8 Formal Methods in DAI: Logic-Based Representation and Reasoning

Munindar P. Singh, Anand S. Rao, and Michael P. Georgeff

8.1 Introduction

It is clear from a reading of the other chapters that agent applications are becoming ever more important. Agents are being deployed in increasingly complex production environments, where the failure or misbehavior of an agent might easily cause loss of life or property. Accordingly, a major challenge is to develop techniques for ensuring that agents will behave as we expect them to—or at least, will not behave in ways that are unacceptable or undesirable.

Of course, ensuring correctness is a challenge for all of computer science. Previous work in computer science has studied formal methods as a good basis for creating systems with minimal errors. These methods have found useful application, but much remains to be understood in terms of specifying complex systems in the first place. Agents are desirable for the very reason that they provide higher-level abstractions for complex systems. These abstractions can lead to simpler techniques for design and development, because they offer an approach to sidestep the complexity inherent in the larger applications.

Formal methods in DAI and elsewhere offer an understanding of the systems being designed at a level higher than their specific implementation. They can provide a way to help debug specifications and to validate system implementations with respect to precise specifications. However, the role of formal methods in DAI—like in the rest of computer science—is somewhat controversial. Despite the above potential advantages, some practitioners believe that formal methods do not assist them in their efforts. This might indeed be true in many cases. Formal methods, because of their call for precision, naturally lag the *ad hoc*, quick-and-dirty approaches to system construction, which are often effective in the short run. Although several powerful formalisms exist, finding the right formalism is a nontrivial challenge. Such a formalism would provide a level of expressiveness that suffices for the practical problems at hand, but would nevertheless be tractable. Also, formal methods are the most effective when included in tools and used by specially trained designers. For that reason, just as software engineers have discovered, there is no substitute for good tools nor for education in formal methods.

Despite the above controversy, there is general agreement that formal methods do help in the long run, in helping developing a clearer understanding of problems and solutions. Indeed, over the years, a number of formal techniques developed in DAI have found their way into practical systems. They usually do not constitute the entire system, but provide key functionality.

This chapter covers the major approaches to formal methods for describing and reasoning about agents and their behavior. It puts a special emphasis on how these methods may be realized in practical systems. It discusses the state of the art in theory and practice, and outlines some promising directions for future research. This chapter is primarily focused on formalizations that involve variants of symbolic logic. Some other mathematical techniques are discussed in Chapters 5 and 12. Although this chapter is self-contained, some familiarity with logic would help the reader.

8.2 Logical Background

In general, formalizations of agent systems can be, and have been, used for two quite distinct purposes:

- as internal specification languages to be used by the agent in its reasoning or action; and
- as external metalanguages to be used by the designer to specify, design, and verify certain behavioral properties of agent(s) situated in a dynamic environment.

The first class of approaches is more traditional in DAI. It presupposes that the agents have the capability to reason explicitly. Such agents are commonly referred to as *cognitive*, *rational*, *deliberative*, or *heavyweight*—some of this terminology is introduced in Chapter 1. The second class of approaches is more recent in the study of agents, although it is more traditional in the rest of computer science. This is to use the formalism to enable a designer to reason about the agent. The agent may or may not be able to reason itself when it is deployed in the field.

Fortunately, although the conceptual basis of the two approaches is radically different, the underlying mathematics is not always as different. We exploit this similarity by presenting most ideas in terms of what reasoning is required and how it may be performed, and only secondarily treating its actual realization as a component for the agent, or as a tool for its designer. Ideally, one would like to have the same logical language serve both of the above purposes. However, the trade-off between expressiveness and computability makes this ideal somewhat infeasible in general. The real-time constraints on agents situated in dynamic environments require the internal language to be computationally efficient, while the variety of complex behaviors that are possible in a system of distributed autonomous agents requires the external language to be more expressive.

We begin with the formalizations of distributed agents from the designer's perspective. We then move on to describe some of the practical tools and systems that have been built by reducing the expressive power of these languages to make them more feasible for direct execution by distributed agents.

8.2.1 Basic Concepts

The techniques used in formalizing DAI concepts make extensive use of propositional, modal, temporal, and dynamic logics. We now review these logics, which have been used in classical computer science to give the semantics of concurrent programs. For reasons of space, we avoid many details of the logics, instead accreting concepts that are of special value to DAI. We combine these into a single logic, which we study in somewhat more detail.

Simply put, there are three aspects to a logic. The *well-formed formulas* of the logic are the statements that can be made in it. These are specified as a formal language that underlies a given logic. The *proof-theory* includes the axioms and rules of inference, which state entailment relationships among well-formed formulas. The *model-theory* gives the formal meaning of the well-formed formulas. The language and proof-theory are called the syntax; the model-theory is also called the semantics.

An important practical consideration is to make the semantics natural. Since logic is used to formalize our intuitions about computational systems, their interactions with each other, or with the environments in which they exist, it is crucial that the formulas refer to the meaning that we wish to formalize.

The purpose of the semantics is to relate formulas to some simplified representation of the reality that interests us. This simplified version of reality corresponds to the nontechnical term "model." However, in logic, a *model* means more than just any simplified version of reality—it is one that is closely related to the formal language that underlies the given logic. Fundamentally, logic can handle only one kind of meaning, namely, the truth or falsity of a given formula. Since models are often quite large and structured, we often need to specify a suitable component of a model with respect to which the truth or falsity of a formula would carry the intuitive meaning one seeks to formalize. We use the term *index* to refer to any such component, be it a piece of the world, a spatial location, a moment or period in time, a potential course of events, or whatever is appropriate.

A formula is *satisfied* at a model and some index into it if and only if it is given the meaning true there. For a model M, index i, and formula p, this is written as $M \models_i p$. A formula is *valid* in a model M if and only if it is satisfied at all indices in the model; this is written as $M \models p$.

The following exposition defines a series of formal languages to capture some pretheoretic intuitions about concepts such as truth, possibility, action, time, beliefs, desires, and intentions. The typical formal languages of interest are context-free, and hence can be specified in the traditional Backus-Naur Form (BNF) [1, chapter 4]. However, for simplicity, and in keeping with most works on logic, we specify their syntax as a set of informal rules. Also, for most of the logics we present, syntactic variants are possible, but it won't be worth our while to discuss them here.

Along with each language, we will define a class of formal models that have the requisite amount of detail. Further, we will give *meaning postulates* or *semantic conditions* defining exactly where in the model (i.e., at what indices) a formula is true. A well-known caveat about logic in general is that the informal meanings of different terms may not be fully captured by the formalization. Sometimes this is because the informal meanings are not mutually consistent, and the formalization helps remove harmful ambiguity. However, sometimes this is because certain nuances of meaning are difficult to capture. If these nuances are not worth the trouble, then nothing is lost; otherwise, one should to consider an alternative formalization.

8.2.2 Propositional and Predicate Logic

Propositional logic is the simplest and one of the most widely used logics to represent factual information, often about the agents' environment. Formulas in this language are built up from *atomic propositions*, which intuitively express atomic facts about the world and *truth-functional connectives*. The connectives $\wedge$, $\vee$, $\neg$, and $\rightarrow$ denote "and," "or," "not," and "implies," respectively. The reader may consult a textbook, such as [26] for additional details.

Example 8.1
The facts "it rains" and "road is wet" can be captured as atomic propositions `rains` and `wet-road`, respectively. The implication that "if it rains, then the road is wet" can be captured by the propositional formula `rain` $\rightarrow$`wet-road`.

$\mathcal{L}_P$ is the language of propositional logic. It is given by the following rules. Here we assume that a set Φ of atomic propositions is given.

SYN-1. $\psi \in \Phi$ implies that $\psi \in \mathcal{L}_P$

SYN-2. $p, q \in \mathcal{L}_P$ implies that $p \wedge q$, $\neg p \in \mathcal{L}_P$

Let $M_0 \stackrel{\mathrm{def}}{=} \langle L \rangle$ be the formal model for $\mathcal{L}_P$. We use $\langle\,\rangle$ brackets around L to highlight similarities with the later development. Here $L \subseteq \Phi$ is an *interpretation* or *label*. L identifies the set of atomic propositions that are *true*. This gives us the base case; the meanings of the nonatomic formulas are recursively defined.

SEM-1. $M_0 \models \psi$ iff $\psi \in L$, where $\psi \in \Phi$

SEM-2. $M_0 \models p \wedge q$ iff $M_0 \models p$ and $M_0 \models q$

SEM-3. $M_0 \models \neg p$ iff $M_0 \not\models p$

The atomic propositions and boolean combinations of them are used to describe states of the system. They do not consider how the system may evolve or has been evolving. Two useful abbreviations are $\mathsf{false} \equiv (p \wedge \neg p)$, for any $p \in \Phi$, and $\mathsf{true} \equiv \neg\mathsf{false}$. As is customary, we define $p \vee q$ as $\neg(\neg p \wedge \neg q)$, and $p \rightarrow q$ as $\neg p \vee q$.

With reference to the caveat mentioned above, the logic operators and their natural language counterparts are different notions. For example, $p \rightarrow q$ is *true* if

p is *false* irrespective of q—thus it identifies potentially irrelevant connections. Alternative, more faithful, formalizations of "implies" do exist, e.g., in *relevance* logic [2]. We will refer to a simple variant in Section 8.2.3. However, most current research in logic and computer science ignores the subtlety and uses the above definition.

Although we do not use predicate logic in the specification languages, we do use it in the metalanguage, which is used in the semantic conditions. The universal ($\forall$) and existential ($\exists$) quantifiers are used to *bind* variables and make claims, respectively, about all or some of their possible values. A variable that is not bound is *free*. Let $Q(x)$ be some expression involving a free variable x, e.g., $x < y$. $(\forall x : Q(x))$ holds if $Q(l)$ holds for each possible object l that may be substituted for x in the entire expression Q. $(\exists x : Q(x))$ holds if $Q(l)$ holds for some possible object l substituted throughout for x.

8.2.3 Modal Logic

Recall the remark in Section 8.2.1 that logic treats truth or falsity of a formula as its exclusive notion of meaning. Modal logic has been used extensively in artificial intelligence to refer to other kinds of meaning of formulas. In its general form, modal logic was used by philosophers to investigate different *modes* of truth, such as *possibly* true and *necessarily* true. In the study of agents, it is used to give meaning to concepts such as belief and knowledge. In modal languages, classical propositional logic is extended with two *modal operators*: $\Diamond$ (for possibility) and $\Box$ (for necessity). The modal language $\mathcal{L}_M$ is defined as follows:

SYN-3. the rules for $\mathcal{L}_P$ (with "$\mathcal{L}_M$" substituted for "$\mathcal{L}_P$")

SYN-4. $p \in \mathcal{L}_P$ implies that $\Diamond p, \Box p \in \mathcal{L}_M$

Example 8.2

We can capture "it is possible that it rains" as $\Diamond$`rain`, and "it is necessary that the sun rises in the east" as $\Box$`sun-rises-in-the-east`.

Models for modal logic require additional structure beyond M_0. The semantics of modal logics is traditionally given in terms of sets of the so-called *possible worlds*. A world can be thought of in several different ways. A simple idea is that a world is a possible state of affairs, corresponding roughly to an interpretation, as in the semantics for $\mathcal{L}_P$. However, a world can also be treated as a history, i.e., a sequence of states of affairs. It can even be treated as a set of all possible histories starting from a given state. The above views—as a history or set of histories—are more common in the philosophical literature. However, in this chapter, we treat a world (in the technical sense) usually as a state of affairs, and sometimes corresponding to a possible history.

With sets of worlds as primitive, the structure of the model is captured by relating the different worlds via a binary *accessibility relation* [54]. Intuitively, this relation tells us what worlds are within the realm of possibility from the standpoint of a given world. A condition is possible if it is true somewhere in the realm of possibility; a condition is necessary if it is true everywhere in the realm of possibility.

Let $M_1 \stackrel{\text{def}}{=} \langle W, L, R\rangle$, where W is the set of worlds, $L : W \mapsto 2^{\Phi}$ gives the set of formulas true at a world, and $R \subseteq W \times W$ is an accessibility relation. Here, since the model is structured, the relevant index is the possible world with respect to which we evaluate a formula.

SEM-4. $M_1 \models_w \psi$ iff $\psi \in L(w)$, where $\psi \in \Phi$

SEM-5. $M_1 \models_w p \wedge q$ iff $M_1 \models_w q$ and $M_1 \models_w q$

SEM-6. $M_1 \models_w \neg p$ iff $M_1 \not\models_w p$

SEM-7. $M_1 \models_w \Diamond p$ iff $(\exists w' : R(w, w') \& M_1 \models_{w'} p)$

SEM-8. $M_1 \models_w \Box p$ iff $(\forall w' : R(w, w') \Rightarrow M_1 \models_{w'} p)$

Example 8.3

Modal logics enable us to represent *strict conditionals*, which offer a more accurate formalization of natural language implication than the propositional operator. $\Box(p \rightarrow q)$ holds not merely when p is *false*, but if p and q are appropriately related at all possible worlds.

Importantly, algebraic properties of the accessibility relation translate into entailment properties of the logic. Some common algebraic properties are the following.

- R is *reflexive* iff $(\forall w : (w, w) \in R)$
- R is *serial* iff $(\forall w : (\exists w' : (w, w') \in R))$
- R is *transitive* iff $(\forall w_1, w_2, w_3 : (w_1, w_2) \in R \& (w_2, w_3) \in R \Rightarrow (w_1, w_3) \in R)$
- R is *symmetric* iff $(\forall w_1, w_2 : (w_1, w_2) \in R \Rightarrow (w_2, w_1) \in R)$
- R is *euclidean* iff $(\forall w_1, w_2, w_3 : (w_1, w_2) \in R \& (w_1, w_3) \in R \Rightarrow (w_2, w_3) \in R)$

We leave it to the reader to verify that models that satisfy the above properties validate the following formulas, respectively.

- $\Box p \rightarrow p$
- $\Box p \rightarrow \Diamond p$
- $\Box p \rightarrow \Box\Box p$
- $p \rightarrow \Box\Diamond p$
- $\Diamond p \rightarrow \Box\Diamond p$

Since the above formulas do not depend on p, they are properly viewed as *schemas* that apply to any condition. In the literature, these are termed the T, D, 4, B, and 5 schemas, respectively [12].

8.2.4 Deontic Logic

Deontic logic is about what ought to be the case or what an agent is obliged to do. Traditional deontic logic introduces an operator Obl for obliged, whose dual is Per

for permitted. Deontic logic is specified as a modal logic with the main axiom that $\mathsf{Obl}p \rightarrow \neg\mathsf{Obl}\neg p$, i.e., the agent is obliged to bring about p only if it is not obliged to bring about $\neg p$. The rest of the logic is fairly straightforward. Unfortunately, this formulation suffers from a number of paradoxes. We shall not study it in detail here, nor the more sophisticated approaches of dyadic deontic logic and logics of directed obligation. Instead, we refer the reader to some important collections of essays on this subject [40, 41, 62].

8.2.5 Dynamic Logic

Dynamic logic can be thought of as the modal logic of action [53]. Unlike traditional modal logics, however, the necessity and possibility operators of dynamic logic are based upon the kinds of actions available. As a consequence of this flexibility, it has found use in a number of areas of DAI.

We consider the propositional dynamic logic of regular programs, which is the most common variant. This logic has a sublanguage based on regular expressions for defining action expressions—these composite actions correspond to Algol-60 programs, hence the name *regular programs*. We define $\mathcal{L}_D$ along with $\mathcal{L}_R$ as an auxiliary definition. Here, $\mathcal{B}$ is a set of atomic action symbols.

SYN-5. the rules for $\mathcal{L}_P$ applied to $\mathcal{L}_D$

SYN-6. $\beta \in \mathcal{B}$ implies that $\beta \in \mathcal{L}_R$

SYN-7. $a, b \in \mathcal{L}_R$ implies that $a; b, (a+b), a* \in \mathcal{L}_R$

SYN-8. $p \in \mathcal{L}_D$ implies that $p? \in \mathcal{L}_R$

SYN-9. $a \in \mathcal{L}_R$ and $p \in \mathcal{L}_D$ implies that $[a]p, \langle a \rangle p \in \mathcal{L}_R$

Intuitively, the atomic actions are what the agent can perform directly. The program $a; b$ means doing a and b in sequence. The program $a + b$ means doing either a or b, whichever works. This is nondeterministic choice—although it might sound a little unintuitive at first, it is logically clean and one gets to appreciate it after some experience. However, a nondeterministic program may not be physically executable, because it can require arbitrary lookahead to infer which branch is really taken. The program $p?$ is an action based on confirming the truth value of proposition p. If p is true, this action succeeds as a noop, i.e., without affecting the state of the world. If p is false, it fails, and the branch of the action of which it is part is terminated in failure—it is as if the branch did not exist. The program $a*$ means 0 or more (but finitely many) iterations of a.

Example 8.4
The Algol-60 program `if q then a else b endif` is translated as $((q?; a) + ((\neg q)?; b))$. If q holds, the $(\neg q)?$ branch fails, so a must be performed. Otherwise b must be performed.

The semantics of dynamic logic is given with respect to a model that includes a set of states (or worlds) related by possible transitions based on the actions in $\mathcal{B}$.

Let $M_2 \stackrel{\text{def}}{=} \langle W, L, \delta \rangle$, where W and L are as before. $\delta \subseteq W \times \mathcal{B} \times W$ is a transition relation. It is convenient to define a class of accessibility relations based on $\mathcal{L}_R$.

RP-1. $R_\beta(w, w')$ iff $\delta(w, \beta, w')$

RP-2. $R_{a;b}(w, w')$ iff $(\exists w'' : R_a(w, w'') \& R_b(w'', w'))$

RP-3. $R_{a+b}(w, w')$ iff $R_a(w, w')$ or $R_b(w, w')$

RP-4. $R_{a*}(w, w')$ iff $(\exists w_0, \ldots, w_n : (w = w_0) \& (w' = w_n) \& (\forall i : 0 \leq i < n \Rightarrow R_a(w_i, w_{i+1})))$

SEM-9. $M_2 \models_w \langle a \rangle p$ iff $(\exists w' : R_a(w, w') \& M_2 \models_{w'} p)$

SEM-10. $M_2 \models_w [a]p$ iff $(\forall w' : R_a(w, w') \Rightarrow M_2 \models_{w'} p)$

We refer the reader to the survey by Kozen & Tiurzyn [53] for additional details.

8.2.6 Temporal Logic

Temporal logic is, naturally enough, the logic of time. There are several variants. Of these, the most important distinctions are the following:

- *Linear versus Branching:* whether time is viewed as a single course of history or as multiple possible courses of history. The branching can be in the past, in the future, or both.
- *Discrete versus Dense:* whether time is viewed as consisting of discrete steps (like the natural numbers) or as always having intermediate states (like the rationals or reals).
- *Moment-Based versus Period-Based:* whether the atoms of time are points or intervals.

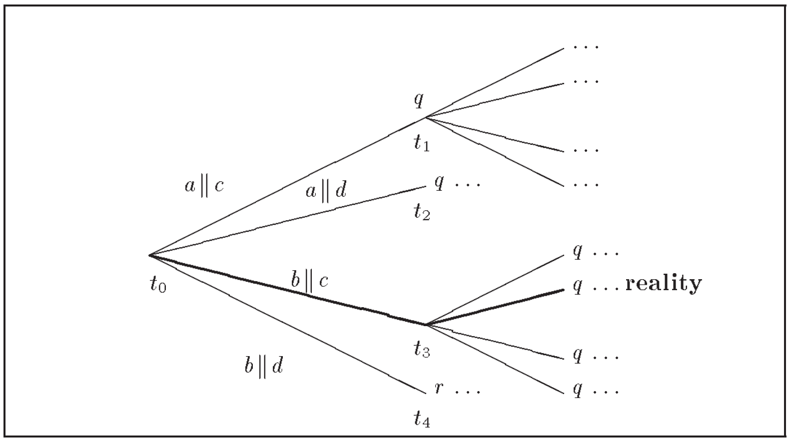

Figure 8.1 An example structure of time.

Although there are advantages to each of the above variants, we will concentrate on discrete moment-based models with linear past, but consider both linear and branching futures. Let us consider an informal view of time before we enter into a formalization. This view is based on a set of *moments* with a strict partial order, which denotes temporal precedence. Each moment is associated with a possible state of the world, identified by the atomic conditions or propositions that hold at that moment. A *path* at a moment is any maximal set of moments containing the given moment, and all moments in its future along some particular branch of $<$. Thus a path is a possible course of events. It is useful for capturing many intuitions about the choices and abilities of agents to identify one of the paths beginning at a moment as the *real* one. This is the path on which the world progresses, assuming it was in the state denoted by the given moment. Constraints on what should or will happen can naturally be formulated in terms of the real path. Figure 8.1 has a schematic picture of this view of time.

Example 8.5
Figure 8.1 is labeled with the actions of two agents. Each agent influences the future by acting, but the outcome also depends on other events. For example, in Figure 8.1, the first agent can constrain the future to some extent by choosing to do action a or action b. If it does action a, then the world progresses along one of the top two branches out of t_0; if it does action b, then it progresses along one of the bottom two branches.

The important intuition about actions is that they correspond to the granularity at which an agent can make its choices. The agent cannot control what exactly transpires, but it can influence it to some extent through its actions.

Example 8.6
In Figure 8.1, the first agent can choose between t_1 and t_2, on the one hand, and between t_3 and t_4, on the other hand. However, it can choose neither between t_1 and t_2, nor between t_3 and t_4.

Linear Temporal Logic

$\mathcal{L}_L$ is a linear-time temporal language.

SYN-10. the rules for $\mathcal{L}_P$

SYN-11. $p, q \in \mathcal{L}_L$ implies that $p\mathsf{U}q$, $\mathsf{X}p$, $\mathsf{P}p \in \mathcal{L}$

$p\mathsf{U}q$ is true at a moment t on a path, if and only if q holds at a future moment on the given path and p holds on all moments between t and the selected occurrence of q. $\mathsf{F}p$ means that p holds sometimes in the future on the given path and abbreviates $\mathsf{trueU}p$. $\mathsf{G}p$ means that p always holds in the future on the given path; it abbreviates $\neg\mathsf{F}\neg p$. $\mathsf{X}p$ means that p holds in the next moment. $\mathsf{P}q$ means that q held in a past moment.

The semantics is given with respect to a model $M_3 \stackrel{\text{def}}{=} \langle \mathbf{T}, <, [\![\,]\!] \rangle$, where $\mathbf{T}$ is the set of moments, $<$ the temporal ordering relation, and $[\![\,]\!]$ gives the denotations of the atomic propositions. It is convenient to use $[\![\,]\!]$, which is the dual of the interpretation L: $w \in [\![\psi]\!]$ iff $\psi \in L(w)$.

SEM-11. $M_3 \models_t \mathsf{P}p$ iff $(\exists t' : t' < t$ and $M_3 \models_{t'} p)$

SEM-12. $M_3 \models_t \mathsf{X}p$ iff $M_3 \models_{t+1} p$

SEM-13. $M_3 \models_t p\mathsf{U}q$ iff $(\exists t' : t \leq t'$ and $M_3 \models_{t'} q$ and $(\forall t'' : t \leq t'' \leq t' \Rightarrow M_3 \models_{t''} p))$

For the later formal development, it is useful to keep in mind that M_3 is linear, i.e., $<$ here is a total ordering.

Branching Temporal and Action Logic

$\mathcal{L}_B$ is a branching-time temporal and action language. It builds on top of $\mathcal{L}_L$ and $\mathcal{L}_D$, and especially uses the ideas of the well-known language CTL* [24]. $\mathcal{L}_B$ captures the essential properties of actions and time that are of value in specifying agents.

Formally, $\mathcal{L}$ is the minimal set closed under the rules given below. Here $\mathcal{L}_s$ is the set of "path-formulas," which is used as an auxiliary definition. Here $\mathcal{X}$ is a set of variables and $\mathcal{A}$ is a set of agent symbols. We give intuitive meanings of the constructs of this formal language after the following syntactic definitions.

SYN-12. the rules of $\mathcal{L}_P$

SYN-13. $p, q \in \mathcal{L}_B$ and implies that $\mathsf{P}p$, $(\bigvee a : p) \in \mathcal{L}_B$

SYN-14. $\mathcal{L}_B \subseteq \mathcal{L}_s$

SYN-15. $p, q \in \mathcal{L}_s$, $x \in \mathcal{A}$, and $a \in \mathcal{B}$ implies that $p \wedge q$, $\neg p$, $p\mathsf{U}q$, $\mathsf{X}p$, $x[a]p$, $x\langle a\rangle p \in \mathcal{L}_s$

SYN-16. $p \in \mathcal{L}_s$ implies that $\mathsf{A}p, \mathsf{R}p \in \mathcal{L}_B$

SYN-17. $p \in (\mathcal{L}_s \setminus \mathcal{L}_B)$ and $a \in \mathcal{X}$ implies that $(\bigvee a : p) \in \mathcal{L}_s$

The formulas in $\mathcal{L}_B$ refer to moments. The formulas in $\mathcal{L}_s$ refer to paths as in the models of $\mathcal{L}_L$. Although $\mathcal{L}_B \subseteq \mathcal{L}_s$, the formulas in $\mathcal{L}_B$ get a unique semantics.

The branching-time operator, A, denotes "in *all* paths at the present moment." Here "the present moment" refers to the moment at which a given formula is evaluated. A useful abbreviation is E, which denotes "in *some* path at the present moment." In other words, $\mathsf{E}p \equiv \neg\mathsf{A}\neg p$.

Example 8.7
In Figure 8.1, $\mathsf{EF}r$ and $\mathsf{AF}(q \vee r)$ hold at t_0, since r holds on some moment on some path at t_0 and q holds on some moment on each path.

The *reality* operator, R, denotes "in the *real* path at the present moment." R is not included in traditional temporal logics, but here helps tie together intuitions about what may and what will happen.

Example 8.8
In Figure 8.1, $\mathsf{RF}q$ holds at t_0, since q holds on some moment on the real path identified at t_0.

$\mathcal{L}_B$ also contains operators on actions. These are adapted and generalized from $\mathcal{L}_D$, in which the action operators essentially yield state-formulas, whereas in $\mathcal{L}_B$ they yield path-formulas. The operators in $\mathcal{L}_B$ capture the operators of $\mathcal{L}_D$. $x[a]p$ holds on a given path S and a moment t on it, if and only if, if x performs a on S starting at t, then p holds along S at the moment where a ends. The formula $x\langle a\rangle p$ holds on a given path S and a moment t on it, if and only if, x performs a on S starting at t and p holds at the moment where a ends.

Example 8.9
In Figure 8.1, $\mathsf{E}\langle b\rangle r$ and $\mathsf{A}[a]q$ hold at t_0, since r holds at the end of b on one path, and q holds at the end of a on each path. Similarly, $\mathsf{A}[d](q \vee r)$ also holds at t_0. Also, $\mathsf{A}[e]\mathsf{true}$ holds at t_0, because action e does not occur at t_0.

The construct $(\bigvee a : p)$ means that there is an action under which p becomes true. The action symbol a typically would occur in p and would be replaced by the specific action which makes p true.

Example 8.10
In Figure 8.1, $(\bigvee e : \mathsf{E}x\langle e\rangle\mathsf{true} \wedge \mathsf{A}x[e]q)$ holds at t_0. This means there is an action, namely, a, such that x performs it on some path starting at t_0 and on all paths on which it is performed, it results in q being true. In other words, some action is possible that always leads to q. This paradigm is used in formalizing know-how.

Let $M_4 \stackrel{\text{def}}{=} \langle \mathbf{T}, <, [\![\,]\!], \mathbf{R}\rangle$ be a formal model for $\mathcal{L}_B$. Unlike M_3, M_4 is branching, and its $[\![\,]\!]$ also applies to actions. In other words, $<$ is branching. It might partition $\mathbf{T}$ into a number of connected components, each of which would then correspond to worlds as traditionally understood. For an atomic proposition, p, $[\![p]\!]$ is the set of moments where p holds; for an action a and an agent x, $[\![a]\!]^x$ is the set of periods over which a is performed by x. These periods are notated as $[S; t, t']$ such that a begins at t and ends at t', where $t, t' \in S$. $\mathbf{R}$ picks out at each moment the *real* path at that moment. This is the notion of relativized reality alluded to above, and which is highlighted by a bold line in Figure 8.1.

For simplicity, we assume that each action symbol is quantified over at most once in any formula. Below, $p|_b^a$ is the formula resulting from the substitution of all occurrences of a in p by b. We also assume that agent symbols are mapped to unique agents throughout the model. Formally, we have:

SEM-14. $M_4 \models_t \psi$ iff $t \in [\![\psi]\!]$, where $\psi \in \Phi$

SEM-15. $M_4 \models_t p \wedge q$ iff $M_4 \models_t p$ and $M_4 \models_t q$

SEM-16. $M_4 \models_t \neg p$ iff $M_4 \not\models_t p$

SEM-17. $M_4 \models_t \mathsf{A}p$ iff $(\forall S : S \in \mathbf{S}_t \Rightarrow M_4 \models_{S,t} p)$

SEM-18. $M_4 \models_t \mathsf{R}p$ iff $M_4 \models_{\mathbf{R}(t),t} p$

SEM-19. $M_4 \models_t \mathsf{P}p$ iff $(\exists t' : t' < t$ and $M_4 \models_{t'} p)$

SEM-20. $M_4 \models_{S,t} \mathsf{X}p$ iff $M_4 \models_{S,t+1} p)$

SEM-21. $M_4 \models_t (\bigvee a : p)$ iff $(\exists b : b \in \mathcal{B}$ and $M_4 \models_t p|_b^a)$, where $p \in \mathcal{L}$

SEM-22. $M_4 \models_{S,t} (\bigvee a : p)$ iff $(\exists b : b \in \mathcal{B}$ and $M_4 \models_{S,t} p|_b^a)$, where $p \in (\mathcal{L}_s \setminus \mathcal{L})$

SEM-23. $M_4 \models_{S,t} p\mathsf{U}q$ iff $(\exists t' : t \leq t'$ and $M_4 \models_{S,t'} q$ and $(\forall t'' : t \leq t'' \leq t' \Rightarrow M_4 \models_{S,t''} p))$

SEM-24. $M_4 \models_{S,t} x[a]p$ iff $(\forall t' \in S : [S;t,t'] \in [\![a]\!]^x \Rightarrow M_4 \models_{S,t'} p)$

SEM-25. $M_4 \models_{S,t} x\langle a\rangle p$ iff $(\exists t' \in S : [S;t,t'] \in [\![a]\!]^x \& M_4 \models_{S,t'} p)$

SEM-26. $M_4 \models_{S,t} p \wedge q$ iff $M_4 \models_{S,t} p$ and $M_4 \models_{S,t} q$

SEM-27. $M_4 \models_{S,t} \neg p$ iff $M_4 \not\models_{S,t} p$

SEM-28. $M_4 \models_{S,t} p$ iff $M_4 \models_t p$, where $p \in \mathcal{L}$

8.3 Cognitive Primitives

As discussed in Chapter 1, in many cases of interest, the agent metaphor is most useful when the agents are given high-level cognitive specifications. This is described as taking an *intentional stance* toward agents [60] or viewing agents at the *knowledge level* [63]. There is sometimes disagreement as to the similarity of the two doctrines, but for our purposes, they are essentially interchangeable. The high-level cognitive specifications involve concepts such as beliefs, knowledge, desires, and intentions (the terms *intentional stance* and *knowledge level* apply to more than just intentions and knowledge). They are high-level, because they enable us to define the current state of an agent, what the agent might do, and how the agent might behave in different situations without regard to how the agent is implemented. Specifications derived from cognitive notions are perhaps the most significant of the AI contributions to agents.

Such high-level specifications serve as natural scientific abstractions for agents. However, to be used effectively, cognitive notions must be given rigorous definitions in general models of action and time. If they are to find broad application, DAI approaches must meet the standards of traditional disciplines such as distributed computing. Much of the material we discussed in Section 8.2 originated in concurrent or distributed computing. Here we build on it by including the concepts of belief, desire, and intention (BDI), and giving them formal definitions. The resulting logics can then be used to reason about agents and the way in which their beliefs, intentions, and actions bring about the satisfaction of their desires. To this end, we introduce the modal operators Bel (belief), Des (desire), $\mathsf{K_h}$ (know-how), and Int (intention). The language $\mathcal{L}_I$ is based on $\mathcal{L}_B$.

SYN-18. $p \in \mathcal{L}_s$ and $x \in \mathcal{A}$ implies that $(x\mathsf{Int}p), (x\mathsf{K_h}p), (x\mathsf{K_t}p), (x\mathsf{Des}p) \in \mathcal{L}_I$

The semantics for $\mathcal{L}_I$ is given with respect to $M_5 \stackrel{\text{def}}{=} \langle \mathbf{T}, <, [\![\,]\!], \mathbf{R}, \mathbf{B}, \mathbf{D}, \mathbf{I}\rangle$. The semantics for the part of $\mathcal{L}_I$ that uses the constructs of $\mathcal{L}_B$ is as given using M_4.

Example 8.11
Consider an agent who has the desire to win a lottery eventually and intends to buy a lottery ticket sometime, but does not believe that he will ever win the lottery. The mental state of this agent can be represented by the following formula: $\mathsf{DesAF}\texttt{win} \wedge \mathsf{IntEF}\texttt{buy} \wedge \neg\mathsf{BelAF}\texttt{win}$.

8.3.1 Knowledge and Beliefs

B, a *belief accessibility* relation, is introduced to give the semantics of the belief operator, which behaves as a modal necessity operator, such as $\Box$ above. **B** assigns to each agent at each moment the set of moments that the agent believes possible at that moment. Knowledge (know-that) is customarily defined as a true belief. Traditionally, to model belief, **B** is assumed to be serial, symmetric, and euclidean (as defined in Section 8.2.3). To model knowledge, it is in addition also assumed to be reflexive. In that case, it becomes an equivalence relation, resulting in $\mathsf{K_t}$ being an S5 modal logic operator [12].

When $\Box$ is treated as belief (or knowledge), the schemas 4 and 5 of Section 8.2.3 have an interesting interpretation. The former means that if an agent believes a condition, it believes that it believes it. The latter means that if an agent does not believe a condition, it believes that it does not believe it. Therefore, these schemas are referred to as *positive* and *negative introspection*, respectively. Negative introspection is a particularly strong assumption for limited agents.

SEM-29. $M_5 \models_t x\mathsf{Bel}p$ iff $(\forall t' : (t, t') \in \mathbf{B}(x, t) \Rightarrow M_5 \models_{t'} p)$

B depends on the given moment. Thus the agent can change its beliefs over time.

8.3.2 Desires and Goals

D associates with each moment a set of moments to represent the desires of the agent. The agent has a desire ϕ in a given moment if and only if ϕ is true in all the **D**-accessible worlds of the agent in that moment.

SEM-30. $M_5 \models_t x\mathsf{Des}p$ iff $(\forall t' : (t, t') \in \mathbf{D}(x, t) \Rightarrow M_5 \models_{t'} p)$

In the philosophical literature, desires can be inconsistent and the agent need not know the means of achieving these desires. Desires have the tendency to 'tug' the agent in different directions. They are inputs to the agent's deliberation process, which results in the agent choosing a subset of desires that are both consistent and achievable. Such consistent achievable desires are usually called *goals*. As a great simplification, the desires as presented here are logically consistent.

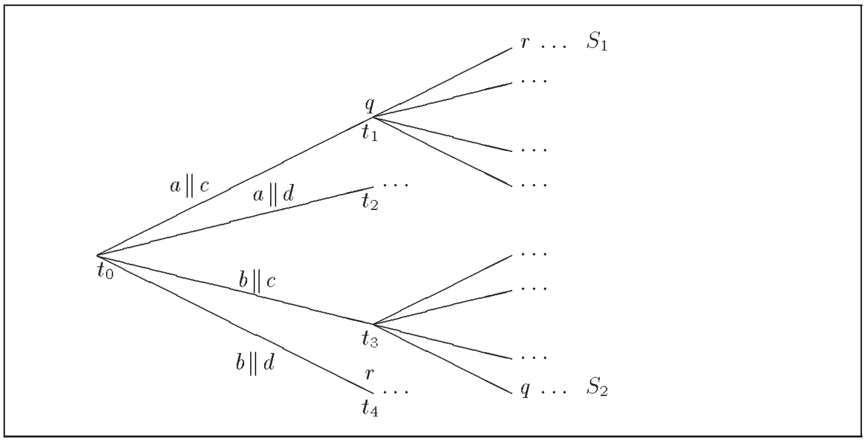

Figure 8.2 Intentions.

8.3.3 Intentions

At each moment in the model, **I** assigns to each agent a set of paths that the agent is interpreted as having selected or preferred. Roughly, intentions are defined as *the conditions that inevitably hold on each of the selected paths*. Here we consider achievement intentions in that these intentions are about achieving various conditions. However, intentions can be defined for maintaining certain conditions as well. Whereas achievement intentions are useful for liveness reasoning, maintenance intentions are useful for safety reasoning. For reasons of space, we will not discuss the latter in this chapter. We now turn to the fairly simple formal definition of achievement intentions:

SEM-31. $M \models_t x\mathsf{Int}p$ iff $(\forall S : S \in \mathbf{I}(x,t) \Rightarrow M \models_{S,t} \mathsf{F}p)$

Example 8.12
Consider Figure 8.2. Assume that $\neg r$ and $\neg q$ hold everywhere other than as shown. Let the agent x (whose actions are written first in the figure) at moment t_0 prefer the paths S_1 and S_2. Then, by the informal definition given above, we have that x intends q (because it occurs eventually on both the preferred paths) and does not intend r (because it never occurs on S_2).

The above definition validates several useful properties of intentions. Some of these were obtained with an entirely different formal structure in [76]—the present development uses fewer conceptual primitives but ignores certain abstraction issues.

IC1. **Satisfiability:**
$x\mathsf{Int}p \rightarrow \mathsf{EF}p$
This says that if p is intended by x, then it occurs eventually on some path. That is, the given intention is satisfiable. This does not hold in general,

since the sets of paths assigned by $\mathbf{I}$ may be empty. We must additionally constrain the models so that $\mathbf{I}(x,t) \neq \emptyset$.

IC2. **Temporal Consistency:**
$(x\mathsf{Int}p \wedge x\mathsf{Int}q) \rightarrow x\mathsf{Int}(\mathsf{F}p \wedge \mathsf{F}q)$
This says that if an agent intends p and intends q, then it (implicitly) intends achieving them in some undetermined temporal order: p before q, q before p, or both simultaneously. This holds because the function $\mathbf{I}$ assigns exactly one set of paths to each agent at each moment. Thus if both p and q, which are path-formulas, occur on all selected paths, then they occur in some temporal order on each of those paths. The formula $(\mathsf{F}p \wedge \mathsf{F}q)$ is true at a moment on a path precisely when p and q are true at (possibly distinct) future moments on the given path.

IC3. **Persistence does not entail success:**
$\mathsf{EG}((x\mathsf{Int}p) \wedge \neg p)$ is satisfiable
This is quite intuitive: just because an agent persists with an intention does not mean that it will succeed. Technically, two main ingredients are missing. The agent must know how to achieve the intended condition and must act on its intentions. We include this here to point out that in the theory of [15], persistence is sufficient for success (p. 233). This is a major conceptual weakness, since it violates the usual understanding that intentions do not entail know-how [75]. The need to state the conditions under which an agent can succeed with its intentions is one of the motivations for the concept of know-how.

Other important constraints on intentions include (a) the absence of closure of intentions under beliefs, (b) the consistency of intentions with beliefs about reality, and (c) the non-entailment of beliefs about reality. Of these, (a) and (b) are jointly termed the *asymmetry thesis* by Bratman [5, p. 38]. He argues that they are among the more basic constraints on the intentions and beliefs of rational agents.

8.3.4 Commitments

As presented, goals and intentions are quite similar in their semantic structure. The difference in these modalities arises in their relationships with other modalities and in terms of how they may evolve over time. One of the properties that separates them is *commitment*.

An agent is typically treated as being committed to its intentions [5]. Such commitments apply within a given individual agent, and are accordingly also termed *psychological commitments* [10, 74]. An agent's commitment governs whether it will persist with its intentions and if so, for how long. There is general agreement that commitment be treated as constraining how intentions are revised and updated, and resides in their processing rather than in their core semantics [36, 65, 76]. A contrasting approach is to include commitment in the core semantical definition of intentions [15]; this approach is criticized by [65, 73, 75]. Constraint IC4 shows how commitment may be expressed in the present framework. This version of commitment is purely qualitative.

IC4. **Persist while succeeding:**
This constraint requires that agents desist from revising their intentions as long as they are able to proceed properly. If an agent selects some paths, then at future moments on those paths, it selects from among the future components of those paths:
$(S \in \mathbf{I}(x,t)$ and $[S;t,t'] \in [\![a]\!]^x) \Rightarrow (\forall S' \in \mathbf{I}(x,t') \Rightarrow (\exists S'' \in \mathbf{I}(x,t)$ and $S' \subseteq S''))$

However, it is believed that handling commitment and the update of intentions will involve greater subtlety than the above, e.g., see [34, 81] for logic-based and probabilistic approaches, respectively.

8.3.5 Know-How

Intentions have an obvious connection with actions—agents act to satisfy their intentions. However, intentions do not ensure success; IC3 above showed that even persistence is not sufficient for success. A key ingredient is know-how, which we now formalize.

Example 8.13
Consider Figure 8.2. At t_0, x may do either action a or action b, since both can potentially lead to one of the preferred paths being realized. However, if the other agent does action d, then no matter which action x chooses, x will not succeed with its intentions, because none of its preferred paths will be realized.

We propose that an agent, x, knows how to achieve p, if it is able to bring about p through its actions, i.e., force p to occur. The agent's beliefs or knowledge must be explicitly considered, since these influence its decision. For example, if an agent is able to dial all possible combinations of a safe, then it is able to open that safe: for, surely, the correct combination is among those that it can dial. On the other hand, for an agent to really know how to open a safe, it must not only have the basic skills to dial different combinations on it, but also know which combination to dial.

A tree of actions consists of an action, called its *radix*, and a set of subtrees. The idea is that the agent does the radix action initially and, then, picks out one of the available subtrees to pursue further. In other words, a tree of actions for an agent is a projection to the agent's actions of a fragment of **T**. Thus a tree includes *some* of the possible actions of the given agent, chosen to force a given condition. Let Υ be the set of trees. Then Υ is defined as follows.

T1. $\emptyset \in \Upsilon$ ($\emptyset$ is the empty tree)

T2. $a \in \mathcal{B}$ implies that $a \in \Upsilon$

T3. $\{\tau_1, \ldots, \tau_m\} \subseteq \Upsilon$, $\tau_1, \ldots, \tau_m$ have different radices, and $a \in \mathcal{B}$ implies that $\langle a; \tau_1, \ldots, \tau_m \rangle \in \Upsilon$

Now we extend the formal language with an auxiliary construct. This extension is only meant to simplify the definitions.

SYN-19. $\tau \in \Upsilon$, $x \in \mathcal{A}$, and $p \in \mathcal{L}_I$ implies that $x[\![\tau]\!]p \in \mathcal{L}_I$

$x[\![\tau]\!]p$ denotes that agent x knows how to achieve p relative to tree τ. As usual, the agent symbol can be omitted when it is obvious from the context. To simplify notation, we extend $\bigvee$ to apply to a given range of trees. Since distinct trees in each such range have distinct radix actions, the extension of $\bigvee$ from actions to trees is not a major step.

SEM-32. $M \models_t [\![\emptyset]\!]p$ iff $M \models_t \mathsf{K_t}p$

SEM-33. $M \models_t [\![a]\!]p$ iff $M \models_t \mathsf{K_t}(\mathsf{E}\langle a\rangle\mathsf{true} \wedge \mathsf{A}[a]\mathsf{K_t}p)$

SEM-34. $M \models_t [\![\langle a; \tau_1, \ldots, \tau_m\rangle]\!]p$ iff
$M \models_t \mathsf{K_t}(\mathsf{E}\langle a\rangle\mathsf{true} \wedge \mathsf{A}[a](\bigvee_{1 \le i \le m} \tau_i : ([\![\tau_i]\!]p)))$

Thus an agent knows how to achieve p by following the empty tree, i.e., by doing nothing, if it knows that p already holds. As a consequence of this knowledge, the agent will undertake no specific action to achieve p. The nontrivial base case is when the agent knows how to achieve p by doing a single action: this would be the last action that the agent performs to achieve p. In this case, the agent has to know that it will know p immediately after the given action.

It is important to require knowledge in the state in which the agent finally achieves the given condition, because it helps limit the actions selected by the agent. If p holds, but the agent does not know this, then it might select still more actions in order to achieve p.

Lastly, an agent knows how to achieve p by following a nested tree if it knows that it must choose the radix of this tree first and, when it is done, that it would know how to achieve p by following one of its subtrees. Thus know-how presupposes knowledge to choose the next action and confidence that one would know what to do when that action has been performed.

SEM-35. $M \models_t x\mathsf{K_h}p$ iff $(\exists \tau : M \models_t x[\![\tau]\!]p)$

Example 8.14

Consider Figure 8.3. Let x be the agent whose actions are written first there. Assume for simplicity that each moment is its own unique alternative for x (this is tantamount to assuming that x has perfect knowledge—the above definition does not make this assumption). Then, by the above definitions, $x\mathsf{K_t}q$ holds at t_3 and t_4. Also, $x\mathsf{K_h}q$ holds at t_1 (using a tree with the single action a) and at t_2 (using the empty tree). As a result, at moment t_0, x knows that if it performs a, then it will know how to achieve q at each moment where a ends. In other words, we can

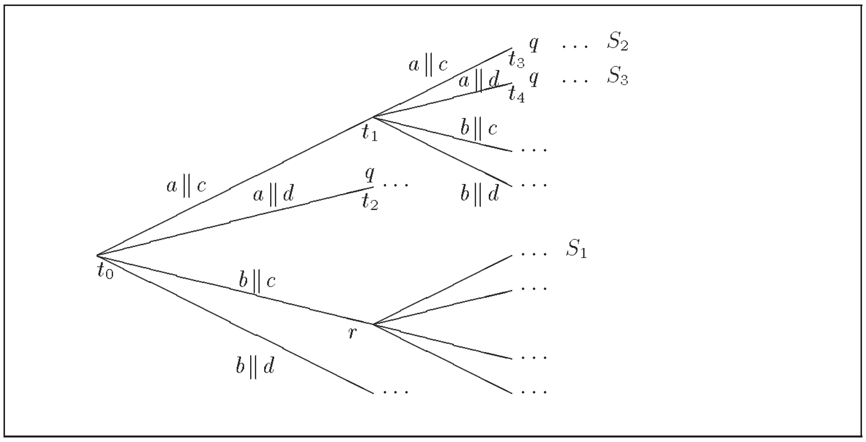

Figure 8.3 Know-how.

define a tree, $\langle a; a, \emptyset \rangle$, such that x can achieve q by properly executing that tree. Therefore, x knows how to achieve q at t_0.

A number of technical approaches to concepts of the know-how family exist. Some of the leading ones are Segerberg's *bringing it about* [68] and Belnap & Perloff's *seeing to it that (STIT)* [4] theories.

8.3.6 Sentential and Hybrid Approaches

The above approaches have used modal logics to formalize various cognitive concepts. Although technically intuitive and elegant, modal approaches have the undesirable feature that they over-estimate the reasoning capabilities of an agent. For example, an agent who knows (or intends) p is automatically assumed to know (or intend) all logical consequences of p. For knowledge, this is termed the problem of *logical omniscience* [42]. Real-life agents cannot be logical omniscient. Consequently, alternative approaches have been proposed to formalizing the cognitive concepts. These approaches include the explicit representations that an agent has for its beliefs or intentions, e.g., [50, 51]. Unfortunately, although these approaches solve the problem of logical omniscience, they do not naturally support any inferences among the cognitive concepts. This too is undesirable, and has accounted for the lack of attention paid to these approaches. Some hybrid approaches the give a possible worlds semantics, but restrict it via some representational mechanism have also been developed, e.g., [27, 82], but these two have not been intensively pursued in the literature.

One way to understand the above issue is as a natural consequence of the knowledge level [63]. Newell observed that the knowledge level (corresponding to the modal approaches) would be inherently inaccurate, whereas the more accurate *symbol* level (corresponding to the representational approaches) would be more accurate, but only as a lower-level, procedural level of discourse.

8.3.7 Reasoning with Cognitive Concepts

Section 8.2 described two main roles for formal methods in DAI. The concepts introduced above may be used in each of those roles. In either case, there is need for efficient reasoning techniques. In the first use, the agent itself applies the logic, and needs methods such as *theorem proving* to decide its actions. In the second use, the designer applies the logic to specify and validate the design of an agent, and needs methods such as theorem proving and *model checking* to relate logical specifications to the construction of the agent. The two uses differ in their complexity requirements. Although both benefit from improved techniques, the first use is by far the more demanding, because it requires an answer in less time than the agent has to respond to its environment or to other agents. For this reason, the second use is the more practical one, at least when the logic is expressive.

There are two main approaches for reasoning with a logic. The more traditional one in logic and AI is theorem proving, which essentially involves establishing that a given formula (the purported theorem) follows through a finite sequence of applications of axioms and inferences rules of a given logic [26]. The other approach, which was invented in logics of programs and is finding increasing application in AI, is model checking. This involves checking if a given formula is satisfied at a given model and index. For certain logics, model checking can be a lot more tractable than theorem proving [24, 14]. However, model checking requires additional inputs in the form of the model and index. This does not prove to be a problem in several cases, where one if trying to validate a given agent design in a given environment. The model can be derived given knowledge of the agent and its environment.

Temporal logics and modal logics of knowledge have been studied for some time, and their complexity issues are well-understood. We lack the space to discuss complexity issues in much detail here, and refer the reader to [24, 28, 53] for details. The μ-calculus is a logical language that has explicit operators for computing greatest and least fixpoints [24, 52]. This can be used to specify various modal and temporal logics in uniform framework, which can be naturally used for model checking [9, 14].

Both of the above classes of techniques are now being extended and applied in DAI. Rao has developed some tableau-based decision procedures for variants of the above BDI logics [64]. The μ-calculus is recently being applied to reasoning about the actions of agents [17, 79].

8.4 BDI Implementations

We now consider some possible ways to realize the above theories of BDI concepts in a computational system.

8.4.1 Abstract Architecture

We first characterize a BDI architecture abstractly and then show how a concrete practical instantiation may be obtained.

A Basic Interpreter

We now describe a basic abstract interpreter for situated systems. The architecture makes use of the underlying concepts of BDI architectures, but implements the entities defined by the modal operators directly as data structures.

The inputs to the system are *events*, received via an *event queue*. The system can recognize (on its event queue) both *external* (environmental) and *internal* events. External events may directly generate particular internal events, such as updating some component of the system state. We assume that the events are atomic and are recognized upon completion (and not during occurrence).

The outputs of the system are atomic *actions*, which are performed by an *execute* function. The system may, but is not required to, recognize events corresponding to the successful or unsuccessful execution of actions. Based on its current state and the events in its queue, the system selects and executes *options*, which correspond to subroutines, production rules, tasks, plans, finite automata, or circuit networks. Correspondingly, the option-invoking events would be subroutine calls or the assertion of antecedents of a production rule.

The abstract interpreter is given below. We assume the procedures and functions appearing in the interpreter operate on the system state, denoted by `S`. The interpreter continually performs the following. First, it determines the available options. Next, it deliberates to commit to some options. It then updates its state and executes appropriate atomic actions. Finally, the event queue is updated to contain all those recognizable events that have occurred during the cycle. Since events are recognized (and thus acted upon) only once per cycle, the system's reaction time is bounded from below by the time taken to perform a cycle.

```
basic-interpreter
initialize-state();
do
    options := option-generator(event-queue, S);
    selected-options := deliberate(options, S);
    update-state(selected-options, S);
    execute(S);
    event-queue := get-new-events();
until quit.
```

This abstract interpreter can be used as a basis for different situated systems, including those in which most of the deliberation is precompiled [67].

An Abstract BDI Interpreter

We now consider the special case of a BDI architecture by refining both the system state and interpreter. The system state comprises three dynamic data structures representing the agent's beliefs, desires, and intentions. For simplicity, we assume that the agent's desires are mutually consistent, although not necessarily all achievable. Such mutually consistent desires are called *goals*. The data structures support query and update operations, which include `b-add`, `b-remove`, `g-add`, `g-remove`, `i-add`, and `i-remove`. The update operations are subject to compatibility requirements, captured in the functions `b-compatible`, `g-compatible`, and `i-compatible`. These functions are critical in enforcing the constraints on the agent's mental attitudes.

The interpreter is refined as follows. Here `get-new-external-events` returns the external events that have occurred since its last invocation. At the beginning of a cycle, the option generator reads the event queue. It returns a list of the best options for further deliberation and possible execution. Next, the deliberator selects a subset of options and adds them to the intention structure. If there is an intention to perform an atomic action now, the agent executes it. Any external events that have occurred during the interpreter cycle are then added to the event queue. Internal events are added as they occur. Next, the agent modifies the intention and goal structures by dropping all successful goals and satisfied intentions, as well as impossible goals and unrealizable intentions.

```
BDI-interpreter
initialize-state();
do
    options := option-generator(event-queue,B,G,I);
    selected-options := deliberate(options,B,G,I);
    update-intentions(selected-options,I);
    execute(I);
    get-new-external-events();
    drop-successful-attitudes(B,G,I);
    drop-impossible-attitudes(B,G,I);
until quit.
```

This interpreter extends the basic interpreter mainly in the last three procedures, which eliminate a number of options that would otherwise be carried over to the next cycle.

8.4.2 Practical System

The above abstract architecture is a useful abstraction of the preceding theoretical model. It illustrates the main components of practical reasoning: option generation, deliberation, execution, and intention handling [5].

However, it is not practical. The architecture assumes a (logically) closed set of beliefs, goals, and intentions. It is not specified how the option generator and deliberation procedures can be made sufficiently fast to satisfy the real-time demands placed upon the system. We now make a number of additional representational choices which, while constraining expressive power, provide a more practical system. The resulting system is a simplified version of the Procedural Reasoning System (PRS) [46].

Beliefs and Goals

The system operates only on *explicit* beliefs and goals and not on their consequential closure. Further, we identify a subset of the agent's beliefs and goals, which we call *current*. These are taken to be ground literals (rather like atomic propositions, but actually predicates applied to constants). Ground literals can be negated, but do not include any binary operators such as disjunction or implication. Intuitively, they represent beliefs and goals that are currently held, but which can be expected to change over time.

It may seem that such a language is too simple to be of practical use. However, implications and variables can be introduced through the plan constructs, resulting in little loss of expressiveness, but for a substantial gain in control.

Plans

The above abstract interpreter represents information about means and options as beliefs. These can be more directly represented as *plans*. A plan has a name or *type*. The *body* of a plan is the method for executing it, and is specified by a *plan graph*, which is a rooted, directed, acyclic graph whose edges are labeled with simple plan expressions. A simple plan expression is either an atomic action or a subgoal. The *invocation condition* (a triggering event) and *precondition* specify when the plan may be selected. The *add list* and *delete list* of a plan respectively specify the atomic propositions to be believed or not believed upon its successful execution.

Plans represent a number of beliefs corresponding to complex modal formulas. Having a plan means that its body is believed to be an option whenever its invocation condition and precondition are satisfied. A plan represents the belief that, whenever its invocation condition and precondition are satisfied and its body successfully executed, the propositions in the add list will become true. Since the preconditions are conditions on the agent's beliefs, the agent can execute plans to compute new consequences. These consequences can trigger further plans to infer further consequences. This gives the agent greater control as to when to compute consequences of its current beliefs, goals, and intentions.

Example 8.15

Suppose John acquires a goal to quench his thirst. He believes he has two ways to satsify it. One, perform a sequence of two atomic actions: open the tap and drink water from the tap. Two, satsify a subgoal (obtain a soda bottle) and then perform an atomic action (drink soda from the bottle). The subgoal can be satisfied

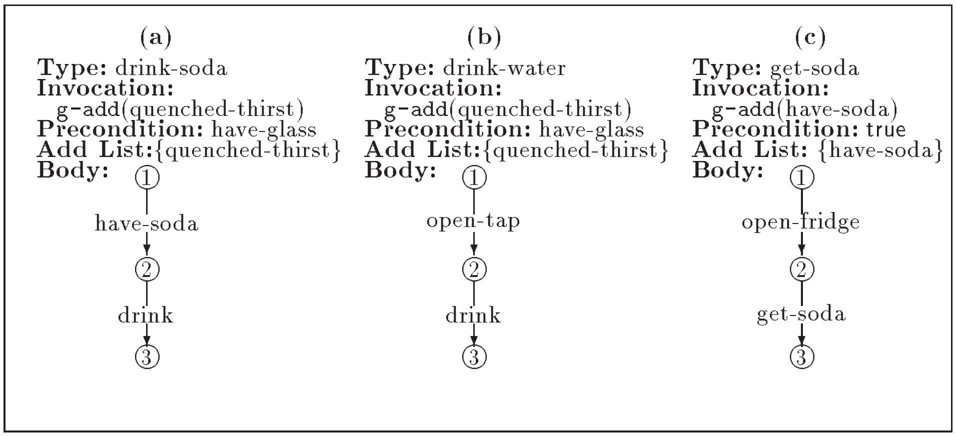

Figure 8.4 Plans for quenching thirst.

by opening the refrigerator and removing a soda bottle. These plans are shown in Figure 8.4.

Intentions

Plans provide a hierarchical structure and allow tractable real-time option generation and means-end reasoning. The options are, in fact, plans. As they are adopted, they are added to the intention structure. Thus, intentions are represented as sets of hierarchically related plans.

To achieve an intended end, the agent forms an intention towards a means for this end; namely, the plan body of an appropriate plan. This means-end pair, together with information about variable bindings and control points, is called an *intention frame*. An intention towards a means results in the agent adopting another end (subgoal) and the means for achieving this end, thus creating another intention frame. This process continues until the subgoal can be directly executed as an atomic action. The next subgoal in the plan is then attempted.

An *intention stack* is used to keep track of variable bindings and control points. Each intention stack represents a separate process or task. These intention stacks are organized into an *intention structure*, which places various ordering constraints on them. Intention stacks can also be created for any event that appears in the invocation condition of a plan. This enables the system to be responsive to external events without mediating everything through goals.

A Practical Interpreter

A practical interpreter can be derived from the above. The main loop for this interpreter is as above. However, as the system is embedded in a dynamic environment, the procedures appearing in the interpreter must be fast enough to satisfy the real-time demands of the appropriate applications.

Given a set of trigger events from the event queue, the option generator iterates through the plan library and returns those plans whose invocation condition matches the trigger event and whose preconditions are believed by the agent. The provability procedure involves simple unification with the beliefs.

```
option-generator(trigger-events)
options := {};
for trigger-event ∈ trigger-events do
   for plan ∈ plan-library do
      if matches(invocation(plan),trigger-event) then
         if provable(precondition(plan),B) then
            options := options ∪ {plan};
return(options).
```

The `deliberate` procedure's execution time should conform with the time constraints of the environment. Under certain circumstances, random choice may be appropriate. Sometimes, however, it is necessary to carry out lengthy deliberation. Such deliberation can be achieved by including metalevel plans in the plan library. Thus the `deliberate` procedure may select, and thus form an intention towards, metalevel plans for performing more complex deliberation than it itself is capable. We give a simplified version of the procedure implemented in PRS [32].

```
deliberate(options)
if length(options) ≤ 1 then return(options);
else metalevel-options := option-generator(b-add(option-set(options)));
   selected-options := deliberate(metalevel-options);
   if null(selected-options) then
      return(random-choice(options));
else return(selected-options).
```

Note that there can be more than one metalevel option, which results in the procedure being called recursively until at most one option remains. If no metalevel options are available, the deliberator chooses randomly.

Option generation can be simplified by inserting `post-intention-status` at the end of the loop. This procedure delays posting events on the queue to avoid the work caused by spurious changes otherwise sent to the event queue. In the abstract interpreter, commitment is achieved by reducing the options generated. Since the options depend on the events in the queue, `post-intention-status` determines the elements of the intention structure that are carried forward. Thus, `post-intention-status` can yield various notions of commitment, which result in different behaviors of the agent. One variant is given next.

```
post-intention-status()
if null(I) then
   for goal ∈ G do
      event-queue := event-queue ∪ g-add(goal);
```

else for stack $\in$ I **do**
    event-queue := event-queue $\cup$ `g-add`(means(top(stack))).

Bel	Goal	Int	*done*	*succeeded*
`glass`	–	–	–	–
unchanged	`quench`	–	–	`g-add(quench)`
unchanged	unchanged	{ `soda; drink`}	–	`g-add(soda)`
¬ `remove-soda`	unchanged	–	`fridge`	`fridge`, `g-add(quench)`
unchanged	unchanged	{ `drink`}	`tap`	`tap`
`quench`	–	–	`drink`	`drink`

Table 8.1 Trace of practical BDI interpreter.

Example 8.16
Consider Example 8.15 with plans as shown in Figure 8.4. Assume that the event `g-add(quench)` has just been added to the event queue. As the invocation conditions of `drink-soda` and `drink-water` match with the trigger event and their context conditions are believed, the option generator returns both these plans as suitable options.

Assume that the deliberator first selects the `drink-soda` option. As this option is to satisfy a new goal, rather than a subgoal of a previous intention, a new intention stack is created. The end (goal) for the top intention frame of the stack is `quench` and the means are given by the `drink-soda` plan. Since the first action in this plan is not atomic, no action is executed. Assume that no external events occur on this cycle. Thus the event queue contains only the internal event corresponding to the creation of the intention for the chosen option. As the system has not succeeded in any of its goals nor discovered that any intentions are impossible, it posts the current intention status. This results in `g-add(soda)` being added to the event queue.

In the next cycle, the option generator selects the plan for getting soda. This is adopted, and its frame added to the intention stack. The agent opens the refrigerator door, but at the next moment discovers that no soda is present. It is thus forced to drop its intention. Finally, the initial goal is reposted by `post-intention-status`.

On the next cycle, the option to drink water is selected, and the plan is completed successfully over further cycles. Table 8.1 shows the trace.

In the above we showed how the logics of the BDI concepts can be mapped into realistic implementations of systems. Although we didn't discuss the interactional aspects in the above, those can be worked in as well [36, 66]. We now our attention to some direct ways of capturing the interactional aspects of multiagent systems.

8.5 Coordination

Coordination is one of the key functionalities needed to implement a multiagent system. This is especially so when the component agents are *heterogeneous*, i.e., of diverse constructions and internal structures, and *autonomous*, i.e., making decisions without regard to the other agents.

A number of techniques for coordination have been developed in DAI. These are discussed in Chapter 3. A thorough logical account of these techniques, however, remains to be developed. A logical account would have the usual benefits of formal methods: a declarative, high-level specification independent of its ultimate realization, and the possibility of rigorously validating the implementations with respect to the specifications.

One formal approach to coordination was developed by Singh [77]. This approach represents each agent as a small *skeleton*, which includes only the *events* or *transitions* made by the agent that are significant for coordination. Coordination requirements are stated as temporal logic formulas involving the events. Formulas have been obtained that can capture the coordination requirements that arise in the literature.

The specific approach uses a temporal logic that is a variant of the linear temporal logic of Section 8.2.6. For that logic, it is possible to compile the specification in such a way as to localize most decision-making information on the individual agents. Effectively, the agents relinquish part of their autonomy (or their designers do it for them) when they decide to be coordinated. This leads to constraints on some of their events. If the agents respect these constraints, then the system as a whole behaves in the desired coordinated manner.

Sometimes, the term *coordination* is taken to mean a bit more than the above. In such cases, coordination involves the agents' beliefs and intentions. We discuss such cases under *collaboration* below.

8.5.1 Architecture

We now discuss the architecture that underlies a distributed coordination scheme based on temporal logic. We assume that agents are designed autonomously, and their internal details may be inaccessible. Also, that agents act autonomously and may unilaterally perform certain actions within their purview. However, in order to be able to coordinate the agents at all, the designer of the multiagent system must have some limited knowledge of the designs of the individual agents. This knowledge is in terms of their externally visible actions, which are potentially significant for coordination. We call these the significant *events* of the agent. In other words, the only events we speak of are those publicly known—the rest are of no concern to the coordination service. These events are organized into *skeletons* that characterize the coordination behavior of the agents. The idea of using events and skeletons is well-known from logics of programs [25].

Event Classes

We allow four classes of events, which have different properties with respect to coordination. Events may be

- *flexible*, which the agent is willing to delay or omit
- *inevitable*, which the agent is willing only to delay
- *immediate*, which the agent performs unilaterally, that is, is willing neither to delay nor to omit
- *triggerable*, which the agent is willing to perform based on external request.

The first three classes are mutually exclusive; each can be conjoined with triggerability. We do not have a category where an agent will entertain omitting an event, but not delaying it, because unless the agent performs the event unilaterally, there must be some delay in receiving a response from the coordination service.

Agent Skeletons

It is useful to view the events as organized into a *skeleton* to provide a simple representation of an agent for coordination purposes. This representation is typically a finite state automaton. Although the automaton is not used explicitly by the coordination service during execution, it can be used to validate specified coordination requirements. The set of events, their properties, and the skeleton of an agent depends on the agent, and is application-specific. The coordination service is independent of the exact skeletons or events used in a multiagent system. Examples 8.17 and 8.18 discuss two common skeletons in information search.

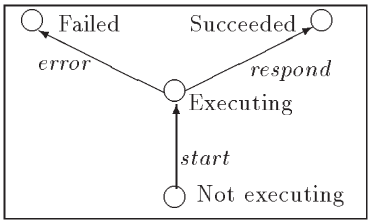

Figure 8.5 Skeleton for a simple querying agent.

Example 8.17
Figure 8.5 shows a skeleton that is suited for agents who perform one-shot queries. Its significant events are *start* (accept an input and begin), *error*, and *respond* (produce an answer and terminate). The application-specific computation takes place in the node labeled "Executing." We must also specify the classes of the different events. For instance, we may state that *error* and *respond* are immediate, and *start* is flexible and triggerable.

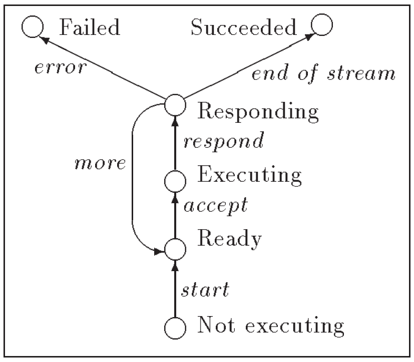

Figure 8.6 Skeleton for an information filtering agent.

Example 8.18
Figure 8.6 shows a skeleton that is suited for agents who filter a stream, monitor a database, or perform any activity iteratively. Its significant events are *start* (accept an input, if necessary, and begin), *error*, *end of stream*, *accept* (accept an input, if necessary), *respond* (produce an answer), *more* (loop back to expecting more input). Here, too, the application-specific computation takes place in the node labeled "Executing." The events *error*, *end of stream*, and *respond* are immediate, and all other events are flexible, and *start* is in addition triggerable.

8.5.2 Specification Language

$\mathcal{L}_C$ is a language for specifying coordinations. It is a variant of $\mathcal{L}_L$, the linear-time language, with some restrictions. $\mathcal{L}_C$ is $\mathcal{L}_P$ augmented with the *before* $(\cdot)$ temporal operator. Before is related to the until operator of $\mathcal{L}_L$: it is used because it is easier to process symbolically for the purpose at hand. The literals denote event types, and can have parameters. Here we only consider the nonparameterized case, for simplicity. Also, in $\mathcal{L}_C$ negation applies only on the atoms, and is written as a $^-$ (bar) to highlight this fact. Further, the atoms are interpreted as events, such as are listed in the agent skeletons.

SYN-20. $\psi \in \Phi$ implies that $\psi, \overline{\psi} \in \mathcal{L}_C$

SYN-21. $p, q \in \mathcal{L}_C$ implies that $p \wedge q$, $p \vee p$, $p \cdot q \in \mathcal{L}_C$

The semantics of $\mathcal{L}_C$ is given with respect to a model $M_6 \stackrel{\text{def}}{=} \langle \mathbf{T}, <, [\![\,]\!] \rangle$. M_6 has the same structure as M_3. However, we restrict M_6 further so that it consists of paths or traces, which are *consistent*. By a consistent trace, we mean one on which no event is repeated and an event and its complement do not both occur. The following semantic definitions take as their index a given trace, τ, not a specific moment on it, as for the previous semantic definitions. The motivation for this is that in giving a specification we only care about the behavior of the system as given by a trace, not by what may or may not have transpired at a given moment. When we execute the coordinations, we do care about the specific moments, of course, but that is not the concern of the specifier. The operator $\odot$ denotes concatenation of two traces, the first of which is finite. The following semantics looks at specific indices of a trace (as in τ_i). This substitutes for the labeling function or $[\![\,]\!]$ used previously, and emphasizes the fact that each event happens at a particular moment.

SEM-36. $M_6 \models_\tau \psi$ iff $(\exists i : \tau_i = \psi)$, where $\psi \in \Phi$

SEM-37. $M_6 \models_\tau \overline{\psi}$ iff $(\exists i : \tau_i = \overline{\psi})$, where $\psi \in \Phi$

SEM-38. $M_6 \models_\tau p \wedge q$ iff $M_6 \models_\tau p$ and $M_6 \models_\tau q$

SEM-39. $M_6 \models_\tau p \vee q$ iff $M_6 \models_\tau p$ or $M_6 \models_\tau q$

SEM-40. $M_6 \models_\tau p \cdot q$ iff $(\exists \sigma, \gamma : (\tau = \sigma \odot \gamma) \& M_6 \models_\sigma p \& M_6 \models_\gamma q)$

$\overline{\psi}$ refers to the complement of ψ. From the above, it is possible that a trace τ may satisfy neither ψ nor $\overline{\psi}$. In this way, negation in $\mathcal{L}_C$ is stronger than in traditional logics. $\overline{\psi}$ means that it is definite that ψ will never occur. Consequently, maximal traces will satisfy $\psi \vee \overline{\psi}$.

Singh [77] presents a set of equations that enable symbolic reasoning on $\mathcal{L}_C$ to determine when a certain event may be permitted, prevented, or triggered.

8.5.3 Common Coordination Relationships

Coordinations are specified by expressing appropriate relationships among the events of different agents. $\mathcal{L}_C$ allows a variety of relationships to be captured.

Table 8.2 presents some common relationships. Some of the relationships involve coordinating multiple events. For example, R8 captures requirements such as that

	Name	**Description**	**Formal notation**
R1	e is required by f	If f occurs, e must occur before or after f	$e \vee \overline{f}$
R2	e disables f	If e occurs, then f must occur before e	$\overline{e} \vee \overline{f} \vee f \cdot e$
R3	e feeds or enables f	f requires e to occur before	$e \cdot f \vee \overline{f}$
R4	e conditionally feeds f	If e occurs, it feeds f	$\overline{e} \vee e \cdot f \vee \overline{f}$
R5	Guaranteeing e enables f	f can occur only if e has occurred or will occur	$e \wedge f \vee \overline{e} \wedge \overline{f}$
R6	e initiates f	f occurs iff e precedes it	$\overline{e} \wedge \overline{f} \vee e \cdot f$
R7	e and f jointly require g	If e and f occur in any order, then g must also occur (in any order)	$\overline{e} \vee \overline{f} \vee g$
R8	g compensates for e failing f	if e happens and f does not, then perform g	$(\overline{e} \vee f \vee g) \wedge (\overline{g} \vee e) \wedge (\overline{g} \vee \overline{f})$

Table 8.2 Example relationships.

if an agent does something (e), but another agent does not match it with something else (f), then a third agent can perform g. This is a typical pattern in applications with data updates, where g corresponds to an action to restore the consistency of the information (potentially) violated by the success of e and the failure of f. Hence the name *compensation*.

8.6 Communications

Communications are a natural way in which the agents in a DAI system may interact with one another other than through incidental interactions through the environment. Communications is discussed in detail in Chapter 2.

Speech act theory, which originated in the philosophy of language, gives the basis for communications. Speech act theory is founded on the idea that with language you not only make statements, but also *perform actions* [3]. For example, when you request something you do not just report on a request, but you actually effect the request; when a justice of the peace declares a couple man and wife, she is not reporting on their marital status, but changing it. The stylized syntactic form for speech acts that begins "I hereby request ..." or "I hereby declare ..." is called a *performative*. With a performative, literally, saying it makes it so! [3, p. 7]. Interestingly, verbs that cannot be put in this form are not speech acts. For example, "solve" is not a performative, because "I hereby solve this problem" just does not work out—or Math students would be a much happier lot! For most computing purposes, speech acts are classified into assertives (informing), directives (requesting or querying), commissives (promising), permissives, prohibitives, declaratives (causing events in themselves, e.g., what the justice of the peace does in a marriage ceremony), expressives (expressing emotions and evaluations).

Austin identified three main aspects of a speech act. The *locution* refers to the lowest level of the speech act, namely, the string that is transmitted. The *illocution* refers to the intrinsic meaning of the speech act. The *perlocution* refers to the possible effects of the speech act on the recipients. The locution can be varied and the perlocutions depend on the recipient. However, the illocution tells us the meaning that is conveyed. For this reason, studies of communication in DAI focus primarily on the illocutions.

8.6.1 Semantics

Formalizing the semantics of communications has proved a longstanding challenge. This is partly because more than one view of what can be formalized is possible. The earliest work was carried out in computational linguistics, and sought to determine the conditions under which the intended meaning of a speech act might be inferred. For example, given a locution in the form of a question (e.g., "can you pass the salt?"), one might infer an illocution that is a request (e.g., "please pass the salt").

There is considerable subtlety involved in this reasoning, but for the most part, it is specific to human languages and can be avoided in DAI.

A different approach was developed by Singh [78]. This approach sought to give the objective criteria under which speech acts of different illocutionary forces could be said to be satisfied. The idea was to identify the conditions in a framework that highlighted the proof-obligations of a designer in showing that different speech acts were satisfied. Following Hamblin [38], Singh defined a notion of *whole-hearted satisfaction*. This was formalized using a modal operator; truth conditions for this operator corresponded to satisfaction conditions for the corresponding speech acts. An example condition is that a directive for p is whole-heartedly satisfied if and only if the recipient adopts and intention to satisfy p, has the know-how to achieve p, and acts resulting in p.

Recently, Labrou & Finin have developed a formal semantics for communications and *conversations* (consisting of a series of communications) that considers the preconditions and postconditions for each speech act. These conditions are stated in terms of the beliefs and wants of the participating agents.

Fundamentally, communication is a social phenomenon. Although this fact is noted in informal discussions, existing approaches have not recognized it in their theoretical development. We believe that the study of social primitives (discussed below) has advanced enough that directly social semantics of communications can now be explored. We leave the development as a significant open research problem in DAI.

8.6.2 Ontologies

An ontology is a representation of some part of the world. Ontologies are thus of interest to knowledge representation. Although ontologies in themselves are not a social concept, they can provide a shared "virtual world" that can serve as the basis for communications [44]. In fact, when many people talk of the "semantics" of a communication, they mean understanding the concepts and terms used in it. Ontologies provide a natural, declarative way of identifying concepts and terms. If two agents agree on the upper nodes of a taxonomy, they can jointly traverse the taxonomy till they find the location of a newly introduced concept. Thus, they can build a shared understanding of their content language. It is this fact that makes ontologies interesting. They found much application in DAI systems, especially those involving access to, or interactions among, information systems and databases [22, 90]. Consequently, ontologies are included in several multiagent architectures.

Ontologies are amenable to formal methods in two main places. One place is in the algorithms for processing ontologies, which exploit the connection between lattice theory and taxonomies [43]. Another place is in approaches to help interlink ontologies developed by different vendors, or incorporated by different agents, who must reconcile them in order to communicate. An interesting class of approaches may be based on algebraic techniques [89]; however, this work is still in its infancy.

8.7 Social Primitives

Arguably, it is the active use of social concepts in its design and implementation that distinguish a DAI system from a traditional distributed computing system [31]. We lump into the category of social primitives those that concern societies of agents as well as those that concern smaller and more heavily structured organizations. Some related social concepts are introduced in Chapter 2, and organizational concepts in Chapter 7.

8.7.1 Teams and Organizational Structure

A *group* or multiagent system is a system of agents that are somehow constrained in their mutual interactions. Typically, these constraints arise because the agents play different *roles* in the group, and their roles impose requirements on how they are to behave and interact with others. A *team* is a group in which the agents are restricted to having a common goal of some sort. Typically, team-members cooperate and assist each other in achieving their common goals. Groups and teams prove to be a fertile ground for the development of formal theories in DAI, especially theories that are unlike the theories in traditional AI or computer science. We emphasize, however, that some of this work is still in an early stage, and the descriptions below, although moderately stable, should not be taken as final.

Some good work has focused on formalizing cooperative problem solving [92], and the representations needed for effective cooperation [21].

8.7.2 Mutual Beliefs and Joint Intentions

One of the oldest ways of lifting single-agent concepts to multiagent concepts is through the use of *mutual beliefs*. A set of agents is said to have a mutual belief that p if they each (a) believe p, (b) believe that condition (a) holds of the others (that they believe p), (c) believe that condition (b) holds of the others, and so on. Mutual belief thus provides a means to achieve the effect of a perfectly shared mental state. It has been argued the mutual beliefs can account for various aspects of human communication [13, 39] and social conventions [58].

Levesque & Cohen developed an approach that generalizes the notion of intentions to *joint intentions* [57]. This theory is extremely complicated, and our presentation can at best be thought of an intuitive approximation of the original. A joint intention for p exists among a group of agents if they (a) each have a goal that p, (b) each will persist with this goal until it is mutually believed that p has been achieved or that p cannot be achieved, (c) conditions (a) and (b) are mutually believed.

Grosz & Kraus develop a formal theory of shared plans [35]. This theory relates the cooperative activities of agents via their individual and shared plans. A distinction is sometimes made between an agent intending to achieve something and an agent intending that some condition be obtained. Usually, actions and propositions are closely related, although they are often treated differently in human languages. Grosz & Kraus adapt this idea to develop a framework in which the agent is itself committed to performing the intentions toward actions, but depending on the situation can act on the intentions for propositions that are held by its team-members (and, similarly, can expect others to take on the propositions it intends).

On the one hand, mutual beliefs play a role in several theories; on the other hand, it is well-known that if communications among the agents are not reliable (in terms of delivery and delay), then mutual beliefs cannot be attained [11, 37]. In other words, the mutual beliefs are limited to the beliefs that the designer hard-wires into the agents at the start, but additional mutual beliefs cannot be attained.

This conflict between some theoretically appealing properties of mutual beliefs and their infeasibility in practical situations has led some researchers to explore alternative ways to achieve the same effect. It has been suggested that social primitives, appropriately formalized, might provide a more direct means to capture the social aspects of multiagent systems, which apparently are the ones that mutual beliefs seek to capture.

8.7.3 Social Commitments

Section 8.3.4 introduced psychological commitments. Here we consider social commitments, which are the commitments that an agent toward another agent [10, 74]. Such commitments related to directed obligations [55] as studied in deontic logic (see Section 8.2.4). Social commitments are a genuinely multiagent concept, since they have no analog in a single-agent system. Social commitments can potentially be used to give clear specifications at the social level of how the agents in a multiagent system ought to interact; such specifications will not delve into implementational details, and give maximal freedom to diverse designers to implement agents that can behave together cohesively.

Although concepts such as social commitments have long been identified, this topic has drawn much interest recently [10, 16, 23, 66, 80]. Castelfranchi introduced the idea of a *witness* of a commitment, which certifies to its creation [10]. Singh generalizes notion to a *context group*, which is usually the multiagent system within which the given commitment exists [80]. The formalization of social commitments involves defining an independent primitive. They also involve the description of associated notions such as the *roles* that may exist in the given multiagent system, and what capabilities and authorities (or authorizations) agents would need to play specific roles. This work is still in its infancy, but we encourage the reader to peruse the cited works for some open research problems.

8.7.4 Group Know-How and Intentions

There is a view that multiagent systems can themselves be treated as agents. These are then referred to as *groups* and distinguished from ordinary *individual* agents.

In many interesting cases, when an agent interacts with another entity, it may have no knowledge or concern that the other entity is an individual or a group. It may have expectations about the other entities as usual, and may enter into social commitments with it. Thus the other entity is justifiably treated as an agent.

A natural question is how may we define the beliefs, knowledge, know-how, and intentions of groups. Some conventional approaches were mentioned in Section 8.7.2. An alternative approach is to define the structure of a group explicitly, and define the intentions and know-how of the group as based on its structure and the intentions and know-how of its members. The structure may itself be formalized in several ways. One way is through a combination of the *reactive* and the *strategic* interactions among the members that are called for by the group [71, 72].

For reasons of space, we only consider group intentions below. Recall the scenarios selected by the model component **I** in formalizing intentions. With reactive interactions, the selected scenarios are restricted to those that satisfy some additionally specified temporal (path) formulas, which intuitively correspond to the habits of interaction of the different members. Similarly, strategic interactions restrict the selected scenarios to those in which the specified communications among the members are satisfied. For example, a group could require that all directives issued by an agent playing the role of leader must be satisfied, or that all commitments created through explicit promises must be discharged. These requirements eliminate unacceptable scenarios, leading to a stronger notion of intentions than if we considered the agents individually. However, this notion is potentially weaker than traditional notions, which always require some form of mutual belief among the members.

Interestingly, when formalized, the above definitions lead to some algebraic properties of group intentions that relate to the underlying structure of the given groups [71].

8.8 Tools and Systems

Now we present a variety of implemented tools and systems for DAI that bear some significant connection with the formal techniques introduced above. We have three categories of these tools and systems: those that follow the above approaches closely; those that are essentially traditional techniques applied to DAI, and those that were informally influenced by the DAI approaches.

8.8.1 Direct Implementations

We now review some of the popular systems that are fairly directly based on the above ideas.

PRS and dMARS

The Procedural Reasoning System (PRS) [33] was one of the first implemented systems to be based on a BDI architecture. As described in the foregoing, PRS provides goal-oriented as well as reactive behavior. It was implemented in LISP and has been used for a wide range of applications in problem diagnosis for the Space Shuttle [46], air-traffic management [59], and network management [46].

dMARS is a faster, more robust reimplementation of PRS in C++. It has been used in a variety of operational environments, including paint shop scheduling in car manufacturing, air combat simulation, resource exploration, malfunction handling on NASA's space shuttle, and management of business processes in Internet and call center applications [49].

COSY

COSY is also a BDI architecture, and bears several similarities to PRS and dMARS [36]. It involves the same concepts, and uses plans as its core representation. However, in addition, COSY has gives importance to both psychological and social commitments. COSY has a strong component of cooperation, which is based on formal protocols built on top of an agent communication language. The formation of commitments is declaratively captured in various rules. The above protocols involve commitments among the agents, and include rules through which tasks may be delegated to and adopted by different agents.

Agent-Oriented Languages

The concepts discussed in the chapter are also finding their way into programming language constructs. Shoham [69] in his proposal for an agent-oriented language called AGENT0 made extensive use of notions such as beliefs, commitments, and know-how. The language was subsequently extended by Thomas [88] to include planning capability similar to that of BDI architectures.

Agent-oriented languages based on alternative formalisms are also gaining ground. Golog and ConGolog [56] are logic programming languages that allow explicit reasoning about actions. The system is based on situation calculus to represent and reason about change [61]. As the Golog interpreter can reason about actions it can avoid "dead paths" that the BDI interpreter cannot. However, it does not offer the reactivity offered by the BDI architecture because of its inability to indirectly invoke the execution of plans.

Concurrent MetateM

An alternative approach uses temporal logic to specify the behavior of agents. A Concurrent MetateM system [29] consists of a set of objects each executing temporal specifications. A rule in this language is of the form "past and present formula" implies "present or future formula." As a result, execution of this rule involves matching the antecedent of these rules against the history of incoming messages and then executing the present and future-time consequents. Enhancements with explicit BDI operators are beginning to be developed [30].

ARTIMIS

Breiter & Sadek have implemented a formal theory of beliefs and intentions in the ARTIMIS system [7]. The ARTIMIS system carries out intelligent dialogue with a user in assisting the user in tasks such as information access. This system, being designed as a user interface, applies the Gricean maxims, whereby the computer attempts to infer the user's intentions and act accordingly. It also uses an agent communication language, Arcol, to carry out a dialogue with the user.

DEPNET

DEPNET is an interpreter for agents who can perform social reasoning [70]. Agents in DEPNET represent knowledge about one another to determine their relative autonomy or dependence for various goals. Dependence leads to joint plans for achieving the intended goals. The underlying theory is based on dependence rather than social commitments. Thus it is more amenable to processing by the agents individually, but is also more limited because it cannot easily capture the normative aspects of social interaction among agents. However, this tool shows how social notions can be realized in tools for simulating and analyzing multiagent systems.

TFM-CAA: Coordinating Autonomous Agents

TFM-CAA is an implementation of a customizable coordination service based on the approach described in Section 8.5. This service (a) takes declarative specifications of the desired interactions, and (b) automatically enacts them. This approach enacts the coordination requirements in a distributed manner with minimal intrusion into the design of the agents being coordinated.

8.8.2 Partial Implementations

These are systems that do not involve a full implementation of the theoretical concepts, but were influenced by the theories and used them in designing their solutions. They are, however, full systems in their own right.

STEAM

STEAM is an architecture for teamwork by agents [87]. STEAM offers abstractions for teams, based on the work on joint intentions and shared plans. STEAM also uses some coordination abstractions. One of STEAM's features is the specification of team plan operators in terms of *role operators*—that is, plan operators for member agents. Three *role-monitoring constraints* are defined, through which STEAM can infer the potential achievability of a team operator. If a team operator becomes unachievable because of a role-monitoring failure, it can be repaired by examining the roles that caused the failure. STEAM is being enhanced with functionality using which an agent can compare its behavior to that of its peers and thereby determine if a failure has occurred. STEAM has been applied in domains such as military helicopter missions and simulated soccer.

Carnot

Carnot was a research project primarily focused on accessing and updating information from heterogeneous databases, such as are common in large enterprises [91]. Carnot was applied on accessing information from legacy databases, automating workflow for service-order processing, and retrieving related information from structured and text databases [83]. In these applications, Carnot adapted formal techniques for ontology management [43] and transaction management [84]. The latter were a precursor of the formal theory later extended to coordinating autonomous agents, as described in Section 8.5.

ARCHON

The ARCHON project developed a domain-independent architecture of multiagent systems, which was applied in an electricity transportation management system and a particle accelerator [48]. This architecture emphasized the role of cooperation among agents through a declarative representation of cooperation, which was reasoned about explicitly. The agents autonomously detected the need to cooperate—this generalizes distributed problem solving, and enhances the autonomy of the agents. The agents maintain *self models* and *acquaintance models* to effectively decide when and how to cooperate. This system adapted the notion of joint intentions mentioned above. It also included a framework for information access similar to Carnot's.

maDes

Ishizaki develops maDes, a *multiagent model of dynamic design*. Design is understood as the creative activity in which a designer constructs a suitable representation for a message [47]. Ishizaki's model is interesting to the design community, because it emphasizes the dynamic or active aspects of modern media, such as computers. It is interesting to the agent community, because it finds a novel application of agents. It considers a number of agents with different abilities who come together to create a composite design. This model uses the theory of group ability as its basis for defining the reactive interaction among design agents [72].

8.8.3 Traditional Approaches

This section reviews some formal approaches that initially were designed for traditional software engineering, but which are being applied to DAI systems. We include these here, because as we have maintained in this Chapter, DAI requires the careful synthesis of traditional and new techniques.

DESIRE

Design and Specification of Interacting Reasoning Components, better known as DESIRE, is a framework for the design and specification of multiagent systems [6]. DESIRE can be thought of as an object modeling framework with enhancements for DAI. The primary unit of representation in DESIRE is a task. The user can specify task composition, sequencing of tasks, and task delegation, in addition to the information exchanged between agents and the knowledge structures that capture the domain knowledge. Tasks are similar to PRS plans, except that when it comes to execution plans are executed indirectly by posting an event to achieve a goal, rather than directly. This has the advantage that any external events can be handled *during* the execution of a plan.

The Z Specification Language

The Z language was developed for the formal specification of software systems [86]. It has found application in DAI as well. One class of uses of Z involves formally specifying properties such as the autonomy and dependence of agents in multiagent systems, as well as the cognitive concepts discussed above [20]. Another use involves formalizing existing systems after the fact to give a mathematical characterization of their behavior that may be more faithful than a pure knowledge-level BDI treatment [19].

8.9 Conclusions

As DAI matures and its applications expand into increasingly critical settings, we will need sophisticated approaches for engineering DAI systems. As in other branches of computer science, these approaches will involve a combination of tools and methodologies. Effective tools and methodologies must not only support a rich variety of powerful abstractions, but also be founded on and respect rigorous treatments of the abstractions they support.

DAI systems involve a variety of concepts. Some of these are the BDI concepts that have been studied for the longest time in DAI. Other relevant concepts involve communications among agents as well as a wide range of coordination and social primitives. Consequently, formal methods in DAI inherently involve mathematical structures that explicate these notions. Although formal methods in DAI are still in their infancy, some interesting results have been obtained. The formal techniques have also been used to influence a variety of practical systems.

However, an important caveat is that most of the present-generation systems that "implement" various theories have only limited fidelity to those theories. They need to go beyond the theories to a significant extent. This deviation is essential because current theories tend to be incomplete in their coverage and somewhat simplistic and top-heavy. Consequently, more than in traditional systems, DAI systems require a greater contribution of insights from their developers. Although the insights are valuable, their insertion detracts from the formal underpinnings of the work, because the insights are typically *ad hoc*, and do not facilitate establishing the kinds of properties that make formal methods attractive.

This speaks to the need for carefully engineered, tractable logics that may not be expressive in general, but have the power needed for a specific class of tasks. Full automation may not be essential, especially at design time, if the insights a human may offer are from a well-understood set of patterns. But, of course, that is what tools and methodologies are all about. Consequently, a range of future challenges is to develop well-honed formal theories that cover the phenomena that emerge in practice, are more accurate in their treatment of real systems, and can be used to analyze and design them.

8.10 Exercises

1. *[Level 1]* Formalize the following conditions in propositional logic:
   (a) it is cold
   (b) it is cold in room 1344
   (c) room 1344 has an air conditioner
   (d) the agent x feels cold
   (e) if it is raining, it is cold
2. *[Level 1]* Formalize the following conditions in temporal logic:
   (a) room 1344 will always be cold
   (b) if room 1344 gets cold, it will stay cold forever
   (c) room 1344 will repeatedly be getting cold and hot
3. *[Level 1]* Formalize the following conditions in dynamic logic:
   (a) turning on the air conditioner makes room 1344 cold
   (b) turning off the air conditioner does not make room 1344 hot
4. *[Level 2]* Formalize the following conditions in predicate logic [26] (requires extra reading):
   (a) every room with an air conditioner is cold

   (b) the agent x feels cold in every room that has an air conditioner
   (c) some agent feels cold in every room that has an air conditioner

5. *[Level 2]* Verify the correspondence between the properties on accessibility relations and inferences in modal logic, as mentioned in Section 8.2.3.
6. *[Level 2]* Translate `while` loops from Algol-60 into regular programs.
7. *[Level 2]* Relate *partial* and *total* correctness of programs (as defined in any introductory text on analysis of programs) with the dynamic logic operators.
8. *[Level 2]* Prove or disprove the following properties about $\mathcal{L}_L$:
   - $\mathsf{FF}p \rightarrow \mathsf{F}p$
   - $\mathsf{G}p \rightarrow \mathsf{F}p$
   - $\mathsf{GG}p \rightarrow \mathsf{G}p$
   - $\mathsf{GG}p \rightarrow \mathsf{GF}p$
   - $\mathsf{GF}p \rightarrow \mathsf{FG}p$
   - $\mathsf{FG}p \rightarrow \mathsf{GF}p$
   - $\mathsf{FGF}p \equiv \mathsf{GF}p$
9. *[Level 2]* Prove or disprove the following properties about $\mathcal{L}_B$:
   - EXtrue
   - $\mathsf{AGAG}p \rightarrow \mathsf{AGAF}p$
   - $\mathsf{E}(p\mathsf{U}q) \rightarrow (q \vee p \wedge \mathsf{EX}(\mathsf{E}(p\mathsf{U}q)))$
   - $(q \vee p \wedge \mathsf{EX}(\mathsf{E}(p\mathsf{U}q))) \rightarrow \mathsf{E}(p\mathsf{U}q)$
10. *[Level 2]* Establish the results mentioned in the context of Constraints cons-i-sat, IC2, and IC3 in Section 8.3.3.
11. *[Level 2]* Prove or disprove the following properties about know-how (the agent is omitted):
    - $\mathsf{K_h}p \rightarrow \mathsf{K_h}\mathsf{K_h}p$
    - $\mathsf{K_h}p \rightarrow (\mathsf{K_t}p \vee (\bigvee a : \mathsf{E}\langle a \rangle \mathsf{true} \wedge \mathsf{A}[a]\mathsf{K_h}p))$
    - $(\mathsf{K_t}p \vee (\bigvee a : \mathsf{E}\langle a \rangle \mathsf{true} \wedge \mathsf{A}[a]\mathsf{K_h}p)) \rightarrow \mathsf{K_h}p$
12. *[Level 3]* Implement a BDI interpreter based on the architecture described above.
    (a) Make turning on the air conditioner makes room 1344 cold
    (b) turning off the air conditioner does not make room 1344 hot
13. *[Level 3]* Implement a deliberation component of a BDI interpreter based on heuristic graph search.
14. *[Level 4]* What might be the nature of a social-level semantics for agent communication languages? Give such a semantics.
    (a) reconcile it with conventional approaches based on the BDI notions

(b) develop a scheme for testing compliance with your semantics of implementations by different vendors.

References

1. Alfred V. Aho and Jeffrey D. Ullman. *Principles of Compiler Design.* Addison-Wesley, Reading, MA, 1977.
2. Alan Ross Anderson and Nuel D. Belnap. *Entailment: The Logic of Relevance and Necessity.* Princeton University Press, Princeton, 1975.
3. John L. Austin. *How to Do Things with Words.* Clarendon Press, Oxford, 1962.
4. Nuel Belnap and Michael Perloff. Seeing to it that: A canonical form for agentives. *Theoria*, 54(3):175–199, 1988.
5. Michael E. Bratman. *Intention, Plans, and Practical Reason.* Harvard University Press, Cambridge, MA, 1987.
6. Frances M. T. Brazier, Barbara M. Dunin-Kęplicz, Nick Jennings, and Jan Treur. Desire: Modelling multi-agent systems in a compositional formal framework. *International Journal of Cooperative Information Systems*, 6(1):67–94, 1997.
7. Phillipe Breiter and M. David Sadek. A rational agent as a kernel of a cooperative dialogue system: Implementing a logical theory of interaction. In *ECAI-96 Workshop on Agent Theories, Architectures, and Languages*, pages 261–276. Springer-Verlag, 1996.
8. Omran A. Bukhres and Ahmed K. Elmagarmid, editors. *Object-Oriented Multidatabase Systems: A Solution for Advanced Applications.* Prentice-Hall, 1996.
9. J. R. Burch, E. C. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the 5th International Symposium on Logic in Computer Science*, pages 428–439, 1990.
10. Cristiano Castelfranchi. Commitments: From individual intentions to groups and organizations. In *Proceedings of the International Conference on Multiagent Systems*, pages 41–48, 1995.
11. K. M. Chandy and Jayadev Misra. How processes learn. *Distributed Computing*, 1:40–52, 1986.
12. Brian F. Chellas. *Modal Logic.* Cambridge University Press, New York, 1980.
13. Herbert H. Clark and Thomas B. Carlson. Speech acts and hearer's beliefs. In *[85]*, pages 1–36. 1982.
14. E. Clarke, O. Grumberg, and D. Long. Model checking. In *Proceedings of the International Summer School on Deductive Program Design*, pages 428–439, 1990.
15. Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.
16. Rosaria Conte and Cristiano Castelfranchi. *Cognitive and Social Action.* UCL Press, London, 1995.
17. Giuseppe De Giacomo and Xiao Jun Chen. Reasoning about nondeterministic and concurrent actions: A process algebra approach. In *Proceedings of the National Conference on Artificial Intelligence*, pages 658–663, 1996.

18. Yves Demazeau and Jean-Pierre Müller, editors. *Decentralized Artificial Intelligence, Volume 2.* Elsevier/North-Holland, Amsterdam, 1991.
19. Mark d'Inverno, David Kinny, Michael Luck, and Michael Wooldridge. A formal specification of dMARS. In *Intelligent Agents IV: Agent Theories, Architectures, and Languages*, 1998.
20. Mark d'Inverno and Michael Luck. Understanding autonomous interaction. In *Proceedings of the European Conference on Artificial Intelligence*, 1996.
21. Mark d'Inverno, Michael Luck, and Michael Wooldridge. Cooperation structures. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 600–605, 1997.
22. Michael L. Dowell, Larry M. Stephens, and Ronald D. Bonnell. Using a domain-knowledge ontology as a semantic gateway among information resources. In *[45]*, pages 255–260. 1997. (*Reprinted from* Proceedings of the IJCAI Workshop on Basic Ontological Issues in Knowledge Sharing, 1995).
23. Barbara Dunin-Kęplicz and Rineke Verbrugge. Collective commitments. In *Proceedings of the International Conference on Multiagent Systems*, pages 56–63, 1996.
24. E. Allen Emerson. Temporal and modal logic. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. North-Holland, Amsterdam, 1990.
25. E. Allen Emerson and Edmund C. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2:241–266, 1982.
26. Herbert B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, San Diego, 1972.
27. Ronald Fagin and Joseph Y. Halpern. Belief, awareness, and limited reasoning. *Artificial Intelligence*, 34:39–76, 1988.
28. Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge.* MIT Press, Cambridge, MA, 1995.
29. Michael Fisher. A survey of concurrent MetateM - the language and its applications. In *Proceedings of the 1st International Conference on Temporal Logic (ICTL)*, 1994.
30. Michael Fisher and Michael Wooldridge. On the formal specification and verification of multi-agent systems. *International Journal of Intelligent and Cooperative Information Systems*, 6(1):37–65, 1997.
31. Les Gasser. Social conceptions of knowledge and action: DAI foundations and open systems semantics. In *[45]*, pages 389–404. 1997. (*Reprinted from* Artificial Intelligence, 1991).
32. Michael P. Georgeff and F. Felix Ingrand. Decision-making in an embedded reasoning system. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1989.
33. Michael P. Georgeff and Amy L. Lansky. Procedural knowledge. *Proceedings of the IEEE*, 74:1383–1398, 1986.
34. Michael P. Georgeff and Anand S. Rao. The semantics of intention maintenance for rational agents. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 704–710, 1995.

35. Barbara J. Grosz and Sarit Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269–357, October 1996.

36. Afsaneh Haddadi. *Communication and Cooperation in Agent Systems : A Pragmatic Theory*. Springer-Verlag, Heidelberg, 1996.

37. Joseph Y. Halpern and Yoram O. Moses. Knowledge and common knowledge in a distributed environment. *Journal of the Association for Computing Machinery*, 37:549–587, 1990.

38. C. L. Hamblin. *Imperatives*. Basil Blackwell, Oxford, 1987.

39. Gilbert Harman. Review of Jonathan Bennett's *Linguistic Behaviour*. *Language*, 53(2):417–424, 1977.

40. Risto Hilpinen, editor. *Deontic Logic: Introductory and Systematic Readings*, volume 33 of *Synthese Library*. D. Reidel, Dordrecht, Holland, 1971.

41. Risto Hilpinen, editor. *New Studies in Deontic Logic: Norms, Actions, and the Foundations of Ethics*, volume 152 of *Synthese Library*. D. Reidel, Dordrecht, Holland, 1981.

42. Jaakko Hintikka. *Knowledge and Belief: An Introduction to the Logic of the Two Notions*. Cornell University Press, Ithaca, 1962.

43. Michael N. Huhns, Christine Collet, and Wei-Min Shen. Resource integration using a large knowledge base in Carnot. *IEEE Computer*, 24(12):55–62, December 1991.

44. Michael N. Huhns and Munindar P. Singh. Ontologies for agents. *IEEE Internet Computing*, 1(6):81–83, December 1997. Instance of the column *Agents on the Web*.

45. Michael N. Huhns and Munindar P. Singh, editors. *Readings in Agents*. Morgan Kaufmann, San Francisco, 1997.

46. F. Felix Ingrand, Michael P. Georgeff, and Anand S. Rao. An architecture for real-time reasoning and system control. *IEEE Expert*, 7(6), 1992.

47. Suguru Ishizaki. Multiagent model of dynamic design: Visualization as an emergent behavior of active design agents. In *[45]*, pages 172–179. 1997. (*Reprinted from* Proceedings of the ACM Conference on Computer Human Interaction, 1996).

48. Nick R. Jennings, E. H. Mamdani, Jose Manuel Corera, Inaki Laresgoiti, Fabien Perriollat, Paul Skarek, and Laszlo Zsolt Varga. Using Archon to develop real-world DAI applications, part 1. *IEEE Expert*, 11(6):64–70, December 1996.

49. David Kinny and Michael P. Georgeff. Modelling and design of multi-agent systems. In *Intelligent Agents III: Agent Theories, Architectures, and Languages*, pages 1–20, 1997.

50. Kurt Konolige. *A Deduction Model of Belief*. Morgan Kaufmann, 1986.

51. Kurt G. Konolige and Martha E. Pollack. A representationalist theory of intentions. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1989.

52. Dexter Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

53. Dexter Kozen and Jerzy Tiurzyn. Logics of program. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 789–840. North-Holland, Amsterdam, 1990.

54. Saul A. Kripke. Semantical analysis of modal logic I: Normal modal propositional calculi. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 9:67–96, 1963.

55. Christen Krogh and Henning Herrestad. Getting personal – some notes on the relationship between personal and impersonal obligation. In *Proceedings of the 3rd International Workshop on Deontic Logic in Computer Science (DEON)*, 1996.

56. Yves Lespérance, Hector J. Levesque, Fangzhen Lin, Daniel Marcu, Raymond Reiter, and Richard B. Scherl. Foundations of a logical approach to agent programming. In *Intelligent Agents II: Agent Theories, Architectures, and Languages*, pages 331–346, 1996.

57. H. J. Levesque, P. R. Cohen, and J. T. Nunes. On acting together. In *Proceedings of the National Conference on Artificial Intelligence*, pages 94–99, 1990.

58. David K. Lewis. *Convention: A Philosophical Study.* Harvard University Press, Cambridge, MA, 1969.

59. Magnus Ljungberg and Andrew Lucas. The OASIS air-traffic management system. In *Proceedings of the 2nd Pacific Rim International Conference on Artificial Intelligence (PRICAI)*, 1992.

60. John McCarthy. Ascribing mental qualities to machines. In Martin Ringle, editor, *Philosophical Perspectives in Artificial Intelligence.* Harvester Press, 1979.

61. John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4.* American Elsevier, 1969.

62. John-Jules Ch. Meyer and Roel J. Wieringa, editors. *Deontic Logic in Computer Science: Normative System Specification.* Wiley, Chichester, UK, 1993.

63. Allen Newell. The knowledge level. *Artificial Intelligence*, 18(1):87–127, 1982.

64. Anand S. Rao. Decision procedures for propositional linear-time belief-desire-intention logics. In *Intelligent Agents II: Agent Theories, Architectures, and Languages*, pages 33–48. Springer-Verlag, 1995.

65. Anand S. Rao and Michael P. Georgeff. Modeling rational agents within a BDI-architecture. In *[45]*, pages 317–328. 1997. (*Reprinted from* Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning, 1991).

66. Anand S. Rao, Michael P. Georgeff, and Elizabeth Sonenberg. Social plans: A preliminary report. In *Proceedings of the 3rd European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW)*, pages 57–76, Amsterdam, 1992. Elsevier.

67. Stanley J. Rosenschein and Leslie Pack Kaelbling. A situated view of representation and control. *Artificial Intelligence*, 7, 1995.

68. Krister Segerberg. Bringing it about. *Journal of Philosophical Logic*, 18:327–347, 1989.

69. Yoav Shoham. Agent-oriented programming. In *[45]*, pages 329–349. 1997. (*Reprinted from* Artificial Intelligence, 1993).

70. Jaime Simão Sichman, Rosaria Conte, Yves Demazeau, and Cristiano Castelfranchi. A social reasoning mechanism based on dependence networks. In *[45]*, pages 416–420. 1997. (*Reprinted from* Proceedings of the 11th European Conference on Artificial Intelligence, 1994).

71. Munindar P. Singh. Group intentions. In *Proceedings of the 10th Workshop on Distributed Artificial Intelligence*, October 1990.

72. Munindar P. Singh. Group ability and structure. In *[18]*, pages 127–145. 1991.

73. Munindar P. Singh. Intentions, commitments and rationality. In *Proceedings of the 13th Annual Conference of the Cognitive Science Society*, pages 493–498, August 1991.
74. Munindar P. Singh. Social and psychological commitments in multiagent systems. In *AAAI Fall Symposium on Knowledge and Action at Social and Organizational Levels*, pages 104–106, 1991.
75. Munindar P. Singh. A critical examination of the Cohen-Levesque theory of intentions. In *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 364–368, August 1992.
76. Munindar P. Singh. *Multiagent Systems: A Theoretical Framework for Intentions, Know-How, and Communications.* Springer-Verlag, Heidelberg, 1994.
77. Munindar P. Singh. A customizable coordination service for autonomous agents. In *Proceedings of the 4th International Workshop on Agent Theories, Architectures, and Languages (ATAL)*, July 1997.
78. Munindar P. Singh. A semantics for speech acts. In *[45]*, pages 458–470. 1997. (*Reprinted from* Annals of Mathematics and Artificial Intelligence, 1993).
79. Munindar P. Singh. Applying the mu-calculus in planning and reasoning about action. *Journal of Logic and Computation*, 1998. In press.
80. Munindar P. Singh. An ontology for commitments in multiagent systems: Toward a unification of normative concepts. *Artificial Intelligence and Law*, 1998. In press.
81. Munindar P. Singh. Semantical considerations on intention dynamics for BDI agents. *Journal of Experimental and Theoretical Artificial Intelligence*, 1998. In press.
82. Munindar P. Singh and Nicholas M. Asher. A logic of intentions and beliefs. *Journal of Philosophical Logic*, 22(5):513–544, October 1993.
83. Munindar P. Singh, Philip E. Cannata, Michael N. Huhns, Nigel Jacobs, Tomasz Ksiezyk, Kayliang Ong, Amit P. Sheth, Christine Tomlinson, and Darrell Woelk. The Carnot heterogeneous database project: Implemented applications. *Distributed and Parallel Databases: An International Journal*, 5(2):207–225, April 1997.
84. Munindar P. Singh and Michael N. Huhns. Automating workflows for service provisioning: Integrating AI and database technologies. *IEEE Expert*, 9(5):19–23, October 1994.
85. N. V. Smith, editor. *Mutual Knowledge.* Academic Press, London, 1982.
86. J. M. Spivey. *The Z Notation.* Prentice-Hall International, Hemel Hempstead, UK, 2nd edition, 1992.
87. Milind Tambe. Agent architectures for flexible, practical teamwork. In *Proceedings of the National Conference on Artificial Intelligence*, pages 22–28, 1997.
88. S. Rebecca Thomas. The PLACA agent programming language. In *Intelligent Agents: Agent Theories, Architectures, and Languages*, pages 355–370, 1995.
89. Gio Wiederhold. Value-added mediation. In *Proceedings of the IFIP TC2/WG2.6 Conference on Database Application Semantics (DS-6).* Chapman and Hall, 1995.
90. Gio Wiederhold. Mediators in the architecture of future information systems. In *[45]*, pages 185–196. 1997. (*Reprinted from* IEEE Computer, 1992).
91. Darrell Woelk, Philip Cannata, Michael Huhns, Nigel Jacobs, Tomasz Ksiezyk, Greg Lavender, Greg Meredith, Kayliang Ong, Wei-Min Shen, Munindar Singh, and Christine Tomlinson. Carnot prototype. In *[8]*, chapter 18, pages 621–648. 1996.

92. Michael Wooldridge and Nick Jennings. Formalizing the cooperative problem solving process. In *[45]*, pages 430–440. 1997. (*Reprinted from* Proceedings of the 13th International Workshop on Distributed Artificial Intelligence, 1994).