Chapter 1
Introduction and Overview

1.1 First Intuitions on Planning

Planning is the reasoning side of acting. It is an abstract, explicit deliberation process that chooses and organizes actions by anticipating their expected outcomes. This deliberation aims at achieving as best as possible some prestated objectives. Automated planning is an area of Artificial Intelligence (AI) that studies this deliberation process computationally.

Some of our actions require planning, but many do not. In our everyday activities we are always acting, and we do anticipate the outcome of our actions, even if we are not fully aware of anticipating [64]. But we act much more frequently than we explicitly plan our actions: we are not often conscious of performing an explicit deliberation process prior to acting. When the purpose of an action is immediate given our knowledge of that action, or when we perform well-trained behaviors for which we have prestored plans, or when the course of an action can be freely adapted while acting, then we usually act and adapt our actions without explicitly planning them.

A purposeful activity requires deliberation when it addresses new situations or complex tasks and objectives or when it relies on less familiar actions. Planning is also needed when the adaptation of actions is constrained, for example, by a critical environment involving high risk or high cost, by a joint activity with someone else, or by an activity to be synchronized with a dynamic system. Since planning is a very complicated, time-consuming, and costly process, we resort to planning only when it is strictly needed or when the trade-off of its cost versus its benefit compels us to plan. Furthermore, we generally seek only good, feasible plans rather than optimal plans[480].

One motivation for automated planning is very practical: designing information processing tools that give access to affordable and efficient planning resources. Some professionals face complex and changing tasks that involve demanding safety and/or efficiency requirements. Imagine, as an example, a rescue operation after a natural disaster such as an earthquake or a flood. That operation may involve a large number of actors and require the deployment of a communication and transportation infrastructure. It relies on careful planning and the assessment of several alternate plans. But it is also time constrained and it demands immediate

decisions that must be supported with a planning tool. The need for such a tool is also felt by organizers of simpler and more mundane tasks such as organizing a social meeting or professional travel for a group of persons. At the individual level, a planning resource that is seamlessly integrated with electronic organizers or web services could be of great benefit in handling constraints, offering alternate plans not yet considered, and pointing out critical actions and what may need to be relaxed [443, 445].

Another motivation for automated planning is more theoretical. Planning is an important component of rational behavior. If one purpose of AI is to grasp the computational aspects of intelligence, then certainly planning, as the reasoning side of acting, is a key element in such a purpose. The challenge here is to study planning not as an independent abstract process but as a fully integrated component of deliberative behavior.

An important combination of the practical and the theoretical motivations for automated planning is the study and design of autonomous intelligent machines. Such a study is of practical concern because some of our complex artifacts such as satellites and spacecraft require autonomy and deliberative behavior. These artifacts cannot always be teleoperated because of operational constraints or because of a demanding interaction with nonexpert humans that is more natural at the task level than at the low level of control signals. The study of autonomous intelligent machines is also of theoretical concern because planning as a fully integrated component of deliberative behavior requires embodiment into a machine that can sense and act as well as it reasons on its actions.

1.2 Forms of Planning

Because there are various types of actions, there are also various forms of planning. Examples include path and motion planning, perception planning and information gathering, navigation planning, manipulation planning, communication planning, and several other forms of social and economic planning.

Path and motion planning is concerned with the synthesis of a geometric path from a starting position in space to a goal and of a control trajectory along that path that specifies the state variables in the configuration space of a mobile system, such as a truck, a mechanical arm, a robot, or a virtual character. Motion planning takes into account the model of the environment and the kinematic and dynamic constraints of the mobile system. It is a well-advanced area that appears today to be quite mature and offers a wide range of efficient and reliable methods.

Motion planning can be seen as a particular case of the generic problem of planning control actions for dynamic systems. While motion planning seeks a trajectory in the configuration space of a mobile system, the generic planning problem is concerned with more abstract state spaces. Consider, as an example, the problem of controlling a ceramic plant or a blast furnace from its current state to a desired state. When these two states are close enough, a corrective control action can be computed

from their difference. If the desired state is too far apart from the current one (e.g., for driving the furnace to a shutdown state), then a sequence of control actions that meets some constraints and criteria must be planned.

Perception planning is concerned with plans involving sensing actions for gathering information. It arises in tasks such as modeling environments or objects, identifying objects, localizing through sensing a mobile system, or more generally identifying the current state of the environment. An example of these tasks is the design of a precise virtual model of an urban scene from a set of images. Perception planning addresses questions such as which information is needed and when it is needed, where to look for it, which sensors are most adequate for this particular task, and how to use them. It requires models of available sensors and their capabilities and constraints. It relies on decision theory for problems of which and when information is needed, on mathematical programming and constraint satisfaction for the viewpoint selection and the sensor modalities.

Data gathering is a particular form of perception planning which is concerned not with sensing but instead with querying a system: e.g., testing a faulty device in diagnosis or searching databases distributed over a network. The issues are which queries to send where and in what order.

Navigation planning combines the two previous problems of motion and perception planning in order to reach a goal or to explore an area. The purpose of navigation planning is to synthesize a policy that combines localization primitives and sensor-based motion primitives, e.g., visually following a road until reaching some landmark, moving along some heading while avoiding obstacles, and so forth.

Manipulation planning is concerned with handling objects, e.g., to build assemblies. The actions include sensory-motor primitives that involve forces, touch, vision, range, and other sensory information. A plan might involve picking up an object from its marked sides, returning it if needed, inserting it into an assembly, and pushing lightly till it clips mechanically into position.

Communication planning arises in dialog and in cooperation problems between several agents, human or artificial. It addresses issues such as when and how to query needed information and which feedback should be provided.

There is a wide range of other kinds of planning problems, particularly in social and economic realms. As examples, urban planning involves the deployment and organization of an urban infrastructure (e.g., public transportation, schools, hospitals) in order to meet the needs of a community, family planning deals with demography, and financial planning focuses on narrow financial optimization.

1.3 Domain-Independent Planning

A natural approach for these diverse forms of planning is to address each problem with the specific representations and techniques adapted to the problem. One develops predictive models for the type of actions to be planned for and for the

states of the system in which they take place. Computational tools for running these models, in order to predict and assess the effects of alternate actions and plans in various situations, exploit the specifics of the domain. For example, geometry, kinematics, and dynamics are the tools needed for motion or manipulation planning. Mathematical programming and optimization techniques are the tools widely used in various forms of economic planning.

These *domain-specific* approaches to specific forms of planning are certainly well justified. They are highly successful in most of the application areas mentioned earlier. However, they are frustrating for several reasons.

- Some commonalities to all these forms of planning are not addressed in the domain-specific approaches. The study of these commonalities is needed for understanding the process of planning; it may help improve the domain-specific approaches.
- It is more costly to address each planning problem anew instead of relying on and adapting some general tools.
- Domain-specific approaches are not satisfactory for studying and designing an autonomous intelligent machine. Its deliberative capabilities will be limited to areas for which it has domain-specific planners, unless it can develop by itself new domain-specific approaches from its interactions with its environment.

For all these reasons, automated planning is interested in *domain-independent* general approaches to planning. For solving a particular problem, a domain-independent planner takes as input the problem specifications and knowledge about its domain. Automated planning is not meant to be opposed to domain-specific planning techniques, just as automated reasoning is not intended to replace every arithmetic and floating-point calculus processor and other specialized reasoning techniques in a computer. Because planning is the reasoning side of acting, the purpose of automated planning is to develop general approaches to this particular form of reasoning that may build on and can be well integrated with domain-specific planning tools.

Domain-independent planning relies on abstract, general models of actions. These models range from very simple ones that allow only for limited forms of reasoning to models with richer prediction capabilities. There are in particular the following forms of models and planning capabilities.

- *Project planning*, in which models of actions are reduced mainly to temporal and precedence constraints, e.g., the earliest and latest start times of an action or its latency with respect to another action. Project planning is used for interactive plan edition and verification. A possible plan is given by the user as input to a project-planning tool that checks the feasibility of the constraints and computes several useful attributes of the given plan such as its critical paths. Here, the models of the actions in the plan (i.e., their effects and their interactions) remain mostly in the mind of the user.

- *Scheduling and resource allocation,* in which the action models include the above types of constraints plus constraints on the resources to be used by each action. A scheduling tool takes as input the actions to be carried out together with resource constraints and optimization criteria. The tool returns a temporal organization and resource allocation for the given actions in order to meet all the constraints and optimize the criteria.
- *Plan synthesis,* in which the action models enrich the precedent models with the conditions needed for the applicability of an action and the effects of the action on the state of the world. A plan synthesis tool takes as input the models of all known actions, a description of the state of the world, and some objective. The tool returns an organized collection of actions whose global effect, if they are carried out and if they perform as modeled, achieves the objective.

Automated planning is concerned with the general form of plan synthesis. Although it is still in its early stages theoretically, automated planning already is advanced enough to be useful in demanding applications, several of which are discussed in Part VI of this book.

Automated planning already has several success stories; one is the control of the spacecraft *Deep Space 1.* This spacecraft was launched from Cape Canaveral on October 24, 1998, and was retired on December 18, 2001, after it completed its mission successfully by encountering Comet Borrelly and returning the best images and other science data ever returned about a comet. The *Deep Space 1* mission successfully tested a number of advanced technologies. Among these was the Autonomous Remote Agent software system. The Autonomous Remote Agent, which was based on automated planning techniques, successfully operated *Deep Space 1* between May 17 and May 21, 1999. Chapter 19 discusses *Deep Space 1* in greater detail.

1.4 Conceptual Model for Planning

A conceptual model is a simple theoretical device for describing the main elements of a problem. It can depart significantly from the computational concerns and algorithmic approaches for solving that problem. However, it can be very useful for explaining basic concepts, for clarifying restrictive assumptions, for analyzing requirements on representations and trade-offs, and for proving semantic properties.

Since planning is concerned with choosing and organizing actions for changing the state of a system, a conceptual model for planning requires a general model for a dynamic system. Most of the planning approaches described in this book rely on a general model, which is common to other areas of computer science, the model of *state-transition systems* [146] (also called *discrete-event systems*).

Formally, a state-transition system is a 4-tuple $\Sigma = (S, A, E, \gamma)$, where:

- $S = \{s_1, s_2, \ldots\}$ is a finite or recursively enumerable set of states;

- $A = \{a_1, a_2, \ldots\}$ is a finite or recursively enumerable set of actions;
- $E = \{e_1, e_2, \ldots\}$ is a finite or recursively enumerable set of events; and
- $\gamma: S \times A \times E \rightarrow 2^S$ is a state-transition function.

A state-transition system may be represented by a directed graph whose nodes are the states in S. If $s' \in \gamma(s, u)$, where u is a pair $(a, e), a \in A$ and $e \in E$, then the graph contains an arc from s to s' that is labeled with u. Each such arc is called a *state transition.* It can be convenient to introduce a neutral event ϵ to account for transitions that are due only to actions and, symmetrically, a neutral action no-op for denoting transitions caused solely by an event. We write $\gamma(s, a, \epsilon)$ as $\gamma(s, a)$ and $\gamma(s, \text{no-op}, e)$ as $\gamma(s, e)$.

Example 1.1 Figure 1.1 shows a state-transition system involving a container in a pile, a crane that can pick up and put down the container, and a robot that can can carry the container and move it from one location to another. Here, the set of states is $\{s_0, s_1, s_2, s_3, s_4, s_5\}$, the set of actions is {take, put, load, unload, move1, move2}, and there are no events. The arc (s_0, s_1) is labeled with the action take, the arc (s_4, s_5) with the action move2, and so forth. Each state transition is *deterministic,* i.e., it leads to just one other state. ■

Both events and actions contribute to the evolution of the system. The difference lies in whether the planner has any control over them or not. *Actions* are transitions that are controlled by the plan executor. If a is an action and $\gamma(s, a)$ is not empty, then action a is *applicable* to state s; applying it to s will take the system to some state in $\gamma(s, a)$. *Events* are transitions that are *contingent*: instead of being controlled by the plan executor, they correspond to the internal dynamics of the system. They should be taken into account by planning, but they cannot be chosen or triggered. If e is an event and $\gamma(s, e)$ is not empty, then e may *possibly* occur when the system is in state s; its occurrence in s will bring the system to some state in $\gamma(s, e)$.

A precise specification of the semantics of the transition function, as defined by $\gamma : S \times A \times E \rightarrow 2^S$, requires a refinement of this conceptual model. One model, the Markov game model, supposes that no action takes place in states where events occur and vice versa. That is, S is partitioned into *action states* and *event states.* An alternative model is to suppose that actions can "compete" with events in the same states. That is, if we apply a to s and $\gamma(s, e)$ is not empty, then the next state can be any element of $\gamma(s, a, e)$.[1]

Given a state transition system Σ, the purpose of planning is to find which actions to apply to which states in order to achieve some objective when starting

1. We will refer in Chapter 14 to a model where an action with time-dependent effects makes a state transition and start a process that triggers, at some later state, an event that causes another state transition [194].

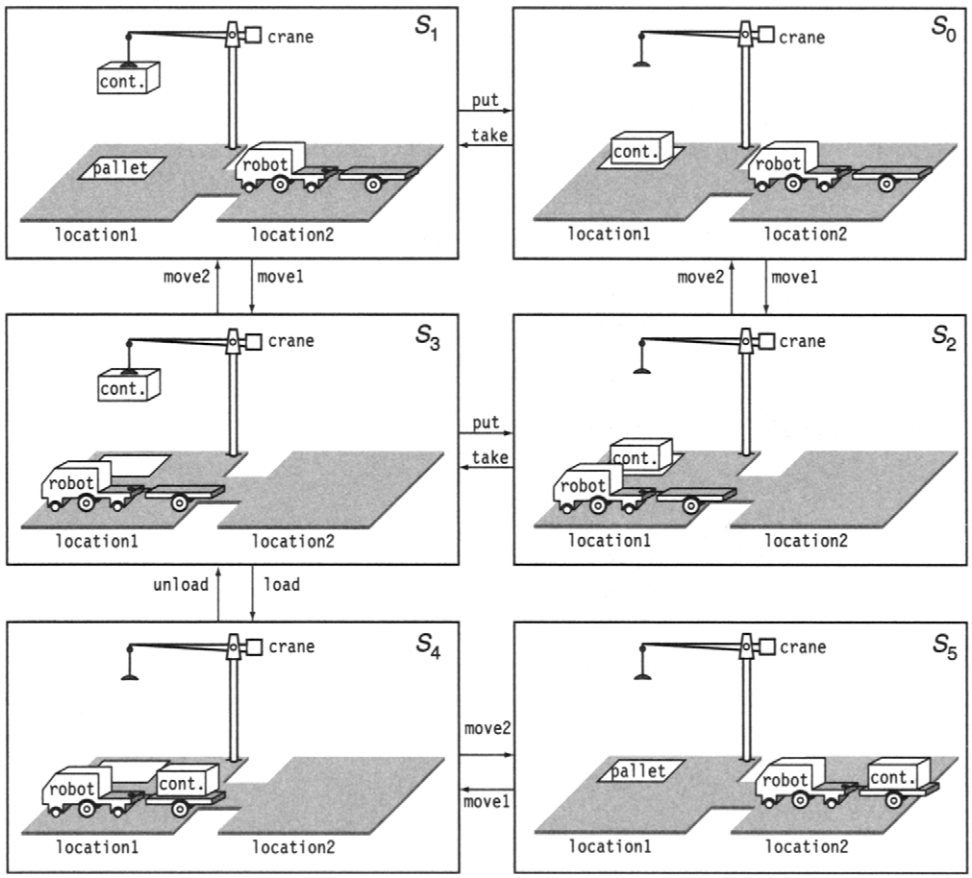

Figure 1.1 A state-transition system for a simple domain involving a crane and a robot for transporting containers.

from some given situation. A *plan* is a structure that gives the appropriate actions. The objective can be specified in several different ways.

- The simplest specification consists of a *goal state* s_g or a set of goal states S_g. In this case, the objective is achieved by any sequence of state transitions that ends at one of the goal states. For example, if the objective in Figure 1.1 is to have the container loaded onto the robot cart, then the set of goal states is $S_g = \{s_4, s_5\}$.
- More generally, the objective is to satisfy some condition over the sequence of states followed by the system. For example, one might want to require states to be avoided, states the system should reach at some point, and states in which it should stay.

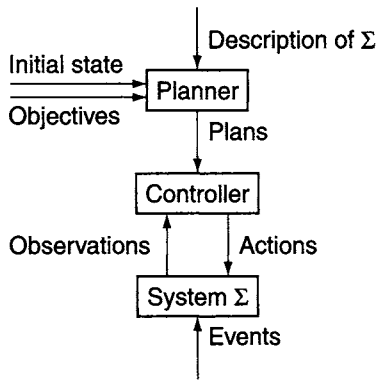

Figure 1.2 A simple conceptual model for planning.

- An alternative specification is through a utility function attached to states, with penalties and rewards. The goal is to optimize some compound function of these utilities (e.g., sum or maximum) over the sequence of states followed by the system.
- Another alternative is to specify the objective as tasks that the system should perform. These tasks can be defined recursively as sets of actions and other tasks.

It is convenient to depict this model through the interaction between three components (Figure 1.2).

1. A *state-transition system* Σ evolves as specified by its state-transition function γ, according to the events and actions that it receives.
2. A *controller*, given as input the state s of the system, provides as output an action a according to some plan.
3. A *planner*, given as input a description of the system Σ, an initial situation, and some objective, synthesizes a plan for the controller in order to achieve the objective.

An important element must be added to this model: the information that the controller has about the current state of the system. In general, this information is not complete. Partial knowledge about the state can be modeled as an observation function $\eta: S \to O$ that maps S into some discrete set $O = \{o_1, o_2, \ldots\}$ of possible observations. The input to the controller is then the observation $o = \eta(s)$ corresponding to current state.

Notice that the controller performs its task along with the dynamics of the state-transition system. It works *online* with Σ. On the other hand, the planner is not directly connected to Σ. It works *offline*. It relies on a formal description of the

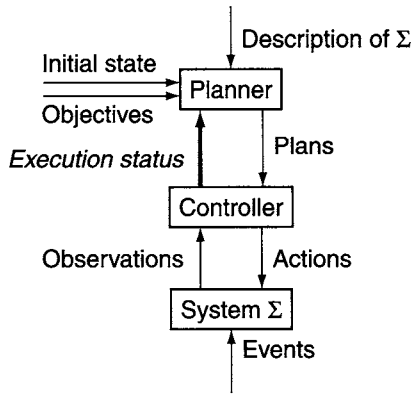

Figure 1.3 A conceptual model for dynamic planning.

system, together with an initial state for the planning problem and the required goal. It is concerned not with the actual state of the system at the time the planning occurs but instead with what states the system may be in when the plan is executing.

Most of the time, there are differences between the physical system to be controlled and its model, as described by Σ. In general, planning is restricted to the model formally represented by Σ. The controller is supposed to be robust enough to cope with the differences between Σ and the real world. Dealing with observations that depart from what is expected in the plan requires more complex control mechanisms than are needed for just observing the state and applying the corresponding action. Several enhancements to this conceptual model are required to address a deliberative goal-directed behavior. A more realistic model interleaves planning and acting, with plan supervision, plan revision, and replanning mechanisms. There is a need for a closed loop between the planner and the controller (Figure 1.3). The latter returns to the planner the *execution status* of the plan to enable dynamic planning.

1.5 Restricted Model

This conceptual model is not meant to be directly operational. Instead, it will be used throughout this book as a reference for our representations. Let us consider it as a starting point for assessing various restrictive assumptions, particularly the following ones.

Assumption A0 (Finite Σ). The system Σ has a finite set of states.

Assumption A1 (Fully Observable Σ). The system Σ is *fully observable*, i.e., one has complete knowledge about the state of Σ. In this case the observation function η is the identity function.

Assumption A2 (Deterministic Σ). The system Σ is *deterministic*, i.e., for every state s and for every event or action u, $|\gamma(s, u)| \leq 1$. If an action is applicable to a state, its application brings a deterministic system to a single other state; similarly for the occurrence of a possible event.

Assumption A3 (Static Σ). The system Σ is *static*, i.e., the set of events E is empty. Σ has no internal dynamics; it stays in the same state until the controller applies some action.[2]

Assumption A4 (Restricted Goals). The planner handles only *restricted goals* that are specified as an explicit goal state s_g or set of goal states S_g; the objective is any sequence of state transitions that ends at one of the goal states. Extended goals such as states to be avoided and constraints on state trajectories or utility functions are not handled under this restricted assumption.

Assumption A5 (Sequential Plans). A solution plan to a planning problem is a linearly ordered finite sequence of actions.

Assumption A6 (Implicit Time). Actions and events have no duration; they are instantaneous state transitions. This assumption is embedded in state-transition systems, a model that does not represent time explicitly.

Assumption A7 (Offline Planning). The planner is not concerned with any change that may occur in Σ *while* it is planning; it plans for the given initial and goal states regardless of the current dynamics, if any.

The simplest case, which we will call the *restricted model*, combines all eight restrictive assumptions: complete knowledge about a deterministic, static, finite system with restricted goals and implicit time. Here planning reduces to the following problem:

> Given $\Sigma = (S, A, \gamma)$, an initial state s_0 and a subset of goal states S_g, find a sequence of actions $\langle a_1, a_2, \ldots, a_k \rangle$ corresponding to a sequence of state transitions $(s_0, s_1, \ldots, s_k)$ such that $s_1 \in \gamma(s_0, a_1)$, $s_2 \in \gamma(s_1, a_2), \ldots, s_k \in \gamma(s_{k-1}, a_k)$, and $s_k \in S_g$.

Since the system is deterministic, if γ is applicable to s then $\gamma(s, a)$ contains one state s'. To simplify the notation, we will say $\gamma(s, a) = s'$ rather than $\gamma(s, a) = \{s'\}$. For this kind of system, a plan is a sequence $\langle a_1, a_2, \ldots, a_k \rangle$ such that $\gamma(\gamma(\ldots\gamma(\gamma(s_0, a_1), a_2), \ldots, a_{k-1}), a_k)$ is a goal state.

The assumption about complete knowledge (Assumption A0) is needed only at the initial state s_0 because the deterministic model allows all of the other states to

2. This assumption is not properly named because the plan is intended precisely to change the state of the system; what is meant here is that the system remains static *unless controlled transitions take place.*

be predicted with certainty. The plan is unconditional, and the controller executing the plan is an *open-loop controller*, i.e., it does not get any feedback about the state of the system.

This restricted case may appear trivial: planning is simply searching for a path in a graph, which is a well-known and well-solved problem. Indeed, if we are given the graph Σ explicitly then there is not much more to say about planning for this restricted case. However even for a very simple application domain, the graph Σ can be so large that specifying it explicitly is not feasible. In the simple example introduced earlier (Example 1.1), in which a robot has to move containers, suppose there are five locations, three piles of containers per location, three robots, and one hundred containers. Then Σ has about 10^{277} states, which is about 10^{190} times as many states as even the largest estimates [562] of the number of particles in the universe! Thus it is impossible in any practical sense to list all of Σ's states implicitly. Consequently, there is a need for powerful *implicit* representations that will enable us to describe useful subsets of S in a way that is compact and can easily be searched.

Classical and *neoclassical* approaches to planning will be covered in Parts I and II of the book, respectively. Most of these approaches rely on an implicit representation based on logic for describing states and on *ad hoc* state-transition operators for representing actions. Specific rules and assumptions are required for defining and easily computing the next state $\gamma(s, a)$.

Planning as theorem proving, within the restricted model of this section, relies entirely on a logic representation for describing both states and state-transition operators. The idea is to use a set of logical formulas to describe not just single states but instead the entire set of states and the relationships among them. This approach will be covered in Part II.

The remaining parts of the book consider other approaches in order to deal with less restricted models of planning.

1.6 Extended Models

Several interesting models are obtained by relaxing some of the restrictive assumptions. We now review briefly the models that will be covered later.

Relaxing Assumption A0 (Finite Σ). An enumerable, possibly infinite set of states may be needed, e.g., to describe actions that construct or bring new objects in the world or to handle numerical state variables. This raises issues about decidability and termination to planners that will be discussed in Parts I and III.

Relaxing Assumption A1 (Fully Observable Σ). If we allow a static, deterministic system to be partially observable, then the observations of Σ will not fully disambiguate which state Σ is in. For each observation o, there may be more than one state s such that $\eta(s) = o$. Without knowing which state in $\eta^{-1}(o)$ is the current state, it is no longer possible to predict with certainty what state Σ will be in after each action. This case will be covered in Part V.

Relaxing Assumption A2 (Deterministic Σ). In a static but nondeterministic system, each action can lead to different possible states, so the planner may have to consider alternatives. Usually nondeterminism requires relaxing assumption A5 as well. A plan must encode ways for dealing with alternatives, e.g., *conditional* constructs of the form "do a and, depending on its result, do either b or c" and *iterative* constructs like "do a until a given result is obtained." Notice that the controller has to observe the state s: here we are planning for *closed-loop control.*

If the complete knowledge assumption (assumption A1) is also relaxed, this leads to another difficulty: the controller does not know exactly the current state s of the system at run-time. A limiting case is *null observability*, where no observations at all can be done at run-time. This leads to a particular case of planning for open-loop control called *conformant planning.*

Part V describes different approaches for dealing with nondeterminism. Some of them extend techniques used in classical planning (like graph-based or satisfiability-based planning), while others are designed specifically to deal with nondeterminism, like planning based on Markov Decision Processes (MDPs) and planning as model checking.

Relaxing Assumption A3 (Static Σ). We can easily deal with a dynamic system Σ if it is deterministic and fully observable, and if we further assume that for every state s there is at most one contingent event e for which $\gamma(s, e)$ is not empty and that e will necessarily occur in s. Such a system can be mapped into the restricted model: one redefines the transition for an action a as $\gamma(\gamma(s, a), e)$, where e is the event that occurs in the state $\gamma(s, a)$.

In the general model of possible events that may or may not occur in a state and "compete" with actions, a dynamic system is nondeterministic from the viewpoint of the planner even if $|\gamma(s, u)| \leq 1$. Deciding to apply action a in s does not focus the planner's prediction to a single state transition. Here again, a conditional plan will be needed.

Relaxing Assumption A4 (Restricted Goals). Controlling a system may require more complex objectives than reaching a given state. One would like to be able to specify to the planner an *extended goal* with requirements not only on the final state but also on the states traversed, e.g., critical states to be avoided, states that the system should go through, states it should stay in, and other constraints on its trajectories. It may also be desirable to optimize utility functions, e.g., to model a system that must function continuously over an indefinite period of time. We will consider such extended goals in several places throughout the book.

Relaxing Assumption A5 (Sequential Plans). A plan may be a mathematical structure that can be richer than a simple sequence of actions. As examples, one may consider a plan to be a partially ordered set, a sequence of sets, a conditional plan that forces alternate routes depending on the outcome and current context of execution, a universal plan or policy that maps states to appropriate actions, or a deterministic or nondeterministic automaton that determines what action to execute depending on the previous history of execution. Relaxing assumption A5 is often required

when other assumptions are relaxed, as we have seen in the case of nondeterministic systems (assumption A3) or when relaxing assumptions A1, A3, A4, and A6. Plans as partially ordered sets or as sequences of sets of actions, studied in Chapter 6 and in Part II, are more easily handled than conditional plans and policies that will be seen in Part V.

Relaxing Assumption A6 (Implicit Time). In many planning domains, action duration and concurrency have to be taken into account. Time may also be needed for expressing temporally constrained goals and occurrence of events with respect to an absolute time reference. However, time is abstracted away in the state-transition model.[3] This conceptual model considers actions or events as instantaneous transitions: at each clock tick, the controller synchronously reads the observation for the current state (if needed) and applies the planned action. To handle time explicitly will require an extension to our conceptual model; this will be considered in Part IV.

Relaxing Assumption A7 (Offline Planning). The control problem of driving a system toward some objectives has to be handled online with the dynamics of that system. While a planner may not have to worry about all the details of the actual dynamics, it cannot ignore completely how the system will evolve. At the least, it needs to check online whether a solution plan remains valid and, if needed, to revise it or replan. Other approaches consider planning as a process that modifies the controller online. Issues with online planning will be surveyed in Part VI.

These extensions cannot be handled directly within the restricted model. However, in some cases that we will illustrate later, they can be handled indirectly, possibly at the cost of a blowup of the domain description.

1.7 A Running Example: Dock-Worker Robots

Whenever possible, we will illustrate the planning procedures and techniques in this book on a simple yet nontrivial running example called the *Dock-Worker Robots* or *DWR* domain. Here we describe it only informally, but it will serve as an example for various planning formalisms throughout the book.

The environment is a generalization of Example 1.1 (see page 6). It represents a harbor with several locations corresponding to docks, docked ships, storage areas for containers, and parking areas for trucks and trains. This harbor is served by several robot carts and cranes that move containers in order to load and unload ships.

---

3. Other formalisms, such as *timed automata*, extend state-transition systems with explicit representation of time.

An abstract version of this domain can be defined by giving five finite sets of constant symbols plus one additional symbol.

- A set of *locations* {l1, l2,...}: A location can be a storage area, a dock, a docked ship, or a parking or passing area.
- A set of *robots* {r1, r2,...}: Each robot is a container carrier cart that can be loaded and can transport just *one* container at a time. When a robot is within some location it can move to any other adjacent location.
- A set of *cranes* {k1, k2,...}: A crane belongs to a single location; it can manipulate containers within that location, between piles and robots.
- A set of *piles* {p1, p2,...}: A pile is a fixed area attached to a single location. At the bottom of each pile is a *pallet*, on top of which are stacked zero or more containers.
- A set of *containers* {c1, c2,...}: A container can be stacked in some pile on top of the pallet or some other container, loaded on top of a robot, or held by a crane.
- A symbol pallet: This denotes the pallet that sits at the bottom of a pile. There is no need to have separate symbols for individual pallets because each pallet is uniquely identified by the pile it is in.

Locations do not necessarily have piles. A location without piles is a parking or passing area for robots. Any location that has piles also has one or more cranes. A crane can move a container from the top of a pile to an empty robot or to the top of another pile at the same location. We will assume that a location can be occupied by at most one robot at a time. The topology of the domain is denoted using instances of the following predicates.

- adjacent(l, l'): Location l is adjacent to location l'.
- attached(p, l): Pile p is attached to location l.
- belong(k, l): Crane k belongs to location l.

Those three predicates denote fixed relationships that do not change over time. The current configuration of the domain is denoted using instances of the following predicates, which represent relationships that change over time.

- occupied(l): Location l is already occupied by a robot.
- at(r, l): Robot r is currently at location l.
- loaded(r, c): Robot r is currently loaded with container c.
- unloaded(r): Robot r is not loaded with a container.
- holding(k, c): Crane k is currently holding container c.
- empty(k): Crane k is not holding a container.

- in(c, p): Container c is currently in pile p.
- on(c, c'): Container c is on some container c' or on a pallet within a pile.
- top(c, p): Container c sits on top of pile p. If pile p is empty, this will be denoted as top(pallet, p).

These nine predicates are not independent. For example, unloaded(r) holds if and only if there is no c such that loaded(r, c) holds. Similarly, empty(k) holds if and only if there is no c such that holding(k, c) holds, and top(c, p) holds if and only if there is no container c' such that on(c', c) holds. The reasons for these relationships and how they are maintained in the domain will be discussed in Chapter 2.

There are five possible actions in the DWR domain.

- *Move* a robot r from some location l to some adjacent and unoccupied location l'.
- *Take* a container c with an empty crane k from the top of a pile p colocated with k in the same location l.
- *Put* down a container c held by a crane k on top of a pile p colocated with k in the same location l.
- *Load* with a container c held by a crane k an unloaded robot r that is within the same location l.
- *Unload* a container c with empty crane k from a loaded robot r within the same location l.

More formal specifications of the DWR domain and several problem instances are given in the rest of this book.[4]

Variants of the DWR Domain. At certain points, in order to give detailed explanations of intricate data structures and algorithms, we will need a simpler version of the DWR domain. In the *Simplified DWR* domain, locations have no piles and no cranes, locations offer unlimited floor space for containers and robots, and every robot is equipped with an arm for loading and unloading a container. In this case, there are only three actions: moving, loading, and unloading.

At various points, we also will enrich the DWR domain in order to illustrate enhanced planning possibilities. Chapter 20 will even describe a realistic instance of the domain that has been deployed experimentally. Here are some examples of the kinds of extensions we will consider.

Space constraints are not taken into account in the basic domain. An explicit handling of space will require metric models. For example, one may slightly enrich the domain by stating that a location can hold at most a given number of robots

4. The web site at http://www.laas.fr/planning/ contains auxiliary material for this book. It includes the Planning Domain Description Language specification and a few problem instances of the DWR domain.

instead of just one. Another extension is to allow more than one action to occur concurrently because there are several independent robots and cranes. Resources will have to be made explicit in order to constrain the concurrency to nonconflicting actions. Realistically, explicit time is also needed because a feasible organization of actions and resources is a function of durations. Goals also have to be constrained in time (e.g., with respect to the arrival and departure of ships). Finally, partial information and uncertainty about the state of the world and about effects of actions needs to be modeled.