic propositions in multiple levels. For example, if some of the atomic propositions are defined at level 0, we cannot guarantee that a plan derived at level 1 or level 2 will satisfy level 0 constraints. This would require working at the lowest level of atomic propositions, thus losing the computational benefit of abstraction and a reduced state space. In all previous hierarchical approaches in this area, when atomic propositions of different levels exist together, the product MDP must be solved at the lowest level (level 0 in this case) to guarantee the satisfaction of the transition constraint that directly affects it. This therefore does not afford the computational benefit of planning at higher levels using AL-MDPs. Our approach, however, employs different depths of AL-MDPs by decomposing the product MDP into subproblems to benefit from the hierarchical structure when the LTL task includes atomic propositions at the lowest level.

AP-MDPs combine the automaton $B$ of the LTL task specification with AL-MDPs. This involves taking an LTL formula in the form of an automaton, converting it to a labeled MDP and decomposing this MDP into several subproblems, each of which are individually solved at the required level of abstraction. We use a running example, as shown in section 4.2 to highlight the process of how decomposed subproblems are solved for the task specification in question. Section ?? then explains how any problem can be decomposed into component subproblems and section 4.3 presents the pseudocode for the algorithm for this process. The language grounding component of the system is discussed in 4.4 and finally, 4.4 describes the
functioning of the end-to-end system.

**An Example Problem**

Consider the example in Fig. 12. This figure shows the DBA for the LTL task specification $\phi = F(\alpha_0^0 \land F(\alpha_0^1 \land F\alpha_1^1))$ and we can see that the atomic proposition $\alpha_0^0$ is in level 0 of the abstraction hierarchy, while $\alpha_0^1$ and $\alpha_1^1$ are in level 1. To deal with these different levels in the abstraction hierarchy, we decompose the entire problem into different subproblems. The first subproblem $\hat{\mathcal{M}}_0$ is defined by a tuple $\hat{\mathcal{M}}_0 = (\hat{S}_0, \hat{A}_0, \hat{T}_0, s_0^0, AP, R^0)$ and here the agent wants to go to $q_1$ while not visiting other states in the DBA. The condition to reach the desired state, $f(q_0, q_1, s, s') = true$ is its goal condition and the condition to stay in the current state, $f(q_0, q_0, s, s') = true$ is its stay condition, where $s$ and $s'$ are the current state and the next state, respectively.

![Figure 12. Deterministic Buchi automaton. Atomic propositions in yellow circles correspond to those in level 0 and atomic propositions in green circles correspond to those in level 1. The transitions of the automaton refer to constraints over the propositions that are satisfied on taking that path.](image)

The function $f$ returns *true* or *false* depending on whether the logical expression on the edge is satisfied by the state. The reward function ensures
that the agent gets a large positive reward if the goal condition is satisfied and gets a large negative reward if the stay condition is violated and the goal condition is not satisfied. In all other cases, it gets a small negative reward as the time taken increases. Since this subproblem contains atomic propositions at level 0, we can solve it at level 0, that is, the lowest level of atomic propositions.

We now consider the latter part of the decomposition, that is, the second subproblem $\hat{M}_1$. This has atomic propositions related to level 1, therefore $\hat{M}_1$ can be formulated at a higher level of abstraction, that is, level 1 ($M_1 = (\hat{S}^1, \hat{A}^1, \hat{T}^1, \hat{s}_{0^1}, AP, 1, R^1)$), allowing for more efficient planning over a smaller state space. In this way, all subproblems $\hat{M}_i$ can be solved at the desired level to allow for full use of the benefits of abstraction where possible.

### 4.3 Algorithm for Abstract MDPs

The entire algorithm is presented as pseudocode in Algorithm 3. The input task is specified as an LTL expression composed of atomic propositions in the environment and the logical operators defined previously. We translate the LTL formula into a DBA using an existing package called Spot2 (line 4) [45]. Note that the DBA may contain infeasible edges because the translator does not consider the real environment (for example, if the $\text{red\_room}$ does not exist on the $\text{first\_floor}$ in a particular gridworld, $\text{red\_room} \land \text{floor\_1}$ cannot be true). We handle this by eliminating edges which have contradictions consisting of a logical incompatibility between two or more propositions (line 5), based on specifications of the environment in question. We check the contradiction by looking at the truth table of the formula.

We then find all possible paths from the initial state to the accepting state in line 6. The AL-MDPs goal and stay conditions are defined through lines 11 to 14, and we then obtain the optimal policy and plan of AL-MDPs with a solver of the AMDP (lines 15-16). We then select the best plan which has the minimum number of actions (lines 20-22).
Algorithm 3 Solve AP-MDPs

LTL task $\phi$ and $s_0$ are given
Initialize the optimal plan, $(s_{seq}, a_{seq})^\ast$.
Initialize the length of the optimal plan, $l^\ast$.

$A \leftarrow \text{LTL2DBA}(\phi)$

$A.\text{RemoveContradiction}()$

$\text{Paths} = A.\text{FindPaths}()$

for $\rho_i \in \text{Paths}$ do

Initialize $s_0$

Initialize the plan $(s_{seq}, a_{seq})^{\rho_i}$

for $j$ in $\{0, \cdots, n_i - 1\}$ do

goal condition $\leftarrow f(\hat{q}_j^i, \hat{q}_{j+1}^i, s, s') = \text{true}$

stay condition $\leftarrow f(\hat{q}_j^i, \hat{q}_{j}^i, s, s') = \text{true}$

$
\ell_j \leftarrow \text{the lowest level of atomic propositions in goal and stay conditions.}$

$\hat{M}_j \leftarrow (\hat{S}^{\ell_j}, \hat{A}^{\ell_j}, \hat{T}^{\ell_j}, \hat{s}_0^{\ell_j}, AP, \ell_j, R^{\ell_j})$.

$\pi \leftarrow \hat{M}_j.\text{Solve}()$

$ss, aa \leftarrow \hat{M}_j.\text{Plan}(\pi, s_0)$

$(s_{seq}, a_{seq})^{\rho_i} \leftarrow (s_{seq}, a_{seq})^{\rho_i} \cup (ss, aa)$

$s_0 \leftarrow s_{seq}(\text{end})$

end for

if $\text{length}(s_{seq}) < l^\ast$ then

$(s_{seq}, a_{seq})^\ast \leftarrow (s_{seq}, a_{seq})^{\rho_i}$

end if

end for

---

4.4 Grounding language to LTL formulae

We train a neural sequence-to-sequence model to translate natural language commands to LTL expressions. We discuss our language corpus and the model architecture below.
Figure 13. Examples, left and right, tested in simulation. In each example, a natural language instruction is converted to an LTL expression, then to a corresponding AP-MDP to find a policy. An agent then executes the policy in the specified environment to reach the correct goal state through the desired path.

Corpus

We use Amazon Mechanical Turk (AMT) to collect non-Markovian natural language commands that also refer to elements in the environment at different levels of abstraction. AMT workers were shown images representing correct and incorrect ways for the robot to complete a task, and asked to give commands that accurately capture the robot’s correct behavior. 810 natural language commands were collected from 120 AMT workers for 27 LTL formulae. We augment these 810 commands to obtain 6185 commands for 343 LTL expressions. Augmentation is done by mapping one training sample (for example, “go to the red room” accompanied by $F(red\_room)$) to similar commands and corresponding LTL expressions for every other possible goal locations. We held aside 20% of the data as the test set to evaluate model performance and trained on all remaining data and perform 5 fold cross-validation in this manner.

1The corpus can be found at https://github.com/h2r/ltl-amdp
Sequence-to-sequence model  As in Gopalan et al. [56], we use a neural sequence-to-sequence model composed of a recurrent neural network (RNN) encoder and decoder to translate each natural language instruction to an LTL formula. It is implemented in PyTorch [108] and trained for 10 epochs over our corpus, with a learning rate of 0.001 using the Adam optimizer [74]. We used a dropout of 0.8 as a regularizer [124].

Planning for an LTL task  Once a natural language command is translated into an LTL formula, it is then converted into a Büchi automaton with multiple paths from the initial state to the accepting state. Each path is represented and solved with an AL-MDP.

Experiments  In this section, we show that our method efficiently generates plans for complex LTL tasks. We evaluate efficiency with the number of backups and the computation time over 100 tasks. We successfully applied the proposed method on a drone.

Environment Setup  For simulations, we consider two 3D grid worlds (\(E_1\) and \(E_2\)) of size \(6 \times 4 \times 3\) and \(30 \times 20 \times 6\), respectively. The smaller world \(E_1\) has three floors, each comprised of six rooms, each the size of \(2 \times 2\) grid cells. The larger world \(E_2\) has six floors, each comprised of six rooms of size \(10 \times 10\). The visually observable elements (grid cells, rooms and floors) form the atomic propositions of the LTL task specifications. Importantly, these elements span different levels of abstraction: landmarks (grid cells) are at level 0, rooms are at level 1, and floors are at level 2. While our simulation environments consist of at least three floors, our robot demonstration is performed in a gridworld with only two floors for compatibility with the maximum height our PiDrone can reach.

Examples in simulation  We consider the tasks below to demonstrate example simulations of our proposed method. We show the language command with the corresponding LTL task specification, the automaton of the LTL expression, and the path found by our proposed approach for each
example. This highlights how our method solves a given task while satisfying the constraints of the task. The tasks in question exhibit the complex constraints with non-Markovian nature and varying levels of abstraction as outlined above. They contain propositions at different levels in the abstraction hierarchy, and contain temporal order constraints by specifying certain subtasks that should be performed before others. The two tasks are:

1. $\phi_1 = \mathcal{F}((\text{floor}_2 \lor \text{red\_room}) \land \mathcal{F}(\text{floor}_1))$
   
   ("First either go to the second floor or the red room, and then go to the first floor")

2. $\phi_2 = \mathcal{F}(\text{floor}_2 \land \mathcal{F}(\text{green\_room}))$

   ("Go to the green room after entering the second floor")

The execution of both tasks is shown in Fig. 13. The process to solve task $\phi_1$ for the given LTL task specification is outlined in the left side of the figure. Upon decomposing this task specification as in our proposed method, there are two paths of automaton states. Consider the path $\rho_0 = q_0q_2$ corresponding to the AL-MDP $\hat{\mathcal{M}}_0$. This has a goal condition of $((\text{red\_room} \land \text{floor}_1) \lor (\text{floor}_2 \land \text{floor}_1))$ and a stay condition of $(\neg \text{floor}_2 \land \neg \text{red\_room})$. For the path $\rho_1 = q_0q_1q_2$, there are two AL-MDPs $\hat{\mathcal{M}}_0$ and $\hat{\mathcal{M}}_1$, where $\hat{\mathcal{M}}_0$ has a goal condition of $((\text{red\_room} \land \neg \text{floor}_1) \lor (\text{floor}_2 \land \neg \text{floor}_1))$ and a stay condition of $(\neg \text{floor}_2 \land \neg \text{red\_room})$, and $\hat{\mathcal{M}}_1$ has a goal condition of $(\text{floor}_1)$ and a stay condition of $(\neg \text{floor}_1)$. Since we can satisfy $\phi_1$ with only two actions with $\rho_0$, the final solution is a plan for $\rho_0$.

For task $\phi_2$, there exists an infeasible path among paths in the automaton. The first AL-MDP in $\rho_0 = q_0q_2$ has goal and stay conditions of $(\text{floor}_2 \land \text{green\_room})$ and $(\neg \text{floor}_2)$, respectively. This problem does not have a solution because the green room is on the second floor, and thus our algorithm does not return a plan. There is, however, a solution for the path $\rho_1 = q_0q_1q_2$. The first AL-MDP has a goal condition of $(\text{floor}_2 \land \neg \text{green\_room})$ and a stay condition of $(\neg \text{floor}_2)$. The second AL-MDP has a goal condition of $(\text{green\_room})$ with a stay condition of $(\neg \text{green\_room})$. The planned path is shown in Fig. 13.
**Language grounding results**  We observe that the accuracy of the model drops on the held-out LTL commands. This problem of zero-shot generalization (specifically, the ability to generalize to samples unseen during training) has been widely studied [56, 75, 84] for neural sequence-to-sequence models that cannot handle compositionality and the ability of models to learn meaning representations for given natural language sentences [39]. We also observe cases where changes in word order affect the translated LTL output of the model. Consider the command “avoid the blue room until you go to landmark 1”, $\neg \text{blue\_room} \cup \text{landmark\_1}$ for example. Variations in our collected data include sentences like “until you go to landmark 1, always avoid the blue room” that change the ordering of referent words ($\text{blue\_room}$ and $\text{landmark\_1}$) which are occasionally confused, and mapped to incorrect expressions such as $(\neg \text{landmark\_1} \cup \text{blue\_room})$. However, in the drone demonstrations, the sequence-to-sequence model correctly translates the given language commands (converted from speech) into LTL task specifications that are then solved using our proposed method.

### 4.5 Robot Experiments

In addition to the simulations described above, we also test our proposed method on a drone. The PiDrone [27] is a quadcopter drone that is equipped with one downward-facing infrared sensor with a maximum range of 60cm to measure the drone’s altitude, and one downward-facing camera for localization over a textured surface. The drone’s flight space is a $3m \times 3m$ surface. We divide the space into a grid-based environment, as shown in Fig. ??, consisting of 2 floors, each with 9 rooms, and each room is a square made up of 4 cells (each cell is 50cm × 50cm). The action space for the drone in the grid-based environment is (north, south, east, west, up, down), where each action changes the drone’s location by 1 cell. We visualize the environment through mixed reality using a Microsoft HoloLens [36]. Colored rooms and landmarks (boxes each with the size of 1 cell) to aid path planning and specify goal positions were set up in a Unity3D virtual environment running on the HoloLens.

In our demonstration, the drone is given a natural language instruction
through speech. This is converted using Google’s speech-to-text, and then translated by our trained sequence-to-sequence model into an LTL formula to be solved by the AP-MDP framework in real time. The action sequence output by AP-MDP for the LTL expression is then used for the drone’s navigation. The natural language commands were: “Navigate to the red room”, “Avoid landmark two until you have been to the blue room”, “Move to the orange room then the purple room”, “Go to landmark three then go to the yellow room”. Video recordings of the drone demonstrations can be found at https://youtu.be/zjtMEGUmkd8.

4.6 Conclusion

This chapter introduces a novel approach to combine the handling of non-Markovian task specifications in large environments by grounding complex language to LTL expressions and then decomposing tasks within an abstraction hierarchy to plan efficiently at higher levels where possible. We show that planning with abstractions allows the robot to correctly reach the goal location more efficiently, in terms of computing time and backups required, in over 95% of tasks in a small environment and over 99% of tasks in a larger environment. We also show that this method of abstraction can handle LTL task specifications. Moreover, we present the largest existing dataset of natural language commands mapped to LTL expressions at different levels of abstraction. We demonstrate our approach with a PiDrone that navigates to the goal location along a correct path when given a human-uttered command.

While the language grounding model works fairly well to translate language to LTL formulae, it cannot fully handle expressions unseen during training and cannot always deal with simple changes in word-ordering and variations in the language. Future work in this direction can explore compositional models that can handle a wide range of expressions by learning to compose subparts together and then execute the required actions. Future work in the hierarchical setup can explore models that go beyond fixed hierarchies and state abstractions. If the AMDP transition hierarchies can be learned with model-learning methods on the fly, this will enable gen-
eration to unseen environments and the ability to handle and properly execute a plan for a wider range of commands.
PART III
5

On the Relationship Between Structure in Natural Language and Models of Sequential Decision Processes

This chapter is based on the paper "On the Relationship Between Structure in Natural Language and Models of Sequential Decision Processes" [112] published at an ICML workshop in 2020, co-led by Rafael Rodriguez, and co-authored with George Konidaris.

In Part II of this thesis, we saw how we could learn a model that translates natural language into temporal logic rewards that an agent can plan with. This communication of useful background knowledge to reinforcement learning (RL) agents is an important and effective method for accelerating learning, however, this communicated information might be relevant to different components of an agent’s model—for example, goals it must achieve or environment dynamics that it must learn about. Moreover, it is important to note that human language is distinguished by powerful semantics, rich structure, and incredible flexibility. It enables us to communicate with each other, thereby affecting the decisions we make and actions we take. While Artificial Intelligence (AI) has made great advances both in sequential decision-making using Markov Decision Processes (MDPs) and in Natural Language Processing (NLP), the potential of language to inform sequential decision-making is still unrealized. We explore how the different functional elements of natural language—such as verbs, nouns and adjectives—relate to decision process formalisms of varying complexity and structure. We attempt to determine which elements of language can be usefully grounded to a particular class of decision process and how partial observability changes the usability of language information. Our work show that more complex, structured models can capture linguistic concepts that
simple MDPs cannot. We argue that the rich structure of natural language indicates that reinforcement learning should focus on richer, more highly structured models of decision-making.

5.1 Why Sequential Decision Processes?

Artificial Intelligence (AI) is concerned with designing agents that exhibit intelligent behaviour. These agents are typically formulated as sequential decision-making processes: systems that perceive their environment via sensors, and then must select actions to maximise a utility function. Markov Decision Processes (MDPs) [113] are widely used to model such tasks, and many extensions have been proposed to model more complex situations by including richer structure.

Language is considered a hallmark of human intelligence—one of the key characteristics that sets us apart from other animals species. The use of language enables the transmission, storage, and evolution of knowledge for humans, and thereby supports sequential decision-making. However, human language is vastly complex. It is marked by semantics, pragmatics, rich syntactic structure, and levels of ambiguity, and the problem of having a computer understand (or generate) it fluently is still unsolved. Therefore, the integration of natural language and decision-making models is of particular interest to researchers aiming to create integrated, general-purpose intelligent agents.

Integrating language information and decision-making agents in the context of Reinforcement Learning (RL) has resulted in marked gains in performance [91]. However, there has been relatively little investigation into how the form and structure of the decision process modeled by MDPs—of which there are several classes modeling varying structure and complexity—might be reflected in the natural language appropriate for communicating with an agent solving one. We consider the following question: what parts of speech are appropriate when communicating about what classes of MDPs? As our tasks and models grow increasingly more complex, does the spectrum of language required also broaden? Similarly, does the complexity of natural language suggest that humans represent
their own decision-making processes using structure models?

5.2 Classes of Markov Decision Processes

In its simplest form, a Markov Decision Process (MDP) [113] is specified by the tuple \((S, A, T, R, \gamma)\), where \(S\) denotes a set of states, \(A\) denotes a set of actions the agent can take, \(T : S \times A \rightarrow \Delta(S)\) denotes a transition probability distribution that represents the probability of transitioning to state \(s' \in S\) when action \(a \in A\) is taken while in state \(s \in S\), \(R\) denotes a task-specific reward function and \(\gamma\) denotes a discount factor. An agent solving an MDP is typically tasked with finding a policy \(\pi\), mapping states to actions, that maximises the discounted cumulative rewards obtained over time: \(\sum_{t=0}^{\infty} \gamma^t r_t\), where \(r_t\) is the reward obtained at time \(t\). In hierarchical reinforcement learning [14], the agent is able to structure its policy to introduce higher-level, temporally abstract, actions, often called options [129].

MDPs model the special case where the agent is able to perceive, at every timestep, all the information it requires to decide which action to take. In the more general case, the agent’s sensors at each timestep only offer a limited view of the state of its environment. Partially Observable Markov Decision Processes (POMDP) [69] model this case by extending the MDP tuple to be \((S, A, T, R, \Omega, O, \gamma)\) where the set \(\Omega\) represents the set of observations that the agent can get and \(O : S \times A \rightarrow \Delta(\Omega)\) is the observation function such that \(O(s, a, z) = p(z|s, a)\), i.e. when the agent in state \(s \in S\) executes action \(a \in A\), \(O(s, a, z)\) is the probability of getting the observation \(z \in \Omega\); the agent never has direct access to \(S\). The remaining elements are defined as in the MDP case.

The basic MDP and POMDP formalisms are essentially unstructured: while they specify the form of the decision process, they do not impose any further structure or complexity on the form of the states, actions, and observations available to the agent. These formalisms have been extended to describe more structured decision processes. We consider the following types of structure in this paper: Factored Factored MDPs [60, 76] and Factored POMDPs [71, 142] struc-
ture the state as a vector of state variables $s = \{s_1, s_2, ..., s_n\}$. These variables can typically be partitioned in factors $s_{z_j} \subset \{s_1, s_2, ..., s_n\}$. This factorisation can be found naturally when modeling natural systems where state variables have clear semantics. This factored representation is also reflected in the transition function that can be written as the product of such factors $z_j$, which satisfy the conditional independence property, i.e. $P_{z_j}(s'|s, a) = P_{z_j}(s'|s_{z_j}, a)$.

**Object Oriented** Object-Oriented MDPs (OO-MDP) [44] and POMDPs (OO-POMDPs) [138] further structure the state space by introducing the concepts of objects and object classes. Each object class is defined by a set of attributes (state variables), and each object instance has a state defined by assigning values to these attributes. The state of the entire environment is the union of the state of its constituent objects, thus allowing a more efficient and understandable representation.

**Parameterised Actions** Parameterised Action MDPs (PAMDP) [96] extend the set of actions $A$ to be parameterised by a vector $x \in \mathbb{R}^{m_a}$. An action selection by the agent is then a pair $(a, x)$ specifying the discrete actions as well as its parametrisation (for example, to kick a ball with a certain amount of force, or to move at a certain velocity). The extension of the model to the POMDP case is straightforward, and text-based games are an existing use of PA-POMDPs [104].

**Decentralised** Decentralised MDPs (Dec-MDP) [19] and POMDPs (Dec-POMDP) [103] extend the MDP and POMDP formalisms to model the multi-agent case, in environments consisting of multiple agents—that collectively maximize the same reward function—selecting actions in a decentralised manner.

Each of these formalisms are obtained by assuming more structure about the basic MDP or POMDP formalism. That structure adds complexity and narrows the set of tasks to which the model is applicable, but at the same time gives the agent the opportunity to exploit the additional structure during learning or planning.
5.3 Syntactic Categories in Language

Syntactic categories, also known as parts-of-speech (POS), are classes of words that have semantic tendencies—for example, nouns describe objects while adjectives refer to properties. They are broadly categorised into closed class (e.g., determiners such as “a, the” or prepositions such as “on, at”, that are rarely coined or expanded as times change) and open class (e.g., verbs such as “zoom, fax” or nouns like “Macbook, Roomba” that are continually created as needed). There are four main open classes i.e., nouns, verbs, adjectives, adverbs each of which are subcategorised. We refer readers to the work of marcus1993building for a full overview of parts-of-speech and the 45 categories annotated by the Penn Tree Bank. POS categories are important for several language understanding tasks, since they reveal important information about properties of the word, as well as its context. For example, for words that have different POS tags in different concepts (e.g., “dash” as a verb versus “dash” as a noun), knowing their POS tags could help resolve ambiguity to understand the meaning of the sentence.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>coordinating conjunction</td>
<td>and, or, if</td>
</tr>
<tr>
<td>V</td>
<td>verb</td>
<td>move, push, pull</td>
</tr>
<tr>
<td>IN</td>
<td>preposition</td>
<td>above, below, on</td>
</tr>
<tr>
<td>NN</td>
<td>common noun</td>
<td>wall, location</td>
</tr>
<tr>
<td>NNP</td>
<td>proper noun</td>
<td>Taxi, Agent</td>
</tr>
<tr>
<td>ADJ</td>
<td>adjective</td>
<td>blue, round, small</td>
</tr>
<tr>
<td>EX</td>
<td>existential ‘there’</td>
<td>there</td>
</tr>
<tr>
<td>MD</td>
<td>modal</td>
<td>can, should</td>
</tr>
<tr>
<td>ADV</td>
<td>adverb</td>
<td>quickly, slowly</td>
</tr>
<tr>
<td>PRP</td>
<td>personal pronoun</td>
<td>your, their</td>
</tr>
</tbody>
</table>

Table 6: Parts of speech (as tagged by the PTB) for word classes important to decision making.
5.4 Different Syntax for Different Languages

We should note that, although the four main POS categories seem like fundamental syntactic constructs, some languages (like Riau Indonesian or Tongan) do not even make a distinction between nouns and verbs [28]. There also exist languages devoid of a certain class e.g., adjectives in Korean, where words that would normally be adjectives in English translate to a subclass of verbs in Korean (“beautiful” → “to be beautiful”). Thus, although the different languages vary in the syntactic categories they cover, their functionality can be replicated, albeit at a cost (e.g., larger number of words).

5.5 Language Elements for Describing Decision Processes

<table>
<thead>
<tr>
<th>MDP</th>
<th>V</th>
<th>CC</th>
<th>IN</th>
<th>NN</th>
<th>NNP</th>
<th>ADJ</th>
<th>EX</th>
<th>MD</th>
<th>ADV</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDP</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Factored MDP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OO-MDP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unstruc. POMDP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Factored POMDP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OO-POMDP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LTL-MDP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dec-MDP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dec-POMDP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dec-OO-POMDP</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 7: Differences in complexities in different classes of MDPs. Each row shows a class of MDPs while columns show POS categories that can be used to ground to existing parts of each model, as enumerated in Section 3.

We now draw connections between which part-of-speech elements can be related to each model class. We summarise these facts in Table 7.
Unstructured MDPs  This is the base formalism for decision processes in AI and the one that includes the least structure. The state representation is completely unstructured and, hence, tying nouns to elements of the model can be difficult. However, **proper nouns** relate to states with certain characteristics such as goal states. Actions are referred to using **intransitive verbs**—as there is no concept of an object to apply the action on. Conditional statements and connecting words i.e., **conjunctions** like “and, or” are used to tie together specifications of parts of the MDP. **Prepositions** denoting order relations, such as “before, after” are used to describe ordering a sequence of actions or sub-policies. **Determiners** such as “more, less, equal”, **cardinal quantities** and **comparative adjectives/adverbs** are used to specify rewards.

Factored MDPs  In addition to the elements above, **proper nouns** can be used to name factors. These nouns are unique in the sense that no two factors are the same. As nouns, we can qualify them by using **adjectives** that can specify properties of the factor such a particular setting of the factor. For instance, if we consider the coordinates of a robot \((x, y)\) to be “location”, “home location” could be the coordinates \((0, 0)\).

OO-MDPs  Object classes in OO-MDPs correspond to the concept of **common nouns**. In this way, we can use **determiners** such as “a, the” to talk about a specific instance of an object or about any object of a class. As before, **adjectives** qualify object instances. In this way, we can map qualified nouns to instances with particular attributes—e.g. *the red ball*. With different numbers of objects, we can use **quantifiers** such as “there is/are, all”. Specific instances of an object can be specified by **proper nouns**. Given the existence of objects, **transitive verbs** can be mapped to actions that affect objects, in this way information about the dynamics of the world can be represented with more complex constructs as the transition function relates to the objects’ state. The fact that humans have rich notions of “nouns" and concepts hints that they are likely using this sort of structure.
**PAMDPs** In this class of MDPs, we have actions that are parameterized, thus calling for **adverbs** as a way to qualify action (**verbs**). In this way, we can now realize actions that correspond to verb phrases such as “to go up slowly”.

**Dec-MDPs** In these models, we have referential expressions, in order to denote concepts of oneself and other existing entities in the world for multi-agent environments. Therefore **pronouns** (e.g., personal, possessive) come into play.

Until now, we have described the parts-of-speech that are relevant to increasingly more structured MDPs. Analogously, we proceed with the partially observable formalisms.

**Unstructured POMDPs** In these models, the important difference is the presence of partial observability. Part-of-speech that relate to MDPs are still pertinent here. However, we can use POS elements that can convey facts and uncertainty about the world such as **modal verbs** like “could, should” that allow to specify information about the world. Facts are necessary in partially observable domains in order to reduce the uncertainty the agent has about the state of the world. This is not the case when the state is completely observable as the agent knows all relevant attributes of its environment at each timestep.

**Factored POMDPs** Along with the above POMDP elements, **proper nouns** are used in the same way as in Factored MDPs. Similarly, **adjectives** can specify properties of state variables, that differentiate them from other states.

**OO-POMDPs** In OO-POMDPs, we need to express facts about the objects. Therefore, **prepositions** that denote order such as “in, at, on, below, above” are necessary to state relative order among object instances’ states. For instance, if we instantiate an object class “cup”, we can use specify facts about an object class: *the cup is in the kitchen.*
Dec-POMDPs  Analogously to the observable case, we have that in this case we need to specify facts about other agents’ state and policies. As in OO-POMDPs, prepositions are necessary to provide both spatial information about the agents and temporal information—e.g. “before, after”—when ordering and coordination among agents’ policies is required.

Non-Markovian Policies  Through the combination of conditional statements and temporal prepositions such as “until, before, after”, it is possible to specify instructions for a task which relates to further structure in the action space of the model. These instructions are related to (sub-)policies in any of the models we have discussed thus far. However, these may be non-Markovian given that the use of prepositions such as “until” implicitly requires the agent to record past actions and states in order to determine if a condition is satisfied. Linear Temporal Logic (LTL) has been used in several works [43, 107] that use LTL-MDPs to handle such dependencies.

5.6 Discussion and Conclusions

In this chapter, we were primarily interested in characterising what elements of natural languages are important to different MDPs. The relation between parts-of-speech and decision making models we have laid out here are by definition true. Which means that it is the case that certain parts-of-speech may become dispensable given the type of decision process the agent must solve. Therefore, our claims are not about trying to establish an empirical performance benchmark, but instead about establishing what elements of language are necessary for the different types of decision process.

We have attempted to use insights from syntactic constructs in natural languages to characterise the differences in the forms of language useful for communicating with agents using different classes of decision-making models. We posit that the richness of language has its roots in the richness of the decision process that humans are solving. From the parallels drawn in the previous section, we can see how much of language can be (partially)
related to elements of the different MDP classes, which in turn suggests the different kinds of structure and abstractions that humans might be implicitly using to tractably solve their own decision problems. This suggests that the form of language used to describe a task has the potential to aid in automatically determining the structure and abstraction necessary for an agent to solve that task.

Moreover, we wish to emphasize that our exploration shows that much of humans’ use of language is for conveying factual information about objects and object classes, which strongly suggests that human decision processes are both partially observable and highly structured. However, the flexible nature of language, both in the way that new elements—such as those of the open classes of syntactic elements—can be defined based on known elements and how verbs and nouns can be qualified in new ways, could result from the flexibility with which we can generate new abstractions to handle new problems and better handle partial observability.

Current RL research does not use highly structured and partially observable models—many current efforts go into designing algorithms with as little structural bias as possible. The richness of human language—and its clear links to richer, more structured representations—suggests that RL research should perhaps focus instead on highly structured formalisms, especially for research on grounding language in RL.
RLang: A Declarative Language for Expressing Prior Knowledge for Reinforcement Learning

This chapter is based on the paper "RLang: A Declarative Language for Expressing Prior Knowledge for Reinforcement Learning with Rafael Rodriguez, Benjamin Spiegel, Jennifer Wang, Stefanie Tellex and George Konidaris.

As motivated in the previous chapter, different types of information that we might want to communicate to an agent might correspond to different components of its decision-making model. However, there currently exists no unifying framework that allows language to be grounded to every component of an MDP—only frameworks that ground to a single component (e.g., rewards) each time. In this chapter, we introduce RLang, a domain-specific language (DSL) for communicating domain knowledge to an RL agent. Unlike other existing DSLs proposed by the RL community that ground to single elements of a decision-making formalism (e.g., the reward function or policy function), RLang can specify information about every element of a Markov decision process. We define precise syntax and grounding semantics for RLang, and provide a parser implementation that grounds RLang programs to an algorithm-agnostic partial world model and policy that can be exploited by an RL agent. We provide a series of example RLang programs, and demonstrate how different RL methods can exploit the resulting knowledge, including model-free and model-based tabular algorithms, hierarchical approaches, and deep RL algorithms (including both policy gradient and value-based methods).

Reinforcement learning (RL) algorithms have seen important successes
such as agents that learn to play the Atari games from pixels [102] and learning to play Go, chess and shogi [118, 122, 123]; demonstrating the capabilities of deep RL methods. However, they require a great amount of experience and, thus, learning in such tabula rasa setting quickly becomes impractical. Moreover, to develop generalist learning agents, we cannot expect them to learn every task they will face without instruction and task-specific knowledge. In fact, it is reasonable to assume that we would communicate to the agent useful task-specific knowledge, e.g., the rules of a game, relevant permitted actions and features of the observation such as relevant observed game pieces, before they begin learning a policy. This information will be oftentimes partial and the agent will need to improve and complete its knowledge in the usual RL loop, but the given prior information should nonetheless improve the learning performance.

Languages, both formal and natural, have been used in various ways to add prior knowledge into decision-making [91]. Formal languages benefit from unambiguous syntax and semantics, and can therefore be reliably used to represent knowledge. These have proven useful in specifying advice to agents in the form of hints about actions [94] or policy structure [5]. Communicating such knowledge using natural language would be more intuitive, though this approach would require converting natural language sentences into grounded knowledge usable by the agent; most of the approaches in this area restrict the possible grounding by translating natural language into expressions of a restricted grammar. For example, for describing task objectives [8, 111], or other individual components of decision-making systems such as rewards [58, 125] and policies [25]. All of the above approaches provide information about a single component of a chosen decision-making formalism; there exists no unified framework able to express information about all the components of a task.

We therefore introduce RLang, a domain-specific language (DSL) with precise syntax and semantics to express information about every component of a Markov decision process (MDP), including flat and hierarchical policies, state factors and features, transition functions, and reward functions. Moreover, we release the RLang Parser\(^2\) that interprets RLang pro-

\(^2\)The RLang source code, documentation, and examples are available at rlang.ai.
Effect:
   if at_workbench_1 and A == use:
   if wood >= 1:
      stick' -> stick + wood
      wood' -> 0
   Factor inventory := S[250:270]
   Feature wood := inventory[0]
   minecraft.rlang
   Feature gold := inventory[1]

Figure 14. RLang programs are parsed by the interpreter to create a partial model of the MDP and its solution.

grams and produces grounded partial models in Python to be used by any learning algorithm. We then demonstrate RLang’s versatility through a series of example programs that express different types of domain knowledge and how such knowledge improves learning performance.

Related Work

There has been a recent surge of interest in methods that use language to inform RL agents [91]. These fall under methods that use natural language to instruct, or to reward, agents as a form of supervision, or methods that use formal languages to represent goals or an MDP component.

Formal Languages in Reinforcement Learning In classical planning it is standard to use the Planning Domain Description Language (PDDL; [53]) and its probabilistic extension PPDDL (probabilistic PDDL; [145]) to specify the complete dynamics of a factored-state environment. RLang is inspired by these but it is intended for a fundamentally different task: providing partial knowledge to a learning agent, where the knowledge might correspond to any component of the underlying MDP. [94] propose an RL paradigm in which the agent may request advice, as provided through a DSL that uses propositional statements to provide policy hints. Similarly, [126] propose to learn a policy conditioned on a program from a DSL. [5] use a simple grammar to represent policies as a concatenation of primitives (sub-policies) to provide RL agents with knowledge about the hierarchical structure of the tasks.

Other languages include linear temporal logic (LTL; [68, 89]) which has been used to describe goals for instruction-following agents. These methods
Table 8: Comparison of DSLs proposed for RL agents and the types of expressible MDP information

<table>
<thead>
<tr>
<th>RL Language</th>
<th>Policy Hint</th>
<th>Action Structure</th>
<th>Policy Constraints</th>
<th>State Structure</th>
<th>Rewards</th>
<th>Transition Dynamics</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALisp [3]</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Advice RL [94]</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Program-guided Agent [126]</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Programable Agents [40]</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Policy sketches [5]</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GLTL [89]</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>SPECTRL [68]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RLang</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Ground LTL formulae to reward functions for the agent. Note that LTL formulae easily express non-Markov tasks and grounding methods modify the state space to cope with this. RLang assumes that the state space is already Markov.

RLang expands on all of these DSLs to include information beyond the policy and the reward function, thus allowing a wider array of information to be parsed and interpreted by the agent. Table 8 summarizes existing DSLs for RL and shows their relative expressive power: no existing DSL is sufficiently powerful to express the wide range of information that can be used by an RL agent.

Natural Language Grounding and Learning Methods There is a significant amount of work that attempts to ground the semantic meaning of language instructions to information usable by RL agents [91]. Some approaches learn to ground natural language advice to a single grounding function type (e.g., reward function) directly from data. For example, for the game of Civilization II, RL agents can be taught to ground linguistic information to features that allow better estimation of the Q-function [24]. Other work shows that grounding textual specifications of goals and dynamics, allows learning a language-conditioned policy [149]. In instruction-following, some approaches learn to map instructions directly to reward functions [13, 59, 101], while others translate natural language to an intermediate formal language that represents rewards [8, 111]. In general, these
languages are restricted grammars that can be easily mapped to the desired grounding element. For example, some methods translate instructions to sequences of primitive actions [100] or to LTL formulae [56, 111, 141]. In future work, we plan to use RLang as the semantic representation language, since it has a higher expressive power.

6.1 Program Guided Agents and DSLs

Domain-Specific Languages Domain-specific languages (DSLs) are formal languages designed to specify information relevant to a target domain. Compared to general-purpose programming languages like Python [135] and C [72], DSLs typically contain a smaller set of narrower semantics that are well-suited to a specific application. That is, DSLs sacrifice computational expressivity for ease-of-use within a particular domain. Commonly-used DSLs include the Standard Query Language (SQL) used for querying relational databases and the Planning Domain Definition Language (PDDL; [53]) for defining planning tasks.

Decision-Making Formalisms Reinforcement learning tasks are typically modeled as Markov Decision Processes (MDPs; [113]), which are defined by a tuple \((S, A, R, T, \gamma)\), where \(S\) is a set of states, \(A\) is a set of actions, \(T : S \times A \times S \rightarrow [0, 1]\) is a transition probability distribution, \(R : S \times A \times S \rightarrow \mathbb{R}\) is a reward function, and \(\gamma \in (0, 1]\) is a discount factor. A solution to an MDP is a policy \(\pi : S \times A \rightarrow [0, 1]\) that maximizes the expected discounted return \(E[\sum_{t=0}^{\infty} \gamma^t r_t]\), where \(r_t\) is the reward obtained at time step \(t\). The value function \(V^\pi : S \rightarrow \mathbb{R}\) for a policy \(\pi\) captures the expected return an agent would receive from executing \(\pi\) starting in a state \(s\). The action-value function \(Q^\pi : S \times A \rightarrow \mathbb{R}\) of a policy is the expected return from executing an action \(a\) in a state \(s\) and following policy \(\pi\) thereafter.

Hierarchical Decision-Making Solving MDPs with high-dimensional state and action spaces can be difficult, especially in domains where long sequences of actions are required to achieve a goal. In these environments, hierarchical reinforcement learning [15] may be more applicable, as temporally-extended actions can reduce the complexity of the space of
solution policies. The options framework [129] formalizes this notion by modeling high-level actions as options: closed-loop policies defined by a tuple \((I, \pi, \beta)\), where \(I \subseteq S\) is a set of states in which the option can be executed, \(\pi\) is an option policy, and \(\beta : S \to [0, 1]\) describes the probability that the option will terminate upon reaching a given state. Let \(O\) be the set of options the agent can execute, then the MDP tuple is extended to \((S, A \cup O, R, T, \gamma)\) in the hierarchical setting.

6.2 RLang: Expressing Prior Knowledge about RL Tasks

If RL is to become widely used in practice, we must reduce the infeasible amount of trial-and-error required to learn to solve a task from scratch. One promising approach is to avoid tabula rasa learning by including the sort of background knowledge that humans typically bring to a new task. Such background knowledge is often easy to obtain—in many cases, it is simply obvious to anyone: try not to fall off cliffs!—and need not be perfect or complete in order to be useful.

Unfortunately, however, there is no standardized approach to communicating such background knowledge to an RL agent. In most cases, the same person who implements the learning algorithm also hand-codes the background knowledge, typically in the same general-purpose programming language in which the algorithm is implemented, typically in an ad-hoc fashion. This has two primary drawbacks. First, prior knowledge is often task-specific, and the lack of a medium to express it hinders the development of general-purpose learning algorithms that can exploit varying types and degrees of background knowledge. Second, this approach is not accessible to end-users or other consumers of RL agents, who do not write the algorithms themselves and cannot necessarily be expected to master the relevant programming languages and mathematical details, but who might nevertheless wish to accelerate learning.

The alternative is to design a standardized, human-interpretable DSL for expressing prior knowledge about reinforcement learning tasks. Such a DSL should have two important properties, which are not present in
existing DSLs [40, 94, 126]. First, it should be agnostic of the learning algorithm used. Separating the question of how to express prior knowledge from how that knowledge is exploited by a learning algorithm introduces a standardized interface that can be used to inform a wide variety of RL agents, even ones based on algorithms that have not yet been developed. Second, it should be *complete*: able to express all the information that could possibly be informative about a particular task. We therefore propose RLang, a new DSL designed to fulfill these criteria.

RLang can be used to prescribe features of the state space (using *Features* and *Propositions*), specify one or more goal states (using *Goals*), define abstract actions (using *Options*), describe solution policies and hierarchical policy structure (using *Policies*), restrict the action space (using *ActionRestrictions*), provide partial models of the world (using *Effects*, which ground to reward functions and transition functions), and shape reward (also using *Effects*). RLang programs can be parsed using the RLang Parser\(^3\) into an algorithm-agnostic data structure (see Section 6.2) that can be integrated into nearly any reinforcement learning algorithm. In this section, we present the main RLang elements and their syntax.

### RLang Elements

An RLang program consists of a set of declarations, where each one grounds to one or more components of an \((S, A, O, R, T, \pi)\) tuple. More specifically, every RLang Element grounds to a function with a domain in \(S \times A \times S\) and a co-domain in \(S, A, R^n\) where \(n \in \mathbb{N}\), or \(\{\top, \bot\}\). We describe the main RLang element types in the rest of this section.

#### State Factors

In Factored MDPs [22], the state space is a collection of conditionally independent variables: \(S = \mathcal{X}_1 \times \ldots \times \mathcal{X}_n\). Some learning algorithms might find it useful to reference these variables individually. For example, consider a 2-D version of Minecraft, where an agent has to collect ingredients to craft new tools and objects. In this environment the state

---

\(^3\)Our website for RLang, which includes installation instructions, can be found at rlang.ai.
is the concatenation of a position vector, a flattened map representation, and an inventory vector: \( s = (\text{pos}, \text{map}, \text{inventory}) \). **Factors** can be used to reference these independent state variables:

```
Factor position := S[0:2]
Factor map := S[2:250]
Factor inventory := S[250:270]
```

\( S \) is a reserved keyword referring to the current state. \( A \) and \( S' \) are also keywords which refer to the current action and the next state, respectively. Factors can be further sliced and indexed:

```
Factor iron := inventory[0]
Factor wood := inventory[1]
```

**State Features** RLang can also be used to define more complex functions of state. For instance, if the agent’s goal is to build axes, we can define a **Feature** that captures the number of axes that can be potentially built at the current state:

```
Feature number_of_axes := wood + iron
```

**Propositions** Propositions in RLang, which are functions of the form \( S \rightarrow \{\top, \bot\} \), identify states that share relevant characteristics:

```
Constant workbench_locations := \{[1, 0], [1, 3]\}
Proposition at_workbench := position in workbench_locations
Proposition have_bridge_material := iron >= 1 and wood >= 1
```

**Goals** Goals can be used to specify goal states given by a proposition. For example, **Goal get_gold := gold >= 1** encodes that the agent must collect at least one gold unit.

**Markov Features** Markov Functions like the action-value function or transition function take the form \( S \times A \times S \rightarrow \mathbb{R} \). We extend the co-domain of this function class to \( \mathbb{R}^n \), where \( n \in \mathbb{N} \), and introduce **Markov Features**, which allow users to compute features on an \((s, a, s')\) experience
tuple. The following Markov Feature represents a change in inventory elements.

**Markov Feature** inventory\_change := inventory ' − inventory

The prime (') operator references the value of an RLang name when evaluated on the next state.

**Policies** Policy functions can also be specified in RLang using conditional expressions:

**Policy** main:
- if iron >= 2:
  - if at\_workbench:
    - Execute Use # This is an action
  - else:
    - Execute go\_to\_workbench # This is a policy
- else:
  - Execute collect\_iron

The **Execute** keyword can be used to execute an action or call another policy. The above policy instructs the agent to craft iron tools at a workbench by first collecting iron and then navigating to the workbench. Policies can also be probabilistic:

**Policy** random\_move:
- Execute up with P(0.25)
- or Execute down with P(0.25)
- or Execute left with P(0.25)
- or Execute right with P(0.25)

Users can specify multiple policy functions in an RLang program and can designate a primary policy by naming it main.

**Options** Temporally-extended actions can be specified using Options, which include initiation and termination propositions:

**Option** build\_bridge:
- init have\_bridge\_material and at\_workbench
  - Execute craft\_bridge
- until bridge in inventory
**Action Restrictions**  Restrictions to the set of possible actions an agent can take in a given circumstance can be specified using **ActionRestrictions**:

```
ActionRestriction dont_get_burned:
    if (position + [0, 1]) in lava_locations:
    Restrict up
```

**Effects**  Effects provide an interface for specifying partial information about the transition and reward functions. When using a factored MDP, RLang can also be used to specify factored transition functions (i.e., transition functions for individual factors):

```
Effect movement_effect:
    if x_position >= 1 and A == left:
    x_position' = x_position - 1

Reward -0.1
```

The above Effect captures the predicted consequence of moving left on the `x_position` factor, stating that the `x` position of the agent in the next state will be 1 less than in the current state. This Effect also specifies a $-0.1$ step penalty regardless of the current state or action. In simpler MDPs, predictions can be made about the whole state vector:

```
Effect tic_tac_toe:
    if three_in_a_row:
    S' = empty_board  # Board is reset
```

Effects can reference previously defined effects using similar syntax:

```
Effect main:
    -> movement_effect
    -> crafting_effect
```

A main Effect designates the primary environment dynamics, and grounds to a partial factored world model $(\bar{T}, \bar{R})$. Similar to policies, Effects can be made probabilistic using with.

Finally, it is important to note that RLang, as we have seen across these examples, does not require the specification of Effects and Policies to be complete (i.e., known for every element transition $(s, a, s')$ or state-action pair $(s, a)$, respectively). Therefore, a user is not constrained to provide extensive and complex programs to fully specify the MDP—although this
is a possibility with RLang—in order to accelerate learning. Therefore, RL agents are meant to learn to fill the missing pieces and improve over the provided knowledge.

**ACCESSING PARSED RLANG KNOWLEDGE**

Using the RLang’s Python API, users can parse RLang programs into the following queryable knowledge Python objects, which can be integrated directly into a learning algorithm: (1) the Dynamics and Task Knowledge object contains a queryable model of the environment and the task (i.e., transition dynamics $T$ and reward function $R$) that are derived from the Effect main declaration and the collection of defined goals; (2) the Solution Knowledge object that contains information about the collection of newly defined options $O$ and the main policy $\pi$. Moreover, these knowledge objects are implemented as partial functions and, hence, when querying for a element of the domain where the RLang programs provides no knowledge, it returns an unknown flag.

**SPECIFYING COMPLEX GROUNDINGS WITH A VOCABULARY FILE**

RLang comes built-in with a set of simple arithmetic, Boolean, and set operations in addition to if, elif, else conditional statements which can be used in various RLang object declarations. However, we recognize that users may want to include more complex grounding functions in their RLang programs. For instance, when dealing with problems with high-dimensional observation spaces (e.g. pixel frames), we might require to provide groundings that abstract away low-level details of the problem. To accommodate these needs, we have made it possible to define RLang objects using our RLang’s Python API which can be imported and referenced in an RLang program. By specifying a vocabulary file (in JSON format)

---

4We chose Python as the implementation language to ensure compatibility with widely used libraries and implementation of RL algorithms
and a corresponding **grounding file** (a Python file defining RLang objects), users can construct RLang objects using the full features of Python and reference them directly in RLang programs. This allows users to provide complex expert groundings or, more generally, learned groundings that hold the necessary semantic information to derive *new* grounded knowledge easily with RLang programs.

### 6.3 Demonstrations

![Figures show the environments we use to demonstrate different types of RLang information.](image)

**Figure 15.** Figure shows the environments we use to demonstrate different types of RLang information.

In this section, we demonstrate RLang use-cases, focusing on examples that show how different types of prior information that can be concisely and easily expressed, for varying degrees of environment complexity and different families of RL methods. We therefore provide examples of information about policy hierarchical structure, policy priors and transition dynamics, and explore how this information can be exploited with RL methods suitable to the type of information provided. We design simple and effective RLang-informed agents based on model-free and model-based tabular RL methods such as Q-Learning [139] and RMax [23], policy gradient methods such as PPO [119] and REINFORCE [143] and hierarchical RL methods based on options and DDQN [134].
Hierarchical Policy Structure: 2D Minecraft

We first consider a 2D version of Minecraft based on [5], consisting of a gridworld that contains workbenches where the agent can craft new objects, and raw materials like wood, stone and gold. To build an item, the agent must have the required ingredients and be in the correct workbench. The agent has the action use to interact with elements, and actions to move in the cardinal directions.

We show how providing the sub-policy structure of the task improves performance. Specifically, we provide the agent with initiation and termination conditions for a few options (to collect wood, go to the three different workshops and to build the required elements), leaving the agent to learn the policy over options. The following program concisely defines 3 options fully and 4 options with uninformative policies. This is an example of a simple RLang program that conveys partial hierarchical structure that can effectively help the agent improve learning.

```
1 Option go_to_workshop_0:
2   init (any):
```
Execute go_to_workshop_0_learnable_policy
until(at_workshop_0)

Option go_to_workshop_1:
init(any):
  Execute go_to_workshop_1_learnable_policy
until(at_workshop_1)

Option get_wood:
init(there_is_wood):
  Execute get_wood_learnable_policy
  until delta_wood >= 1

Option build_plank:
init(wood >= 1 and at_workshop_1):
  Execute use
  until (delta_plank >= 1)

Option build_stick:
init(wood >= 1 and at_workshop_1)
  Execute use
  until (delta_stick >= 1)

Option build_ladder:
init(stick >= 1 and plank >= 1)
  Execute use
  until (delta_ladder >= 1)

To exploit this information, the agent must learn both the policy over options to maximize reward, and the option policies that achieve each option’s termination condition. For both the high-level and low-level agents, we use the DDQN algorithm [134] (implementation details are in Appendix ??).

Figure 16a shows the average return of RLang-informed hierarchical DDQN [134] vs. the uninformed (flat) performance of a DDQN agent. The results show that providing a concise program partially describing a hierarchical solution was sufficient to successfully learn to solve the task, in stark contrast with the uninformed DDQN agent.

**Policy Prior: Lunar Lander** Next, we consider programs that provide prior policy knowledge. Such policy information need not be optimal or complete, but it can still improve learning performance. We first consider the Lunar Lander environment openaigym, which requires learning
an optimal control policy to gently land a ship on the moon. The environment has a dense reward signal encoding both the goal of the system and cost constraints, a continuous state space, and four discrete actions that either do nothing, fire the main engine, or fire the left or right thruster. We provide the agent with an initial policy using the following RLang program:

```r
Policy land:
  if (left_leg_in_contact == 1.0) or (right_leg_in_contact == 1.0)
    if (velocity_y/2 * -1.0) > 0.05:
      Execute main_engine
    else:
      Execute do_nothing
  elif remaining_hover > remaining_angle and remaining_hover > -1 * remaining_angle and remaining_hover > 0.05:
    Execute main_engine
  elif remaining_angle < -0.05:
    Execute right_thruster
  elif remaining_angle > 0.05:
    Execute left_thruster
  else:
    Execute do_nothing
```

We implemented an RLang-informed agent using PPO [119], a policy gradient method, as our base method. We probabilistically mixed the RLang-defined advice policy with a learnable policy network using mixing parameter $\beta \in [0, 1]$, following [47]. This mixing parameter is annealed during learning process. In this way, the RLang policy and the learnable policy shared control stochastically.

```r
Policy gain_momentum:
  if velocity < 0:
    Execute go_left
  else:
    Execute go_right
```

Figure 16b shows the average return curves resulting from an uninformed PPO agent schulman2017proximal and the RLang-informed version. The
informed agent is able to leverage the initial performance of the given policy and learn how to improve it further, resulting in a clear performance improvement.

We also considered two classic control problems: CartPole and Mountain Car. For CartPole, we obtain analogous results using REINFORCE [143] as the base method. In Mountain Car, a hard exploration problem in RL, a very concise RLang policy results in near-optimal performance; the simple program on the right gets a $-119$ average return over 100 episodes, where the task is considered solved with a $-110$ average return.¶

```python
Effect moving_effect:
    if A == up:
        x' -> x + 1
        y' -> y
    elif A == down:
        x' -> x - 1
        y' -> y
    elif A == left:
        x' -> x
        y' -> y - 1
    elif A == right:
        x' -> x
        y' -> y + 1

Effect dynamics:
    if at_wall:
        S' -> S
    else:
        -> moving_effect

Effect reward:
    if in_lava:
        Reward -1
    elif at_goal:
        Reward 1.
    else:
        Reward 0.

Effect main:
    -> dynamics
```
Dynamics and Rewards: Lava-Gap

We now show how to provide information about the dynamics and the rewards to an agent using RLang. To do so we use the Lava-Gap environment, a gridworld in which an agent is tasked to navigate to a goal position. The agent can move in the cardinal directions but each action has a probability of failure of 1/3. Moving into walls causes the agent to stay in the same position and falling into a lava pit results in a high negative reward. The agent would typically need to fall into lava pits many times to learn to avoid it. With RLang, however, we can easily inform agents about the dynamics and the high cost of lava pits.

The program on the right defines an effect moving_effect that predicts the effect of an action in most cases—i.e., when walls are not in the way. The effect dynamics extends this by adding the effect of walls and the reward function is provided through the effect reward. Tabular Q-Learning is suitable here. We designed a Q-Learning agent that exploits the transition dynamics and reward information. The agent first estimates an initial Q-table using Value Iteration based on the partial transition and reward models—when information is unknown for a transition tuple \((s, a, s')\) the Q-value defaults to 0.

Average return curves for an RLang-informed Q-Learning agent [139] are in Figure 16c. This shows that the informed agent leverages the information to gain high return early in the training process. Analogous results for an RMax agent, a model-based reinforcement learning method [23].

6.4 Discussion

RLang is a precise, concise and unambiguous domain-specific language designed to enable a human to provide prior knowledge to an RL agent. It provides syntax and semantics tailored for MDPs and the RL setting where partial information can significantly improve learning performance of established RL methods, as shown in our experiments. Moreover, these experiments highlight the main assumption in current RL algorithm design: agents must learn tabula rasa and, therefore, we were required to design ad-hoc informed variations. In future work, RL methods must also
consider informed formulations in which humans can provide information about the task definition, the relevant dynamics and policy/action advice. RLang is algorithm-agnostic at its core and we envision it as a standard interface that will enable research in such general informed RL agents.

Finally, we envision RLang as a first step in our approach to enable flexible communication between humans and RL agents; a necessary feature to deploy embodied agents. Natural language is one of the most flexible and readily available communication tool we have and RLang can be used in systems that ground natural language semantics for RL agents’ use. We foresee that negative impacts of our work are related to the understanding of RL and robotics that can bring about the development of sophisticated technology for harmful purposes.
PART IV
7

Mapping Language Models to Grounded Conceptual Spaces

This chapter is based on the paper "Mapping Language Models to Grounded Conceptual Spaces" [109] published at ICLR 2022 with Ellie Pavlick.

In Parts III and IV, we saw how could train models to ground natural language by structuring it to formal languages tied to the environments that agents exist in, or components of an agents world. However, there has been significant progress in textual language models, that only learn from text corpora, however, have shown a remarkable understanding of language, and the world when tested on a range of natural language understanding tasks [29, 115]. Such models have demonstrated their ability to generate fluent dialogue [29], make commonsense inferences [147], and reconstruct taxonomies and word relations [32]. However, it has been argued that true meaning cannot be learned from the form of language alone (i.e., from text) because it is a word’s use in the non-linguistic world that imparts it meaning [17, 20]—that forms the crux of the motivation of this thesis. For example, although LMs might learn from textual co-occurrences that the words north and south are opposites, the grounded meaning of these words, i.e., the direction that you should travel if you are told to go north, is something which these models, by definition, do not have access to during training.

While it is indisputable that text-only models do not learn representations of concepts that are grounded in the non-text world, it is possible for the structure of relations between concepts in text form to be identical to what a grounded model would learn. In principle, it is therefore possible for a text-only model’s conceptual space to be isomorphic to the “true”, grounded conceptual space [99]. In this work, we investigate whether this is the case by asking whether models can learn to ground an entire domain...
Figure 17. Figure illustrates the difference between (a) learning the meaning of a word through other word forms, that could potentially be self-referential within a symbol system and (b) learning word meaning by grounding to some external context outside of the symbol system.

(e.g., colour) after grounding only a subset of the points in that domain (e.g., red), as illustrated in Figure 17. Specifically, for generative LMs that have been trained only on large text corpora, we “orient” the models by showing them how some word forms (that they have learned during training) are used in simple text worlds—for example, what the direction north maps to in a textual representation of a grid world (see Figure 18). We then evaluate two types of generalisation. First, we evaluate generalisation to unseen worlds. For example, if the model has seen several realisations of the word north in different grid worlds, can it correctly identify north in an unseen world (e.g., one of a different size or shape)? Second, we evaluate generalisation to unseen but related concepts. For example, if the model has been shown grounded representations of north and east, can it correctly identify south and west, even though it was never shown to them? We find that although the small language models (GPT-2 models that contain an order of 100M parameters) cannot perform either generalisation well, the largest model (a GPT-3 model containing 175B parameters) can indeed learn groundings in the conceptual worlds we build. We analyse the pre-
dictions and errors made by models and find that the errors made by the small models often come from them struggling to produce in-domain references and defaulting to random words. In contrast, the errors made by the largest model are quite intuitive, as they predict in-domain concepts that are often close to the true concept in the world (e.g., maroon versus dark red).

**Related Work**

There is a significant amount of work that focuses on understanding how language models represent and reason about concepts in the world, as well as work that directly attempts to build models that take text inputs and ground them to elements in the world. We situate our work within these two bodies of literature: one that investigates how LMs understand linguistic phenomena and word meaning, and another, that attempts to connect models of language to models of the world in order to solve tasks. We describe each body of work below.

**Meaning and Understanding in LMs**  With the advent of large LMs of increasing orders of magnitude, there has been increasing speculation on the capabilities of such models, and whether they truly understand the meaning of the words they are learning representations for. Several works that attempt to probe linguistic phenomena in LMs, show that the representations learned by such models encode syntactic dependencies and coreference information [63, 132] and word-sense information [32]. Work that investigates the ability of models to form word associations, finds that large LMs can indeed perform such a task; for e.g., [66] reports that GPT-3 produces such tuples with 73% accuracy, suggesting that pre-trained LMs not only recognize that entities are related, but can differentiate how they are related. In this work, we specifically investigate whether or not such models can learn to ground conceptual domains (for e.g., a world of colours or spatial directions) and analyse when and where they fail to do so.
Natural Language Grounding  There is an increasing amount of work, usually at the intersection of NLP and fields like vision and reinforcement learning, that aims to use natural language to instruct agents about aspects of the world. In the case of vision, this could be in order to learn correspondences between language descriptions and pixels in an image [73], for example, to learn how to caption an image [137]. In the case of reinforcement learning or robotics, this could be to build agents that understand natural language instructions in order to take actions in a world that follow the instruction—for example to navigate to a goal [9], or to solve tasks in different languages [81]. Most of these tasks focus on training LMs from scratch, however usually with inputs that contain both textual information as well as grounded world information. Our work is different in that we attempt to take an LM that was previously trained only on text, and attempt to teach it a concept in the world without re-training it. With only a few samples of what the concept grounds to, we investigate how well large LMs can use the structure of language and associations between word forms in order to generalise to grounded concepts.

7.1 Experimental Design

Models  We test five autoregressive Transformer language models [136] of varying size, specifically the GPT-2 [115] and GPT-3 [29] models. The smallest model we use contains 124M parameters, and the others follow increasing model sizes (355M, 774M, 1.5B and 175B parameters). All models are pre-trained on the OPENAI-WT dataset, composed of 40GB of English web text available on the internet. We generate up to 5 tokens per prompt and, to improve the robustness of our analyses, generate 3 samples for every prompt. We use a temperature of 1 during generation and sample from the softmax probabilities produced at each time step using nucleus sampling [65] with $p = 0.85$.

In-Context Learning  Several studies [29, 116] have shown that instead of fine-tuning generative LMs—i.e., a process that updates parameters learned by the model during pre-training, it is possible to achieve competi-
tive performance by giving the model a small number of training examples within the prompt. This is often referred to as “in-context learning” or “few-shot prompting.” Specifically, a prompt includes $n$ task examples that include a question prefix (e.g., “World:"), followed by the question, and an answer prefix (e.g., “Answer:"), followed by the answer to the question. After giving the model $n$ examples in this manner, the prompt ends with a new question and only an answer prefix after which the model is expected to generate an answer to the last question, following the prompt format it has seen. By enumerating over all questions in the test set, we can obtain a model-generated answer for every test set question that we wish to evaluate. There are no gradient updates to any model parameters using this approach.

Figure 18. Figure shows example worlds and groundings for three concept categories: colours, cardinal directions, and spatial terms.

Grounded Concept Domains The models that we test, by construction, can receive only text inputs. Thus, we focus on a set of grounded domains for which it is possible to faithfully represent the grounded meaning in text form. We briefly describe these domains below, summarise them in Figure 18, and describe in detail, the prompt creation process for each generalisation task in the below sections.

Spatial Terms We consider 6 spatial concepts: left, right, up, down, top, bottom. Each of the above concepts can be represented in a grid world using the position of a special character (here, a ‘1’).

To do this, we create a grid world with a special character to represent each concept. For example, we can use a ‘1’ to represent the top of the world and a ‘0’ for the bottom. The model is then trained to predict the correct answer to the question based on the position of the special character.

As a control, we perform the same experiments by varying the special character, e.g., replacing a 1 with a 2 or replacing them with letters such as a or b. This does not affect the performance of the model.
grid world environments of varying sizes (where the number of rows and columns ranges from 1 to 8), where each world consists of ‘0’s and a single ‘1’.

**Cardinal Directions** We consider eight cardinal directions: *north, south, east, west, northeast, northwest, southeast, southwest*. These are similar to spatial terms, except that they include compositional terms (e.g., *north-east*). We use grid-worlds of the same format as spatial terms to represent these concepts.

**Colour Terms** We consider colour terms in a three-dimensional space, using a dataset of 367 RGB colours [1] that contains colour names (e.g., *red, cyan, forest green*) each associated with an RGB code (e.g., *(255, 0, 0)*). Therefore, the world representation in this case is not a grid-world, but an RGB code associated with every colour name. Figure 18 shows example RGB codes and colours that serve as part of a prompt given to the models we test.

### 7.2 Isomorphism and Controls for Memorisation

**Motivation** The GPT-x models that we use have been trained on the CommonCrawl corpus [115], a collection of documents that contains web text freely available on the internet. Since we provide instantiations of grounded concepts in text form, it is very plausible that the domains described above have been encountered verbatim during training. For example, for spatial terms such as *left*, a model might have seen instances of matrices and linear algebra terms with the word *left* in close proximity; for colour terms, tables that map RGB colour codes to colour names are pervasive in web-text. We therefore include a control task in our experimental setup such that the model cannot succeed using simple memorisation.
Rather, success on a task requires the model to truly perform a conceptual mapping between the ungrounded and grounded representations.

**Isomorphism as a Control** We use the concept of *isomorphism* to control for memorisation. Intuitively, imagine a situation where you are lost in the woods. Once pointed in the direction of north, you instantly know which way is south. However, this ability is not dependent on having been correctly pointed north—if someone were to incorrectly point east and tell you this was north, you would readily infer west to be south. This is because your reasoning depends on your knowledge of the relation between north and south, and between north and the world, rather than on having memorized the “true” grounding of each concept independently.

By the same logic, if a model is learning a grounding function, this should be dependent on the world that it is being grounded in. For example, in different worlds where the word *red* grounds to different points in space, the word *blue*, by analogy, shares a fixed conceptual relation to the word *red*. Therefore, it should ground in correspondingly equidistant ways in the two worlds. A model’s ability to learn two different grounding functions $f$ vs. $g$, should not be dependent on what the actual points ground to, as long as the structural relations between concepts in the space are preserved. Further, this should hold for all such isomorphic transformations that preserve the structure of the space, and importantly, it should *not* hold for random perturbations that distort the structure of the space, since such distortions would break the assumption that the relation between *red* and *blue* in ungrounded space is analogous to that in grounded space.

**Mechanism** In the colour domain, since colour concepts exist in a 3D world of RGB codes, we rotate each point around a fixed axis by a certain degree to create a new isomorphic world. We repeat this control three times (for 90°, 180° and 270° rotations) and average over the rotations in our evaluations. For cardinal directions, we rotate each cardinal concept by 90° in two dimensions, and do this three times and average over rotations. Since the spatial terms exist as pairs, we simply swap the groundings of alternate terms (e.g., *left* and *right*). For random worlds that do not preserve the
structure between word forms, we randomly assign a concept name (e.g., red) to a point in the world, and we do this for all concept names and categories to obtain a random world for each. Figure 19 shows example transformations of colours on rotating by $90^\circ$, as well as randomly rotating points.

Figure 19. Figure shows how colours and modifiers transform in rotated worlds. The leftmost figure shows a full 3-D colour space of 367 colours. The three figures on the right, show four sample colours in their original world, a world rotated by $90^\circ$, and a randomly rotated world, showing us how the structure of the space is preserved in an isomorphic rotation, but distorted in a random rotation.

7.3 Evaluations

Experimental Logic We report model performance in three settings: the original (“true”) world (e.g., that in which red maps to the actual RGB code for red), an average over three rotated worlds (e.g., worlds in which red maps to some other RGB code, but relations between colors are preserved), and a random world (in which relations between mappings are not preserved). If a model is performing the conceptual mapping in the desired way, we expect that performance should be high in the true world and that there should be no significant degradation in performance when moving to rotated worlds. We also expect that performance should be low in the random world.
Metrics When given a prompt, a generative LM is free to generate any number of tokens until having generated the EOS token that halts further generation. Since classification tasks usually correspond to labels that contain only a few words, the standard approach is to let the model generate an entire sequence of text and to then cut off the generation to the first \( n \) tokens (where typically \( n < 10 \)). From the prompting mechanism, the model should learn to follow the prompt format to generate one label (e.g., instead of every label in succession), and since it does not receive any gradient updates during this learning, there is no incentive for it to predict all related labels within a \( n \)-token generation. We set \( n = 5 \) and then report the following metrics.

**Top-1 Accuracy** If the ground-truth answer or any substring thereof\(^6\) lies in the generated answer (i.e., the first \( n \) tokens of the full generation), the model gets perfect accuracy. If the ground-truth answer does not exist in the generated answer, or exists in the generation outside of the cutoff limit, the model gets a 0 accuracy. For example, for the ground truth answer *deep tuscan red*, if the model generated answer is *tuscan red* or *red* the model gets a perfect accuracy, but if the model generated answer is *deep red* or *wine* or *vermilion*, the model gets an accuracy of 0.

**Top-3 Accuracy** This metric is analogous to Top-1 except that instead of only considering the most probable generation from the model, we collect the second and third most probable answer sequences as well. The model gets a perfect accuracy if the correct answer exists in any of the three generated answers. If the correct answer does not exist in any of the 3 generated answers, or exists in any of the generations outside of the cutoff limit, the model gets an accuracy of 0.

**Grounding Distance** For analysis purposes (S7.5), we wish to assess how far off models are, in cases where they are wrong. For this, we need to

\(^6\)We report results on exact match, instead of substring matches, for the best-performing model as well.
quantify the distance between the model’s predicted answer and the ground truth answer. For example, in the colour domain, the distance between two points can be computed as the Euclidean distance between two RGB codes. For every answer generated by the model (e.g., the word \textit{pink} in the colour domain), if the answer is an acceptable grounded term that exists in the world, we can compute its distance to the true grounding (e.g., the colour \textit{red}). However, if the generated answer was an unrelated word (e.g., the word \textit{cat}) that does not exist in the same domain, no distance can be computed in the world. Therefore, we calculate a distance metric as the Euclidean distance between two points in space when the generated answer falls in the domain. When the generated answer does not fall in the domain, we set the distance to a number significantly higher than the largest distance between two in-domain concepts.

**Baselines** Given that the language models are free to generate any word that exists in their vocabulary as a potential answer, we choose two random baselines over vocabulary words against which to compare model performance. We use R-IV (i.e., random in-vocabulary) to denote a baseline that randomly selects from among all words in the model’s vocabulary. We use R-ID (i.e., random in-domain) to denote a baseline that randomly selects from amongst only the in-domain words (e.g., from colour terms); this is 6, 8 and 367 words respectively for the spatial, cardinal and colour categories. Note that the generative LMs do not have such a domain restriction over words, as they are free to choose any token from the full vocabulary (like R-IV), thus making R-IV, a fairer baseline than R-ID.

**Experiments**

7.4 **Generalisation to Unseen Worlds**

Our first investigation looks into how well models generalise concepts that have been taught to them to unseen worlds. For example, if a model has seen a few examples of a concept (such as \textit{left}) as depicted in some grid worlds, can it correctly identify an instance of the same concept in a
Figure 20. Figure shows example worlds and groundings for the spatial domain. The left panel shows example grid worlds where the location of the 1 denotes a point grounded in the world, with a corresponding text instantiation of that particular concept. The two panels on the right show example outputs from the smallest and largest models. We show the top three most probable words from each model along with the probability of that word. When computing Top-1 accuracy, we only consider the first, however, we consider all 3 when computing Top-3 accuracy.

different grid world (e.g., one with a different size or orientation)? Note that we can only conduct this type of evaluation in the spatial and cardinal domains, since, for the colour domain, there is only ever one world, with one grounding for each concept (e.g., the colour red has exactly one grounding to a 3-digit RGB code in a 3-D RGB space).

Data We create prompts that include 20 examples of grounded concepts in a set of grid worlds. For each domain (e.g., cardinal directions that contain 8 concepts, and spatial terms that contain 3 pairs of 2 concepts), we include a (roughly) equal sample of concepts among these 20 examples. Then, for held-out grid worlds, we append it to the end of the prompt, to test whether or not models generate the correct concept label for these unseen worlds. Figure 20 shows example prompts given to the model and example generations from three different models. We report results averaged over all generations in Table ??.
above that on the random world. We do not see consistent or significant performance degradation when moving from the original to the rotated world, suggesting performance is not due to simple memorisation. Comparing across models, we see that the smaller models struggle to even learn concepts that were taught to them in a few-shot manner. For example, for the spatial category, the performance of the smallest model is below that of a baseline which guesses randomly among in-domain words, suggesting the model even fails to learn the general domain of the task (discussed more in Section 7.5). In contrast, the largest model (GPT-3), has a 45% Top-1 accuracy and a 76% Top-3 accuracy for the spatial category.

7.5 Generalization to Unseen Concepts

Our primary interest here is in the model’s ability to map its ungrounded conceptual structure to a grounded world. Specifically, we want to see that the model is able to ground an entire conceptual domain when only taught how to ground a small subset of the points in that domain. To test this, we show models example concepts in a domain (e.g., north, east), while holding out other concepts (e.g., south, west). We then test them on instances of held-out concepts. Figure 21 shows, for the colour domain, an example prompt with training examples and the associated held-out concept.

Data For the spatial and cardinal domains, for every concept we wish to test (e.g., left) we create prompts that contain 20 examples of all the remaining $n - 1$ concepts in the domain. We ensure that each concept appears at least once in the prompt. Then, we append a grid-world containing an instance of the held-out concept and evaluate whether models can generalise to this concept. For each concept, we do this for grid-worlds of different sizes and then average over concepts to report one metric of generalisation performance. For the colour domain, since the total number of concepts (colours) are too large to hold-out just one concept, we include only 60 examples in a prompt and average results over the remaining colours. Here, the prompt contains 3 primary and 3 secondary colours, as
Figure 21. Figure shows example worlds and groundings for the colour domain. The left panel shows example RGB codes and associated colour names, while the right shows example outputs from the smallest and largest models, with the top three most probable words from each model along with the probability of that word. When computing Top-1 vs. Top-3 accuracy, we consider only the first vs. all 3 outputs respectively.

well as 57 other randomly selected colours. For each held-out colour (i.e., one that does not appear in the prompt that the model is fine-tuned on), we append the RGB code to the end of the prompt, to assess how well the model can generalise to unseen colours.

We report results on all concept categories, and as before, we see that models do not appear to be exploiting simple memorisation (evidenced by similar performance in original vs. rotated worlds) and that only the largest models appear capable of getting reasonable performance on the task. That said, the largest GPT-3 model achieves impressive results given the difficulty of the task. For example, it achieves over 40% Top-1 accuracy in the color domain, which requires generating a label like violet despite having never seen such a label during training (Figure 21).

Error Analysis Given the results above, we investigate “how wrong” models are when they fail to produce the expected ground-truth label. One clear trend of the large model is the ability to generate “on topic” (i.e., in-domain) responses, regardless of whether or not those responses are correct.
For example, if the expected response to an input is “left”, a model which produces right is wrong in a very different way than a model that produces nonsensical outputs such as “[0][0]” or function words such as “the” (see Figure 20). We see that the largest 175B parameter model almost always produces in-domain answers. We evaluate this by checking whether the generation lies in the set of related words for that category (e.g., all colours, spatial, cardinal words respectively). We see that the smaller models fail to do this i.e., their generations tend to be unrelated words that might have had high prominence (e.g., function words). On evaluating accuracy of generations being “in-domain”, we see that the smallest model has only a 53% accuracy while the largest has a 98% accuracy of generating in-domain answers.

Second, we evaluate grounded distance (S7.3) to measure the degree of correctness of a response. There are often multiple reasonably correct ways to label a given instance of a grounded world. For example, in the colour domain, the colours dark red and wine are close enough in space that they might be intuitive alternate answers for one another. However, our top-1 and top-3 metrics only assess strict matches and do not account for this. Thus, we look at the distance between the color denoted by the predicted label (when it exists, i.e., when the model generated a legitimate color name) and the color the model was asked to label. The lower this distance is, the “less wrong” the model’s prediction is.

Table 3 reports evaluations measured by grounding distance for the colour domain. For every test instance, we compute the distance between the model predicted grounding and the true grounding as defined in Equation 7.5. We then average over all computed distances to report one number that tells us how close, on average, model predictions are to the true grounding in the world. We show example visualisations of colours in RGB space that lie within a certain distance threshold of each other. We see that the distances of predicted groundings from the true groundings reflect the models’ ability on the task and offers intuitive insights into the models reasoning process. Specifically, for the largest model, the distances between model-predicted groundings from the true grounding are significantly lower than random.
\[ d(c_1, c_2) = \begin{cases} \sqrt{(c_{1x} - c_{2x})^2 + (c_{1y} - c_{2y})^2 + (c_{1z} - c_{2z})^2} & \text{for } c_1 \in C \\ 500 & \text{for } c_1 \notin C \end{cases} \]

<table>
<thead>
<tr>
<th></th>
<th>124M</th>
<th>355M</th>
<th>774M</th>
<th>1.5B</th>
<th>175B</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>328.3</td>
<td>309.5</td>
<td>209.6</td>
<td>190.7</td>
<td>96.3</td>
</tr>
<tr>
<td>R-IV</td>
<td>334.9</td>
<td>334.9</td>
<td>334.9</td>
<td>334.9</td>
<td>334.9</td>
</tr>
<tr>
<td>R-ID</td>
<td>174.9</td>
<td>174.9</td>
<td>174.9</td>
<td>174.9</td>
<td>174.9</td>
</tr>
</tbody>
</table>

Table 9: Table shows average distance (lower is better) between model-predicted groundings and true groundings in the world, averaged over all instances in the test set. We see that the largest model has an average distance of predictions significantly lower than random.

<table>
<thead>
<tr>
<th>True G</th>
<th>Predicted G &amp; Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>dark red</td>
<td>wine (76.5), light crimson (208.1)</td>
</tr>
<tr>
<td></td>
<td>dark slate gray (144.7)</td>
</tr>
<tr>
<td>light green</td>
<td>beige (126.6), light sea green (129.7)</td>
</tr>
<tr>
<td></td>
<td>cerulean (185.7), violet (262.6)</td>
</tr>
</tbody>
</table>

Table 10: Table shows example model predictions (from the GPT-3 model) and distances from the true groundings in RGB space. The first column shows the true concept while the second column shows model predictions and their distances from the true concept.

7.6 Discussion

Our empirical results suggest that very large LMs (specifically, GPT-3), even when trained only on text, can learn to ground words to conceptual spaces to some degree. The fact that these models succeed even in
isomorphic rotated worlds, suggests that these models are not succeeding via naive memorisation. Rather, this suggests that they may be exploiting something about the conceptual structure of the space learned from text in order to map onto a new space that was not explicitly encountered during training.

A limitation of our approach however, is that there are some grounded concepts (e.g., visual and sensory inputs) that cannot be easily encoded in text form. By construction, the LMs that we use are restricted to text-only inputs, thus our focus is on domains (e.g., colours and directions) that have a well-defined world-space. This is only a small set of all the potential grounded concepts we would wish to teach LMs. Although many forms of data can be coerced into text format (e.g., we represent color using discrete digits to represent RGB space), complex concepts may lose fundamental aspects of their meaning when represented in this way. For example, for color, a coarse-grained notion of numeric proximity, derivable from text wallace-etal-2019-nlp, naik-etal-2019-exploring, may be sufficient to differentiate the concepts we explore, but for more complex visual inputs (e.g., the output of a CNN image encoder), a text-based numeric representation is unlikely to capture the necessary degree of nuance. Future work would need to consider ways of adapting GPT-3-like models to accept non-textual inputs while still exploiting the text-based conceptual structure.

If such limitations were addressed, our results are suggestive of a potentially promising way in which text-only training could support general purpose, grounded models of meaning. Specifically, our results imply that the conceptual space a model learns from text might be nearly isomorphic to what it would learn from interacting in a grounded world, and that models can be taught to map between those conceptual spaces without requiring explicit grounding for every concept. This is exciting, as domain-general text corpora are readily available, while domain general multimodal corpora—e.g., containing sufficient information on abstract concepts such as emotions (happy) or time (used to)—might be difficult or impossible to collect. If models like GPT-3 could be adapted to receive non-text prompts (as discussed above), our results suggest that the rich conceptual structure such models learn from text could be bootstrapped into powerful grounded
models of language.
CONCLUSION

In this dissertation, we have presented a family of techniques that allow language to be grounded to components of an agent’s model, when solving tasks in a world. We introduce a family of techniques for language grounding to the sequential decision processes of agents. We show that elements of language can be grounded not only to rewards, but to many different components of an MDP. A summary of the contributions of each part of this thesis is as below.

Part 1  We build weakly supervised algorithms to translate natural language instructions into temporal logical forms that agents can use to plan in complex environments. This method therefore allows natural language information to be incorporated into an agents reward function.

Part 2  We create a new formal language, one that grounds not only to rewards, but any component of an agent’s model i.e., a Markov Decision Process. We show the benefits of having such a framework, and what information might be lost when only grounding to one component of an MDP, as is prevalent in the literature.

Part 3  We show how we can take an un-grounded pre-trained language model and teach it grounded concepts in small grid worlds. We test the zero-shot and few-shot generalisation of models to uncover what types of concepts models can generalise on.

LIMITATIONS OF THIS THESIS

Despite the progress outlined in each section of this thesis, there are several limitations of these avenues of work that we outline in this section.
Limitations of grounding to logical forms  In Chapters 1 and 2, we saw the benefits of using different formal languages as intermediate representations of the meaning of natural language instructions. Using such logical languages might be preferable in terms of interpretability of language understanding systems or safety of the task-solving or planning mechanisms that take in logical forms. However, it is worth considering that the restricted syntax, semantics, and closed-form vocabulary of such languages differs so vastly from the fluidity of natural language, that it is sometimes impossible to represent the full meaning of a natural language sentence in the form of a logical form.

Limitations of current language modeling systems  In Chapter 4, we show how we can take an un-grounded pre-trained language model and teach it grounded concepts—and especially for the larger language models, they can transfer un-grounded knowledge to grounded knowledge surprisingly well. However, it is worth noting that these models are fundamentally, by virtue of their training process, limited in the knowledge that they have learned. Moreover, several works have pointed out clear fallacies in their reasoning approaches, and ways in which they can be deceived, thus portraying their lack of understanding of language. Future work that explores transferring un-grounded language models to grounded models, should explore the theoretical feasibility of such approaches, given how flawed the models are to begin with.

Limitations of single agent systems  When grounding language in agents’ worlds, all the chapters in this thesis focus only on single-agent settings i.e., where one agent interacts with the world to solve tasks specified in natural language. However, the world that we live in, is not a single-agent system but is inherently multi-agent. We learn behaviours through interacting not only with elements in the world, but other individuals, and this brings about a variety of desired behaviours (e.g., being prosocial or coordinating with others) and allows the realisation of a number of important tasks (e.g., communication between agents). Future work should build language grounding systems that allow for multiple agents to under-
stand and generate language for the environments and tasks discussed in this thesis.

8.1 What Comes Next

The recent success of large pre-trained language models, even though trained only in an ungrounded manner, opens up many possibilities for future work to build off of. Although most efforts in thesis focus on building grounded models of language (Chapters 3, 4 and 6) train models from scratch to learn language, in Chapter 7 we showed that it is possible to take a previously ungrounded model (e.g., a model trained on only textual data) and teach it to be grounded in small, simulated worlds.

Moreover, although this thesis explores several ways of grounding language to components of a single agent’s decision process, there is much work left to be done, when considering multiple agents. Although generative language models have proved to be remarkably adept at producing fluent streams of text, their ability to adapt to new communicative contexts is still lacking. However, their generative ability, as well as all the knowledge learned during pretraining, is an enticing factor, since if the context grounding problem is solved, these models should effectively be able to fluently converse with agents in new situations—something that has long been a challenge.

8.2 Closing Remarks

In conclusion, this thesis provides insight into the language grounding problem, this time situated in the context of existing neural language models that can be trained on large amounts of data. This thesis has explored two avenues of grounding such models—either by training them to learn such groundings from scratch, or by transferring textual information to be grounded after training. Considering the latter, this is an exciting time in the field of NLP, where large language models that have already been trained on large text corpora can be fine-tuned, or prompted, to use in new tasks. Give the rise of such models, there is much to be explored in
learning to transfer such information from textual models, in order to allow them to be grounded in the world.
References


for complex and cross-domain semantic parsing and text-to-sql task. 


