Contents

CHAPTER 1 Introduction

1.1 WHAT IS AI PLANNING?

Making plans is considered a sign of intelligence, and for this reason automated planning has been one of the goals of research in Artificial Intelligence (AI) since its beginning. With this goal, AI planning can be described as the study of computational models and methods of creating, analysing, managing, and executing plans.

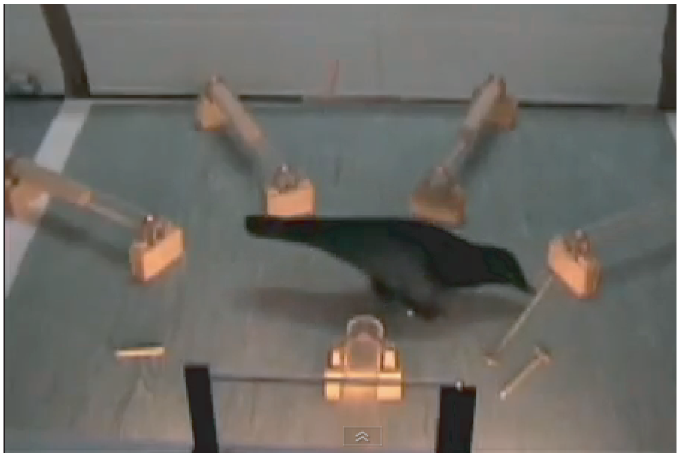

Figure 1.1: The New Caledonian crow has been observed, both in laboratory experiments and in the wild, to not only use tools to get food, but also to find and use tools to acquire better tools, and use those to get food. This could be considered a sign of planning. However, researchers are cautious about ascribing human-like cognitive processes to animals; other experiments have shown the crow failing to solve problems when it cannot directly observe the reward getting nearer. Image from Wimpenny et al. [2009].

From a more practical perspective, planning brings together AI knowledge representation and AI problem solving techniques. To apply a particular AI method (for example, heuristic search) to a problem, we must formulate a problem representation, and, most likely, devise an informative heuristic if we want the solver to be efficient. AI Planning places a third element between the problem to be solved and the solving method: a problem description language. This is a formal representation language (deriving some features from logic but adapted to express state transformation problems) in which we model the problem (see Figure 1.2).

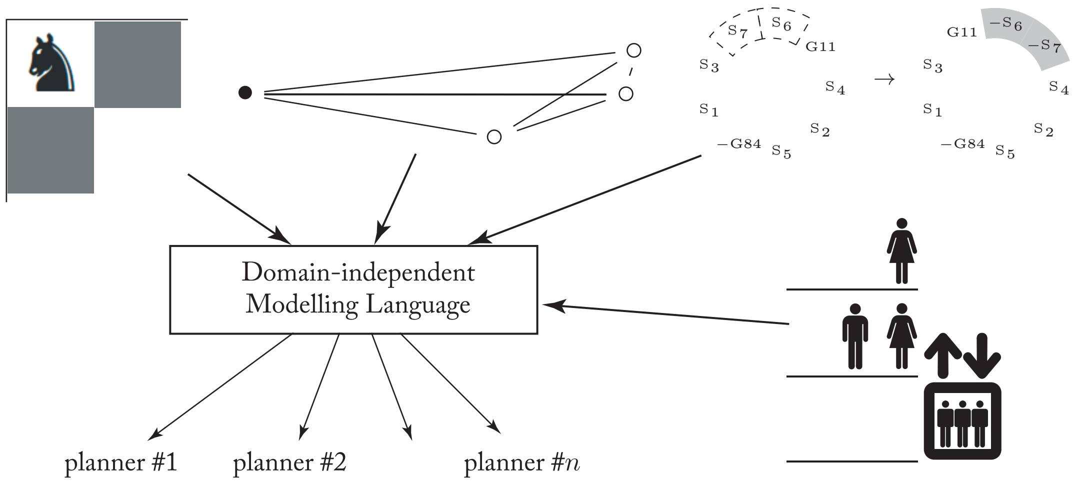

Figure 1.2: The idea of domain-independence in AI planning is to place a modelling language between the planning problem to be solved and the planning system that solves it. A problem, once modelled, can be solved (or at least we can try to solve it) with many different planners, and a planner can be applied to any problem that can be expressed in the modelling language.

An AI planning system ("planner", for short) takes the problem formalisation, or *model*, as input and uses some problem solving technique, such as heuristic search, propositional satisfiability, or other, to work out its solution. Turning the model into a search space or logical reasoning problem, creating heuristics to solve it efficiently, and so on, are problems for the designer of the planning system. The planner does not know, or need to know, what the formal problem description is about. It can be applied to any problem that can be expressed in the modelling language, though of course, not every planner will be able to solve every problem that can be given to it. This property of planners is known as *domain-independence*.

The *Planning Domain Definition Language* (PDDL) is one such problem description language. The syntax, semantics, and practical use of PDDL are the topic of this book. PDDL is not the only modelling language for planning problems (we briefly review other languages in Section 1.4.1), but owing to its role in the International Planning Competition (IPC), it is one of the most widely supported languages by planning systems. As we demonstrate throughout this book, it is also a language that has been extended in many different directions, to capture different kinds of planning problems.

The ability to plan hinges on two other capabilities: the ability to predict the consequences that the actions that can be taken will have in the world, and the ability to find, efficiently, in the space of all possible courses of action, the one that will lead to a good result. AI planning is very much focused on the second of these problems, that of finding a plan. The formalised problem description provides a predictive model from which the possible actions and the results of taking them can be deduced.

1.2 PLANNING MODELS

The plan creation aspect of AI planning deals mainly with *state transformation problems*. These are problems characterised by an initial state, a desired goal, and a set of possible actions that can be taken to change the state. A plan consists of actions that can be taken from the initial state, and which when taken will transform it into a state where the goal has been achieved[1]. Depending on the specific nature of the planning problem, the plan may be simply an ordered sequence of actions, or a schedule where actions are executed at particular times.

By making different assumptions about the model of actions, and its relation to the problem that is modelled, we arrive at different kinds of planning problems, with different difficulties. Different fragments of PDDL support the expression of several classes of planning problems.

- The simplest form of planning problem normally considered is the so-called "classical planning problem", which assumes a deterministic, discrete, and essentially non-temporal world model. We describe this fragment of PDDL in Chapters 2 and 3.
- Numeric planning relaxes the assumption of a discrete and finite model, allowing for the modelling of resources, general counting, positions in space, and many other things. We describe this fragment of PDDL in Chapter 4.
- Temporal planning extends the classical planning problem with scheduling planned actions in time. This introduces problems such as action (non-)concurrency, coordination, and scheduling actions in the presence of predictable events (for example, sunrise and sunset). We describe this fragment of PDDL in Chapter 5.
- Hybrid-system planning includes the problems above and in addition handles models with discrete modes and continuous change over time, by introducing continuous processes and exogenous events that can be triggered by plan actions or by changes in the environment. We describe this fragment of PDDL in Chapter 6.

1.3 EXAMPLES

1.3.1 THE KNIGHT'S TOUR

The knight's tour is a classic chess puzzle. In chess, the knight piece moves two spaces in any cardinal direction (horizontally or vertically) followed by one space in a perpendicular direction.

[1]As we will see in the following sections, planning does not always imply that the plan will be executed to bring about the goal condition. Many planning problems are hypothetical or counterfactual, in the sense that we seek a plan that could be, or could have been, performed by another actor.

Unlike other chess pieces, the knight "jumps" to the target position without stepping on intermediate squares. A knight's tour is a sequence of knight's moves that causes the piece to visit every square of a chess board, without stepping on any square twice. (Thus, it a special case of the Hamiltonian path problem.) An example of a tour, with the knight starting in the top-left corner of the chess board, is shown in Figure 1.3.

Puzzles like this are of course not the primary application of domain-independent planning, but they are useful illustrations of a certain class of planning problems[2]. The problem state can be represented as a set of facts (where the knight piece is, and which squares it has already visited) and the plan consists of a sequence of actions (the knight's moves) which transforms the starting state (the piece in its starting position, and no other square visited) to a final state satisfying the goal condition (every square visited). The actions must be formulated so that they capture all constraints of the problem: in particular, each action must not only correspond to a valid knight's move, but also enforce that the target square of the move is not already visited.

We will show how this problem can be modelled in PDDL in Chapter 2.

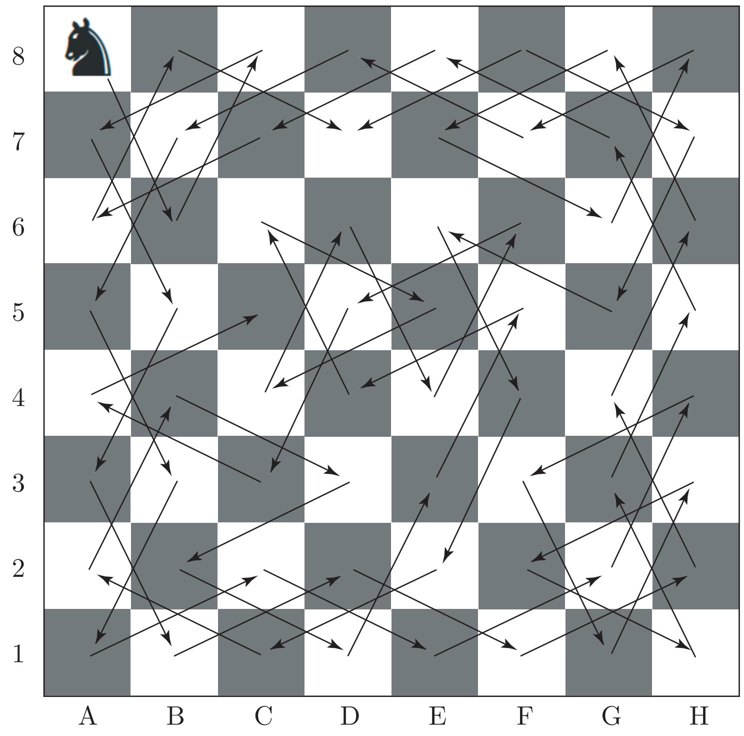

Figure 1.3: A knight's tour. The tour begins in A8 and ends in C5.

[2]This is the class of discrete and deterministic problems, which we will address in Chapters 2 and 3.

1.3.2 LOGISTICS

Planning problems arise from common operational decision problems in many business sectors. For example, managing logistics involves making decisions about when, where, and how to move and store goods. Whether it is the movement of conveyors in an automated greenhouse [Helmert and Lasinger, 2010], routing of shipments by a fleet of trucks, trains, and ships [Flórez et al., 2011], or serving passengers with a bank of elevators [Koehler and Ottiger, 2002], these can often be viewed and expressed as planning problems.

The vehicle routing problem (VRP) is a classic optimisation problem, which involves routing vehicles delivering goods to customers from a central depot so as to minimise the total distance driven [Dantzig and Ramser, 1959]. The VRP and many of its variations have a long history of study in operations research, leading to many highly efficient specialised algorithms. It is unlikely that the current generation of AI planning systems would be the best performing for this problem. However, most practical logistics problems feature additional complexities, such as time windows for deliveries, compatibility constraints between goods and vehicles (for example, refrigerated goods must be transported in a refridgerated truck), or between different types of goods (food cannot be mixed with hazardous chemicals), delivery in parts, or pickups as well as deliveries, different modes of transport (road, rail and ship, for example), and so on. A recent survey indicates that both research and commercial VRP software have yet to address many of these complications [Drexl, 2012].

Planning modelling languages, like PDDL, have the flexibility to express both the basic VRP and a wide range of extensions. We will look at modelling one particular VRP, investigated by Kilby et al. [2015], in Chapter 2.

1.3.3 PLANS AS EXPLANATIONS

Plans are not always intended for execution. In some applications of planning, the generated plan is hypothetical, demonstrating a sequence of actions that *may* have happened, or counterfactual, showing what actions could have happened if the starting state or goal had been different. Such hypothetical planning occurs in diagnostic reasoning, where the given information is a time-ordered set of observations of events that have occurred and a system model that defines which (observable and unobservable) events can occur. The task in this setting is to find one or more histories that explain the observations. Examples include alarm processing in technical networks [Haslum and Grastien, 2011], verifying and restoring consistency rules in a database [Boselli et al., 2014], and checking conformance with business process descriptions [De Giacomo et al., 2016]. The planning model typically includes some notion of "fault" events (for example, an incorrect entry made in a database), which must be hypothesised to have occurred when the observations are not consistent with the model of what should happen; the planning objective is to minimise the number, or aggregate likelihood, of the assumed faults occurring.

Another use of hypothetical plans is to measure their complexity. For example, cyber attack planning is a means of estimating how vulnerable a computer network is to cyber attacks, by planning from the perspective of a hypothetical attacker [Boddy et al., 2005, Hoffmann, 2015, Lucangeli Obes et al., 2010]. In phylogenetic analysis, the evolutionary distance between organisms can be estimated by the (weighted) shortest sequence of mutation events that can transform the genome of a hypothetical common ancestor into that of the organisms seen today. The problem of minimising such sequences of events can be formulated as a planning problem [Erdem and Tillier, 2005, Haslum, 2011].

1.3.4 PLAN-BASED CONTROL

Traditional control systems are reactive, though in many situations more efficient control can be achieved by optimising over a longer horizon. Planning can be seen as an extrapolation of receding horizon control, where a plan of control actions is computed to a goal state. Like in receding horizon control, uncertainty is dealt with by replanning. Plan-based control has been applied to realise the smart grid vision of power systems, solving problems such as the Unit Commitment problem, which is the problem of deciding which generators to switch on/off and when so as to meet forecasted demand [Piacentini et al., 2016], and voltage control [Piacentini et al., 2015]. On the energy consumer side, foresighted control of heating, ventilation, and air-conditioning (HVAC) systems in large commercial buildings can reduce energy usage by exploiting strategies such as cooling ahead of use and separating control of zones within a building, particularly when integrated with scheduling of the building's use, such as meetings or classes [Lim et al., 2015].

Control in many industrial or infrastructure systems is centred around flows, such as the flow of power in the electricity grid. Other examples include planning operations in chemical process industry [Aylett et al., 1998], and control of traffic flows via the traffic lights at intersections [Vallati et al., 2016].

In these applications, temporal and numeric planning, as discussed in Chapters 4, 5, and 6, is often required to express the dynamics of the system under control. The planners used have often been integrated with special-purpose solvers [e.g., Aylett et al., 1998, Piacentini et al., 2015], but as the capability of general-purpose planners to efficiently solve problems in more expressive fragments of PDDL improves, more and more such applications can be formulated purely in PDDL. For example, the traffic control problem modelled by Vallati et al. [2016] is formulated using processes in PDDL+. Management of multiple batteries, in which plan-based control is used decide which battery to use to serve the current load, is another example where processes are used to model the flow of charge [Fox et al., 2012].

1.3.5 PLANNING IN ROBOTICS

Planning the actions of a mobile robot is an appealing problem that has inspired AI planning research since its beginning. (The STRIPS planning system, which pioneered the precondition-effect model of classical planning, was intended for use on the robot Shakey; cf. Fikes and Nilsson [1971].) The problem of planning physically feasible, collision-free movements of a robot, whether it is a Mars rover or a multi-jointed arm, where the actions are low-level actuator controls, such as the speed of turning wheels or the degrees of joint movement, is known as *motion planning* and is well studied in robotics and AI [e.g., LaValle, 2006]. Motion planning uses models which are typically focused on geometry and kinematic constraints, and typically different from those naturally expressed in PDDL. (It is possible to express some forms of motion planning in the numeric and hybrid fragments of PDDL. We will see a simple example in Chapter 4.) However, the two forms of planning share many fundamental algorithmic ideas, and integrating motion planning with the more abstract, task-oriented planning usually represented in PDDL has been an active area of research [e.g., Cambon et al., 2009, Lagriffoul et al., 2012, Plaku and Hager, 2010, Srivastava et al., 2014, Toussaint, 2015]. Moreover, planning in a robotics context must often take account of uncertainty, due to the robot's noisy sensors and imprecise actuators. Nevertheless, mission- or task-level AI planning has found applications in planning for industrial robots [Asai and Fukunaga, 2014, Crosby et al., 2017], autonomous aircraft [Bernardini et al., 2017a] and underwater vehicles [Cashmore et al., 2014, McGann et al., 2008], space missions [Ai-Chang et al., 2004, Jonsson et al., 2000], and more. The ROSPlan library [Cashmore et al., 2015] allows any PDDL-capable planner to be integrated in the ROS framework. It handles invoking the planner, and plan dispatch and monitoring, and includes interfaces to several common ROS libraries.

1.3.6 NARRATIVE PLANNING

The plot of a story has some similarity with a plan, in that they are both sequences of events that change the state of the (story) world. For a story to be coherent and plausible, the event sequence must be logically possible and connected by causes and effects. This has inspired a long line of AI research into planning-based approaches to automating the creation or presentation of stories (cf. Gervás [2009] for a survey of early work, and Brenner [2010], Ware and Young [2014], and Porteous et al. [2013] for more recent examples).

Planning-based story generation raises many challenges: most actions in a story are taken by characters of the story, who, in order to be believable, must be seen as acting in a manner that is directed by their own goals and desires. Yet characters' goals are often in conflict, and in conflict with the goals of the author. For example, in a Shakespearian tragedy it is often the author's goal that all the main characters die, but that is certainly not the intention of the characters, who fight against their fate. If instead they went helpfully to their graves in Act 1, the story would be both boring and unbelievable. This has been addressed by modifying planners to consider character goals [e.g., Ware and Young, 2014], multi-agent planning and plan simulation [e.g., Brenner, 2010], or re-formulating the problem to make character's goals and reasoning explicit [e.g., Chang and Soo, 2008, Haslum, 2012].

Another difficulty is the scale of the problem. To permit the planner to come up with varied stories, so that its creative potential is not unnecessarily restricted, the planning model must include a sufficient breadth of events that could occur in the story world. Creating this model by hand places an impractical burden on system designers, driving investigations into methods to automatically or semi-automatically build or extend planning models [see, e.g., Chambers and Jurafsky, 2009, Porteous et al., 2015, Sil and Yates, 2011].

Finally, plot creation differs from most other applications of planning in that it is unclear how to measure the quality of a candidate plan. While in most other cases the objective is to minimise plan cost or complexity, a story must have a certain amount of complexity to make it interesting. A different way to approach the difficulty of judging what is a good story plan is to generate a collection of plans that is *diverse* and leave humans to make the final choice (see, for example, Srivastava et al. [2007] or Goldman and Kuter [2015] for discussion of measures of plan diversity and how to generate diverse plans).

1.4 THE ORIGINS OF PDDL AND THE SCOPE OF THIS BOOK

Working out a course of action to achieve a goal is one of the earliest applications of logical reasoning in AI [e.g., Green, 1969]. The STRIPS planner ["*STanford Research Institute Problem Solver*", Fikes and Nilsson, 1971] is often credited as the first system to implement a dedicated algorithm for planning, and although this algorithm has long since been superseded, its approach to modelling actions, with preconditions, positive and negative effects expressed as sets of atomic facts, survives to this day. (The simplest subset of PDDL, which we present in Chapter 2, is commonly referred to as "STRIPS".)

PDDL was created as a common language for the first International Planning systems Competition (IPC) in 1998 [McDermott, 2000]. PDDL version 1 was defined by a committee of researchers in the field of AI planning, led by Drew McDermott [McDermott et al., 1998]. New versions of PDDL have been proposed in conjunction with later editions of the planning systems competition. The subsequent versions have expanded the language with new features to express kinds of planning problems of different kinds (e.g., numeric and temporal planning), with the aim of directing researchers to develop planning systems capable of handling these problem classes. Over time, however, language features that failed to gain popularity have also been removed. The PDDL versions published so far are as follows.

- PDDL 1 [McDermott et al., 1998].
- PDDL 1.2 [Bacchus, 2000]. This version trimmed some unused features from the language, and corresponds to most of the classical planning formalism presented in Chapters 2–3.
- PDDL 2.1 [Fox and Long, 2003]. This version extended the language to represent numeric planning (presented in Chapter 4) and temporal planning problems (presented in Chapter 5). Exploiting the expressivity of the numeric planning extension, it also introduced a syntax for specifying an objective function for optimisation.
- PDDL+ [Fox and Long, 2006]. This version further extended the numeric and temporal representation to hybrid planning (presented in Chapter 6).

- PDDL 2.2 [Edelkamp and Hoffmann, 2004]. This version added two features: Axioms (described in Chapter 3), which add more expressive conditions to classical planning, and timed initial literals, which provide a syntactic convenience for defining a schedule of predictable events in temporal planning.
- PDDL 3.0 [Gerevini et al., 2009]. This version added syntax for temporally extended goals and preferences to classical planning. This is covered in Chapter 3.
- PDDL 3.1 [Helmert et al., 2008]. This version defined the restricted syntax for specifying action costs (described in Chapter 2). It also introduced general finite-domain state variables, known as "object fluents". We do not cover this extension in this text.

Although PDDL is intended to be a common modelling language, for a certain class of planning problems, it is important to recognise that it is not a standard. It is better understood as a community effort to promote interchangeability and ease of application of planning systems. Discrepancies between the different versions, and differing styles and levels of formality in the articles and technical reports cited above, mean that there is no complete and unambiguous specification of the syntax or semantics of all of PDDL. Moreover, planners usually support only a subset of (some version of) PDDL, and different systems' implementors have interpreted the ambiguous aspects of the language in different ways.

Yet, there is a core of PDDL about which there is a reasonably broad consensus of understanding. Our approach in this book is to focus on that core. We cover elements from all PDDL versions, but have omitted some of the unclear, unused, or as yet underdeveloped parts of the language from each. As much as practical, we also point out specific areas where we are aware that different planners may interpret the language differently.

1.4.1 OTHER PLANNING LANGUAGES

Although PDDL is the modelling language used by many planners, it is far from the only planning language. What defines domain-independent planning is the idea of using some declarative problem definition formalism, not what that formalism is. Many different classes of planning problems can be distinguished, and while different fragments of PDDL capture some of them, there are others that do not fit into PDDL at all. This book is about PDDL, and planning problems that can be expressed in PDDL. In this section we briefly review some alternative planning languages, and the kinds of problems that they are intended for.

Extensions of PDDL have been defined to model problems with uncertainty, in the form of a partially known initial state and actions with nondeterministic effects. This uncertainty may be quantified with probabilities [Younes and Littman, 2004] or express only a set of possibilities. Planning, when viewed as a state transformation problem, with probabilistic uncertainty closely resembles solving an MDP or POMDP [Mausam and Kolobov, 2012, Puterman, 1994]. RDDL [Sanner, 2011] is a language designed for probabilistic planning. While it's syntax is quite different from PDDL, it is based on the same state-transition model. Another example that extends PDDL is MA-PDDL [Kovacs, 2012], which is designed to model multi-agent planning problems for planning both *for* and *by* several agents. We describe some of the languages most closely related to PDDL in greater detail later in Chapter 7.

The timeline-based planning model [e.g., Ghallab and Laruelle, 1994, Muscettola, 1994] was developed for temporal planning problems with a significant scheduling component. Fundamental to this model is a timeline, which represents the evolution of a state variable over time. State persistence and state change are treated uniformly, and actions are modelled mainly through sets of constraints that they impose across timelines. Initial conditions and goals are also expressed as constraints over timelines, both possibly referring to several points in time. Languages for modelling problems of this kind are mostly specific to planning systems [e.g., Barreiro et al., 2012, Chien et al., 2000, Frank and Jonsson, 2003]. ANML [Smith et al., 2008] is a recent effort to create a unifying modelling language for timeline-based planning.

Hierarchical task network (HTN) planning embodies a different view of what planning is. Instead of changing the state of the world model, a HTN planning problem is defined as a set of abstract *tasks* to be done, and set of methods for each task that represent different ways in which it can be carried out (although methods may also have conditions and effects on state). A plan recursively decomposes the given abstract tasks into primitive tasks, which are concrete, executable actions. There is no widely shared modelling language for HTN planning, as different HTN planners use their own input languages [e.g., Nau et al., 2005, Tate et al., 1994, Wilkins, 1986]. However, ANML also includes some hierarchical planning features.

Related to both timeline-based and hierarchical planning, languages like RMPL [Effinger et al., 2009, Williams and Ingham, 2002] and Golog [Levesque et al., 1997] express planning problems as nondeterministic programs. They can use programming constructs, such as conditional branching or iteration, as well as temporal constraints, to express a high-level strategy. The task of the planning system becomes to instantiate the unspecified choices in the program to achieve a successful execution from a given initial situation.

1.4.2 RELATION TO NON-PLANNING FORMALISMS AND OTHER FIELDS

AI planning is not the only area of computer science that defines a class of problems via a modelling framework and aims to develop general solvers for the class. Propositional satisfiability (SAT) [Biere et al., 2009], its extension to quantified boolean formulas (QBF) [Nethercote et al., 2007, Schaefer, 1978], (finite-domain) constraint programming (CP) [Rossi et al., 2006], and mathematical programming formalisms such as linear programming (LP) and mixed-integer programming (MIP) [Dantzig, 1960, Wolsey, 1998] are prominent examples. The emphasis on the modelling language varies: for example, there are many different formats for specifying a MIP, yet there is a common, unambiguous definition of what is the class of MIP problems. Likewise, the emphasis on domain-independence, in the sense of fully automatic generality, of solvers varies. SAT solvers, like AI planners, are typically intended to work on any input without problem-specific configuration; in CP, and to some degree MIP, on the other hand, the modelling of a problem and customisation of the solver often go hand in hand.

In terms of the complexity of problems that can be expressed, AI planning, even when restricted to only the classical subset of PDDL, sits in a higher complexity class than many other general-purpose formalisms (PSPACE-complete for grounded PDDL, EXPSPACE-complete for parameterised PDDL). Some of the more expressive fragments of PDDL are undecidable in general. In contrast, SAT, CP, and MIP are all in NP, and thus comparatively "easy". QBF is also PSPACE-complete, yet very different in the types of problems it naturally models. Many AI planners are built on heuristics, and often are able to find some solution (plan) for a complex problem quickly, but offer no guarantee of optimality, or even completeness. Complete and optimal planners typically target subclasses of planning problems, such as classical planning with additive action costs. In spite of the difference in worst-case complexity, reductions of planning into other formalisms have been used effectively [Kautz and Selman, 1996, van Beek and Chen, 1999]. Fixing a low polynomial bound on plan length (which in theory sacrifices completeness, but often works in practice) allows, for example, classical planning to be reduced to SAT or CP, and numeric planning to be reduced to MIP.

Another computer science area that is closely related is model checking [Clarke et al., 1993], which is concerned with verifying (or disproving) properties of reactive, usually discrete-state, systems, such as for example communications protocols. These properties, usually expressed in temporal logics such as LTL, correspond to the existence or non-existence of execution traces, i.e., sequences of transitions, in the system. There is a clear analogy with planning, as a plan is also a sequence of transitions in the state space of the planning problem. Reductions of both classical planning and other planning variants to model checking have been shown [e.g., Giunchiglia and Traverso, 1999], as well as translations in the other direction, i.e., stating a verification problem as a planning problem [Camacho et al., 2018, Ghosh et al., 2015, Patrizi et al., 2013]. We will show an example, encoding the famous "dining philosophers" deadlock detection problem in PDDL, in Section 2.4.2.

However, there is a clear difference in emphasis: in planning, the emphasis is on efficiently finding a plan, and in some cases on finding a plan of good quality, while in model checking the emphasis is on proving that the specification is satisfied, which in many cases translates to proving that no plan exists. The problem of proving plan non-existence has received increased attention in planning research recently [see, e.g., Muise and Lipovetzky, 2016], and many of the techniques developed in the field of model checking, which are useful not only for proving plan non-existence but also in proving plan optimality, have been adapted into planners.

1.5 PLANNING SYSTEMS AND MODELLING TOOLS

The main purpose of modelling planning problems in PDDL is to apply automated planning systems to find solution plans. However, there are also other tools that can be valuable in the process of writing (and debugging) a PDDL model.

http://planning.domains is an on-line repository of planning benchmark models, which also includes an on-line editor with PDDL-specific features such as syntax highlighting and semi-automatic instantiation of some common model patterns. Throughout the book, we provide links that preload examples into this editor. It also provides an interface to a planner as a web service, and from within the editor. However, the planner it makes available does not support all of PDDL, and as it has a runtime limit of 15 s it is restricted to solving small problems.

For readers who want to try other planners, we can suggest exploring the archive of planning competition web pages at http://icaps-conference.org/index.php/Main/Competitions (the "deterministic" tracks). Each of the competitions since 2008 has provided links to the source code of participating planning systems. However, these are "snapshots" of the planners at the time of the competition, and many of them are not maintained.

The VAL tool suite (https://github.com/KCL-Planning/VAL) includes a PDDL syntax checker and a plan validator. A plan validator is a tool that takes as input a problem definition (in PDDL) and a plan, and determines if the plan solves the problem. Validating manually written plans can be a useful approach to debug the problem definition. An alternative implementation of a plan validator for PDDL is INVAL (https://github.com/patrikhaslum/INVAL).

As we have mentioned, PDDL is not a standard, and there are some aspects of the language where its specification is vague, ambiguous, or even missing. Moreover, planners and plan validators are research prototypes. They usually support only a subset of PDDL, their interpretation of some aspects of the language may differ, and, like any complex software, they may have bugs. Even the plan validators occasionally disagree on whether a problem definition or a plan is valid. When modelling a problem in PDDL, one must always be prepared to reformulate the model to work around planners' limitations and quirks.