To Appear in the Journal of
Artificial Intelligence 1993

Agent Oriented Programming

Yoav Shoham
Robotics Laboratory
Computer Science Department
Stanford University

A new computational framework is presented, called *agent oriented programming* (AOP), which can be viewed as an specialization of *object oriented programming.* The state of an agent consists of components such as beliefs, decisions, capabilities, and obligations; for this reason the state of an agent is called its *mental state.* The mental state of agents is described formally in an extension of standard epistemic logics: beside temporalizing the knowledge and belief operators, AOP introduces operators for obligation, decision and capability. Agents are controlled by *agent programs*, which include primitives for communicating with other agents. In the spirit of *speech act theory*, each communication primitives is of a certain type: informing, requesting, offering, and so on. This article presents the concept of AOP, discusses the concept of mental state and its formal underpinning, defines a class of agent interpreters, and then describes in detail a specific interpreter that has been implemented.

1 Introduction

This manuscript proposes a new programming paradigm. The paradigm promotes a societal view of computation, in which multiple 'agents' interact with one another, although in this

This work was supported in part by a grant from AFOSR.

document we will concentrate on the design of the individual agent. Many of the ideas here intersect and interact with the ideas of others. For the sake of continuity, however, I will not place this work in the context of other work until the end.

1.1 What is an agent?

The term 'agent' is used frequently these days. This is true in AI, but also outside it, for example in connection with data bases and manufacturing automation. Although increasingly popular, the term has been used in such diverse ways that it has become meaningless without reference to a particular notion of agenthood. Some notions are primarily intuitive, others quite formal. Some are very austere, defining an agent in automata-theoretic terms, and others use a more lavish vocabulary. The original sense of the word, of someone acting on behalf of someone else, has been all but lost in AI (an exception that comes to mind is the use of the word in the intelligent-interfaces community, where there is talk of 'software agents' carrying out the user's wishes; this is also the sense of *agency theory* in economics [63]). Most often, when people in AI use the term 'agent', they refer to an entity that functions continuously and autonomously in an environment in which other processes take place and other agents exist. This is perhaps the only property that is assumed uniformly by those in AI who use the term. The sense of 'autonomy' is not precise, but the term is taken to mean that the agents' activities do not require constant human guidance or intervention. Often certain further assumptions are made about the environment, for example that it is physical and partially unpredictable. In fact, agents are sometimes taken to be robotic agents, in which case other issues such as sensory input, motor control and time pressure are mentioned. Finally, agents are often taken to be 'high-level.' Although this sense is quite vague, many take some version of it to distinguish agents from other software or hardware components. The 'high level' is manifested in symbolic representation and/or some cognitive-like function: agents may be 'informable' [25], may contain symbolic plans in addition to stimulus-reponse rules [14, 30, 19, 53], and may even possess natural-language capabilities. This sense is *not* assumed uniformly in AI, and in fact a certain counter-ideology deliberately denies the centrality or even existence of high-level representation in agents [8, 2].

Clearly, the notion of agenthood in AI is anything but crisp. I should therefore make it clear what *I* mean by the term 'agent', which is precisely this: An agent is an entity whose

state is viewed as consisting of of mental components such as beliefs, capabilities, choices, and commitments. These components are defined in a precise fashion, and stand in rough correspondence to their commonsense counterparts. In this view, therefore, agenthood is in the mind of the programmer: What makes any hardware or software component an agent is precisely the fact that one has chosen to analyze and control it in these mental terms.

The question of what an agent is is now replaced by the question of what entities can be viewed as having mental state. The answer is that *anything* can be so described, although it is not always advantageous to do so. This view is not original to me. For example, in [16] and other publications, D. Dennett proposes the "intentional stance," from which systems are ascribed mental qualities such as intentions and free will. The issue, according to Dennett, is not whether a system really is intentional, but whether we can coherently view it as such. Similar sentiments are expressed by J. McCarthy in [48], who also distinguishes between the 'legitimacy' of ascribing mental qualities to machines and its 'usefulness':

> To ascribe certain *beliefs, free will, intentions, consciousness, abilities* or *wants* to a machine or computer program is legitimate when such an ascription expresses the same information about the machine that it expresses about a person. It is useful when the ascription helps us understand the structure of the machine, its past or future behavior, or how to repair or improve it. It is perhaps never logically required even for humans, but expressing reasonably briefly what is actually known about the state of the machine in a particular situation may require mental qualities or qualities isomorphic to them. Theories of belief, knowledge and wanting can be constructed for machines in a simpler setting than for humans, and later applied to humans. Ascription of mental qualities is most straightforward for machines of known structure such as thermostats and computer operating systems, but is most useful when applied to entities whose structure is very incompletely known.

In [66] I illustrate the point through the light-switch example. It is perfectly coherent to treat a light switch as a (very cooperative) agent with the capability of transmitting current at will, who invariably transmits current when it believes that we want it transmitted and not otherwise; flicking the switch is simply our way of communicating our desires. However,

while this is a coherent view, it does not buy us anything, since we essentially understand the mechanism sufficiently to have a simpler, mechanistic description of its behavior. In contrast, we do not have equally good knowledge of the operation of complex systems such robots, people, and, arguably, operating systems. In these cases it is often most convenient to employ mental terminology; the application of the concept of 'knowledge' to distributed computation, discussed below, is an example of this convenience.[1]

1.2 On the responsible use of pseudo-mental terminology

When I (like Dennett and McCarthy before me) state that in principle anything can be ascribed intensionality, I do not mean that there are no rules for this ascription. We do not arbitrarily label one component of the machine a 'belief' and another a 'commitment'; that would be gratuitous fancy naming, against which McDermott has already warned [51]. Indeed, if the labelling were arbitrary, there would be no reason not to exchange the labels 'belief' and 'commitment,' or for that matter not call the first component 'F112' and the second 'F358'.

When then are we justified in ascribing a particular mental quality such as belief to a particular component of a machine? It seems to me reasonable to require the following elements as justification:

- A precise theory regarding the particular mental category; the theory must have clear semantics (or, to quote McDermott in [51], "No Notation without Denotation"), and should correspond to the commonsense use of the term.
- A demonstration that the component of the machine obeys the theory.
- A demonstration that the formal theory plays a nontrivial role in analyzing or designing the machine (or, to coin a new phrase, "No Notation without Exploitation").

[1] In [66] I discuss how the gradual elimination of animistic explanations with the increase in knowledge is correlated very nicely with both developmental and evolutionary phenomena. In the evolution of science, theological notions were replaced over the centuries with mathematical ones. Similarly, in Piaget's stages of child development, there is a clear transition from animistic stages around the ages of 4-6 (when, for example, children claim that clouds move because they follow us around), and the more mature later stages.

As an example, it is instructive to consider the use of logics of knowledge and belief in distributed computation (cf. [28, 29]). The initial motivation behind that line of research was to prove properties of distributed protocols. Researchers noted that intuitive reasoning about protocols included statements such as: "Processor A does not yet know that the network is back up, but since processor B knows that processor A doesn't know it, it (processor B) will not send the next message." Wishing to formalize this sort of reasoning, researchers adopted a logic of knowledge which had been introduced in analytic philosophy by Hintikka [33] and imported to AI by Moore [54] (subsequent work on knowledge and belief within AI includes [38, 42, 55, 67]).

The logic widely (though not universally) adopted, the S5 modal system, is at best an idealization of the commonsense notion of knowledge. Some of its more counter-intuitive properties include tautological closure (you know all that follows from your knowledge), positive introspection (if you know something then you know that you know it) and negative introspection (if you do not know something then you know that you do not). The important point is that, for the applications considered in distributed computation, the deviation from commonsense was harmless. Intuitively, in those applications the knowledge possessed by agents was so simple that complexity of internal reasoning could be neglected.

The first lesson, then, is that while the formal theory of the mental category should correpsond to commonsense, this correspondence will not be exact. It is up to the consumer of the theory to consider the application at hand, and judge whether the correspondence between theory and commonsense is sufficiently close that s/he will not be misled by allowing commonsense intuition to guide reasoning with the formal construct. For example, if in the application all the beliefs are propositional Horn clauses, and if in the application all linear-time computation is negligible, then the axiom of tautological closure is quite reasonable, since propositional Horn theories admit a linear-time decision procedure. On the other hand, this same axiom renders the logic useless for reasoning about number-theoretic cryptographic protocols.[2]

The distributed computation application serves also to illustrate the second element required to justify the use of a mental term, a demonstration that the machine obeys the

[2]Indeed, there have since been a number of proposals to escape the property of tautological, by modifying the logic.

properties of the formal construct. The semantics adopted for the knowledge operator were standard possible-worlds semantics [39], but here possible worlds were given a very concrete interpretation: A possible world was a possible global state of the system (that is, a possible combination of local states of the various processors) given a fixed protocol, and the worlds accessible to each agent from a given world consisted of all global states in which its local state was the same as in the given world. It was easy to show that this concrete definition obeyed the S5 properties.

Finally, using this definition of knowledge, it became possible to prove certain properties of distributed protocols. The logic of knowledge was in principle dispensible – one could theoretically replace the knowledge operator by the corrsponding statement about the states of the various processors – but the statements would get complex, and intuition would be lost. In practice, therefore, logics of knowledge proved invaluable, satisfying the third requirement of ascribing a mental attitude to a machine.

In this article we will consider mental constructs that are somewhat more involved than knowledge. We will consider belief, obligation, and capability, and add a temporal component to each of those. As in in the case of knowledge, however, we will not reach exact match between the formal properties of these formal constructs and commonsense, but rather will aim to strike a balance between computational utility and commonsense.

1.3 AOP versus OOP

It was mentioned in the previous section that the ascription of mental constructs must be coherent and useful. The application of the logic of knowledge in distributed computation, given there as an example, used the mental construct 'knowledge' in a particular way: It mapped it onto an existing computational framework (a distributed network of processors), and used it to reason about the system. The use we will make of mental constructs is different: Rather than use them for mere analysis, we will employ them to *design* the computational system. The various mental cateogries will appear in the programming language itself, and the semantics of the programming language will be related to the semantics of the mental constructs. This is similar in spirit to a development within the distributed computation community, where a proposal has been made to include tests for epistemic properties in the protocols themselves [74]; at this time the proposal has not yet been followed up on.

Specifically, I will propose a computational framework called *agent oriented programming* (AOP). The name is not accidental, since from the engineering point of view AOP can be viewed as an specialization of the *object oriented programming* (OOP) paradigm. I mean the latter in the spirit of Hewitt's original Actors formalism [32], rather than in the more specific sense in which it used today. Intuitively, whereas OOP proposes viewing a computational system as made up of modules that are able to communicate with one another and that have individual ways of handling in-coming messages, AOP specializes the framework by fixing the state (now called *mental state*) of the modules (now called *agents*) to consist of components such as beliefs (including beliefs about the world, about themselves, and about one another), capabilities, and decisions, each of which enjoys a precisely defined syntax. Various constraints are placed on the mental state of an agent, which roughly correspond to constraints on their commonsense counterparts. A computation consists of these agents informing, requesting, offering, accepting, rejecting, competing, and assisting one another. This idea is borrowed directly from the *speech act* literature [5, 65, 27]. Speech act theory categorizes speech, distinguishing between informing, requesting, offering and so on; each such type of communicative act involves different presuppositions and has different effects. Speech act theory has been applied in AI, in natural language research as well as in plan recognition [59, 37]. To my knowledge, AOP and McCarthy's Elephant2000 language (see Section 8) are the first attempts to base a programming language in part on speech acts. Figure 1 summarizes the relation between AOP and OOP.[3]

1.4 Organization of the document

The rest of the document is organized as follows. In Section 2 I provide further motivation for the AOP paradigm by looking at two scenarios in which AOP is expected to prove useful. In Section 3 I outline the main ingredients of the AOP framework. The bulk of the paper then describes progress made to date towards realizing the concept. In Section 4 I discuss the mental categories that are essential to AOP, and as many of the details involved in formalizing those that are needed for the remainder. In Section 5 I discuss a general family

[3]There is one more dimension to the comparison, which I omitted from the table, and it regards inheritance. Although absent from Hewitt's proposal, inheritance among objects is today one of the main features of OOP, constituting an attractive abstraction mechanism. I have not discussed it since it is not essential to the idea of OOP, and even less so to the idea of AOP. Nevertheless a parallel can be drawn here too, and I discuss it briefly in the final section.

Framework:	OOP	AOP
Basic unit:	object	agent
Parameters defining state of basic unit:	unconstrained	beliefs, commitments, capabilities, choices, ...
Process of computation:	message passing and response methods	message passing and response methods
Types of message:	unconstrained	inform, request, offer, promise, decline, ...
Constraints on methods:	none	honesty, consistency, ...

Figure 1: OOP versus AOP

of agent interpreters. In Section 6 I define a simple programming language called AGENT-0, and its specific interpreter. In Section 7 I briefly discuss the process of 'agentification,' or transforming an arbitrary device into a programmable agent. In Section 8 I discuss related work by others in AI. Finally, in the last section I discuss some of the directions in which the work described here can be extended.

2 Two scenarios

Below are two scenarios. The first is fairly complex, and serves illustrates the type of future applications envisioned. The second is a toy example, and serves three purposes: It illustrates a number of AOP ideas more crisply, it is implementable in the simple language defined later in this article, and it illustrates the fact that agents need not be robotic agents.

2.1 Manufacturing automation

Alfred and Brenda work at a car-manufacturing plant. Alfred handles regular-order cars, and Brenda handles special-order ones. The plant has a welding robot, Calvin. The plant is controlled by a coordinating program, Dashiel. The following scenario develops, involving communication between Alfred, Brenda, Calvin and Dashiel. It contains communication acts such as informing, requesting, committing, permitting and commanding, and requires agents to reason about the beliefs, capabilities and commitments of other agents.

8:00: Alfred requests that Calvin promise to weld ten bodies for him that day; Calvin agrees to do so.

8:30: Alfred requests that Calvin accept the first body, Calvin agrees, and the first body arrives. Calvin starts welding it and promises Alfred to notify him when it is ready for the next body.

8:45: Brenda requests that Calvin work on a special-order car which is needed urgently. Calvin reponds that it cannot right then, but that it will when it finishes the current job, at approximately 9:00.

9:05: Calvin completes welding the Alfred's first car, ships it out, and offers to weld Brenda's car. Brenda ships it the car, and Calvin starts welding.

9:15: Alfred enquires why Calvin is not yet ready for his (Alfred's) next car. Calvin explains why, and also that it (Calvin) expects to be ready by about 10:00.

9:55: Calvin completes welding Brenda's car, and ships it out. Brenda requests that it reaccept it and do some painting, but Calvin refuses, explaining that it does not know how to paint. Calvin then offers to weld another car for Alfred, and proceeds to weld Alfred's cars for a while.

12:15: Brenda requests that Calvin commit to welding four more special-order cars that day. Calvin replies that it cannot, since that conflicts with its commitment to Alfred, who still has six unwelded cars. Brenda requests Alfred to release Calvin from its commitment to Alfred. Alfred refuses. Brenda requests that Dashiel (remember Dashiel?) order Calvin to accept her important request and revoke its commitment to Alfred. Dashiel orders Calvin to weld two of Brenda's cars, and then as many of Alfred's as time allows.

3:30: Calvin senses it is low on lubricant, and, without interrupting its work, asks Alfred for when the next lubrication is scheduled. Alfred reponds that he doesn't know, but that Dashiel should know. Calvin asks Dashiel, who informs it that the next lubrication is scheduled for 4:00. Since that is sufficiently soon, Calvin continues to weld.

2.2 Airline reservation

This example is a minor modification of one due to John McCarthy [49], who uses it to illustrate his Elephant2000 programming language (see Section 8). The example involves three agents: a passenger P, an airline clerk C, and C's supervisor S. P and S are human, and C is a program (or is it the other way around? Or perhaps all three are programs?). Note that by confirming a reservation the airline enters into a commitment to issue a boarding pass to the passenger at the appropriate time. Again, the following exchange involves requesting (a query is a request to inform), informing, permitting and denying, as well as well as mental components such as knowledge and commitment.

March

P to C: Please inform me what flights you have from San Francisco to New York on April 18.

C to P: Flight #354 departs at 08:30, flight #293 departs at 10:00, flight #441 departs at noon, ...

P to C: Please book me on #354.

C to P: That is sold out.

P to C: Please book me on #293.

C to P: That is confirmed; your reservation number is 112358.

P to C: Please book me also on #441.

C to P: That conflicts with #293; I am not allowed to double book a passenger.

P to C: Please get permission to do so.

C to S: I request permission for the following double booking: ...

S to C: Permission denied.

C to P: Sorry, I cannot get approval.

April 18, at the airport

P to C: My name is P; I have a reservation for flight #293.

C to P: Here is your boarding pass.

3 Overview of the AOP framework

A complete AOP system will include three primary components:

- A restricted formal language with clear syntax and semantics for describing mental state. The mental state will be defined uniquely by several modalities, such as belief and commitment.
- An interpreted programming language in which to define and program agents, with primitive commands such as `REQUEST` and `INFORM`. The semantics of the programming language will be required to be faithful to the semantics of mental state.
- An 'agentifier,' converting neutral devices into programmable agents.

In this article I will fall short of supplying a complete system. My main purpose in this article is to discuss the second component. However, since it relies in part on the first one, I will start with a short discussion of mental state. I will limit discussion of logic to the minimum needed for the discussion of the interpreter. There is, however, substantial literature on the logic of various mental categories, and I will provide references to it.

In contrast to the first two components, of which I have relatively good understanding, the third component is still somewhat mysterious to me, and I will discuss it only briefly at the end.

4 Mental categories and their properties

The first step in the enterprise is to define the various components of mental state and their properties. There is not a unique 'correct' selection of mental categories, nor a correct

theory regarding them, as different applications can be expected to call for specific mental properties.[4] In this section I will discuss what could be viewed as a bare-bones theory of mental state, a kernel that will in the future be modified and augmented (for in-depth treatment within our research group of various logical aspects of mental state, see [71, 44, 67, 15]).

4.1 Components of mental state

In related past research by others in AI (see Section 8), three modalities were explored: belief, desire and intention (giving rise to the pun on BDI agent architectures). Other similar notions, such as goals and plans, were also pressed into service. These are clearly important notions; however, I propose starting with a slightly different set of modalities, which are more modest and, I find, more basic.

By way of motivation, here is an informal view of the world which underlies the selection. At any point in time, the future is determined by two factors: The past history, and the current actions of agents. For example, past history alone does not (in this view) determine whether I raise my arm; that is determined by whether in fact I take the appropriate action. The actions of an agent are determined by its *decisions*, or *choices*[5]. In other words, some facts are true for natural reasons, and other facts are true because agents decided to make them so. Decisions are logically constrained, though not determined, by the agent's *beliefs*; these beliefs refer to the state of the world (in the past, present or future), to the mental state of other agents, and to the *capabilities* of this and other agents. For example, given that the robot believes that it is incapable of passing through the narrow doorway, it will not decide to go through it. Decisions are also constrained by prior decisions; the robot cannot decide to be in Room 5 in five minutes if it has already decided to be in Room 3 at that time.

This perspective motivates the introduction of two basic mental categories, *belief* and *decision* (or *choice*), and a third cateogry which is not a mental construct *per se*, *capability*. These are precisely the categories I will adopt, with one modification: rather than take choice

[4] In this respect our motivation here deviates from that of philosophers. However, I believe there exist sufficient similarities to make the connection between AI and philosophy mutually beneficial.

[5] The term *choice* is somewhat ambiguous; I discuss various senses of choice later.

as basic, I will start with the notion of *obligation*, or *commitment*, and will treat decision simply as obligation to oneself.

By restricting the components of mental state to these modalities I have in some informal sense excluded representation of motivation. Indeed, I will not assume that agents are 'rational' beyond assuming that their beliefs, obligations and capabilities are internally and mutually consistent. This stands in contrast to the other work mentioned above, which makes further assumptions about agents acting in their own best interests, and so on. Such stronger notions of rationality are obviously important, and I am convinced that in the future we will wish to consider them. However, neither the concept of agenthood nor the utility of agent oriented programming depend on them.

In the remainder of this section I will describe the various modalities in more detail. My goal here is not to provide a comprehensive analysis of them; that is is the subject of the papers mentioned earlier. Here I will discuss them only to the extent that they bear on the development of the interpreter defined later.

4.2 A language for belief, obligation and capability

Time. Time is basic to the mental categories; we believe things both *about* different times and *at* different times, and the same is true of other modelities. We will adopt a simple point-based temporal language to talk about time; a typical sentence will be

$$\texttt{holding(robot,cup)}^{\texttt{t}}$$

meaning that the robot is holding the cup at time `t`.

Action. Actions take place at different points in time, and, depending on the circumstances at the time they are taken, have certain effects. However, for the purposes of this article, we will not distinguish between actions and facts, and the occurrence of an action will be represented by the a corresponding fact holding. For example, strictly speaking, rather than say that the robot took the action `raise-arm` at time `t`, we will say that the sentence $\texttt{raise-arm(robot)}^{\texttt{t}}$ is true. (However, to retain the agency behind the action, we will

introduce the notion of decision; see below.) Given that actions are facts, they too are instantaneous. This too is a limitation in the current language.

There is substantial literature on representing action; the best known representative in AI is the *situation calculus* [50]. There are a number of proposals to allow time, durational facts, and durational actions all in the same language. In fact, in [43] we represent time and parallel action in the situation calculus itself. However, the details are somewhat involved, and these extensions are not essential to the current discussion. Nonetheless, since actions are such a natural concept, in the programming language discussed later we will introduce them as syntactic sugar.

Belief. We now augment the language with a modal operator B, denoting belief. As mentioned above, an agent believes things both at certain times and about certain times. The general form of belief statement is

$$\mathtt{B}^{\mathtt{t}}_{\mathtt{a}}\varphi$$

where a is an agent, t a time term, and φ a (recursively defined) sentence. For example, $\mathtt{B}^{3}_{\mathtt{a}}\mathtt{B}^{10}_{\mathtt{b}}\mathtt{like(a,b)}^{7}$ will mean that at time 3 agent a believes that at time 10 agent b will believe that at time 7 a liked b.

Obligation. So far I have used largely well-known notions: Temporal languages are standard fare, operators denoting belief common, and their combination, although somewhat novel, is straightforward. We now depart more radically from past constructions and introduce new modal operator, OBL. OBL has one more argument than B:

$$\mathtt{OBL}^{\mathtt{t}}_{\mathtt{a,b}}\varphi$$

will mean that at time t agent a is obligated, or committed, to agent b about φ. Notice that, since actions are represented simply as facts, the agent is obligated to a fact holding rather than to taking action.

Decision (choice). The freedom to choose among several possible actions is central to the notion of agenthood,[6] and earlier on in the research we indeed took decision, or choice, to be a primitive notion. The current definition of obligation provides an alternative, however: decision is defined to be simply obligation, or commitment, to oneself:

$$\mathtt{DEC}^{\mathtt{t}}_{\mathtt{a}}\varphi =_{def} \mathtt{OBL}^{\mathtt{t}}_{\mathtt{a},\mathtt{a}}\varphi$$

Again, the term 'choice' carries many connotations, and I emphasize that I mean it in the sense of 'decision'; an agent has chosen something if it has decided that that something be true. Thus, most of us can decide to own a new pair of shoes, but few of us can decide to own yacht.

Capability. Intimately bound to the notion of agenthood is also that of capability. I may decide to move my arm, but if I am not capable of it then it will not move. I will not decide to do anything I believe myself incapable of. Similarly, I will not request a two-year-old, nor a mobile robot, to climb a ladder, since I do not believe they are capable of it.

It is debatable whether capability is best defined in mental terms or not. For example, one definition would say that agent a is capable of φ just in case the following is true: If a were to decide φ, then φ would be true. There are also reasonable philosophical arguments against this definition (cf. [20, 6, 9]). Here I will simply introduce the notation

$$\mathtt{CAN}^{\mathtt{t}}_{\mathtt{a}}\varphi$$

to represent the fact that at time t a is capable of φ. We will place certain constraints on such sentences, but not take a stance on whether CAN is reducible to mental notions or not.[7] Note that, like the other modalities, capabilities refer to specific times; a typical sentence is

$$\mathtt{CAN}^{5}_{\mathtt{robot}}\mathtt{open(door)}^{8}$$

[6] To quote Isaac Bashevis-Singer, "We *must* believe in free will; you see, we have no choice."

[7] However, [71] does take a stance; capability is taken to be a primitive notion defined in terms of future-branching structures, not reducible to other notions. This is in the spirit of recent philosophical treatments of agency [20, 6, 9] mentioned earlier.

Thus at time 5 the robot might be able to ensure that the door is open at time 8, but at time 6 it might no longer have that capability. We may define ABLE to be the 'immediate' version of CAN. First, for any sentences φ, we define $time(\varphi)$ to be the outermost time occurring in it; for example, $time(\texttt{open(door)}^{\texttt{t}}) = time(\texttt{B}^{\texttt{t}}_{\texttt{a}}\varphi) = \texttt{t}$. We now define ABLE as follows:

$$\texttt{ABLE}_{\texttt{a}}\varphi =_{def} \texttt{CAN}^{time(\varphi)}_{\texttt{a}}\varphi$$

and thus

$$\texttt{ABLE}_{\texttt{robot}}\texttt{open(door)}^{5} \equiv \texttt{CAN}^{5}_{\texttt{robot}}\texttt{open(door)}^{5}$$

4.3 Properties of the various components

I have so far not placed any constraints on the various modalities defined, and therefore have not guaranteed that they in any way resemble their commonsense counterparts. We will now place such constraints. Just as there is no objectively 'right' collection of mental categories, there is no 'right' list of properties for any particular mental category. It was already stated in the introduction that the correspondence between the formal definition and commonsense will always be only approximate, and that we would like to strike a balance between commonsense and utility. Indeed, I expect different applications of AOP to call for different properties of belief, commitment and capability. In this section I will define a number of properties I assume about the modalities. These properties are quite weak, but they are sufficient to justify the terminology, and necessary for the design of the interpreter. The weakness of the assumptions ensures that the interpreters apply to a wide variety of applications. Still, even these assumptions will be inappropriate for some purposes, in which case a new type of interpreter will be required.

Internal consistency. We assume that both the beliefs and the obligations are internally consistent. Specifically, we assume

for any t,a: $\{\varphi\text{: } \texttt{B}^{\texttt{t}}_{\texttt{a}}\varphi\}$ is consistent.

for any t,a: $\{\varphi\text{: } \texttt{OBL}^{\texttt{t}}_{\texttt{a,b}}\varphi$ for some b$\}$ is consistent.

Good faith. We further assume that agents commit only to what they believe themselves capable of, and only if they really mean it:

for any t,a,b,φ: $\mathtt{OBL}^t_{a,b}\varphi \supset \mathtt{B}^t_a((\mathtt{ABLE}_a\varphi)\wedge\varphi)$

Introspection. Although in general we do not assume that agents have total introspective capabilities, we do assume that they are aware of their obligations:

for any t,a,b,φ: $\mathtt{OBL}^t_{a,b}\varphi \equiv \mathtt{B}^t_a\mathtt{OBL}^t_{a,b}\varphi$

for any t,a,b,φ: $\neg\mathtt{OBL}^t_{a,b}\varphi \equiv \mathtt{B}^t_a\neg\mathtt{OBL}^t_{a,b}\varphi$

On the other hand, we do not assume that agents are necessarily aware of commitments made *to* them.

Persistence of mental state We have only placed restrictions on mental attitudes at a single instant of time. We conclude this section by discussing how mental states change or persist over time. However, unlike in the previous discussion, we will not place precise constraints, but rather informal guidelines.

Consider, for example, belief. Our restrictions so far allow agents which at one time believe nothing at all, shortly afterwards have a belief about *every* sentence, and then again become quite agnostic. Commonsense suggests that beliefs tend to be more stable than that, and it would indeed be difficult to rely on the behavior of agents with such volatile beliefs. We will now place a strong condition on belief; we will assume that agents have perfect memory of, and faith in, their beliefs, and only let go of a belief if they learn a contradictory fact. Beliefs therefore persist *by default.* Furthermore, we will assume that the *absence* of belief also persists by default, although in a slightly different sense: if an agent does not believe a fact at a certain time (as opposed to believing the negation of the fact), then the only reason he will come to believe it is if he learns it.

How to formally capture these two kinds of default persistence is another story, and touches on issues that are painfully familiar to researchers in nonmonotonic temporal reasoning and belief revision. In fact, a close look at the logical details of belief (or knowledge)

persistence reveals several very subtle phenomena, which have so far not been addressed in the literature [44].

Obligations too should persist; they wouldn't be obligations otherwise. As in the case of belief, however, the persistence is not absolute. Although by default obligations persist, there are conditions under which obligations are revoked. These conditions presumably include explicit release of the agent by the party to which it is obligated, or alternatively a realization on the part of the agent that it is no longer able to fulfill the obligation. (In their discussion of the persistence of commitment, Cohen and Levesque [10] actually propose a more elaborate second condition, one that requires common knowledge by the committer and committee of the impossibility; however further discussion of their position and arguments against it would be too long a detour; see a brief discussion of their work in subsection 4.4.)

Since decision is defined in terms of obligation, it inherits the default persistence. Notice, however, an interesting point about the persistence of decision: While an agent cannot unilaterally revoke obligations it has towards others, it *can* cancel obligations held towards it – including obligations it holds towards itself, namely decisions. An agent is therefore free to modify an existing decision, but unless he explicitly does so the decision will stand.

Finally, capabilities too tend to not fluctuate wildly. In fact, in this document I assume that capabilities are fixed: What an agent can do at one time it can do at any other time. However, I will allow to condition a capability of an action on certain conditions that hold at the time of action.

4.4 A short detour: Comparison with Cohen and Levesque

In several publications (*e.g.*, [11, 10]) Cohen, Levesque and several associates have investigated the logical relationships between several modalities such as the above-mentioned ones. As mentioned earlier, I have deferred discussion of related work to a later section. However, since there is room for confusion between Cohen and Levesque's definitions of mental categories and our own, I will make an exception in this case. The reader may skip this subsection, although I believe the discussion may provide further intuition about our definition as well as Cohen and Levesque's.

Cohen and Levesque employ two basic modalities, `BEL` (belief) and `G` (goal, or choice).

Although these bear a resemblance to our belief and choice modalities, important differences exist. Their belief modality is really the same as ours, except that they are not as explicit about its temporal aspect. In their language, one may use the $\Diamond$ tense operator to specify that "Sometime in the future an agent will believe a fact" or "Sometime in the future the agent will believe a fact either about that time or about a time yet further into the future," but one is not able to talk about (e.g.) the agent believing in the future something about the past. This capability could be achieved by adding other tense operators, if the authors insisted on a tense logic.[8]

The primary intuition offered about the G modality is that it denotes choice ("Consider the desire that the agent has chosen to pursue as put into a new category. Call this chosen desire, loosely, a goal"). However, I have already noted that term 'choice' is multi-facetted, and indeed G is quite different from our DEC. For Cohen and Levesque, the choices of an agent constitute a consistent subset of the agent's desires, those that the agent has adopted as goals. In contrast, our OBL modality (and hence the derived DEC modality) reflects absolutely no motivation of the agent, and merely describes the actions to which the agent is obligated.

The difference in senses is the difference between a decision to act and a decision to pursue a goal. This difference is reflected in the different interactions between choice (or decision) and belief. In our construction, obligation (and therefore also decision) implies belief: $\mathtt{DEC}^t_{aP} \supset \mathtt{B}^t_{aP}$ (if an agent decides on an action he believes it will take place). The converse implication does not hold in our construction (the agent may believe that the sun will rise tomorrow without making a choice in this regard). For Cohen and Levesque, on the other hand, belief does imply choice: (BEL a p) $\supset$ (G a p) (see, e.g., their Proposition 17). The intuition there is that the G modality specifies possible worlds chosen by the agent, and these worlds are selected among the ones the agent believes are possible; therefore if a fact is true in all worlds believed possible by the agent, it must be true in the subset he selects. Note that both senses of choice guarantee that an agent does not choose something he believes impossible: both $\mathtt{DEC}^t_{aP} \supset \neg \mathtt{B}^t_a \neg p$ and (G a p) $\supset \neg$(BEL a $\neg$p) hold in the respective systems.

Both senses of choice are worthwhile, although one can imagine other ways of capturing

[8]However, the authors do find it necessary to mention explicit dates, which they represent by propositions such as '1/1/90/12:00'. That being the case, I do not see the the utility of retaining the tense operators.

a decision to pursue a goal. In particular, I am not sure that it is best to use a single operator to capture both 'having a goal' and 'deciding to adopt a goal.' For example, it may prove advantageous to start with a G modality denoting 'having a goal,' and define 'goal adoption' by $\texttt{DEC}_\texttt{a}\texttt{G(a,p)}$ (with the appropriate temporal arguments added). However, I have deliberately avoided such more complex notions in this document as they are not needed for the fundamentals of AOP, and will not pursue the issue further.

In summary, the pioneering work of Cohen and Levesque introduces mental categories that are different from ours. The two frameworks share the essential view of belief and time. They each introduce modalities absent from the other: obligation and capability in our framework, goals and intentions in theirs. Even two notions that at first appear to be similar – our decision and their choice – turn out to be quite different.

5 A generic agent interpreter

In the previous section I discussed the first component of the AOP framework, namely the definition of agents. I now turn to the central topic of this paper, the programming of agents. In this section I will outline a generic agent interpreter. In the next section I describe a specific programming language and its implemented interpreter.

The role of agent programs is to control the evolution of an agent's mental state; actions occur as a side effect of the agent's being committed (that is, obligated) to an action whose time has come. Since the mental state of agents is captured in a formal language, it might be tempting to view AOP as a form of logic programming. In this view, the program would consist of logical statements about the mental state of the agent, and through a process of theorem proving the mental state at each point in time will be determined. However, this is not what I intend; although it might be possible in principle to do develop such a programming language[9], multi-modal temporal theorem proving is sufficiently daunting to discourage me from attempting it at this point. Instead, I will allow standard operationally-defined languages; indeed, I will define such a language myself in the next section. These languages, however, will include data structures which represent various logical sentences (for example, those denoting beliefs and commitments), and the (extra-logical) operations on

[9]Indeed, an early design of agent programs by J. Akahani was entirely in the style of logic programming.

these data structures will be required to obey the properties of the various logical operators. For example, that no two data structures denoting contradictory beliefs may be instantiated by the interpreter at the same time.

The basic loop

The behavior of agents is, in principle, quite simple. Each agent iterates the following two steps at regular intervals:

1. Read the current messages, and update your mental state, including your beliefs and commitments (the agent program is crucial for this update);
2. Execute the commitments for the current time, possibly resulting in further belief change (this phase is independent of the agent's program).

Actions to which agents can be committed include communicative ones such as informing and requesting, as well as arbitrary 'private' actions. The process is illustrated in Figure 2; dashed arrows represent flow of data, solid arrows temporal sequencing.

Assumption about message passing

Agent programs will include, among other things, communication commands. In order that those be executable I will assume that the platform is capable of passing messages to other agents addressable by name, whether those reside in the same machine or in others. The programming language will define the form of these messages, and the interpreter will determine when messages are sent.

Assumption about the clock

Central to the operation of the interpreter is the existence of a clock; agents are inherently 'real time' (to use another overloaded term). The main role of the clock is to initiate iterations of the two-step loop at regular intervals (every 10 milliseconds, every hour, *et cetera*); the length of these intervals, called the 'time grain,' is determined by the settable variable `timegrain`. I do not discuss the implementation of such a clock, which will vary among

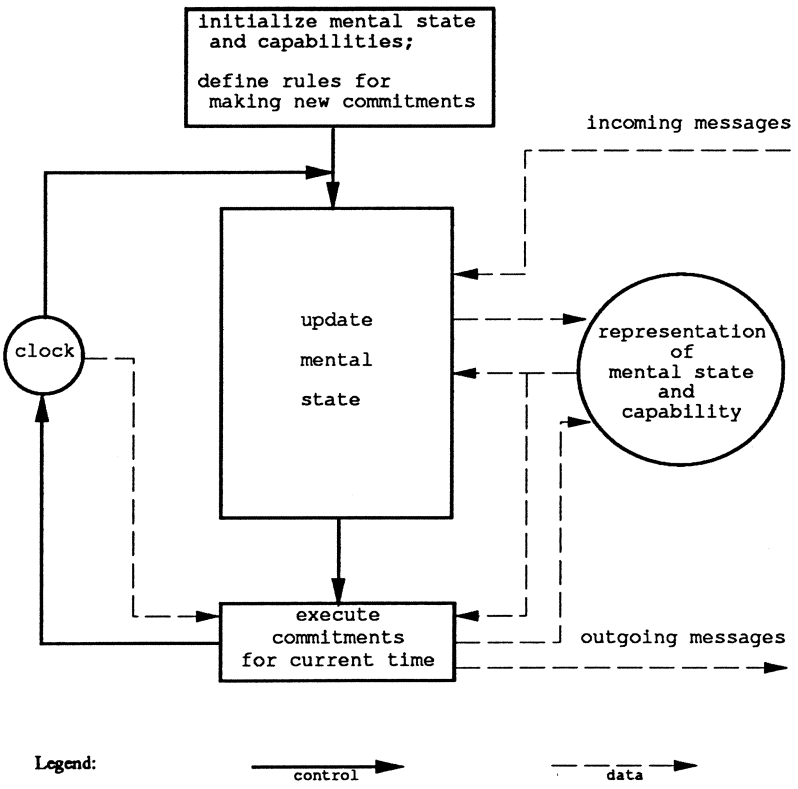

Figure 2: A flow diagram of a generic agent interpreter

platforms, and simply assume that it exists. We also assume a variable **now**, whose value is set by the clock to the current time in the format defined in the programming language (an integer, *date:hour:minute, et cetera*).

In the remainder of the description we make the very strong assumption that a single iteration through the loop lasts less than the time grain; in future versions of the language we will relax this assumption, and correspondingly will complicate the details of the loop itself.

Of course, the fact that agents use the same temporal language does not ensure that their clocks are synchronized. If all are agents are running on the same machine there will be no problem, but otherwise the possibility of clock drift exists. Although synchronization does not impact the design and programming of single agents, it is crucial for ensuring that a society of agents is able to function usefully. Fortunately, there exist synchronization protocols which ensure limited drift among clocks (for an overview, see [64]), and we expect to use these in our applications. However, since the focus in this article is on the design of single agents, I will not discuss this issue further.

6 AGENT-0, a simple language and its interpreter

Agent interpreters may vary along many dimensions, and in general pose many challenging problems. In this section I describe a particular programming language called AGENT-0, whose interpreter is an extremely simple instance of the generic agent interpreter. In fact, the simplifications embodied in AGENT-0 are so extreme that it may be tempting to dismiss it as uninteresting. However, it was recognized early on that one would not gain good insight into the strengths and weaknesses of AOP without writing actual programs. It was decided therefore to implement a simple interpreter first, and design more complex languages and interpreters based on this experience. It turned out the design of AGENT-0 itself posed some challenges, and we have been surprised by the diversity of applications that even this simple language admits. Furthermore, AGENT-0 is designed in a way that suggests obvious extensions; a few are being currently pursued, and will be described in the last section.

6.1 The syntax of AGENT-0

In the programming language itself one specifies only conditions for making commitments; commitments are actually made, and later carried out, automatically at the appropriate times (see discussion of the interpreter in Subsection 6.2 below). Commitments are only to primitive actions, those that agent can directly execute. In other words, in AGENT-0 an agent cannot commit to achieving any condition which requires some sort of planning. Before we define the syntax of commitments we need a few preliminary definitions. I will first develop the syntax of AGENT-0 in a bottom-up fashion, and then summarize it in BNF notation. The reader may wish to refer forward to the formal definition while reading the following description.

Fact statements

Fact statements are fundamental to AGENT-0; they are used to specify the content of actions as well as conditions for their execution. Fact statements constitute a tiny fragment of the temporal language described earlier; they are essentially the atomic objective sentences (that is, no conjunction or disjunction, nor modal operators). Typical fact statements will be `(t (employee smith acme))` and `(NOT (t (employee jones acme)))`.

Private and communicative action statements

Agents commit to action, and so we now specify what actions are. We make two orthogonal distinctions between types of action: Actions may be *private* or *communicative*, and, independently, they may be *conditional* or *unconditional.*

The syntax for private actions is

```
(DO t p-action)
```

where `t` is a time point and `p-action` is a private action name. Private action names are idiosyncratic and unconstrained; a data base agent may have retrieval primitives, a statistical computation agent may run certain mathematical procedures, and a robot may servo itself. The effects of private actions may be invisible to other agents, as in the data base example, but need not be so, as in the robot example.

Private actions may or may not involve IO. Communicative actions, on the other hand, always involve IO. Unlike private actions, communicative actions are uniform, and common to all agents. While in a general AOP system we can expect many types of communicative action, the restricted version AGENT-0 has only three types of communicative action: informing, requesting, and cancelling a request.

The syntax of informing is

```
(INFORM t a fact)
```

where `t` is a time point, `a` is an agent name and `fact` is a fact statement. Note that `t` is the time at which the informing is to take place, and `fact` itself contains other temporal information, as in `(INFORM 5 b (1 (employee smith acme)))`.

The syntax of requesting is

```
(REQUEST t a action)
```

where `t` is a time point, `a` is an agent name and `action` is an action statement, defined recursively. So, for example, `(REQUEST 1 a (DO 10 update-database))` constitutes a legitimate request. Again, one should distinguish between the time at which the requesting is to be done (1, in this example) and the time of the requested action (10, in the example). Requests can be embedded further, as in `(REQUEST 1 a (REQUEST 5 b (INFORM 10 c fact)))`.

The syntax of cancelling a request is:

```
(UNREQUEST t a action)
```

where `t` is a time point, `a` is an agent name and `action` is an action statement.

The last unconditional action in AGENT-0 is really a nonaction. Its syntax is:

```
(REFRAIN action)
```

where `action` is an action statement. The role of refraining will be to prevent commitment to particular actions.

Conditional action statements

In AGENT-0 we distinguish between commitments for conditional actions, which include conditions to be tested right before acting, and conditions for entering into commitments in the first place. I now discuss only conditional actions, and will discuss conditions for entering into commitments later.

Conditional actions rely on one form of condition, called a *mental condition.* Mental conditions refer to the mental state of the agent, and the intuition behind them is that when the time comes to execute the action, the mental state *at that time* will be examined to see whether the mental condition is satisfied. For this reason the agent- and time-components of the mental state are implicit and can be omitted in the specification of mental conditions. A mental condition is thus any combination of modal statements in the temporal-modal language, with the primary 'agent' and 'time' arguments omitted.

Specifically, a mental condition is a logical combination of *mental patterns.* A mental pattern is one of two pairs:

`(B fact)` or `((CMT a) action)`

where `fact` is a fact statement, `a` is an agent name and `action` is an action statement (we use the term `CMT` rather than `OBL` since, for historical reasons, this is the notation used in the actual implementation). An example of a mental pattern is `(B (t (employee smith acme)))`.

Given the syntax of mental conditions, the syntax of a conditional action is

`(IF mntlcond action)`

where `mntlcond` is a mental condition and `action` is an action statement. An example of a conditional action is

```
(IF (B (t' (employee smith acme))) (INFORM t a (t' (employee smith acme))))
```

The intuitive reading of this action is "if at time `t` you believe that at time `t'` `smith` is an employee of `acme`, then at time `t` inform agent `a` of that fact."

As was said, mental conditions may contain logical connectives. These connectives are AND, OR and NOT. The following three actions illustrate the use of NOT; together they constitute a QUERY about whether fact is true (b is the one being queried, a is the one he is asked to inform):

```
(REQUEST t b (IF (B,fact) (INFORM t+1 a fact)))
(REQUEST t b (IF (B (NOT fact)) (INFORM t+1 a (NOT fact))))
(REQUEST t b (IF (NOT (BW fact)) (INFORM t+1 a (NOT (t+1 (BW a fact))))))[10]
```

Variables

In the style of logic programming and production systems, in AGENT-0 procedures are invoked in a pattern-directed fashion. Specifically, we will see that commitment rules are 'activated' based on certain patterns in the incoming messages and current mental state. Variables play a crucial role in these patterns.

A variable is denoted by the prefix '?'. Variables may substitute agent names, fact statements or action statements. Thus the following is a legitimate conditional action:

```
(IF (NOT ((CMT ?x) (REFRAIN sing))) sing)
```

In the tradition of logic programming, variables in action statements (including the mental condition part) are interpreted as existentially quantified. The scope of the quantifier is upwards until the scope of the first NOT, or it is the entire statement, if the variable does not lie in the scope of a NOT. Thus the last statement reads informally as "if you are not currently committed to anyone to refrain from singing, then sing."

It is advantageous to allow other quantifiers as well. The one quantifier included in AGENT-0 is a limited form of the universal quantifier, but in the future others, such as the "latest (earliest) time point such that" quantifier, may be introduced. The universally-quantified variables will be denoted by the prefix '?!'. The scope of these variables is always the entire formula, and thus the conditional action

```
(IF (B (t (emp ?!x acme))) (INFORM a (t (emp ?!x acme))))
```

[10] BW is the "believe whether" operator, defined by (t (BW a p))≡(t (B a p))∨(t (B a (NOT p))).

results in informing a of all the individuals who the agent believes to be acme employees.[11]

Having discussed action statements, we can now finally discuss the type of statements that actually appear in the program, namely commitment rules.

Commitment rules

Since action statements contain information about what needs to be done, about when it needs to be done, and even the preconditions for doing it, one might have expected a collection of action statements to constitute a program. However, there is another crucial layer of abstraction in AGENT-0. Most of the action statements are unknown at programming time; they are later communicated by other agents (one of which may be the "user," in situations where that concept is applicable). The program itself merely contains conditions under which the agent will enter into new commitments. Some of these conditions may be trivial, resulting in a priori commitments, but most commitments will be in response to messages.

The conditions under which a commitment is made include both mental conditions, discussed above, and message conditions, which refer to the current incoming messages. A message condition is a logical combination of *message patterns.* A message pattern is a triple

```
(From Type Content)
```

where `From` is the sender's name, `Type` is `INFORM`, `REQUEST` or `UNREQUEST`, and `Content` is a fact statement or an action statement, depending on the type. The other information associated with each incoming message, its destination and arrival time, are implicit in this context and are thus omitted from the message pattern (of course, the `content` will include reference to time, but that is the time of the fact or action, not the arrival time of the message). An example of a message pattern is `(a INFORM fact)`, meaning that one of the new messages is from a informing the agent of `fact`. An example of a more complex message condition is `(AND (a REQUEST (DO t walk)) (NOT (?x REQUEST (DO t chew-gum))))`, meaning that there is a new message from a requesting the agent to `walk`, and there is no new request from anyone that the agent `chew-gum`.

The syntax of a commitment rule is

[11]This feature of AGENT-0 that was not included in its actual implementation, described below.

```
(COMMIT msgcond mntlcond (agent action)*)
```

where **msgcond** and **mntlcond** are respectively message and mental conditions, **agent** is an agent name, **action** is an action statement, and * denote repetition of zero or more times.[12] Note that the action statement itself may be conditional, containing its own mental condition.

An example of a simple commitment rule is

```
(COMMIT  (?a REQUEST ?action)
         (B (now (myfriend ?a)))[13]
         (?a ?action ))
```

Finally, a program is simply a sequence of commitment rules, preceded by a definition of the agent's capabilities and initial beliefs, and the fixing of the time grain.

A BNF description of the AGENT-0

Before describing the interpreter for AGENT-0 and providing an example, let me summarize the discussion of the syntax by giving its BNF definition. (In accordance with standard conventions, * denotes repetition of zero or more times.)

[12]For no good reason, the actual implementation described below allows only one agent-action pair.

[13]**now** is a global variable that evaluates to the current time. The reader might have expected other conditions, such as the absence of contradictory prior commitments. However, as is explained below in Subsection 6.2, these conditions are verified automatically by the interpreter and therefore need not be mentioned explicitly by the programmer.

```
<program>      ::= timegrain := <time>
                   CAPABILITIES := (<action> <mntlcond>)*
                   INITIAL BELIEFS := <fact>*
                   COMMITMENT RULES := <commitrule>*
<commitrule>   ::= (COMMIT <msgcond> <mntlcond> (<agent> <action>)*)
<msgcond>      ::= <msgconj>     | (OR <msgconj>*)
<msgconj>      ::= <msgpattern> | (AND <msgpattern>*)
<msgpattern>   ::= (<agent> INFORM  <fact>)   |
                   (<agent> REQUEST <action>) |
                   (NOT <msgpattern>)
<mntlcond>     ::= <mntlconj>     | (OR <mntlconj>*)
<mntlconj>     ::= <mntlpattern> | (AND <mntlpattern>*)
<mntlpattern> ::= (B <fact>) | ((CMT <agent>) <action>) | (NOT <mntlpattern>)
<action>       ::= (DO        <time> <privateaction>)  |
                   (INFORM    <time> <agent> <fact>)   |
                   (REQUEST   <time> <agent> <action>) |
                   (UNREQUEST <time> <agent> <action>) |
                   (REFRAIN   <action>)                |
                   (IF        <mntlcond> <action>)
<fact>         ::= (<time> (<predicate> <arg>*))
<time>         ::= <integer> | now | <time-constant> |
                   (+ <time> <time) | (- <time> <time>) |
                   (x <integer> <time>)   ; Time may be a <variable> when
                                          ; it appears in a commitment rule
<time-constant> ::= m | h | d | y   ; m (minute) = 60, h (hour) 3600, etc.
<agent>        ::= <alphanumeric_string> | <variable>
<predicate>    ::= <alphanumeric_string>
<arg>          ::= <alphanumeric_string> | <variable>
<variable>     ::= ?<alphanumeric_string> | ?!<alphanumeric_string>
```

A note about time: The programming language allows use of symbolic dates and times; in the actual implementation, described below, each date is represented internally by the

number of seconds that have passed since 1900.

6.2 The AGENT-0 interpreter

Since it is an instance of the generic interpreter, the AGENT-0 interpreter inherits its two-step loop design. However, since in AGENT-0 the mental state is made up of specific three components, one of which (capabilities) is fixed, the first step in the loop may be specialized as follows:

1a. Update the beliefs.

1b. Update the commitments.

We now look at the various substeps in more detail.

Updating beliefs and commitments

In AGENT-0 the beliefs, commitments, and capabilities of an agent are each represented by a data base. The belief data base is updated either as a result of being informed, or as a result of taking a private action. There is little to say about the latter; a database agent will come to believe a fact after performing a retrieval operation, and a robotic agent will come to believe something after performing a visual routine. These updates are implemented by the analogue of brain surgery, that is, by providing the appropriate routines with the ability to directly modify the belief database. More interesting is the former sort of update. In its full generality, the assimilation of new information into an existing belief base poses difficult problems, both semantical and algorithmic. It is not obvious what in general the semantics of this assimilation should be. Indeed, normative theories have been proposed for at least two different sorts of assimilation, *revision* [22] and *update* [36], and a number of new results on these operations have recently been discovered.

Beside the semantical issue, one is faced with an algorithmic one as well. Consider a given database Γ and a new fact φ. Most theories of belief assimilation require that, if φ is consistent with Γ, then assimilation amounts to simply adding φ to Γ. But checking consistency for unconstrained theories is a notoriously hard problem, either intractable (in the

propositional case) or undecidable (in the first-order case). Furthermore, if φ is inconsistent with Γ, most theories of assimilation require that Γ be 'minimally' modified so as to restore consistency, and this is even a harder problem. What then are we to do?

There are at least to approaches to getting around the computational complexity. The first is to relax the requirements, and adopt a heurisitic assimilation algorithm which compromises either soundness, or completeness, or both. The second approach, which is the one taken in AGENT-0, is to restrict the sentences in the languages sufficiently so that the problem becomes tractable. In fact, as we have seen, AGENT-0 imposes an extreme restriction, which is to disallow logical connectives other than negation (this is in addition to disallowing modal operators, necessary for representing nested beliefs such as "I believe that you believe ..."). This makes the consistency checking trivial – at most linear in the size of the database, and much less with good data structuring.

This still leaves open the question of what do with new information. We will ultimately require a theory of authority, which will dictate whether or not new information is believed. However, in AGENT-0 agents are completely gullible; they incorporate any fact of which they are informed, retracting the contradictory atomic belief if that were previously held.

We now turn to the process of updating commitments. For that we need to explain the structure of the commitment and capability databases. Items in the data base of commitments are simply pairs `(agent action)` (the agent to which the commitment was made, and the content of the commitment). Items in the data base of capabilities are pairs `(privateaction mntlcond)`. The mental condition part allows one to prevent commitment to incompatible actions, each of which might on its own be possible. An example of an item in the capability data base is

```
((!?time (rotate wheelbase ?degrees))
 (NOT ((CMT ?x) ?!time (service wheelbase)))).
```

Existing commitments are removed either as a result of the belief change, or as a result of `UNREQUEST` messages. Considering the former first, recall that agents must believe in their ability to perform the actions to which they are committed. Belief change may affect capabilities, since the capability of each private action depends on mental preconditions. And thus whenever a belief update occurs, the AGENT-0 interpreter examines the current commitments to private actions, and removes those whose preconditions in the capability data

base have been violated. Exhaustive examinations of all current commitments upon a belief change can be avoided through intelligent indexing, but I will not pursue this optimization issue. It is recommended that in such a case the agent add a commitment to immediately inform the agents to whom he was committed of this development, using commitment rules, but AGENT-0 does not enforce this.

The handling of `UNREQUEST` messages is trivial: The agent removes the corresponding item from the commitment data base if it exists, and otherwise does nothing.

Note that the removal of existing commitments is independent of the program. The addition of commitments, on the other hand, depends on the program very strongly. The algorithm adding commitments is as follows:

Check all program commitment-statement; for each program statement
`(COMMIT msgcond mntlcond (a`$_i$ `action`$_i$`)*)`, if :

- the message conditions `msgcond` hold of the new incoming message,
- the mental condition `mntlcond` holds of the current mental state,
- for all i, the agent is currently capable of the `action`$_i$, and
- for all $_i$, the agent is not committed to `REFRAIN action`$_i$, and, if `action`$_i$ is itself of the form `REFRAIN action`$_i$`'`, the agent is not committed to `action`$_i$`'`,

then for all i, commit to a$_i$ to perform `action`$_i$.

Although I am not explicit about it here, it is clear what it means for the message conditions and mental conditions to hold, given their definitions. An agent is capable of an action under the following conditions:

An agent can request and unrequest anything from anyone.

An agent can inform anyone of a fact he (the agent) believes. An agent can inform *itself* of any fact whatsoever (this is useful to implement reasoning in the agent; of course it presents a danger as well, and it is up to the programmer of commitment rules to prevent self delusion).

An agent is capable of any private action in the capability data base provided the mental condition associated in the data base with that private action holds at that time.[14]

An agent can refrain from any action, provided he is not already committed to that action.

An agent can perform a conditional action (`IF mntlcond action`) if he can perform `action` under the condition `mntlcond`.

Carrying out commitments

We have so far discussed the first of the two steps in each iteration of the interpreter, updating the mental state. We now discuss the second step, which is less complex by far. Recall that each commitment in the commitment data base has a time associated with it: (`INFORM t a fact`), (`IF mntlcond (DO t privateaction)`), *et cetera*. In this second step the interpreter simply executes all the actions whose time falls in the interval (`now-timegrain` , `now`]. The meaning of "execute" depends on the type of action:

`INFORM`, `REQUEST`, `UNREQUEST`: Send the appropriate message.

`REFRAIN`: No effect on execution (`REFRAIN` commitments play a role only in preventing commitment to other actions).

`DO`: Consulting the belief and commitment data bases, check the mental condition associated in the capability data base with the primitive action; if it holds then perform the primitive action.

`IF`: Consulting the belief and commitment data bases, test the mental condition; if it holds then (recursively) execute the action.

6.3 A sample program and its interpretation

As an example of AGENT-0 programs, consider the flight-reservation scenario described in Section 2. We now present an annotated program implementing the airline representative.

[14]This last mental condition is separate from the mental condition `mntlcond` mentioned above; the one mentioned above is a condition for making a commitment regardless of whether the agent is capable of the action; in contrast the mental condition currently discussed determines whether the agent is capable of it in the first place.

Although the scenario was simple to begin with, here I simplify it further by ignoring the exchange relating to the supervisor as well as other aspects of the communication. The idea behind the program is that the relevant activity on the part of the airline is issuing a boarding pass to the passenger, and that confirming a reservation is in fact a commitment to issue a boarding pass at the appropriate time.

Since some of the low-level definitions are long, it will be convenient to use abbreviations. We will therefore assume that AGENT-0 supports the use of macros (the actual implementation, mentioned below, does not). We define the following macros:

```
(issue_bp pass flightnum time) ⇒
    (IF (AND (B ((- time h) (present pass)))
             (B (time (flight ?from ?to flightnum))))
        (DO time-h (physical_issue_bp pass flightnum time)))
```

Explanation: This no-frills airline issues boarding passes precisely one hour prior to the flight; there are no seat assignments. `physical_issue_bp` is a private action involving some external events such as printing a boarding pass and presenting it to the passanger.

```
(query_which t asker askee q) ⇒
    (REQUEST t askee (IF (B q) (INFORM (+ t 1) asker q)))
```

Explanation: `query_which` requests only a positive answer; if `q` contains a universally-quantified variable then `query_which` requests to be informed of all instances of the answer to the query `q`.

```
(query_whether t asker askee q) ⇒
    (REQUEST t askee (IF (B q) (INFORM (+ t 1) asker q)))
    (REQUEST t askee (IF (B (NOT q)) (INFORM (+ t 1) asker (NOT q))))
```

Explanation: `query_whether` expects either a confirmation or a discomfirmation of a fact.

We now define the airline agent. To do so we need to define its initial beliefs, capabilities, and commitment rules. Of the initial beliefs, the ones relevant here refer to the flight schedule, and the number of available seats for each flight. The former are represented in the form `(time (flight from to number))` (ignoring the fact that in practice airlines have a more-or-less fixed weekly schedule), and the latter in the form `(time (remaining_seats time1 flight_number seats))`. We also assume that the agent can evaluate arithmetic comparisons, such as `4>0`.

There are two relevant capabilties here: Issuing boarding passes, and updating the count of the available seats on flights. Thus the capability data base contains two items:

```
((issue_bp ?a ?flight ?time) true)
```

```
((DO ?time (update_remaining_seats ?time1 ?flight_number ?additional_seats))
 (B (?time (remaining_seats ?time1 ?flight_number ?current_seats))))
```

Explanation: `update_remaining_seat` is a private action which changes the belief regarding `remaining_seats`.

Finally, the airline agent has two commitment rules:

```
(COMMIT (?pass REQUEST (IF (B,?p) (INFORM ?t ?pass ?p)))
        true
        ?pass
        (IF (B,?p) (INFORM ?t ?pass ?p)))

(COMMIT (?cust REQUEST (issue_bp ?pass ?flight ?time))
        (AND (B (?time (remaining_seats ?flight ?n)))
             (?n>0)
             (NOT ((CMT ?anyone) (issue_bp ?pass ?anyflight ?time))))
        (myself (DO (+ now 1) (update_remaining_seats ?time ?flight -1)))
        (?cust (issue_bp ?pass ?flight ?time)))
```

In a more realistic example one would have other commitment rules, notifying the passenger whether his reservation was confirmed, and the reasons for rejecting it in case it was not accepted. In the current implementation the passenger must query that separately.

This concludes the definition of the simple airline agent. Below is a sample exchange between a passenger, `smith`, and the airline agent. The messages from the passenger are determined by him; the actions of the airline are initiated by the agent interpreter in response. The times are given in a convenient date/hh:mm format, rather than the number of seconds that have passed since 1900.

```
agent     action
smith     (query_which 1march/1:00 smith airline
                       (18april/?!time (flight sf ny,?!num)))
airline   (INFORM 1march/2:00 smith (18april/8:30 (flight sf ny #354)))
airline   (INFORM 1march/2:00 smith (18april/10:00 (flight sf ny #293)))
airline   (INFORM 1march/2:00 smith (18april/ ...
smith     (REQUEST 1march/3:00 airline (issue_bp smith #354 18april/8:30))
smith     (query_whether 1march/4:00 smith airline
                         ((CMT smith) (issue_bp smith #354 18april/8:30)))
airline   (INFORM 1march/5:00 smith
                  (NOT ((CMT smith) (issue_bp smith #354 18april/8:30))))
smith     (REQUEST 1march/6:00 airline (issue_bp smith #293 18april/10:00))
smith     (query_whether 1march/7:00 smith airline
                         ((CMT smith) (issue_bp smith #293 18april/10:00)))
airline   (INFORM 1march/8:00 smith
                  ((CMT smith) (issue_bp smith #293 18april/10:00)))
...
smith     (INFORM 18april/9:00 airline (present smith))
airline   (DO 18april/9:00 (issue_bp smith #293 18april/10:00))
```

6.4 Implementation

A prototype AGENT-0 interpreter has been implemented in Common Lisp, and has been installed on Sun/Unix, DecStation/Ultrix and Macintosh computers. Both the interpreter and the programming manual [72] are available to the scientific community. A separate implementation has been developed by Hewlett Packard as part of a joint project to incorporate AOP in the New Wave™ architecture. The interpreter for AGENT-1, which extends AGENT-0 in a number of ways (see below), is under development.

7 Agentification

In the previous two sections I discussed the second component of the AOP framework, agent programs and their interpretation. In this section I discuss, briefly, the process of agentification. My purpose in this section is not to make a substantial contribution to the topic, but to clarify some of the issues involved and point to some related work.

Agentification refers to bridging the gap between the low-level machine process and the intensional level of agent programs. Of course, the interpreter itself is one such bridge, but it requires a direct mapping between the constructs in the agent language and the machine implementing the agent. In particular it requires explicit representation in the machine of beliefs, commitments and capabilities. When we are the ones creating the agents we indeed have the luxury of incorporating these components into their design, in which case the interpreter is adequate. However, we intend AOP as a framework for controlling and coordinating general devices, and those – cars, cameras, digital watches, spreadsheets – do not come equipped with beliefs and commitment rules.

Even if we were in a position to persuade (say) General Motors, Finmeccanica and Matsushita to equip every single product with a mental state, we would be ill-advised to do so. AOP offers a perspective on computation and communication that has its advantages, but it is not proposed as a uniform replacement of other process representations. It would be ridiculous to require that every robot-arm designer augment his differential equations with beliefs, or that the digital-watch design verifier augment finite automata with commitments.

However, releasing the manufacturers from the requirement to supply a mental state

creates a gap between the intensional level of agent programs on the one hand, and the mechanistic process representation of a given device on the other hand. The role of the agentifier is to bridge this gap.

We inherit this decoupling of the intensional level from the machine level from *situated automata*, introduced by Rosenschein in [61] and further developed by him and Kaelbling [62, 35]. In situated automata there is a low-level language for describing the device, and another, high level language for the designer to reason about the device. The compiler takes a program written in the high-level language and produces a description of a device in the low-level language.

Like the 'knowledge based' camp in distributed computation we adopt Rosenschein and Kaelbling's insight that intensional notions can be viewed as the designer's way of conceptualizing a device (as was discussed also in the introductory section, in connection with McCarthy's and Dennett's ideas). Having accepted this decoupling, however, we depart from situated automata in important ways. First, we consider different high-level and low-level languages. Concerning the high-level language, situated automata has had several versions; published versions have included a knowledge operator (K) and a tense operator ($\Diamond$, or "eventually"). Our intensional language has already been discussed – it is the AGENT-0 language defined in the previous section, which is quite richer. In fact, we are currently engaged in the enriching the language even further

The low-level process languages in the two cases are also different. Many process languages exist - synchronous Boolean circuits with or without delays (the choice of situated automata as a process language), finite automata and Turing machines, and various formalisms aimed at capturing concurrency. Our requirements of the process language included the following:

- Representation of process time, including real-valued durations
- Asynchronous processes
- Multiple levels of abstraction

We found that no existing process models met all requirements, and have developed an

alternative process model, called *temporal automata.*[15] The details of temporal automata are not relevant to the current discussion, so I will omit them; they appear in [41, 40].

The choice of intensional languages and process description is important, but more crucial than anything is the translation process envisioned. As was mentioned, in situated automata the idea is to generate a low-level process model from a high-level, intensional description. In contrast to that, the goal of agentification is to agentify a particular, given machine. The input to the translator will include a description of a machine in the process language, and the output will be an intensional program. From this perspective, the process considered in situated automata is *de*-agentification. Since we have not tackled the problem of agentification in a substantial way, it is premature to assert with certainty how difficult it is. However, I expect that the unconstrained problem will be quite difficult. By this I mean that, given only a particular device (say, a camera) and general constraints on mental state of the sort described in the article, it will be hard to generate an intensional description of the device which fully captures is functioning. After all, who is to say where in the camera lie the beliefs? Is it in the state of a particular component, or perhaps in a complicated sequence of state changes? It seems to me more fruitful to include in the input information about the location of at least some mental attitudes ("when the light meter registers x, the camera believes that..."), and attempt to synthesize a high-level program based on this information (this is closer in spirit to Rosenschein's original writing, but farther from the subsequent work). Again, I stress that we do not have sufficient experience with the problem to report any results.

8 Related work

Except occasionally, I have so far not discussed related past work. This body of related work is in fact so rich that in this section I will mention only the most closely related work, and briefly at that. I do not discuss again past work on logics of knowledge and belief, which the logic of mental state extends, since I already did that in the introduction. For the same reason, I will not discuss object oriented programming and Hewitt's work. The following is ordered in what I see as decreasing relevance to, and overlap with, AOP. The order (or, for

[15] However, in recent years the specification and verification community has taken much interest in real-time computation, and some of the recent proposals make come closer to meeting our needs.

that matter, inclusion in the list) reflects no other ranking, nor is it implied that researchers high up on the list would necessarily endorse any part of AOP.

- McCarthy's work on Elephant2000 [49]. This language under development is also based on speech acts, and the airline-reservation scenario I have discussed is due to McCarthy. One issue explored in connection with Elephant2000 is the distinction between illocutionary and perlocutionary specifications, which I have not addressed. In contrast to AOP, in Elephant2000 there is currently no explicit representation of state, mental or otherwise. Conditional statements therefore refer to the history of past communication rather than to the current mental state.

- There is related work within Distributed AI community (cf. [1]). Although AOP is, to my knowledge, unique in its definition of mental state and the resulting programming language, others too have made the connection between object-oriented programming and agenthood [31, 21].

- The Intelligent Communicating Agents project (1987-1988), carried out jointly at Stanford, SRI and Rockwell International (Nilsson, Rosenschein, Cohen, Moore, Appelt, Buckley, and many others). This ambitious project had among its goals the representation of speech acts and connection between the intensional level and the machine level. See discussion of some of the individual work below.

- Cohen and Levesque's work on belief, commitment, intention and coordination [11, 10]. This work was discussed in detail in subsection 4.4. To summarize that discussion, Cohen and Levesque too have investigated the logical relationships between several modalities such as belief and choice. Although they have not approached the topic from a programming-language perspective as I have, they too have been interested in speech acts and mental state as building blocks for coordination and analysis of behavior. Their work has its roots in earlier work in natural language understanding by Allen, Cohen and Perrault [3, 12]. Despite some similarities, crucial differences exist between the mental categories employed by Cohen and Levesque and ours.

- AOP shares with early work on *contract nets* [70] the computational role of contracts among agents. The similarity ends there, though. Contract nets were based on broadcasting contracts and soliciting bids, as opposed to the intimate communication in

AOP. Contract nets had no other notion of mental state, no range of communicative speech acts, nor the asynchronous, real-time design inherent in AOP.

- Rosenschein and Kaelbling's situated automata [61, 62, 35]. I already discussed this work in the previous section. To summarize, it is relevant in connection with the process of agentification; we adopt their idea of decoupling the machine language from the programmer's intensional conceptualization of the machine, but differ on the specific details.

- Research on coordination. Several researchers have been concerned with the process of coordination in modern environments. For example, as a part of their more global project, Winograd and Flores have developed a model of communication in a work environment. They point to the fact that every conversation is governed by some rules, which constrain the actions of the participants: a request must be followed by an accept or a decline, a question by an answer, and so on. Their model of communication is that of a finite automaton, with the automaton states corresponding to different states of the conversation. This is a macro theory, a theory of societies of agents, in contrast to the micro theory of AOP. In related work, Malone and his associates are aiming towards a general theory of coordination, drawing on diverse fields such as computer science and economics [47].

- Genesereth's work on informable agents [25, 24]. Genesereth's interest lies primarily in agents containing declarative knowledge that can be informed of new facts, and that can act on partial plans. In this connection he has investigated also the compilation of declarative plans and information into action commands. Genesereth uses the term 'agents' so as to include also low-level finite-automaton-like constructs. AOP's structure of mental state is consistent with Genesereth's declarative regime, but is not required by it.

- Recent work on plan representation and recognition by Kautz, Pollack, Konolige, Litman, Allen and others (e.g., [45, 37, 59, 7]). This literature also addresses the interaction between mental state and action, but it is usually concerned with finer-grained analyses, involving the actual representation of plans, reasoning limitations, and more complex mental notions such as goals, desires and intentions.

- Nilsson's action nets. ACTNET [58] is a language for computing goal-achieving actions that depends dynamically on sensory and stored data. The ACTNET language is based on the concept of action networks [57]. An action network is a forest of logical gates that select actions in response to sensory and stored data. The connection to AOP, albeit a weak one, is that some of the wires in the network originate from data-base items marked as 'beliefs' and 'goals'. The maintenance of these data bases is not the job of the action net.

9 Discussion

I have described the philosophy behind agent oriented programming, and progress made towards realizing it – both in terms of formal underpinning and in terms of algorithm design.

This is clearly only a beginning. Beside debugging and fine-tuning the logic (which has not been the main focus of this article) and programming language (which has been), the framework can be extended dramatically in a number of directions. Below are some of the more important ones.

- Mental categories. The language for describing mental state can be augmented to include more complex notions such as desires, intentions and plans, allowing a richer set of communicative commands and more structure on the behavior of agents. In this effort we hope to build on previous work mentioned in the previous section; [71] explores adds a notion of intention and planning, and in [18] we take a stab a the notion of desire, building on Doyle and Wellman's earlier work.

- Groundedness of mental categories. One of the contributions of distributed computation to the formal theory of knowledge is the concrete grounding of the semantics: What were formerly purely formal constructs, possible worlds, became the set of possible global states of a collection of finite-state processors, given a particular protocol. In connection with the process of agentification, it will be satisfying to the able to anchor belief and commitment similarly.

- Probability and utility. As in most recent work on knowledge and belief, we have adopted very crisp notions of mental attitude; there is no representation of graded belief

or commitment. This stands in contrast to game-theoretic work on rational interaction among agents in economics (e.g., [4, 23]) and AI (e.g., [60]), where uncertainty and utility play a key role. This is a natural direction in which to extend our framework.

- Inheritance and groups. In the analogy between OOP and AOP I did not mention inheritance, a key component of OOP today. In OOP, if an object is a specialization of another object then it inherits its methods. One analogous construct in AOP would be 'group agents'; that is a group of agents will itself constitute an agent. If we define the beliefs of this composite agent as the 'common beliefs' of the individual agents and the commitments of the composite agent as the 'common commitments' (yet to be defined) of the individual agents, then mental attitudes of the group are indeed inherited by the individual.

- Persistence of mental states. At the end of section 4 I mentioned that dealing formally with the persistence of mental state is even harder that dealing with the familiar frame problem: If I believe that you don't believe x, do I believe that you will not believe in a little while? Do I believe that I will believe that you don't? Will I believe then that you don't? Will I believe then that I believed in the past that you didn't know? Answers to these questions depend on some subtle assumptions; our preliminary results appear in [44].

- Resource limitations. In the definition of the interpreter I assumed that the belief and commitment updates all happened fast enough before the next cycle was to start. While often reasonable, this assumption is violated in many real-time applications. In these cases the manipulation of data structures (such as beliefs) must be shortened or supressed in favor of rapid action. There is much interest nowdays in intelligent real-time problem solving, including issues such as tradeoff between quality and timeliness. From the agent interpreter's standpoint this means that the belief and commitment update cannot proceed blindly, but must take into account the elapsed time, choosing wisely among mental operations.

- Belief revision and update. AGENT-0 adopts an extreme form of belief revision, accepting all new information. Obviously there are situations that call for more discriminating agents, raising the question of what constitutes a reasonable policy of belief

update. We are interested both in semantical and algorithmic questions; our results on the former appear in [15].

- Temporal belief maps. AGENT-0 restricts beliefs to 'objective' sentences, so one cannot represent beliefs of agents about the beliefs or commitments of other agents. AGENT-0 keeps tracks of these beliefs by a *time map* mechanism [13], essentially recording the points of transition and assuming default persistence between them. In the new interpreter under development, AGENT-1, we allow nested modalities in the belief database. For this purpose we introduce a new computational construct, called *mental time maps*, which is essentially a high-dimensional time map. *Temporal belief maps* are a special case, and are described in [34].

- Societies. Both the theoretical development of mental categories and the AGENT-0 programming language concentrated on a single agent. Indeed, the view promoted was of agents functioning autonomously. However, if a society of agents is to function successfully, some global constraints must be imposed. These include social *rules* as well as social *roles*; both reduce the problem solving required by agents and the communication overhead. There is a rich body of literature on computer societies, examples of which include Minsky's informal Society of Mind metaphor [52], Winograd's studies of societal roles, both human and machine [73], Moses and Tennenholtz's recent discussion of the computational advantages of social laws [56], and Doyle's pioneering work on the relationship between AI, rational psychology, and economics [17]. In recent work we have investigated the off-line design of social laws which strike a good balance between preventing chaos on the one hand, and allowing sufficient freedom to individual agents on the other hand [68, 69]; we are currently investigating the automatic on-line learning of such laws.

These are some of the directions we intend to continue to explore. Above all, it is important to experiment with significant applications. At this time there are about a half dozen projects experimenting with variants of AGENT-0, and it will be important to continue this activity. I have been deliberately conservative so far in the scope of the work, but, I believe, more ambitious explorations, involving, for example, other mental attitudes, will benefit from a

clear and rigorous basis of the kind I have defined.

Acknowledgements. I have discussed AOP in general and this document in particular with many people, and have benefitted from their comments. Members of the Nobotics group, including Jun-ichi Akahani, Nita Goyal, Hideki Isozaki, George John, Jean-Francois Lavignon, Fangzhen Lin, Eyal Moses, Anton Schwartz, Dominique Snyers, Moshe Tennenholtz, Becky Thomas, Mark Torrance and Alvaro del Val have contributed in many ways. I have discussed agents and agenthood also with Vint Cerf, Tom Dean, Mike Genesereth, Joe Halpern, Barbara Hayes-Roth, Leslie Kaelbling, Bob Kahn, Jean-Claude Latombe, Yoram Moses, Nils Nilsson, Stan Rosenschein, Rich Thomason, Terry Winograd, and many others; I apologize for not mentioning everyone. I thank Phil Cohen, Kurt Konolige, Martha Pollack and an annonymous referee for critical comments. Finally, special thanks to John McCarthy for enlightening conversations.

References

[1] Proceedings of the 10th International Workshop on Distributed Artificial Intelligence. Technical Report ACT-AI-355-90, MCC, Austin, Texas, October 1990.

[2] P. Agre and D. Chapman. Pengi: An implementation of a theory of activity. In *Proceedings 6th AAAI*, pages 268–272, 1987.

[3] J. F. Allen. Recognizing intentions from natural language utterances. In M. Brady and R. C. Berwick, editors, *Computational Models of Discourse*, pages 107–166. MIT Press, Cambridge, MA, 1983.

[4] R. Aumann. Agreeing to disagree. *The Annals of Statistics*, 4:1236–1239, 1976.

[5] J. L. Austin. *How to Do Things with Words.* Harvard University Press, 1955/1975.

[6] N. D. Belnap and M. Perloff. Seeing to it that: a canonical form of agentives. *Theoria*, 54:175–199, 1989.

[7] M. E. Bratman. *Intention, Plans, and Practical Reason.* Harvard University Press, Cambridge, MA, 1987.

[8] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1), March 1986.

[9] B. F. Chellas. Time and modality in the logic of agency. *Studia Logica*, to appear, 1991.

[10] P. R. Cohen and H. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(3), 1990.

[11] P. R. Cohen and H. Levesque. Rational interaction as the basis for communication. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication.* MIT Press, Cambridge, Ma., In press.

[12] P. R. Cohen and C. R. Perrault. Elements of a plan-based theory of speech acts. *Cognitive Science*, 3:177–212, 1979.

[13] T. Dean and D. V. McDermott. Temporal data base management. *Artificial Intelligence*, 32(1):1–55, 1987.

[14] T. Dean and M. P. Wellman. *Planning and Control.* Morgan Kauffman, 1991.

[15] A. del Val and Y. Shoham. Deriving the postulates of belief update from theories of action. Stanford working document, 1992.

[16] D. C. Dennett. *The Intentional Stance.* MIT Press, Cambridge, MA, 1987.

[17] R. Doyle. Artificial intelligence and rational self-government. Technical Report CMU-CS-88-124, Carnegie-Mellon University, Computer Science Department, 1988.

[18] R. Doyle, Y. Shoham, and M. Wellman. A theory of relative desire. In *Proc. ISMIS*, 1991.

[19] M. Drummond. Situated control rules. In *Proceedings First International Conference on Knowledge Representation and Reasoning.* Morgan Kaufmann, 1989.

[20] D. Elgesem. He would have done it anyway: the logic of agency, ability and opportunity. Stanford University, CSLI, manuscript, 1990.

[21] J. Ferber and P. Carle. Actors and agents as reflective concurrent objects: a Mering IV perspective. In [1].

[22] P. Gärdenfors. *Knowledge in Flux: modeling the dynamics of epistemic states.* MIT Press, Cambridge, MA, 1987.

[23] J. Geanakoplos. Common knowledge, bayesian learning, and market speculation with bounded rationality. memo, Yale University, 1988.

[24] M. R. Genesereth. A comparative analysis of some simple architectures for autonomous agents. Technical Report Logic-89-2, Stanford University, Computer Science Department, 1989. also to appear as [26].

[25] M. R. Genesereth. A proposal for research on informable agents. Technical Report Logic-89-4, Stanford University, Computer Science Department, 1989.

[26] M. R. Genesereth. A comparative analysis of some simple architectures for autonomous agents. In K. VanLehn, editor, *Architectures for Intelligence.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1991.

[27] P. Grice. *Studies in the Ways of Words.* Harvard University Press, 1989.

[28] J. Y. Halpern. Using reasoning about knowledge to analyze distributed systems. In J. F. Traub, editor, *Annual Review of Computer Science, Volume 2.* Annual Reviews Inc., 1987.

[29] J. Y. Halpern and Y. Moses. A guide to the modal logics of knowledge and belief: Preliminary draft. In *Proceedings of 9th IJCAI*, pages 480–490, 1985.

[30] B. Hayes-Roth, R. Washington, R. Hewett, M Hewett, and A. Seiver. Intelligent monitoring and control. In *Proceedings 11th IJCAI*, pages 243–249, 1989.

[31] C. Hewitt. Towards open information systems semantics. In [1].

[32] C. Hewitt. Viewing control structures as patterns of passing messages. *Artificial Intelligence*, 8:323–364, 1977.

[33] J. Hintikka. *Knowledge and Belief.* Cornell University Press, 1962.

[34] H. Isozaki and Y. Shoham. A mechanism for reasoning about time and belief. In *Proc. FGCS*, Japan, 1992.

[35] Leslie Pack Kaelbling. Goals as parallel program specifications. In *Proceedings AAAI-88*, pages 60–65. AAAI, 1988.

[36] H. Katsuno and A. O. Mendelzon. On the difference between updating a knowledge base and revising it. In *Proc. Second Conference on Knowledge Representation and Reasoning*, Boston, MA, 1991.

[37] H. A. Kautz. A circumscriptive theory of plan recognition. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication.* MIT Press, Cambridge, MA, 1990.

[38] K. Konolige. *A Deduction Model of Belief.* Pitman / Morgan Kaufmann, 1986.

[39] S. Kripke. Semantical considerations of modal logic. *Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik*, 9:67–96, 1963.

[40] J.-F. Lavignon. A simulator for temporal automata. Technical Report in press, Stanford University, Computer Science Department, 1990.

[41] J.-F. Lavignon and Y. Shoham. Temporal automata. Technical Report STAN-CS-90-1325, Stanford University, Computer Science Department, 1990.

[42] H. Levesque. All I know: an abridged report. In *Proceedings of 6th AAAI*, pages 426–431, 1987.

[43] F. Lin and Y. Shoham. Concurrent actions in the situation calculus. In *Proc Fourth Intl Workshop on Nonmonotonic Reasoning*, 1992.

[44] F. Lin and Y. Shoham. On the persistence of knowledge and ignorance. Stanford working document, 1992.

[45] D. J. Litman and J. F. Allen. Discourse processing and commonsense plans. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication.* MIT Press, Cambridge, Ma., 1990.

[46] W. Litwin. A model for computer life. Stanford University, Computer Science Department, manuscript, 1990.

[47] T. W. Malone. Toward an interdisciplinary theory of coordination. Technical Report CCS Technical Report 120, MIT Sloan School of Management, April 1991.

[48] J. McCarthy. Ascribing mental qualities to machines. Technical Report Memo 326, Stanford AI Lab, 1979.

[49] J. McCarthy. Elephant 2000: A programming language based on speech acts, 1990. unpublished manuscript.

[50] J. M. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[51] D. V. McDermott. Tarskian semantics, or no notation without denotation! *Cognitive Science*, 2(3):277–282, 1978.

[52] M. Minsky. *The Society of Mind.* Simon and Schuster, 1986.

[53] T. M. Mitchell. Becoming increasingly reactive. In *Proceedings 8th AAAI*, pages 1050–1058, 1990.

[54] R. C. Moore. A formal theory of knowledge and action. In J. R. Hobbs and R. C. Moore, editors, *Formal Theories of the Commonsense World.* Ablex Publishing Corporation, Norwood, New Jersy, 1985.

[55] Leora Morgenstern. *Foundations of a Logic of Knowledge, Action, and Communication.* PhD thesis, New York University, 1988.

[56] Y. Moses and M. Tennenholtz. In favor of a society. manuscript, Weizmann Institute of Science, Department of Applied Mathematics, 1990.

[57] N. J. Nilsson. Action networks. In *Proceedings of the Workshop on Planning*, University of Rochester, NY, 1989.

[58] N. J. Nilsson, R. Moore, and M. Torrance. Actnet: an action-network language and its interpreter, 1990.

[59] Martha E. Pollack. Plans as complex mental attitudes. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication.* MIT Press, Cambridge, MA, 1990.

[60] J. S. Rosenschein and M. R. Genesereth. Deals among rational agents. In *Proceedings 9th IJCAI*, pages 91–99, 1985.

[61] S. J. Rosenschein. Formal theories of knowledge in AI and robotics. Technical Report 362, SRI International, 1985.

[62] S. J. Rosenschein and L. P. Kaelbling. The synthesis of digital machines with provable epistemic properties. In *Proceedings Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 83–86. Morgan Kaufmann, 1986.

[63] S. Ross. The economic theory of agency. *American Economic Review*, 63:134–139, 1973.

[64] F. Schneider. Understanding protocols for byzantine clock synchronization. Technical report, Cornell University, Computer Science Department, August 1987.

[65] J. R. Searle. *Speech Acts: An Essay in the Philosophy of Language.* Cambridge University Press, Cambridge, 1969.

[66] Y. Shoham. Time for action. In *Proceedings 11th IJCAI*, pages 954–959, 1989.

[67] Y. Shoham and Y. Moses. Belief as defeasible knowledge. In *Proceedings 11th IJCAI*, pages 1168–1172, 1989.

[68] Y. Shoham and M. Tennenholtz. On the synthesis of useful social laws for artificial agents. Stanford working document, 1992.

[69] Y. Shoham and M. Tennenholtz. On traffic laws for mobile robots. Stanford working document, 1992.

[70] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, December 1980.

[71] S. R. Thomas. A logic for representing action, belief, capability, and intention, 1992. Stanford working document.

[72] M. Torrance. The AGENT-0 programming manual (revise), 1991. Computer Science Department, Stanford University.

[73] T. Winograd. A language/action perspective on the design of cooperative work. *Human-Computer Interaction*, 3(1):3–30, 1987-88.

[74] J. Y. Halpern L. D. Zuck. A little knowledge goes a long way: simple knowledge-based derivations and correctness proofs for a family of protocols. In *Proc of 6th ACM Symp on Principles of Distributed Computing*, pages 269–280, 1987.