9. AI Reasoning Methods for Robotics

Joachim Hertzberg, Raja Chatila

Artificial intelligence (AI) reasoning technology involving, e.g., inference, planning, and learning, has a track record with a healthy number of successful applications. So, can it be used as a toolbox of methods for autonomous mobile robots? Not necessarily, as reasoning on a mobile robot about its dynamic, partially known environment may differ substantially from that in knowledge-based pure software systems, where most of the named successes have been registered.

This Chapter sketches the main robotics-relevant topics of symbol-based AI reasoning. Basic methods of knowledge representation and inference are described in general, covering both logic- and probability-based approaches. Then, some robotics-related particularities are addressed specially: issues in logic-based high-level robot control, fuzzy logics, and reasoning under time constraints. Two generic applications of reasoning are then described in some detail: action planning and learning.

General reasoning is currently not a standard feature onboard autonomous mobile robots. Beyond sketching the state of the art in robotics-related AI reasoning, this Chapter points to the involved research problems that remain to be solved towards that end.

The Chapter first reviews knowledge representation and deduction in general (Sect. 9.1), and then goes into some detail regarding reasoning issues that are considered particularly relevant for applications in robots (Sect. 9.2). Having presented reasoning *methods*, we then enter the field of generic reasoning *applications*, namely, action planning (Sect. 9.3) and machine learning (Sect. 9.4). Section 9.5 concludes.

Historical debates about the necessity and wisdom of employing symbolic reasoning in autonomous mobile robots notwithstanding, there now seems to be agreement that some part or layer in the control system of such robots should or could include reasoning. Hybrid control architectures (Chap. 8) would be typical software structures for solving the difficult problem of amalgamating their contributions with those of other parts of the controller, while guaranteeing sufficiently short control cycle times, so that the robot can act safely in a dynamic environment.

Symbolic reasoning is understood here in the classical sense of artificial intelligence (AI), i. e., deliberation based on symbols as in first-order predicate logic

(FOPL) or Bayesian probability theory, but typically with restrictions and/or extensions thereof, which have to be traded off against each other to achieve a suitable combination of expressivity and inference speed.

9.1 Knowledge Representation and Inference

Reasoning requires that the reasoner – in our case, a robot – has an explicit representation of parts or aspects of its environment to reason about. Two questions come up immediately: what are suitable formats for such an explicit representation, and where does the represented knowledge come from?

The second question refers to the problem of generating and maintaining in real time a symbolic description of the robot's environment, or at least of some part of it, based on a previous symbolic description and on recent environment information obtained from sensors and communication with other agents. In full generality, this problem is currently unsolved, involving AI fundamentals such as *symbol grounding* [9.1] and *object anchoring* [9.2]. So practical symbolic reasoning in a robot is restricted to that part of its knowledge that can be kept sufficiently recent. This includes, obviously, static knowledge about the environment, such as the topological elements and their relations in a building; transient knowledge available in symbolic forms, such as in facility management databases; and, most challenging, symbolic data that has to be distilled from sensor data. Object recognition from camera data (Chap. 23) is a relevant method here.

This section deals with answers to the first question, i. e., formalisms suitable for representing knowledge. Suitability has two aspects here that are to be treated as two sides of one coin. One is *epistemological adequacy*: does the formalism allow the targeted aspects of the environment to be expressed compactly and precisely? The other is *computational adequacy*: does the formalism allow typical inferences to be drawn effectively and efficiently? There is a tradeoff between these two adequacies, given that very rich, expressive, and hence epistemologically appealing formalisms are typically accompanied by intractable or even undecidable inference problems, and vice versa. Knowledge representation (KR), then,

> *is the field of AI that focuses on the design of formalisms that are both epistemologically and computationally adequate for expressing knowledge about a particular domain. [9.3, p. xiii]*

In this section, we focus on two families of formalisms, namely, logic and probability theory, and respective inference procedures. For introductions to the field of KR and for sources for recent results, we refer to the general AI and KR literature as given at the end of this chapter. In addition, there is the *International Conference on Knowledge Representation* (KR), held biannually every *even* year.

9.1.1 Logic

First-order predicate logic (FOPL) is the archetype of KR formalisms in AI. Having said this, let us add immediately that it does not qualify for such a formalism in the aforementioned sense of being both epistemologically and computationally adequate for typical application domains – in many cases, it is neither of the two. However, it is the background for conceptually and mathematically understanding formalisms for representing and reasoning with definite knowledge.

We assume familiarity with the basic notions of FOPL, and rather than giving another sketch here, we refer to the substantial body of textbooks and introductions. Every typical AI textbook introduces FOPL, and [9.4, 5] are no exceptions. [9.6] is a principled introduction to logics; [9.7] a practical one, and [9.8] a concise mathematical one.

Representing knowledge in some variant of logic facilitates drawing inferences from that knowledge using provably sound and complete calculi. Automated deduction is a subfield of logic and AI that offers a healthy number of very powerful implemented deduction systems that can readily be used. (See [9.4, Chap. 9] for an introduction; [9.9] is a recent, comprehensive source book.)

Relying on logical inference, a robot could deduce a large number of facts that might otherwise be hard or impossible to obtain. For example, assume that a robot perceives by acquiring and interpreting sensor data, that the door D_{509} in an office building is currently closed: $Closed(D_{509})$ expressed in some ad hoc FOPL language, assuming intuitive interpretations of symbols. Let us further assume that the robot's base of static knowledge

about the building contains the sentences

$$
\begin{aligned}
&Connects(D_{509}, C_5, R_{509}) ,\\
&Connects(D_{508}, C_5, R_{508}) ,\\
&Connects(D_{508a}, R_{508}, R_{509}) ,\\
&\forall d, l_1, l_2[Connects(d, l_1, l_2)\\
&\qquad\qquad \leftrightarrow Connects(d, l_2, l_1)] ,\\
&\forall d.[Closed(d) \leftrightarrow \neg Open(d)] ,\\
&\forall l.[At(l) \rightarrow Acessible(l)] ,\\
&\forall l_1, l_2.[Accessible(l_1)\\
&\qquad \rightarrow (\exists d.[Connects(d, l_1, l_2) \wedge Open(d)]\\
&\qquad\qquad \rightarrow Accessible(l_2))]
\end{aligned}
$$

Constants D_i and variable d denote doors; R_i denote rooms; C_i corridors; variables l, l_i denote locations, i. e., rooms and corridors. Assume that the robot's localization tells it its current room or corridor location as $At(\cdot)$.

Then, assuming the robot knows itself to be $At(C_5)$, observing $Closed(D_{509})$ entails $\neg Open(D_{509})$, and, more interesting, $Accessible(R_{509})$ only if $Open(D_{508}) \wedge Open(D_{508a})$ is true. So, performing, for example, a delivery task to room R_{509}, the robot may replan its route through R_{508}, unless at least one of D_{508} and D_{508a} is known to be closed. If the status of one or both of them is unknown (neither $Open(\cdot)$ nor $Closed(\cdot)$ is entailed by the current knowledge base), then accessibility of R_{509} can be neither proven nor disproven. That would leave open the option of planning the route through D_{508} and D_{508a}, gathering their required statuses on site.

So, as this little example shows, FOPL is a powerful representation and reasoning tool. Moreover, its theory is very well understood. In particular, it is well known that consequence in FOPL is in general undecidable, which means that a sound and complete deduction algorithm cannot even guarantee termination for some particular inference attempt, let alone speedy results.

However, that does not mean logic as a representation and inference tool needs to be completely disposed of. Many applications do not require the full expressivity of FOPL. Moreover, there are many interesting language subsets of FOPL that are decidable and can be handled by algorithms that are efficient in most practical cases. So some effort by the KR community has gone into identifying FOPL subsets and fitting inference procedures that qualify for being both epistemologically and computationally adequate for a broad number of applications. We will consider two of these in more detail: propositional logic and description logics.

Propositional Theories

In quite a number of practical cases, FOPL theories (sets of formulas) in fact represent finite domains. Logically speaking, they have finite Herbrand universes or can at least be recast in a form such that they have. In this case, it may still be handy to express the domain theory in FOPL syntax, but all that goes beyond a purely propositional theory is just for notational convenience. For example, an axiom stating that a robot can only be in one location (room or corridor) at a time

$$\forall l_1, l_2.[At(l_1) \rightarrow (\neg At(l_2) \vee l_1 = l_2)]$$

can be *flattened* for a finite building into the, clumsy but equivalent, form handling all locations explicitly, for example in a six-storey building

$$
\begin{aligned}
At(R_{001}) \rightarrow [&\neg At(R_{002}) \wedge \neg At(R_{003}) \wedge \cdots\\
&\neg At(R_{514}) \wedge \neg At(C_{0a}) \wedge\\
&\neg At(C_{0b}) \wedge \cdots \wedge \neg At(C_5)] ,\\
At(R_{002}) \rightarrow [&\neg At(R_{001}) \wedge \neg At(R_{003}) \wedge \cdots\\
&\neg At(R_{514}) \wedge \neg At(C_{0a}) \wedge\\
&\neg At(C_{0b}) \wedge \cdots \wedge \neg At(C_5)] ,\\
&\ldots
\end{aligned}
$$

where every ground (variable-free) instance of a predicate, such as the At instances above, are to be considered as propositional variables, regarding textual identity.

The good news here is that the corresponding *flat* theory to a FOPL theory over a finite Herbrand universe is propositional; hence, it is decidable. Moreover, under some practical conditions – e.g., if the variables in the FOPL theory are sorted (that is, the information is available that the variables l_1, l_2, e.g., range over rooms and corridors) – it can even be generated mechanically from a more compact FOPL syntax.

The potentially bad news is that the now propositional theory may of course consist of a huge number of propositional sentences: In general, as all combinations of variable substitutions in all FOPL sentences need to be generated, the growth is multi-exponential in terms of the domain sizes of the FOPL variables.

However, the technology for propositional satisfiability checking or model checking is making considerable progress, allowing propositional theories with thousands or even tens of thousands of variables to be handled in the order of seconds of computation time on regular hardware. These methods are particularly efficient if many models exist for a satisfiable propositional formula or if the truth or falsity of many ground facts is

known a priori (such as $At(C_5)$ for a robot by independent localization). Both are often the case in practical KR.

[9.4, Chap. 7] gives an introduction to model checking. One family of methods is based on the classical Davis–Putnam (DPLL) algorithm [9.10]. It attempts to construct systematically a propositional model of a given theory, efficiently propagating interpretation constraints for variables. Another family of algorithms applies local search techniques, attempting to generate interpretations using random (Monte Carlo) variable assignments. [9.11] is a recent collection of papers summarizing the state of the art, including a paper about the results of the SAT2002 competition that discusses in detail the performance of recent satisfiability checkers on synthetic (i. e., typically unnaturally hard) problems.

Description Logics

Description logics (DL) have emerged as a rigorous formalization of somewhat intuitive KR forms of the AI of the 1970s, namely, semantic networks and frame systems. Logically speaking, DLs, of which there is a considerable variety, form a certain family of decidable subsets of FOPL.

There are two parts of representing knowledge about a particular domain using a DL language. First, the *upper ontology* of the domain is formulated. It introduces the general domain concepts as well as relations between these concepts. A particularly interesting type of relations occurring in all ontologies is the superclass–subclass relation. Given that DLs – as part of the means for enforcing decidability – strictly disallow cyclic relations between concepts to be specified, the superclass relation imposes a hierarchical taxonomy on concepts, which serves to define property inheritance, much like in object-oriented programming. This first part of a DL-based domain representation is, for historical reasons, often called the *TBox* (or *terminological knowledge*).

A *concept* in the DL language corresponds to a unary predicate. To give an example, an ontology of robot navigation domains might include concepts like *Door*, *Location*, and so on. The concept hierarchy is built by defining concept equality $=$ or a subconcept property, e.g., $Room \sqsubseteq Location$, and $Corridor \sqsubseteq Location$. Concepts can be combined using concept conjunction, disjunction and, negation ($\sqcap$, $\sqcup$, $\neg$, respectively), allowing, e.g., concept definitions such as $Location = Room \sqcup Corridor$, $Door = Closed \sqcup Open$, and $Open = \neg Closed$.

Roles in a DL language correspond to binary predicates, such as *leadsTo* for a door and a location. Inversity, intersection, and union of roles are defined as expected, where $leadsTo = leadsFrom^{-1}$ is an example for defining inverse roles. Roles can be composed, such as in defining $adjacent = leadsFrom \circ leadsTo$ (location l is adjacent to m if and only if some door leads from l to m). Finally, concepts and roles can be combined for defining new concepts and roles. In particular, it is possible to quantify over *role-fillers*, i. e., the individual objects (see below) that can be consistently substituted for role arguments, for example, one could define $BlockedLoc = Location \sqcap \neg\exists leadsFrom.Open$ (assuming the intuitive binding rules of the operators). Different variants of DLs differ in what other operators they make available. See [9.3] for details.

As the second part of domain representation using DLs, individual objects have to be introduced into the language of concepts and roles. This part of the domain representation is called *ABox* or *assertional knowledge*, for example, $Room(R_{509})$, $leadsTo(D_{509}, R_{509})$, and $Closed(D_{509})$ could be asserted.

DLs have a number of reasoning services to offer, which are based on logical inference in a given TBox and ABox. They include consistency of the concept definition, subsumption and disjointness of concepts, consistency of the ABox with respect to the TBox, concept and role instances, all of which are decidable in a DL. Given the TBox and ABox rudiments above, it would, e.g., be concluded that everything is consistent and that $BlockedLoc(R_{509})$ (note that only door D_{509} is known here to lead to R_{509}). These DL inferences are theoretically intractable, but run very efficiently in most practical cases.

Reference [9.3] provides a comprehensive overview of the state of the art in DL. In 2004, the WWW consortium (W3C) defined the web ontology language (OWL) [9.12] as a technical basis for the semantic web. Part of the language is OWL-DL, a classical DL in the sense just described. OWL-DL ontologies are starting to become publicly available over the web; see [9.13] for a tutorial introduction.

9.1.2 Probability Theory

KR formalisms based on logics are worth considering whenever factual knowledge is to be represented, from which deductions are to be made. Part of the knowledge that a robot might use about its environment does, however, not really have this character. Choosing among KR

formalisms is – just like choosing among programming languages – to some degree a matter of taste, experience, familiarity, or system integration considerations. In KR terms the epistemological adequacy of a formalism for some domain is not an issue that can be judged with precision.

However, logic, at least its classical variants, has a number of weak spots in terms of KR requirements for some application types. Handling *uncertainty* is one, or rather, a whole family of them, as uncertainty is in itself an overloaded concept. *Lack of knowledge* is one of its aspects. Logics can handle this insofar as truth or falsity of some facts may remain undetermined. However, if too much is undetermined in a knowledge base, logics will no longer be able to make interesting deductions, as *everything is possible* logically. However, intuitively the different possibilities may differ considerably in likelihood.

The field of KR has adopted Bayesian probability as a means of representing and reasoning with this type of uncertainty, using correlations between facts rather than strict implications. Note that this approach amalgamates different sources of lack of precise and complete knowledge. Some piece of knowledge may be unknown because it is in principle unknowable, or because it was judged too costly to build a precise theory or determine all information relevant for making a sound deduction. In either case, working with probabilities rather than binary truth values can serve as an approximation. Note that the conception of truth is still the same here as in classical logics: *objectively*, a fact is supposed to be either true or false; probability just models the subjective degree of belief that the fact is true. Note that this is a difference from fuzzy logics, which will be addressed briefly below (Sect. 9.2.2).

In analogy to Sect. 9.1.1, we here assume familiarity with the basic notions of probability theory. Reference [9.4, Chap. 13] gives an excellent basic introduction; for a more comprehensive treatment, there are a large variety of introductory textbooks around, of which [9.14] is a recent example.

Inference in Bayesian probability theory basically means to infer the probability of some event of interest, given other prior and dependent relevant probabilities. Practically, an important type of probabilistic inference is *diagnostic reasoning*: reasoning from observed effects back to hidden causes, given causal rules that specify conditional probabilities from causes to effects. So the problem is, for a potential cause C and an observed effect E: given prior probabilities $P(C)$ and $P(E)$ and the conditional probability $P(E|C)$, determine the posterior $P(C|E)$. The solution is of course given by the Bayes rule as

$$P(C|E) = \frac{P(E|C)P(C)}{P(E)} .$$

However, as in logical reasoning, this theoretically appealing principle turns out to be impractical if applied naively. Consider that not just one effect E may be observed, but n of them; moreover, not all of these are conditionally independent. A generalized form of the Bayes rule to calculate the correct posterior is straightforward, but who can specify all the conditional probabilities involved – in the worst case $O(2^n)$ of them, where n may easily range in the hundreds in practical cases?

Until the late 1980s, this problem has more or less been circumvented. One way is to treat the E_i in bad faith as independent, simply use the n individual conditional probabilities $P(E_i|C)$ and use them straightforwardly to approximate the full joint probability distributions. Reference [9.4, Chap. 14.7] reviews this and other alternative approaches.

The solution used ever since it has appeared [9.15] is *Bayes networks* (BNs). The idea is to represent the random variables in a directed acyclic graph, where a node is directly preceded by a set of parent nodes iff it is directly conditionally dependent on the corresponding parent variables. So the huge full joint probability distribution is broken down into many, typically very small, *local joint probability distributions* without loss of information, the trick being to use the typically many known conditional independencies among variables to reduce the representational and inferential effort drastically.

Figure 9.1 shows a simple BN expressing that D can be caused by B or C, with probabilities given by a conditional probability table that specifies locally the joint probability distribution. In addition, the structure says

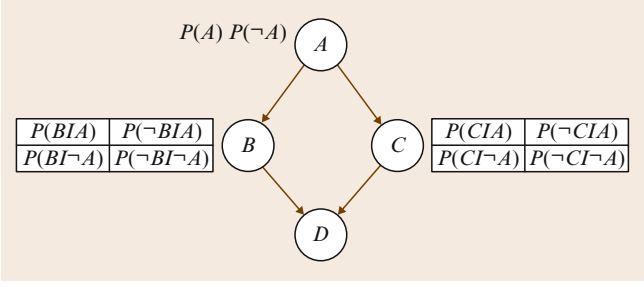

Fig. 9.1 Structure of a simple Bayesian network associated with conditional probability tables. (The ones for D, dependent on B and on C, are omitted.)

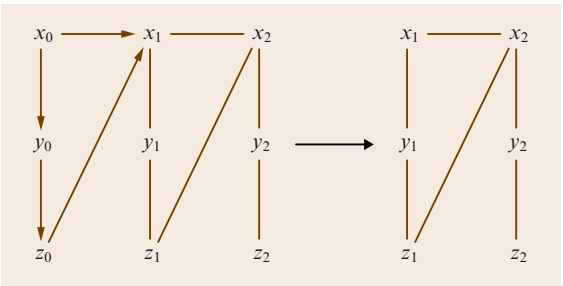

Fig. 9.2 Example of a dynamic Bayesian network at two successive time steps. The structure is invariant over time

that D is independent from A knowing B and C (a node is independent from its ancestors knowing its parents), and that B is independent from C knowing A, i. e., that $P(C|A, B) = P(C|A)$ and $P(B|A, C) = P(B|A)$. The network can be exploited in both directions. Bottom up, it enables one to explain an observed phenomenon using the known conditional probabilities (diagnostic). For example, if D is observed, its more probable cause (B or C) can be inferred. Top down, it enables one to propagate evidence to compute the probability of a phenomenon, e.g., to compute the probability of D given A (causation).

For systems evolving over time, it is important to use an invariant structure of the Bayes network in order to update the values of the variables while using the same reasoning process. Such networks are called dynamic Bayesian networks (DBNs). Figure 9.2 shows such a network at time T, then at time $T + 1$.

In Sect. 9.3.3 we shall see specific probabilistic techniques for planning under uncertainty such as Markov decision processes.

9.2 KR Issues for Robots

The reasoning methods and generic applications in the rest of this chapter all make use, to some degree, of variants of the basic formalisms of logic or Bayesian probability theory. Both formalisms, in particular if variants are included, are so general and powerful that they can contribute to any KR application.

However, reasoning in a robot control system differs from reasoning in other knowledge-based systems in a number of aspects. A knowledge-based robot is an embedded system, working in closed-loop control, typically in a dynamic environment with quite limited resources; it has to interpret itself its sensor input, and should execute at least some of the actions autonomously that it has judged appropriate. Accordingly, knowledge-based robot control has special needs that are not present in, say, an offline expert system, which gets its input by prompting an experienced human in its domain of expertise and writes its output into some document for more experienced humans to deliberate about and possibly act accordingly if they feel it right. Naturally, this difference poses particular demands of robot control on the side of KR that may not be of interest for many other KR *customers* and that lie outside the mainstream of the field. In this section, we very briefly address three topics.

Before doing so, a word about the role of representation in designing *intelligent* or *smart* robots. It has been convincingly demonstrated that some forms of *intelligence without representation* [9.16] can be achieved in robots; Chap. 38 provides the background to this. Clearly, the most sophisticated representation and inference system is of no help for a robot, if the reasoning in terms of symbols cannot be grounded in real sensor data and actuator control. On the other hand, representation and inference have much to offer for improving the performance, the software design, and the user interface of robots, for example in field, service, and intervention applications (Part F). Hybrid robot control systems (Chap. 8) are currently the means for amalgamating instances of KR technology with reactive control elements in a robot.

9.2.1 Logics for High-Level Robot Control

Robot domains are dynamic by nature, including at least one physical agent, namely, the robot. Capturing this in a logic-based formalism poses some conceptual and technical problems, which are introduced in [9.4, Chap. 10.3].

Very briefly, one of these comes from the requirement of expressing concisely and reasoning efficiently with logical models of action. Logically speaking, an individual action changes the truth value of a limited number of facts, leaving everything else as it was. For example, modeling on some abstract level the action of going from one location to another as atomic, it changes the robot's current location before and after; depending on the modelization, it may also change the battery state and the mileage count; but it does not change the lay-

out of the building or the name of the president. The problem of formalizing actions concisely in a logical language so that they allow facts that may or may not have changed after applying a sequence of actions to be inferred efficiently, has been termed the *frame problem*. It has received much attention in the literature; there are now a number of practical solutions to it.

Another problem concerns knowledge-base update, made necessary by independently changing facts. Consider, for example, a robot sitting – as it is told by its self-localization – in front of some door D, which is believed to be open, but the robot perceives a closed door. Logically, this is a contradiction. Now there are in theory several ways to make the knowledge and the perception consistent. One is to assume that D got closed since learning that it was open – probably the most intuitive explanation. Logically just as good would be, e.g., that the perception is faulty, or that the robot has been teleported in front of a door that is known to be closed. Among these *explanations*, some are more intuitively plausible than others; logically, some would require less formulas of the knowledge-base to be withdrawn and should therefore be preferred. Ideally, after replacing an old information by a new one, one would have to make sure that the consequences of a retracted formula are no longer believed.

Theoretically, these problems are arbitrarily difficult. Practically, they can be sufficiently restricted to allow solutions within a neat logical framework. Typical solutions would introduce some notion of situation or holding period for formulas, following the classical example of the situation calculus [9.17]. This allows change to be tracked. Typical solutions would also give up completeness of inference, resorting to a PROLOG-like inference engine. Three examples for such solutions with reported applications in robot control are GOLOG [9.18], event calculus [9.19], and FLUX [9.20].

9.2.2 Fuzzy Logic Approaches

For historical reasons, a number of methods and techniques dealing with classical AI questions have been living outside the *inner circle* of AI, sometimes under the different field names of *soft computing* or *computational intelligence*. This relates in particular to *fuzzy logic*, which is clearly a symbolic reasoning method. Moreover, it shares its fundamentals with fuzzy control, which gets used, in one way or another, in quite a number of robot controllers, so it is certainly a formalism to consider for KR on a robot.

To start with, the ontological basis of fuzzy logic is different to that of FOPL and probability theory. The latter assume that predicates are *objectively* either true or false; the truth may not be known, but it is in fact there. Fuzzy logic assumes that truth *objectively* comes in degrees. Many natural language categories have a fuzzy flavor, e.g., the statement that some person is *big* could be interpreted as objectively true or false in some extreme cases – but what should one *objectively* say about a male of 185 cm and 80 kg?

Fuzzy logic is conceived as a generalization of propositional logic (often called *crisp* logic in the field). Consequently, fuzzy logic uses the regular junctors $\wedge$, $\vee$, $\neg$ etc., but generalizes their truth functional definitions from the set $\{0, 1\}$ to the interval $[0, 1]$. There are several plausible ways of doing this; a frequent one is to define

$$\text{value}(\neg P) = 1 - \text{value}(P)$$
$$\text{value}(P \vee Q) = \max\{\text{value}(P), \text{value}(Q)\} .$$

The other junctors can be defined in the same spirit, where a number of differing definitions are in use.

Fuzzy deduction then typically has a different purpose, and hence a different machinery than propositional deduction – satisfiability of some fuzzy logic theory is normally not an interesting question as it too would come in degrees, so most fuzzy theories would normally be at least *a bit satisfiable*. Fuzzy logic knowledge bases are normally applied in a forward chaining manner over sets of fuzzy inference rules, inferring fuzzy values of fuzzy variables from other known fuzzy values. This is how fuzzy knowledge bases are successfully used in robot controllers, e.g., in the Saphira control architecture [9.21]. For example, it would make sense to determine the steering angle of a mobile robot using rules like

if (freeRange *right is_narrow*) $\wedge$
(freeRange *front is_medium*) $\wedge$
(freeRange *left is_wide*) $\wedge$
then set angle *medium_left*

where the italicized components (*right*, *is_narrow* etc.) are fuzzy variables. The fuzzy truth values of the antecedent variables at a particular instance in time would be determined, e.g., from the readings of a distance sensor via definitions of the variables mapping, e.g., *is_medium* to a normal distribution with mean of 2 m; the set angle *medium_left* might be mapped to a normal distribution with mean 30° to the left. The actual

set angle to apply would finally be calculated by *defuzzification* of the combined fuzzy set values of the best applicable rule or rules; a plausible scalar value would be the center of gravity of the resulting fuzzy value.

Weltanschauung debates between soft computing and *core* AI notwithstanding, fuzzy reasoning components and other AI components can live together well in a robot controller, each contributing their respective strengths. The Saphira architecture is an example.

9.2.3 Reasoning under Time Constraints

Unlike many usual AI systems, robots have to take decisions in real time based on sensory data, in order to cope with the dynamics of their environments. Hence their decisions should be produced in limited time. If these decisions require some lookahead and anticipation, i. e., if planning is required, a problem arises: real-time decision-making is indeed incompatible with the unbounded nature of planning, which in its general form is an intractable problem, and being based on FOPL inference mechanisms, is in addition only semi-decidable.

In order to take real-time constraints into account, approaches such as reactive planning have been proposed [9.22], which actually abandons to plan too much ahead (only one step at a time). However, the solution to time constraints usually comes from the *architectural design* of the robot system (Chap. 8), which integrates a decision-making level and a reactive level that would react within a bounded time. The decision-making level often includes a planner and a supervisory control system that ensures task execution supervision using predefined procedures or scripts, with explicit triggering conditions. This supervisory component has to be both goal and event driven: goal driven to accomplish the task and event driven to be reactive to environment changes. An example of such a system is the procedural reasoning system (PRS [9.23, 24]), which includes a set of procedures, a database updated by sensing and by the evolution of system state, and an interpreter that decides to launch the applicable procedures according to the contents of the database.

Anytime algorithms [9.25, 26] were developed as a possible response to time constraints. The basic idea is that the longer the algorithm can operate, the closer to optimal its solution will be. However, if time is shortened, the algorithm will still produce an answer, which may be far from optimal, but will enable the system to act instead of being blocked awaiting a decision. Anytime algorithms evaluate the quality of their results and a performance profile enables one to predict how this quality can be improved over time.

9.3 Action Planning

For the rest of this Chapter, we turn from reasoning *methods* to two generic reasoning *applications*, namely action planning and learning.

Planning, in the traditional AI sense, means deliberating about a course of action for an agent to take for achieving a given set of goals. *Scheduling* means allocating time and potentially other resources to a set of actions such that given deadlines are met and resource constraints respected. The two activities are conceptually different and have historically often been treated in a cascaded way: plan first, then schedule. Recently, with the advent of more efficient temporal reasoning methods and more efficient planning algorithms, there is a tendency to deal with them in an integrated way. The rationale is that generating an optimal plan according to some metric first, and then designing an optimal schedule for that plan, may not lead to a time- and resource-optimal schedule. It does still also make sense, however, to consider the two in isolation as their combination is not always required. In this overview, we skip the integration of scheduling. Reference [9.27, Part IV] provides a comprehensive overview.

Robot control has been among the first intended applications of planning: the STRIPS system [9.28,29] was used to generate plans, i. e., sequences of abstract high-level actions to execute, for the robot SHAKEY [9.30]. However, it has turned out to be nontrivial to design a robot control architecture in which plans containing abstract actions are broken down into physical operation that is flexible enough to follow the plan but cope with contingency, serendipity, and failure as one would intuitively expect from an *intelligent* usage of plans.

This overview starts with how to model planning domains for planners. In Sect. 9.3.2, we sketch planning algorithms that work efficiently under a number of simplifying assumptions. Section 9.3.3 relaxes these assumptions, dealing with uncertainty such as lack of

observability. We close with a brief discussion of robot planning Sect. 9.3.4.

For comprehensive coverage of the topic, we refer to [9.27]. Introductions to the topic can be found in AI textbooks, such as [9.4, Chap. 11,12,16,17]. Reference [9.31] concisely reviews the more recent developments in planning. The *International Conference on Automated Planning and Scheduling*) (ICAPS) is currently the main international conference of the field, held annually. The PLANET online research database [9.32, Service/Repositories] contains a set of continually updated links to state-of-the-art planners, schedulers, and related tools. Reference [9.33] contains an application-oriented perspective of the current state and perspectives of the field as of the end of 2003.

9.3.1 Planning Domain Descriptions

Deliberating about the course of action of an agent requires representing its possible actions, including their effects, keeping track of changes in the environment over time, and dealing rationally with any remaining uncertainty or lack of information. In this sense, planning combines the reasoning methods described previously in this Chapter.

The extreme form of deliberation would be to simulate in advance the environment, including the agent's possible actions, and then do what seems best. In practice, that is impossible for at least two reasons. First, typically not all information that would be needed for a truthful simulation is available. Planning is meant for real environments in which many parameters are unknown or unknowable and which are not under the agent's control. Second, even if everything for a complete simulation were known, then it would very likely be so computationally intensive that the real world would run ahead of the simulation all the time. In consequence, planning domain representations deliberately abstract away many, or most, details, to make effective planning possible. As in all forms of KR, precision and completeness of domain models need to be traded off against reasoning speed.

Exactly how many or how few details, and which, are represented, is an engineering decision to take when a planning system is built. Accordingly, a number of simplifying assumptions on the domain may or may not be made, including

- *finiteness* (the domain has only finitely many objects, actions, and states)
- *information completeness* (the planner has all relevant information at planning time)
- *determinism* (actions have deterministic effects)
- *instantaneousness* (actions have no relevant duration)
- *idleness* (the environment does not change during planning)

As usual, restrictive assumptions lead to efficient algorithms, but may cause problems when a simplified solution makes contact with the real world – in this case at execution time. Note that using a simplifying domain model does *not* mean assuming that the world objectively *is* that way. It may, e.g., make sense to plan only for standard cases, assuming maximal simplicity, and handle any occurring exceptions at execution time if and when they occur. In robot planning, this philosophy goes back as far as SHAKEY [9.29].

Currently, two main families of domain representations are used. The PDDL-type description is described next. In Sect. 9.3.3, a representation for nondeterministic domains will be sketched. In addition, there are various forms of deductive planning orthogonal to these two that use variants of *pure* logical representations. See [9.27, Chap. 12] for an introduction.

PDDL-Type Descriptions

Remaining on the simple side of all the above assumptions, a *domain description* contains mainly two parts. First, a specification of a restricted first-order language $\mathcal{L}$ from predicate symbols and constants representing the relations and objects considered relevant in modeling the environment; and, second, a set $\mathcal{A}$ of *action descriptions*. These consist for each action $a \in \mathcal{A}$ of pre(a), the preconditions, i. e., a set of facts in $\mathcal{L}$ that need to be true to execute a; and post(a), the postconditions, i. e., a set of (positive or negative) facts in $\mathcal{L}$ that become true by executing a.

To give a very simple example (taken from [9.27, p. 35]), Table 9.1 shows an *action schema* for moving a robot r between adjacent locations l and m. Action schemata are instantiated in the planning process to ac-

Table 9.1

```
(:action move
 :parameters (?r - robot ?l ?m - location)
 :precondition (and (adjacent ?l ?m)(at ?r ?l)
   (not (occupied ?m)))
 :effect (and (at ?r ?m) (occupied ?m)
   (not (occupied ?l))(not (at ?r ?l))))
```

tions occurring in plans, by substituting variables by constant objects.

A specific *planning problem*, given a domain, is then specified by the following components over $\mathcal{L}$:

- I, the *initial situation*, a set of all facts in $\mathcal{L}$ true of the environment before execution starts
- G, the *goal conditions*, a set of facts in $\mathcal{L}$ to bring about by executing a plan

Plans in this framework are typically *partially ordered* sets of ground (variable-free) actions to be executed in accordance with the ordering.

The *planning domain description language* (PDDL, [9.34]) is the current de facto standard for describing planning domains. In fact, PDDL stands for a whole family of languages, allowing or disallowing features such as argument typing, equality handling, conditional action effects, and some restricted form of FOPL statements. To give an example of a conditional effect, consider that the robot in the `move` action may or may not be loaded with some container `?c`, then we specify as an *additional* effect that `?c` would move with the robot:

```
(when (loaded ?r ?c)
                                % condition
 (and (at ?c ?m) (not(at ?c ?l))))
                                % effect.
```

Beyond such variants, there are extensions to the original PDDL of which PDDL2.1 [9.35] is most notable, allowing information to be expressed for temporal planning, e.g., allowing actions to have durations.

Planning is computationally hard. For propositional variants of PDDL-type problems, i. e., those with a finite $\mathcal{L}$, plan existence is of course decidable. However, solving planning problems is PSPACE-complete [9.36].

9.3.2 Partial-Order Plan Generation

Many planning systems and their underlying planning algorithms accept the aforementioned restrictive assumptions of information completeness, determinism, instantaneousness, and idleness. In consequence, the definition of a planning problem just given applies. The sort of plan to find, then, is a set of actions whose execution transforms the initial situation into a situation with the goal conditions true. (Goal situations need not be unique.) Finiteness need not be assumed for this planning model, but let us do so for simplicity, meaning that all action instances can be assumed to be ground.

Normally, actions in a plan cannot be executed in arbitrary sequence, but have to obey an ordering, making sure that all preconditions of each action are true at the time of its execution. This order need not be total or linear, but may be partial or nonlinear. The corresponding type of plan is called a nonlinear or *partial-order plan*,which is, in sum, defined as a pair $\langle A, \prec \rangle$ of a set A of action instances and an ordering relation $\prec$ on A. Unordered actions may be executed in either sequence. With some care in the action modeling, they may also be executed *concurrently*, which, assuming instantaneousness, means within the same atomic *time step*.

For a plan to be a *solution* of the given planning problem, every action $a \in A$ needs to have its preconditions true in every possible execution sequence compatible with $\prec$, and the goal conditions must be true at the end of any such execution.

Starting from the first partial-order planning approaches in the 1970s and 1980s, culminating in the SNLP (systematic nonlinear planning) algorithm [9.37] and its descendants, algorithms that tried to achieve efficiency by systematic, backward-chaining, least-commitment, relatively reasoning-intensive assembly of partial-order plans were developed. Reference [9.4, Sec. 11.3] provides an introduction and references. This line of work is often called *classical planning*.

Here is the algorithmic idea to generate a solution plan: start with the empty plan containing only the initial facts and the goal conditions; iteratively check any condition occurring in the current plan, whether it is true under all $\prec$-admissible execution sequences; if some condition c is found open, fix it by inserting a new action generating c as an effect or by ordering an existing action in the plan so that c becomes true at the time needed. This would work efficiently if all actions were independent of each other, but usually they are not: a c once achieved in a plan may be threatened later by some action with $\neg c$ in its effects. Much of the algorithmic or heuristic effort in classical planning systems has gone into preventing or handling subgoal interactions in plans efficiently.

A technique employed in many application systems is using predefined subplans as *macros*, which induce a hierarchical structure in plans, motivating their name *hierarchical task networks* (HTNs). Reference [9.27, Chap. 11] gives an introduction. Formally, a subplan can be expressed as a *task*, i. e., an action with preconditions and postconditions, that can be expanded at planning time into its underlying subplan. Tasks may be layered hierarchically. A plan is unfinished as long as it contains unexpanded tasks. The rationale for using HTNs is that the planning domain modeler knows typically – in fact, in all real applications – recurring subplans that the planner need not reinvent every time from scratch.

Moreover, the hierarchical nature of HTNs allows even large plans to be displayed in a way that is manageable for humans. HTNs have been used for a long time, with good results. SIPE (system for interactive planning and execution monitoring) [9.38] was among the first planners used for real applications. More recently, SHOP (simple hierarchical ordered planner) [9.39] is among the top-performing planners.

Starting with GRAPHPLAN [9.40], the field has focused on a set of techniques for partial-order planning algorithms (dubbed *neoclassical planning* by [9.27]), which deviate from the classical algorithms and keep generating top-performance planners. We will elaborate on two of their ingredients, namely, planning graphs and translation into satisfiability problems.

Much of the performance improvement of recent planners is owed to the formulation of planning as a particular family of search problems, for which powerful heuristics exist. For details, see [9.27, Part III].

Planning Graphs

One of GRAPHPLAN's key ideas is to start plan construction with a preprocessing phase, that:

1. determines a necessary criterion for solvability in n time steps (where n is incremented until some answer is found), and
2. yields a structured representation, based on which an n-time-step partial-order plan can be extracted relatively efficiently, if it does exist.

This structure is the *planning graph* (PG), i. e., a bipartite, layered DAG of ground fact nodes and ground action nodes, where layers of fact nodes and action nodes occur alternatingly. Figure 9.3 sketches three subsequent layers.

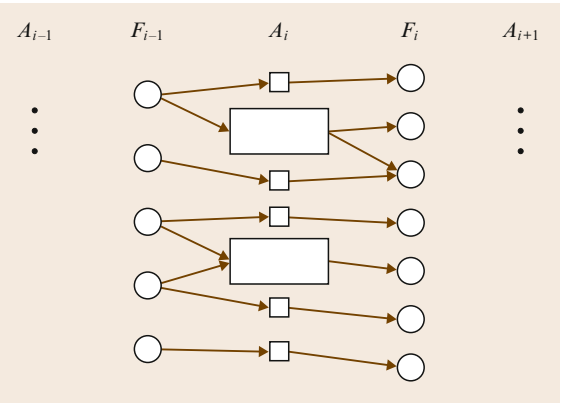

Fig. 9.3 Structure of a planning graph. *Circles* represent fact nodes, *boxes* represent action nodes, and *small square boxes* represent pseudo-operators

The first layer F_0 consists of all fact nodes to represent completely the initial situation of the problem. The last layer is also a fact layer. An edge connects a fact node with an action node iff the corresponding fact is among the action preconditions. An action node is connected with a fact node iff the fact is in the action effects. In addition to the actions $\mathcal{A}$ of the domain description, PGs contain pseudo-actions serving to copy each and every fact of a fact layer to the next fact layer. (Syntactically, a pseudo-action for f has exactly f as its precondition and its effect.) A PG contains in any layer A_i the action nodes corresponding to all actions that are applicable, given the facts of F_{i-1}, and F_i contains all facts contained in all A_i action postconditions.

For a given finite planning problem, the PG is unique up to node sequence within layers. It can be expanded without search by forward-chaining from the last fact layer, inserting all applicable actions in the next action layer and their effects in the next fact layer. The sets of actions and facts grow monotonically over layers, so PG generation terminates for finite domains.

A necessary condition for solvability of the planning problem is the existence of a fact layer containing all goal facts. If such a layer is found at time step n, PG expansion is suspended and *plan extraction* is attempted, assembling a solution plan backwards. A set of actions is selected from each A_i that generate the facts needed in F_i, starting with the goal facts in F_n and needing the preconditions of the A_i actions in F_{i-1}.

Plan extraction is not guaranteed to succeed, as needed actions may be mutually exclusive in a common action layer – the analog to subgoal interaction in classical planning. So, occurrence of the goal conditions in some F_n is a necessary, but not sufficient, criterion for solvability. In general, search is involved in plan extraction from a PG, yet in a much smaller search space than in classical planning algorithms, which typically outweighs by far the effort for constructing the PG first.

The main drawback of PGs is that they require domains to be finite (i. e., propositional), and, moreover, their languages $\mathcal{L}$ have to be limited in size, as all fact and action ground instances are explicitly constructed for expanding the PG.

Planning as Satisfiability

From the early history of planning, researchers have been tempted to express the problem of plan generation as one of inference in classical logic [9.41]. This might be useful for two reasons. First, existing efficient

logical reasoners might be used for planning, saving the effort of inventing dedicated planning algorithms. Second, logical reasoning within the domain would be seamlessly integrated into planning, obtaining powerful domain representations for free.

Expressing change and inertia efficiently in FOPL has turned out to be problematic due to the *frame problem*; see [9.4, Sec. 10.3] for an introduction. Meanwhile there are several methods to represent planning problems in different logical formalisms, with some *deductive* planners under the top performers of their times. For example, *TALplanner* [9.42] uses a representation in a temporal logic; Blackbox [9.43] uses an encoding in propositional logic, its planning algorithm incorporating PGs. A propositional logic encoding offers the option of using an efficient model checker (Sect. 9.1) for inference. This is applicable whenever PGs are. Reference [9.27, Chap. 7] provides an introduction.

A propositional representation of a finite planning problem can be derived relatively schematically from a given domain and problem description, as developed for Blackbox. Facts are given an additional situation argument, with actions transiting between situations. The propositional logic representation of initial facts, goal facts, and action effects is straightforward. The interesting part (Sect. 9.2.1) consists of the *explanatory frame axioms*, i. e., a set of axioms formalizing that any change of the truth of a fact between two situations needs to be *explained* by applying an action that has the changed fact values among its effects.

9.3.3 Planning Under Uncertainty

While information completeness, determinism, and idleness are often useful approximations for planning domains at some abstract level, they may not help in other cases. However, that does not mean that all their exact opposites are true (in which case planning would hardly make sense). Many domains show aspects of ignorance, randomness or dynamics, but still offer some degree of information or control. Exploiting this to compensate for any lacking information or control is the point of planning under uncertainty. According to different aspects and grades of imperfection in domain models, it comes in different forms. We sketch one of these forms here and refer to [9.27, P. V] and [9.4, Chaps. 12, 16, 17] for comprehensive introductions. Reference [9.44, P. IV] introduces probabilistic planning techniques in a robotics context. Reference [9.45] develops a unifying view of classical planning as previously described and planning under various forms of uncertainty.

MDPs

Assume that some actions may have nondeterministic effects, and information (which may be approximate or estimated) is available about the relative frequencies of these effects. In this case, representation in a probabilistic framework, such as *Markovian decision processes* (MDPs) is natural. An MDP represents the domain by a finite set of states S, a finite set of actions A, and an action model, which specifies for each $a \in A$ the conditional probability distribution $P_a(s' \mid s)$ of ending in state s' when executing a in s. For example, assume that the robot's current location is part of the state description; the high-level action of moving from its current location l to an adjacent, free location m (Sect. 9.3.1) produces the standard effect of being at m in 95% of cases; unspecified, maybe unknown, conditions cause the robot to stay at location l in the remaining 5% of cases.

States have associated utilities, where action cost may be modeled as negative utility of the resulting states. Planning in this framework means finding a course of action that takes not only a unique standard outcome for every operator into account, but regards all likely outcomes, including those that are unlikely but would produce very costly effects (think of the possible motion errors of a wheeled platform in the vicinity of a steep staircase).

A plan in this framework is a *policy* as in decision theory, i. e., a function from states into actions. The basic planning algorithms are value iteration (VI) and policy iteration (PI, see [9.4, Chap. 17]), both of which converge to the optimal policy for finite state sets. The difference is that VI does so by approximating the true utility of an optimal policy in a given state, whereas PI finds just the optimal policy, saving the effort of determining its precise utility value.

MDPs still make an assumption that may be inconvenient in some robotic applications, namely that state is observable at execution time, i. e., the agent can precisely determine in which state it is, and can pick the corresponding branch in its policy with certainty. Further skipping the observability assumption leads to *partially observable MDPs* (POMDPs). A POMDP adds to an MDP an observation model consisting of a finite set O of possible observations for the agent to make and of the conditional probabilities $P(o \mid s)$ of observing o in s. POMDPs are also used as the formal model in mobile robot localization and mapping (see Chap. 37).

Optimal policies for POMDPs are defined in analogy to MDPs, but finding them is computationally hard. A standard approach is to consider the *belief space* rather

than the state space, i. e., the space of probability distributions over the state space, which correspond to the agent's belief of where in state space it probably is after performing an action or an observation. Belief space is fully observable (the agent knows its beliefs with certainty); hence a POMDP can be translated into a belief space MDP and the VI and PI algorithms applied. The belief space is exponentially larger than the state space, and it is continuous. As a consequence, only POMDPs over small state spaces can be effectively handled in this way. Approximation approaches exist, see [9.4, Chap. 17].

9.3.4 Robot Planning

The term *robot planning*, or *plan-based robot control*, in AI refers to the usage of the planning methods just described in the control software of a robot, typically an autonomous mobile robot. It is not meant to replace existing motion or path-planning methods in robotics (Chaps. 5, 26, and 35), but to complement them for more general action planning. There are several purposes for using plans in robot control – be they automatically generated online by the robot or taken from a plan library. Offering a highly granular description of the robot's operation, they can be useful for *optimizing* the overall behavior, for *interacting* with humans as in task-level teleoperation, for *communicating* with fellow robots as in multirobot coordination (Chap. 40), for *learning* complex action structures in highly granular chunks, for *adapting* the behavior to environment changes, and, on a different scale of usefulness, for *engineering* the robot control code by providing a meaningful abstraction level for the programmer. Reference [9.27, Chap. 20] elaborates on these purposes; [9.46] is a collection of research papers spanning them. Reference [9.33, Chap. 5] presents a recent application-oriented perspective on the field.

Corresponding to the multiplicity of purposes and abstraction levels where action planning can be of help, there is no single method, formalism or approach for robot planning; rather, any of the planning methods can be of use. As a consequence, the term *robot plan* is mostly used in the broad sense formulated in [9.47], namely *that part of the robot's program whose future execution the robot reasons about explicitly*. The specificity of robot planning from the viewpoint of planning methodology comes from the requirement for closed-loop execution and monitoring of whatever plan the robot adopts. Technically, this poses constraints on robot planning, in particular on the domain language, on the action formalism, and on the robot control architecture.

The *domain language* $\mathcal{L}$ for a given planning domain is normally carefully handcrafted, as bad design decisions may make the difference between short and unacceptable planning time of the same planner. For robot planning, there is another issue: the propositions (at least those that need to be observed for plan generation and execution monitoring) need to be effectively determined based solely on sensor data. In general, this is the fundamental *symbol grounding* problem in AI [9.1, 2] (Sect. 9.1). The current pragmatic approach to this is to restrict the dynamic part of the domain language to propositions that can be effectively monitored.

The *action formalism* should ideally support both plan generation and execution monitoring. So it should be expressive enough to resemble, e.g., real-time programming languages, but restricted enough to make reasoning about plans effective. Reference [9.48] discusses these issues and reviews the relatively small range of existing approaches.

Finally, the *robot control architecture* has to make sure that planning and plans are integrated appropriately into the overall robot control. In particular, the advice coming from the recent plan (if any) has to be effectively negotiable with any activity coming from other control components, such as concurrent reactive behaviors; see Chap. 8 for details.

9.4 Robot Learning

Machine learning has been one of the first concerns of AI research, the link between learning and intelligence appearing to be so obvious. One of the first instances of a learning program was Samuel's Checkers program in 1959. The definition of learning is itself as general as the definition of intelligence. It can be basically stated as *the ability to improve the system's own performance or knowledge based on its experience*. In AI, the main classical learning schemes are based on manipulating symbolic representations to produce new knowledge either by deduction, by induction or by analogy. Statistical learning covers methods that enable the classification

of data, i. e., to deduce models from samples, which is a widespread learning scheme used in robotics and other domains. When the representations are probabilistic, Bayesian reasoning is used. Neural networks, based on formal models of the biological neurons, cover techniques for data classification related to statistical learning.

Another popular learning scheme in robotics is *reinforcement learning*, which is closely related to MDPs and dynamic programming. It is mainly based on a reinforcement signal (a reward) associated to each robot action that is used to reinforce a correct response.

Learning can also be classified as being *supervised* or *unsupervised* depending on whether there is a teacher able to provide the correct answer explicitly after the system's action, i. e., to reward the system, or only the system's own performance is available for this.

This section overviews the most important learning schemes of interest to robotics. Applications of learning in robot control appear throughout this Handbook, e.g., for learning maps in Chap. 37, learning intricate motor coordination patterns in Chap. 60, and evolutionary learning in Chap. 61.

9.4.1 Inductive Logic Learning

One of the main supervised symbolic learning schemes is *inductive logic programming* (ILP) [9.49]. This process is based on logic representations and first-order logic inference, and uses contextual knowledge, i. e., already acquired knowledge. The purpose of ILP is to synthesize (or learn) logic programs given a set of variables for which they return true or false values, and so-called background knowledge. The programs form an explanation of the observations. These programs are of course not unique. Therefore the difficulty is to find a minimal (necessary and sufficient) program that is compatible with the data, i. e., that returns true for the positive variables and false for the negative ones. This program must also be able to generalize, i. e., to produce the correct answer when applied to new instances. In general, ILP is not tractable. Reference [9.4, Sect. 19.5] provides an introduction.

9.4.2 Statistical Learning and Neural Networks

Statistical learning is very widely used in robotics and computer vision. It covers several techniques such as *Bayesian learning*, *kernel methods*, and *neural networks* that classify patterns into categories with the purpose of learning models from data considered as random variables, the values of which provide a sampling of the model. First there is a training period during which the algorithm is presented with a set of data and associated labels (supervised learning). Then the algorithm will label the data on its own. The purpose of the training phase is to minimize the classification error (or risk or expected loss). We refer to [9.4, Chap. 20] for a general introduction.

Bayesian learning is the application of the Bayes rule to compute the probability of given hypotheses – which are the labels of the classes to which the data belong – given the data. The training period provides the likelihood of the data for each class $P(d|C_i)$. The classification also requires a knowledge of the prior class distribution $P(C_i)$ to apply the Bayes rule. There are several variations of Bayesian learning including *maximum likelihood* (ML) and *expectation maximization* (EM). Bayes techniques have medium computational cost and are easy to implement, but they are not online.

Neural networks (NN) are composed of elementary computational units (neurons) linked by weighted connections. Each unit computes the weighted sum of its inputs and activates or fires if the input is larger than a given threshold (a sigmoid function is often used). Hence this provides for the base of a classifier. The basic structure of a feedforward network, the most commonly used, includes an input layer, one or more hidden layers, and an output layer. Through the connection weights, the output of a NN is a function of its inputs. The training of the NN consists of adjusting the weight to classify the data correctly (supervised learning). Contrary to Bayesian learning, neural networks give no insight into the classifier, but they can be used online.

9.4.3 Reinforcement Learning

Reinforcement learning (RL) is unsupervised learning through acting in the environment, which returns a reward signal. A main reference for RL is [9.50], while [9.4, Chap. 21] gives a general introduction. In RL, the purpose of the robot or the agent is to learn the best *policy*, i. e., sequence of actions to reach a given goal. To do this it has to maximize the cumulated reward. The formal framework of RL is therefore the same as that of MDPs. The robot perceives (in some cases partially) environment states in a set S and is able to accomplish actions in a set A, the sequence of which forms a policy π. There are transition probabilities from a state to the

next upon performing a given action $P(s_{t+1} = s'|s_t, a_t)$. The robot attempts to maximize a value function V expressing the expected cumulated rewards r_t over time. The basic equation expressing the value function and enabling learning of the optimal policy π^* is the Bellman equation

$$V^{\pi^*}(s) = \max_a \left[r(s,a) + \gamma \sum_{s'} P(s'|s,a) V^{\pi^*}(s') \right] , \quad (9.1)$$

where $r(s, a)$ is the immediate reward, and the discount factor γ enables future anticipated rewards to be taken into account. A variant to compute the optimal policy is Q-learning [9.51], the idea of which is to learn values of actions rather than those of states.

Most RL work assumes that the action space is discrete, and the application to continuous action spaces is not straightforward given the formulation of the learning process itself. To apply RL to continuous spaces, one usually resorts to sampling.

9.5 Conclusions and Further Reading

From its early days, autonomous robots have been included on the AI research agenda. The first experiments took place in many laboratories in the 1990s, when mobile robot platforms became widely available and affordable. AI's motivation for dealing with robotics is clear: it provides a useful test bed for computational models of agents that include perception, reasoning, and action, as robots allow them to be examined in full integration.

This is similar to AI's *long-term* perspective for robotics: to develop methods and tools that will contribute to closed-loop controllers for smart autonomous robots. But what is the contribution of AI reasoning methods to robotics *today*?

Two lessons have been learned here. First, the described reasoning methods are sufficiently well understood and efficient that they can be applied safely and profitably as ingredients in closed-loop control of robotic agents today. Mid-size league matches in RoboCup soccer are spectacular examples of this; the autonomous control features in the National Aeronautics and Space Administration (NASA) Remote Agent mission [9.52] are another.

Second, the full closed-loop integration of perception, (symbol-based) reasoning, and action would require the symbol-grounding problem [9.1], or a substantial part of it, to be solved. That is, we would have to understand how a robot could perceive its environment in terms of the semantic categories that it uses for reasoning and for planning actions. Understanding the symbol-grounding process fully is among the big problems of AI, and of all of cognitive science for that matter. So robotics had better not wait for AI to find the general solution first. In the meantime, robots can profitably use AI reasoning methods wherever the required data are available in closed-loop control.

To conclude this chapter, let us return to the issue of the robot control architecture to highlight an essential point of functional and physical integration. For the purpose of this overview, we have treated reasoning as a method or set of methods that has to rely on information given by, first, a (human) domain modeler, second, online perception for delivering up-to-date information in symbol form about the current state of the domain, and/or, possibly third, a learner that helps with the recognition or the reasoning itself by generalizing past experience. This view is in fact short-sighted. A fully integrated robot reasoner should help other modules by providing hypotheses, e.g., about what the perception module might currently look for in its sensor data. (*If you tell me you have identified a sink and a dishwasher, then I tell you we might be in a kitchen, and you could watch out for a fridge.*) A two-way integration of reasoning and perception is currently beyond the state of the art in robotics, but is moving into focus. A source of inspiration might be related work from *cognitive vision*, of which [9.53] contains a collection of recent papers.

Further Reading

For a more complete coverage of the methods described here, as well as the of the rest of AI, we refer to AI textbooks, of which the book by *Russell* and *Norvig* [9.4] is currently most widely used. Reference [9.5] is to be recommended as a recent introduction specifically to knowledge representation and its issues. Among the sources for first-hand material, let us mention the two journals *Artificial Intelligence* and the *Journal of Artificial Intelligence Research* (JAIR) [9.54], both of which cover the topics addressed here well. The main international conference, the *International Joint Conferences on Artificial Intelligence* (IJCAI), is held biannually every *odd* year.

References

9.1 S. Harnad: The symbol grounding problem, Physica D **42**, 335–346 (1990)

9.2 S. Coradeschi, A. Saffiotti: An introduction to the anchoring problem, Robot. Auton. Syst. **43**(2–3), 85–96 (2003)

9.3 F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider (Eds.): *The Description Logic Handbook* (Cambridge Univ. Press, Cambridge 2003)

9.4 S. Russell, P. Norvig: *Artificial Intelligence: A Modern Approach*, 2nd edn. (Prentice Hall, Englewood Cliffs 2003)

9.5 R.J. Brachman, H.J. Levesque: *Knowledge Representation and Reasoning* (Morgan Kaufmann, San Francisco 2004)

9.6 W.V.O. Quine: *Methods of Logic*, 4th edn. (Harvard Univ. Press, Cambridge 1955)

9.7 Z. Manna, R. Waldinger: *The Deductive Foundations of Computer Programming: A One-Volume Version of "The Logical Basis for Computer Programming"* (Addison-Wesley, Reading 1993)

9.8 W. Hodges: Elementary predicate logic. In: *Handbook of Philosophical Logic*, Vol. I, ed. by D. Gabbay, F. Guenthner (D. Reidel, Dordrecht 1983)

9.9 A. Robinson, A. Voronkov (Eds.): *Handbook of Automated Reasoning* (Elsevier Science, Amsterdam 2001)

9.10 M. Davis, G. Logemann, D. Loveland: A machine program for theorem proving, Commun. ACM **5**(7), 394–397 (1962)

9.11 J. Franco, H. Kautz, H. Kleine Büning, H. v. Maaren, E. Speckenmeyer, B. Selman: Special issue: theory and applications of satisfiability testing, Ann. Math. Artif. Intell. **43**, 1–365 (2005)

9.12 The Web Ontology Language OWL. http://www.w3.org/TR/owl-features/

9.13 G. Antoniou, F. v.Harmelen: *A Semantic Web Primer* (MIT Press, Cambridge 2004)

9.14 K.L. Chung, F. AitSahila: *Elementary Probability Theory* (Springer, Berlin 2003)

9.15 J. Pearl: *Probabilistic Reasoning in Intelligent Systems* (Morgan Kaufmann, San Mateo 1988)

9.16 R. Brooks: Intelligence without representation, Artif. Intell. **47**, 139–159 (1991)

9.17 J. McCarthy, P. Hayes: Some philosophical problems from the standpoint of artificial intelligence, Machine Intell. **4**, 463–507 (1969)

9.18 H. Levesque, R. Reiter, Y. Lespérance, F. Lin, R. Scherl: Golog: a logic programming language for dynamic domains, J. Logic Programm. **31**, 59–83 (1997)

9.19 M. Shanahan, M. Witkowski: High-level robot control through logic, ATAL '00, 7th Intl. Workshop Intell. Agents VII. Agent Theories Architectures and Languages 2000 (Springer, Berlin 2001) pp. 104–121

9.20 M. Thielscher: *Reasoning Robots. The Art and Science of Programming Robotic Agents* (Springer, Berlin 2005)

9.21 A. Saffiotti, K. Konolige, E.H. Ruspini: A multivalued logic approach to integrating planning and control, J. Artif. Intell. **76**, 481–526 (1995)

9.22 R.J. Firby: An investigation into reactive planning in complex domains, AAAI 1987 (Morgan Kaufmann, San Mateo 1987) pp. 202–206

9.23 M.P. Georgeff, A.L. Lansky: Reactive Reasoning and Planning, AAAI 1987 (Morgan Kaufmann, San Mateo 1987)

9.24 M.P. Georgeff, F.F. Ingrand: Decision-making in an embedded reasoning system, IJCAI 1989 (Morgan Kaufmann, San Mateo 1989)

9.25 M. Boddy, T.L. Dean: Solving time-dependent planning problems, IJCAI 1989 (Morgan Kaufmann, San Mateo 1989)

9.26 S. Zilberstein: Operational rationality through compilation of anytime algorithms, AI Mag. **16**(2), 79–80 (1995)

9.27 M. Ghallab, D. Nau, P. Traverso: *Automated Planning: Theory and Practice* (Morgan Kaufmann, San Francisco 2004)

9.28 R.E. Fikes, N.J. Nilsson: strips: a new approach to theorem proving in problem solving, J. Artif. Intell. **2**, 189–208 (1971)

9.29 R.E. Fikes, P.E. Hart, N.J. Nilsson: Learning and executing generalized robot plans, J. Artif. Intell. **3**, 251–288 (1972)

9.30 N.J. Nilsson: Shakey the Robot. SRI International, Tech. Note TN 323, 1984. www.ai.sri.com/shakey/

9.31 J. Rintanen, J. Hoffmann: An overview of recent algorithms for AI planning, KI **15**(2), 5–11 (2001)

9.32 PLANET: Euopean Network of Excellence in AI Planning. http://www.planet-noe.org/

9.33 PLANET Technological Roadmap on AI Planning and Scheduling. http: //www.planet-noe.org/service/Resources/Roadmap/Roadmap2.pdf. 2003

9.34 D. McDermott, M. Ghallab, A. Howe, A. Ram, M. Veloso, D. S. Weld, D. E. Wilkins: *PDDL – The Planning Domain Definition Language*, Tech Report, Vol. CVC TR-98-003/DCS TR-1165 (Yale Center for Computational Vision and Control, New Haven 1998)

9.35 M. Fox, D. Long: PDDL2.1: an extension to PDDL for expressing temporal planning domains, J. Artif. Intell. Res. **20**, 61–124 (2003)

9.36 T. Bylander: The computational complexity of propositional strips planning, J. Artif. Intell. **69**, 165–204 (1994)

9.37 D. McAllester, D. Rosenblitt: Systematic nonlinear planning, AAAI 1991 (Morgan Kaufmann, San Mateo 1991)

9.38 D. Wilkins: Domain-independent planning: representation and plan generation, J. Artif. Intell. **22**, 269–301 (1984)

9.39 D.S. Nau, T.C. Au, O. Ilghami, U. Kuter, M. Murdock, D. Wu, F. Yaman: Shop2: an HTN planning system, J. Artif. Intell. Res. **20**, 379–404 (2003)

9.40 A.L. Blum, M.L. Furst: Fast planning through plan graph analysis, J. Artif. Intell. **90**, 281–300 (1997)

9.41 C. Green: Application of theorem proving to problem solving, IJCAI 1969 (Morgan Kaufmann, San Mateo 1969)

9.42 P. Doherty, J. Kvarnström: TALplanner: a temporal logic based planner, AI Mag. **22**(3), 95–102 (2001)

9.43 H. Kautz, B. Selman: Unifying SAT-based and graph-based planning, IJCAI, Stockholm 1999 (Morgan Kaufmann, San Mateo 1999)

9.44 S. Thrun, W. Burgard, D. Fox: *Probabilistic Robotics* (MIT Press, Cambridge 2005)

9.45 B. Bonet, H. Geffner: Planning with incomplete information as heuristic search in belief space, AIPS 2000 (AAAI, Menlo Park 2000)

9.46 M. Beetz, J. Hertzberg, M. Ghallab, M.E. Pollack (Eds.): *Advances in Plan-Based Control of Robotic Agents*, Vol. 2466 (Springer, Berlin 2002)

9.47 D. McDermott: Robot planning, AI Mag. **13**(2), 55–79 (1992)

9.48 M. Beetz: Plan representation for robotic agents, AIPS, Toulouse 2002 (AAAI, Menlo Park 2002)

9.49 N. Lavrac, S. Dzeroski: *Inductive Logic Programming: Techniques and Applications* (Ellis Horwood, New York 1994)

9.50 R.S. Sutton, A.G. Barto: *Reinforcement Learning: An Introduction* (MIT Press, Cambridge 1998)

9.51 C.J. Watkins: Models of Delayed Reinforcement Learning. Ph.D. Thesis (Cambridge Univ., Cambridge 1989)

9.52 N. Muscettola, P. Nayak, B. Pell, B.C. Williams: Remote Agent: to boldly go where no AI system has gone before, J. Artif. Intell. **103**, 5–47 (1998)

9.53 H.I. Christensen, H.H. Nagel (Eds.): *Cognitive Vision Systems – Sampling the Spectrum of Approaches LNCS* (Springer, Berlin 2006)

9.54 J. Artif. Intell. Res. http://www.jair.org