Verifying that Agents Implement a Communication Language

Michael Wooldridge
Queen Mary & Westfield College
Department of Electronic Engineering
London E1 4NS, United Kingdom
M.J.Wooldridge@qmw.ac.uk

Abstract

In recent years, a number of attempts have been made to develop standardized agent communication languages. A key issue in such languages is that of conformance testing. That is, given a program which claims to semantically conform to some agent communication standard, how can we determine whether or not it does indeed conform to it? In this article, we present an expressive agent communication language, and give a semantics for this language in such a way that verifying semantic conformance becomes a realistic possibility. The techniques we develop draw upon those used to give a semantics to reactive systems in theoretical computer science. To illustrate the approach, we give an example of a simple agent system, and show that it does indeed respect the semantics.

Introduction

Perhaps the biggest single obstacle that stands in the way of the wider industrial take-up of agent technology is the issue of interoperability. That is, it must be possible for agents built by different organisations, using different hardware and software platforms, to communicate, cooperate, and negotiate using commonly agreed communication languages and protocols. This concern has lead to the development of several standardized agent communication languages (ACLs), including KQML (Patil et al., 1992) and FIPA's communication language (FIPA, 1997).

As part of these standardisation initiatives, attempts have been made to give a precise formal semantics to these ACLs (e.g., (Labrou and Finin, 1997)). Typically, these formal semantics have been developed using techniques adapted from speech act theory (Cohen and Perrault, 1979; Cohen and Levesque, 1990). If these ACL standardisation initiatives are to succeed, then the issue of *semantic conformance testing* must be successfully addressed. The conformance testing problem can be summarised as follows (Wooldridge, 1998): We are given program π_i, and an agent communication language $\mathcal{L}_C$ with the semantics $[\![\ldots]\!]_C$. The aim is to determine whether or not π_i respects the semantics $[\![\ldots]\!]_C$ whenever it communicates using $\mathcal{L}_C$. We say a program *implements* a communication language $\mathcal{L}_C$ if it respects its semantics. (Syntactic conformance testing is of course trivial.)

The importance of conformance testing *has* been recognised by the ACL community (FIPA, 1997, p1). However, to date, little research has been carried out either on how verifiable communication languages might be developed, or on how existing ACLs might be verified. One exception is (Wooldridge, 1998), where the issue of conformance testing is discussed from a formal point of view. (Wooldridge, 1998) points out that ACL semantics are generally developed in such as way as to express *constraints* on the senders of messages. For example, the semantics for an "inform φ" message in some ACL might state that the sender of the message is respecting the semantics of the language if it truly believes φ. This constraint — that the sender believes the message content — can be viewed as a *specification*. Verifying that an agent respects the semantics of the ACL then reduces to a conventional program verification problem: show that the agent sending the message satisfies the specification given by the ACL semantics.

(Wooldridge, 1998) notes that this poses the following problem for ACL conformance testing. The formalisms used to give a semantics to ACLs are typically quantified multi-modal logics, with modalities for referring to the "mental state" of agents. In the FIPA case, this mental state consists of beliefs, intentions, and the like. However, we do not currently understand how to attribute FIPA-like mental states to programs, and so we cannot verify whether or not such programs implement "mentalistic" semantics.

In this paper, we present an expressive ACL that overcomes this problem. The language, (which is intended as a proof of concept for the basic approach to ACL semantics, rather than as a serious ACL proposal), contains performatives similar to those of both KQML and FIPA. In addition, the language has a rigorous formal semantics, which superficially resemble those of (Cohen and Levesque, 1990; Labrou and Finin, 1997; FIPA, 1997). However, the language semantics are based on the semantics of concurrent systems from theoretical computer science (Manna and Pnueli, 1992; Manna and Pnueli, 1995; Fagin et al., 1995). Specifically, the semantics are given in terms of a quantified epistemic temporal logic, (QUETL). The paper begins in the next section by fully defining QUETL. We subsequently define our ACL, which we shall call $\mathcal{L}_C$. We then present a general computational model of multi-agent systems, and use QUETL to give a semantics to this model of multi-agent systems and our agent communication framework. To illustrate the approach, we present an agent program that respects the semantics we give. Finally, note that this paper is *not* concerned with giving a semantics to *human* speech acts. We are only concerned with *software* agents.

A Quantified Epistemic Temporal Logic

In this section, we define a quantified epistemic temporal logic (QUETL). This logic is essentially classical first-order logic augmented by a set of modal connectives for representing the *temporal ordering* of events and an indexed set of unary modal connectives for representing the *knowledge* possessed by agents in a system. QUETL is thus a quantified version of the epistemic temporal logics studied in (Halpern and Vardi, 1989; Fagin et al., 1995).

QUETL provides the following temporal connectives:

- the nullary temporal operator "**start**" is satisfied only at the beginning of time;
- $\bigcirc\varphi$ is satisfied now if φ is satisfied at the next moment;
- $\Diamond\varphi$ is satisfied now if φ is satisfied either now or at some future moment;
- $\Box\varphi$ is satisfied now if φ is satisfied now and at all future moments;
- $\varphi\,\mathcal{U}\,\psi$ is satisfied now if ψ is satisfied at some future moment, and φ is satisfied until then — $\mathcal{W}$ is a binary connective similar to $\mathcal{U}$, allowing for the possibility that the second argument might never be satisfied.

To express the knowledge possessed by each agent i, QUETL contains a unary modal operator $\mathcal{K}_i$. The intended reading of a formula $\mathcal{K}_i\varphi$ is "agent i knows φ" (Fagin et al., 1995).

In addition to these temporal and epistemic operators, QUETL contains the usual truth-functional connectives of classical logic, and the usual apparatus of first-order quantification.

Syntax

Formulae of QUETL are constructed from the (denumerable) sets *Pred* (predicate symbols), *Const* (individual constants), and *Var* (logical variables). In addition, QUETL contains the truth constant "**true**", the binary connective "$\vee$" (or), unary connective "$\neg$" (not), equality symbol "$=$", universal quantifier "$\forall$", and punctuation symbols "$)$", and "$($", and "$\cdot$". In addition, QUETL contains the unary modal epistemic connective "$\mathcal{K}$" (knows), a denumerable set $Ag = \{1,\ldots,n\}$ of *agent identifiers*, (used to index epistemic connectives), the binary temporal connective "$\mathcal{U}$" (until), unary temporal connective "$\bigcirc$", and nullary temporal connective **start**. The syntax of QUETL is defined by the grammar in Figure 1.

In the interests of simplicity, we assume QUETL contains no functional terms other than individual constants. Let $Term = Var \cup Const$ be the set of all terms. We use τ (with decorations: $\tau', \tau_1, \ldots$) to stand for arbitrary terms. We assume each predicate symbol is associated with a natural number called its arity, which determines the number of arguments it takes — it is assumed that predicate symbols are only applied to the appropriate number of arguments.

$$
\begin{array}{rcl}
\langle const\rangle & ::= & \text{any element of } Const \\
\langle var\rangle & ::= & \text{any element of } Var \\
\langle term\rangle & ::= & \langle const\rangle \mid \langle var\rangle \\
\langle pred\rangle & ::= & \text{any element of } Pred \\
\langle ag\text{-}id\rangle & ::= & \text{any element of } Ag \\
\langle wff\rangle & ::= & \textbf{true} \mid \textbf{start} \\
 & \mid & \langle pred\rangle(\langle term\rangle,\ldots,\langle term\rangle) \\
 & \mid & \mathcal{K}_{\langle ag\text{-}id\rangle}\langle wff\rangle \\
 & \mid & (\langle term\rangle = \langle term\rangle) \\
 & \mid & \neg\langle wff\rangle \mid \langle wff\rangle \vee \langle wff\rangle \\
 & \mid & \forall\langle var\rangle\cdot\langle wff\rangle \\
 & \mid & \bigcirc\langle wff\rangle \mid \langle wff\rangle\,\mathcal{U}\,\langle wff\rangle
\end{array}
$$

Figure 1: Syntax of QUETL

Semantics

The temporal model that underpins QUETL is $(I\!N, \leq)$, i.e., the natural numbers ordered by the usual "less than" relation. This model is widely used in theoretical computer science for representing the semantics of concurrent and distributed systems (Manna and Pnueli, 1992; Manna and Pnueli, 1995).

A *domain*, D, is a non-empty set. If D is a domain and $u \in I\!N$, then by D^u we mean the set of u-tuples over D. In order to interpret QUETL, we need various functions that associate symbols of the language with semantic objects. The first of these is an *interpretation for predicates*

$$\Phi : Pred \times I\!N \to \wp(\bigcup_{u \in I\!N} D^u)$$

which for every predicate P at every time n determines a set of tuples over D denoting the extension of P at n. (We assume Φ respects the arity of its arguments.) An *interpretation for constants* is a function $I : Const \times I\!N \to D$ which gives the denotation of a constant at some time. Note that constants are *not* assumed to be rigid designators: they may have different denotations at different times. A *variable assignment* is a function $V : Var \to D$, which gives the semantic value of every variable. We introduce a derived function $[\![\ldots]\!]_{V,I}$, which gives the denotation of an arbitrary term with respect to a particular interpretation for constants, variable assignment, and time:

$$[\![\tau]\!]^u_{V,I} \mathrel{\hat{=}} \begin{cases} I(\tau,u) & \text{if } \tau \in Const \\ V(\tau) & \text{if } \tau \in Var. \end{cases}$$

As V and I will generally be understood, reference to them will usually be suppressed.

Finally, in order to give a semantics to epistemic connectives, we require an indexed set of binary equivalence relations, $\sim_i \subseteq I\!N \times I\!N$, one for each $i \in Ag$ (Fagin et al., 1995).

Models for QUETL are $(n+3)$-tuples of the form

$$M = \langle D, \sim_1, \ldots, \sim_n, I, \Phi\rangle$$

where:

- D is a domain;

$$
\begin{array}{ll}
\langle M,V,u\rangle \models \mathbf{true} & \\
\langle M,V,u\rangle \models \mathbf{start} & \text{iff } u = 0 \\
\langle M,V,u\rangle \models P(\tau_1,\ldots,\tau_n) & \text{iff } \langle [\![\tau_1]\!]^u,\ldots,[\![\tau_n]\!]^u\rangle \in \Phi(P,u) \\
\langle M,V,u\rangle \models \mathcal{K}_i\varphi & \text{iff } \langle M,V,v\rangle \models \varphi \text{ for all } v \in I\!N \text{ such that } u \sim_i v \\
\langle M,V,u\rangle \models (\tau = \tau') & \text{iff } [\![\tau]\!]^u = [\![\tau']\!]^u \\
\langle M,V,u\rangle \models \neg\varphi & \text{iff } \langle M,V,u\rangle \not\models \varphi \\
\langle M,V,u\rangle \models \varphi \vee \psi & \text{iff } \langle M,V,u\rangle \models \varphi \text{ or } \langle M,V,u\rangle \models \psi \\
\langle M,V,u\rangle \models \forall x \cdot \varphi & \text{iff } \langle M,V \dagger \{x \mapsto d\},u\rangle \models \varphi \text{ for all } d \in D \\
\langle M,V,u\rangle \models \bigcirc\varphi & \text{iff } \langle M,V,u+1\rangle \models \varphi \\
\langle M,V,u\rangle \models \varphi\,\mathcal{U}\,\psi & \text{iff } \exists v \in I\!N \text{ such that } (v \geq u) \text{ and } \langle M,V,v\rangle \models \psi, \\
 & \quad \text{and } \forall w \in I\!N, \text{ if } (u \leq w < v) \text{ then } \langle M,V,w\rangle \models \varphi
\end{array}
$$

Figure 2: Semantics of QUETL

- $\sim_i \subseteq I\!N \times I\!N$ is a knowledge accessibility relation, one for each agent $i \in Ag$;
- $I : Const \times I\!N \to D$ interprets constants; and
- $\Phi : Pred \times I\!N \to \wp(\bigcup_{n \in I\!N} D^n)$ interprets predicates.

As usual, we define the semantics of the language via the satisfaction relation, "$\models$". For QUETL, this relation holds between triples of the form $\langle M,V,u\rangle$, (where M is a model, V is a variable assignment, and $u \in I\!N$ is a temporal index into M), and QUETL-formulae. The rules defining the satisfaction relation are given in Figure 2 (note that if f is a function, then $f \dagger \{x \mapsto d\}$ denotes the same function as f except that x maps to d). Satisfiability and validity for QUETL are defined in the standard way.

The remaining temporal connectives of QUETL are introduced as abbreviations:

$$
\begin{array}{rcl}
\Diamond\varphi & \hat{=} & \mathbf{true}\,\mathcal{U}\,\varphi \\
\Box\varphi & \hat{=} & \neg\Diamond\neg\varphi \\
\varphi\,\mathcal{W}\,\psi & \hat{=} & \varphi\,\mathcal{U}\,\psi \vee \Box\varphi
\end{array}
$$

It should be clear that QUETL inherits the expected proof theoretic properties of its temporal, epistemic, and first-order fragments.

An Agent Communication Language

We now define our agent communication language, $\mathcal{L}_C$. Like KQML and the FIPA ACL, this language has two main parts:

- An "outer" language, which defines a number of *performatives* such as "inform". In speech act terminology, the outer language is used to define the illocutionary force of a message (Searle, 1969). The performative of a message defines how the content of the message should be interpreted. In addition to the performative, the outer language contains some "housekeeping" information such as the sender and recipient of the message.
- A *content language*, which is used to define the actual content of the message. In $\mathcal{L}_C$, the content language is actually QUETL itself. QUETL is a powerful, highly expressive language, which will afford agents much greater expressive power than (for example) first-order logic.

In addition, we will assume that, (unlike KQML and FIPA) our language is *synchronous*, in the sense of Hoare's Communicating Sequential Processes (CSPs) (Hoare, 1978). Intuitively, this means that whenever an agent i attempts to execute one of the $\mathcal{L}_C$ performatives by sending a message to some other agent, the intended recipient of the message must execute a "receive message" action. To represent this, every agent is assumed to be able to perform a special action "`recv`", indicating that it has received a message. When an agent attempts to execute a performative, it will block (i.e., suspend its activities) until the recipient executes a `recv` action.

In general, a message in $\mathcal{L}_C$ has the form $p_{i,j}(\varphi)$, where p is the performative, $i \in Ag$ is the sender of the message (i.e., the agent that performs the communicative act), $j \in Ag$ is the intended recipient, and φ is the message content, expressed as a formula of the content language, i.e., QUETL. $\mathcal{L}_C$ provides four performatives: see Table 1.

The "`inform`" performative is the basic mechanism through which agents communicate information. The intuitive semantics of the `inform` performative are identical to that of the FIPA `inform` (FIPA, 1997, p25). Thus an agent will use `inform` to communicate information to another agent. More formally, an agent i that sends the message $\texttt{inform}_{i,j}(\varphi)$ must *carry the information* φ in order to satisfy the semantics of $\mathcal{L}_C$. That is not to say that φ must be present in an internal database, or that i has some variable called φ. Rather, we mean that i must *know* φ in the sense of knowledge theory (Fagin et al., 1995).

Note that the semantics of `inform` refer to the *knowledge* of the agent sending the message. Although this is a kind of "mentalistic" terminology, it should be understood that knowledge theory provides us with a precise way of attributing knowledge to arbitrary programs (Fagin et al., 1995). Unlike the mentalistic terminology of, (for example), the FIPA ACL semantics, our semantics are clearly *grounded* in the sense that they have a well-defined interpretation in terms of the states of programs. We see how this grounding works in the following section.

The "`ask-whether`" performative is the question-asking performative of $\mathcal{L}_C$. The intuitive semantics of `ask-whether` are the same as the semantics of the FIPA "`query-if`" performative (FIPA, 1997, p30): an agent i ex-

Performative	Informal meaning	Classification
$\texttt{inform}_{i,j}(\varphi)$	i informs j of φ	information passing
$\texttt{ask-whether}_{i,j}(\varphi)$	i asks j whether φ	information passing
$\texttt{commit}_{i,j}(\alpha,\varphi)$	i commits to performing action α before φ	action performing/commissive
$\texttt{refrain}_{i,j}(\alpha,\varphi)$	i will refrain from α while φ	action performing/commissive

Table 1: Summary of Performatives

ecutes the performative $\texttt{ask-whether}_{i,j}(\varphi)$ in an attempt to find out from j whether or not φ is true. More formally, an agent i executing the performative $\texttt{ask-whether}_{i,j}(\varphi)$ will respect the semantics of $\mathcal{L}_C$ if it does not currently carry the information that φ, and it does not currently carry the information that $\neg\varphi$.

Neither the original KQML language nor the FIPA ACL provide *commissive* performatives, which commit an agent to a course of action (Searle, 1969). It has been argued that the provision of such performatives is essential to the process of coordination in many multi-agent systems (Cohen and Levesque, 1995). $\mathcal{L}_C$ provides two commissives, called "`commit`" and "`refrain`" respectively.

An agent i will execute the performative $\texttt{commit}_{i,j}(\alpha,\varphi)$ in order to assert to agent j that it guarantees to perform the action denoted by α before the condition φ is true. (This *does not* assert that eventually φ *will* be true.) The sender i will be respecting the semantics of $\mathcal{L}_C$ if it carries the information that before φ becomes true, it will perform α.

The "`refrain`" performative provides agents with a way of committing never to perform some action. Thus if i executes $\texttt{refrain}_{i,j}(\alpha,\varphi)$, then i will be respecting the semantics of $\mathcal{L}_C$ if it carries the information that it will not perform α until after the condition φ becomes true. Notice that there is a kind of duality between `commit` and `refrain`.

Our next step is to define a model of multi-agent systems that is in some sense generic, and to show how agents modeled in this framework can be seen to satisfy the informal semantics discussed above.

A Model of Multi-Agent Systems

In this section, we develop a formal model of multi-agent systems that, we claim, is sufficiently general, intuitive, and powerful that it can be considered as a model for multi-agent systems implemented in most programming languages. The formal model is based upon a model of concurrent systems from theoretical computer science, which has been widely studied for several decades (Manna and Pnueli, 1992).

Formally, a multi-agent system will be considered to be the parallel composition of n programs $\pi_1,\ldots,\pi_n$. We model each program π_i as a labelled multi-graph (i.e., a graph that can have more than one arc connecting two nodes). Nodes in the graph correspond to *control points*. Let Lab_i be the set of control points for agent i, and let $Lab = \bigcup_{i\in Ag} Lab_i$ be the set of all labels. Arcs in a program correspond to the execution of actions by agents. Actions are assumed to be atomic. The paradigm example of an action would be an assignment statement. However, the performatives discussed above are also actions. Thus $\texttt{inform}_{i,j}(\varphi)$ would be one of agent i's actions. Let Ac_i be the actions that agent i can perform, and let $Ac = \bigcup_{i\in Ag} Ac_i$ be the set of all actions.

The *memory* of a program π_i is defined by a finite set of program variables, $Pvar_i$, each of which is associated with a domain (type) from which it can take a value. For simplicity, we assume that the type of every program variable is $I\!N$. Let $Pvar = \bigcup_{i\in Ag} Pvar_i$ be the set of all program variables. Formally, the memory state of an agent i at any instant is defined by a total function $ms_i : Pvar_i \to I\!N$. Let $MS_i = Pvar_i \to I\!N$ be the set of all memory states for agent i.

The effect of an action is to modify the memory state of the program that executes it. Thus an action $\alpha \in Ac_i$ can be viewed as a function $\alpha : MS_i \to MS_i$.

A *program statement* for an agent $i \in Ag$ is a four tuple: (ℓ, C, α, ℓ'). Here, $\ell \in Lab_i$ is a control point that marks the starting point of the statement, $\ell' \in Lab_i$ is the end point, C is a predicate over ms_i known as the *guard* of the statement, and α is the *action* of the statement. Following the usual convention, we write the statement (ℓ, C, α, ℓ') as $(\ell, C \longrightarrow \alpha, \ell')$. We let $Stmt$ be the set of all such statements.

Collecting these components together, a program is a structure $\pi_i = (Lab_i, Ac_i, Pvar_i, \ell_i^0, ms_i^0, Stmt_i)$ where Lab_i is the set of control points of the program, Ac_i is the set of actions that the program can perform, $Pvar_i$ is the set of program variables, $\ell_i^0 \in Lab_i$ is a distinguished member of Lab_i that marks the point at which the program starts executing, ms_i^0 is the initial memory state of the program, and finally, $Stmt_i \subseteq Stmt$ are agent i's actual program statements.

The *local state* of a program π_i at any instant is uniquely determined by a pair (ℓ_i, ms_i), where $\ell_i \in Lab_i$ is a control point (i.e., a node in the program's graph, which intuitively corresponds to a program counter), and $ms_i \in MS_i$ is a memory state. The set of all such local states for an agent i is denoted S_i, i.e., $S_i = Lab_i \times MS_i$.

The set G of all *global states* of a multi-agent system $\pi_1,\ldots,\pi_n$ is the cross product of all local states: $G = S_1 \times \cdots \times S_n$. We use $g, g', \ldots$ to stand for members of G. If g is a local state, then we write $\ell_i(g)$ to denote agent i's control point in g, and $ms_i(g)$ to denote i's memory state in g.

Now, suppose the system $\pi_1,\ldots,\pi_n$ is in state $g = (s_1,\ldots,s_n)$. Then a state $g' = (s'_1,\ldots,s'_n)$ will represent an acceptable transition from g iff some agent $i \in Ag$ has a statement $(\ell, C \longrightarrow \alpha, \ell')$ such that

1. $C(ms_i(g))$ (the guard is satisfied);
2. $\ell = \ell_i(g)$ and $\ell' = \ell_i(g')$;
3. $\ell_j(g) = \ell_j(g')$ for all $j \neq i$;
4. $ms_i(g') = \alpha(ms_i(g))$;
5. $ms_j(g') = ms_j(g)$ for all $j \neq i$.

We write $g \rightsquigarrow g'$ to denote the fact that g' is an acceptable transition from g. A run, or computation of a multi-agent system $\pi_1, \ldots, \pi_n$ is then an infinite sequence of global states

$$g_0 \xrightarrow{\alpha_0} g_1 \xrightarrow{\alpha_1} g_2 \xrightarrow{\alpha_2} \cdots$$

such that g_0 is the initial state of the system, and for all $u \in I\!N$ we have $g_u \rightsquigarrow g_{u+1}$.

Before leaving the model of multi-agent systems, we introduce an indexed set of binary relations over global states. If $g = (s_1, \ldots, s_n)$ and $g' = (s'_1, \ldots, s'_n)$ are global states, then we write $g \sim_i g'$ if $s_i = s'_i$ (Fagin et al., 1995). We will refer to $\sim_i$ as a *knowledge accessibility relation* for agent i. Intuitively, if $g \sim_i g'$, then states g and g' are indistinguishable to agent i, as it has exactly the same information in both. These relations are the mechanism through which we can attribute knowledge to programs.

Semantics for Multi-Agent Systems

Models for QUETL and runs of multi-agent systems are very closely related: both are infinite sequences of states, isomorphic to the natural numbers. This is the key to the use of temporal logic for reasoning about non-terminating programs (Manna and Pnueli, 1992). The idea is that, given a multi-agent system $\pi_1, \ldots, \pi_n$, we can systematically derive a QUETL formula $\mathcal{M}(\pi_1, \ldots, \pi_n)$, which characterizes the behaviour of the system, in the sense that every QUETL model which validates $\mathcal{M}(\pi_1, \ldots, \pi_n)$ corresponds to one of the possible runs of $\pi_1, \ldots, \pi_n$. The formula $\mathcal{M}(\pi_1, \ldots, \pi_n)$ is known as the *theory* of the system $\pi_1, \ldots, \pi_n$. In order to demonstrate that $\pi_1, \ldots, \pi_n$ satisfies specification φ, it suffices to show that $\mathcal{M}(\pi_1, \ldots, \pi_n) \vdash \varphi$, i.e., that φ is a theorem of the theory $\mathcal{M}(\pi_1, \ldots, \pi_n)$. Formally, the semantic function $\mathcal{M}$ can be understood as a mapping

$$\mathcal{M} : \text{Multi-agent system} \to \text{QUETL-formula}.$$

The procedure for generating the temporal semantics of a multi-agent system is well-documented, see e.g., (Manna and Pnueli, 1992). Here, we will simply sketch out the key aspects of the process.

The basic idea is to use two domain predicates, $at_i(\ldots)$ and $do_i(\ldots)$. The predicate $at_i(\ell)$ is used to indicate that agent i is currently at location $\ell \in Lab_i$. The predicate $do_i(\alpha)$ is used to indicate that the agent i now performs the action $\alpha \in Ac_i$. Formally, let

$$g_0 \xrightarrow{\alpha_0} g_1 \xrightarrow{\alpha_1} g_2 \xrightarrow{\alpha_2} \cdots$$

be a computation of the system $\pi_1, \ldots, \pi_n$, and let M be a QUETL model. Then:

$$\begin{array}{lll} \langle M, V, u \rangle \models at_i(\ell) & \text{iff} & \ell_i(g_u) = \ell \\ \langle M, V, u \rangle \models do_i(\alpha) & \text{iff} & \alpha = \alpha_u \text{ and } \alpha \in Ac_i \end{array}$$

We also require that the $\sim_i$ relations in M correspond to those obtained from the run using the techniques described earlier.

It is convenient to introduce a proposition $next_i$ which expresses the fact that agent i is the next one to act.

$$next_i \mathrel{\hat{=}} \exists \alpha \cdot do_i(\alpha)$$

Semantics for Agent Communication

Our task is now to give a semantics for the agent communication framework introduced above. First, we must ensure that communication is truly synchronous: every message send action is matched on the recipient's part by a corresponding receive action, and that between the message send and the message receive, the sender executes no other actions. Formally, we have

$$do_i(\alpha) \Rightarrow (\neg next_i)\, \mathcal{U}\, do_j(\texttt{rcv}) \tag{1}$$

where α is one of the performative actions introduced above, of which i is the sender and j is the recipient. Note that (1) is a *liveness property* (Manna and Pnueli, 1992).

Corresponding to this liveness property, we have a *safety property* which states that an agent cannot execute a `recv` instruction unless it is preceded by a corresponding message send operation:

$$(\neg do_j(\texttt{recv})) \Rightarrow (\neg do_j(\texttt{recv}))\, \mathcal{U}\, do_i(\alpha) \tag{2}$$

where α is once again a performative in which i is the sender and j is the recipient.

We now move on to the performatives proper. First, we give a semantics for the `inform` performative. Recall that an agent executing $\texttt{inform}_{i,j}(\varphi)$ must know φ.

$$do_i(\texttt{inform}_{i,j}(\varphi)) \Rightarrow \mathcal{K}_i(\varphi) \tag{3}$$

(Strictly speaking, this axiom is a cheat, since the argument to the $do_i(\ldots)$ predicate contains a formula of QUETL. However, as long as we are careful not to quantify over such nested formula, it will cause no problems — the argument to $do_i(\ldots)$ can simply be understood as a QUETL constant that denotes an action.)

Corresponding to this constraint on the sender of the `inform`, we have a "rational effect" constraint, which specifies the behaviour of the system if the communicative act is successful (FIPA, 1997). Note that the recipient of a message is *not* required to respect the rational effect condition; instead, it is merely intended to characterise an "ideal" situation.

$$do_i(\texttt{inform}_{i,j}(\varphi)) \Rightarrow \Diamond \mathcal{K}_j \varphi \tag{4}$$

If successful, therefore, an inform message will lead to the recipient of the message knowing the content.

An agent executing $\texttt{ask-whether}_{i,j}(\varphi))$ must not know whether φ or $\neg\varphi$.

$$do_i(\texttt{ask-whether}_{i,j}(\varphi)) \Rightarrow \neg(\mathcal{K}_i \varphi \vee \mathcal{K}_i \neg\varphi) \tag{5}$$

The rational effect of a message $\texttt{ask-whether}_{i,j}(\varphi)$ will lead to j informing i either that φ, or that $\neg\varphi$, or that it does not know either φ or $\neg\varphi$.

$$do_i(\texttt{ask-whether}_{i,j}(\varphi)) \Rightarrow \Diamond do_j(\texttt{inform}_{j,i}(\psi)) \tag{6}$$

where $\psi = \varphi$, or $\psi = \neg\varphi$, or $\psi = \neg(\mathcal{K}_j \varphi \vee \mathcal{K}_j \neg\varphi)$.

An agent executing $\texttt{commit}_{i,j}(\alpha, \varphi)$ must know that it will perform α before condition φ is true.

$$do_i(\texttt{commit}_{i,j}(\alpha, \varphi)) \Rightarrow \mathcal{K}_i \neg((\neg do_i(\alpha))\, \mathcal{W}\, \varphi) \quad (7)$$

Note that the use of $\mathcal{W}$ allows for the possibility that φ is *never* satisfied, and hence that α is never executed. The rational effect of commit is simply to make the recipient aware of the commitment.

$$do_i(\texttt{commit}_{i,j}(\alpha, \varphi)) \Rightarrow \Diamond \mathcal{K}_j \neg((\neg do_i(\alpha))\, \mathcal{W}\, \varphi) \quad (8)$$

Finally, an agent performing $\texttt{refrain}_{i,j}(\alpha, \varphi)$ must know that it will not do α until φ becomes true.

$$do_i(\texttt{refrain}_{i,j}(\alpha, \varphi)) \Rightarrow \mathcal{K}_i(\neg do_i(\alpha)\, \mathcal{W}\, \varphi) \quad (9)$$

As with commit, the rational effect is to make the recipient aware of the commitment.

$$do_i(\texttt{refrain}_{i,j}(\alpha, \varphi)) \Rightarrow \Diamond \mathcal{K}_j(\neg do_i(\alpha)\, \mathcal{W}\, \varphi) \quad (10)$$

An Example

To illustrate the communication language and its semantics, we present a small example. Consider the following program (it is straightforward to derive the formal program from the text given below):

0 : $x := 5$
1 : if $x = 5$ then goto 3
2 : $\texttt{inform}_{i,j}(x \neq 5)$
3 : $\texttt{refrain}_{i,j}(\texttt{inform}_{i,j}(x > 6), (x = 7))$
4 : if $x > 6$ then goto 7
5 : $x := x + 1$
6 : goto 4
7 : $\texttt{inform}_{i,j}(x > 6)$

We claim that this program respects the semantics defined above. The proof is straightforward, although somewhat lengthy. The basic idea is to use temporal reasoning to derive the theory of the program, using conventional techniques (Manna and Pnueli, 1992). We then use some simple epistemic rules, such as

$$(\mathrm{x}_i = n) \Rightarrow \mathcal{K}_i(\mathrm{x}_i = n)$$

(if x_i is one of i's program variables, then i knows the value of x_i), to derive an epistemic temporal theory of the program. Verifying conformance involves proving that the axioms defining the semantics of the performatives follow from this theory. The proof is done using the proof theory of QUETL, which combines that of linear discrete temporal logic, epistemic logic, and first-order logic.

Conclusions

Conformance testing is a critical issue for agent communication languages that aspire to status as international standards. If there is no practical method via which the conformance (or otherwise) to a particular ACL may be verified, then this ACL is unlikely to be accepted into the international software engineering community. Despite this, comparatively little attention has been paid to this problem. In this paper, we have demonstrated that semantic conformance testing for an ACL is possible, if the semantics of that language have a computational interpretation. We have presented a simple but, we argue, useful ACL, containing several performatives whose intuitive interpretation closely resembles that of their FIPA counterparts. In addition, we have defined the semantics of this ACL using a quantified epistemic temporal logic QUETL, demonstrated how this logic can be used to reason about multi-agent systems, and finally, given a simple agent system that respects the semantics of our language.

References

Cohen, P. R. and Levesque, H. J. (1990). Rational interaction as the basis for communication. In Cohen, P. R., Morgan, J., and Pollack, M. E., editors, *Intentions in Communication*, pages 221–256. The MIT Press: Cambridge, MA.

Cohen, P. R. and Levesque, H. J. (1995). Communicative actions for artificial agents. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 65–72, San Francisco, CA.

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

Fagin, R., Halpern, J. Y., Moses, Y., and Vardi, M. Y. (1995). *Reasoning About Knowledge*. The MIT Press: Cambridge, MA.

FIPA (1997). Specification part 2 — Agent communication language. The text refers to the specification dated 23 October 1997.

Halpern, J. Y. and Vardi, M. Y. (1989). The complexity of reasoning about knowledge and time. I. Lower bounds. *Journal of Computer and System Sciences*, 38:195–237.

Hoare, C. A. R. (1978). Communicating sequential processes. *Communications of the ACM*, 21:666–677.

Labrou, Y. and Finin, T. (1997). Semantics and conversations for an agent communication language. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 584–591, Nagoya, Japan.

Manna, Z. and Pnueli, A. (1992). *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag: Berlin, Germany.

Manna, Z. and Pnueli, A. (1995). *Temporal Verification of Reactive Systems — Safety*. Springer-Verlag: Berlin, Germany.

Patil, R. S., Fikes, R. E., Patel-Schneider, P. F., McKay, D., Finin, T., Gruber, T., and Neches, R. (1992). The DARPA knowledge sharing effort: Progress report. In Rich, C., Swartout, W., and Nebel, B., editors, *Proceedings of Knowledge Representation and Reasoning (KR&R-92)*, pages 777–788.

Searle, J. R. (1969). *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press: Cambridge, England.

Wooldridge, M. (1998). Verifiable semantics for agent communication languages. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS-98)*, pages 349–365, Paris, France.