Semantics of communicating agents based on deduction and abduction

K. V. Hindriks, F. S. de Boer,
W. van der Hoek, J.-J. Ch. Meyer

UU-CS-1999-09

Semantics of Communicating Agents Based on Deduction and Abduction

Koen V. Hindriks, Frank S. de Boer,
Wiebe van der Hoek and John-Jules Ch. Meyer
University Utrecht, Department of Computer Science
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
tel. +31-30-2539267
{koenh,frankb,wiebe,jj}@cs.uu.nl

Abstract

Intelligent agents in the agent language 3APL are computational entities consisting of beliefs and goals which make up their mental state. In this paper, we integrate communication at the agent level into the language 3APL. Communication at the agent level consists of communication between agents of their beliefs and goals. We propose two pairs of communication primitives for agent level communication. The semantics of these primitives are based on two distinct types of reasoning: deduction and abduction. Deduction serves to derive information from a received message. Abduction serves to explain facts and to find plans to fulfil requests. We illustrate the use of these communication primitives for agents by means of an example concerning agents negotiating about meeting schedules. An implementation of the multi-stage negotiation protocol is presented. The implementation is both natural and concise which illustrates the expressiveness of the communication primitives. We also discuss the relation of our approach and approaches to communication based on speech act theory.

1 Introduction

Intelligent agents in the agent language 3APL (pronounced "*triple-a-p-l*") are computational entities consisting of beliefs and goals which make up their mental state. 3APL agents are rule-based and use so-called *practical reasoning rules* to find plans for their goals, to revise their goals, and to create new goals (cf. [9], [10]). In this paper, we integrate communication at the agent level into the language 3APL. Communication at the agent level consists of communication between agents of their beliefs and goals. We propose two pairs of communication primitives for agent level communication. The first pair of communication primitives, tell and ask, are used to exchange information between agents. The second pair of communication primitives, req and offer, are used to communicate requests and offers between agents. We provide a clear and formal semantics for these communication primitives. The semantics of these primitives are based on two distinct types of reasoning: deduction and abduction. Deduction serves to derive information from a received message. The use of deduction in the specification of the semantics of information exchange is similar to that of other communication primitives proposed in the literature (see in particular [20] for such primitives in a constraint-based agent language). Abduction serves to explain facts and to find plans to fulfil requests. A semantics based on abduction is provided for the req and offer primitives and is the main contribution of this paper.

We illustrate the use of and the differences between the two communication primitives in a multi-agent meeting scheduling example. Agents used for meeting scheduling need to negotiate about meeting schedules. An implementation of the multi-stage negotiation protocol ([4]) in 3APL is presented and is used by these agents to negotiate. The implementation of the protocol in 3APL is both natural and concise, which illustrates the expressive power of the agent programming language. Furthermore, we show how to implement semaphores, a facility to guarantee mutual exclusion, by means of the communication primitives. We also implement a new operator for posting goals in the agent language.

We discuss our approach to communication and other approaches based on speech act theory, like KQML ([11]) and FIPA ([7]). We argue that speech act theory should not be viewed as a repository of many different speech acts for which computational equivalents should be provided. Instead, we aim to keep the agent communication language as simple and clear as possible, while maintaining that speech act theory may be useful for the specification and verification of agent systems.

The paper is organised as follows. First, we give an overview of the agent language 3APL. Then we propose an extension of the agent language with the four communication primitives mentioned above. The operational semantics of these primitives is specified in a transition style semantics ([14]). We present an extended example concerning agents negotiating about meeting times by using the multi-stage negotiation protocol (cf. [4], [17]). Finally, we discuss our approach and other approaches to agent communication languages based on speech act theory.

2 The Agent Language 3APL

In this section, we give an overview of the agent programming language 3APL for single agents as proposed in [9].

2.1 Intelligent Agents

The beliefs of an agent are formulas from some logical language $\mathcal{L}$. We assume that a logical consequence operator, denoted by $\models$, is associated with this language. Moreover, we assume the logical language has associated sets of variables Var, terms T, and atoms At. A substitution is defined as a set of bindings X/t such that $X \in \mathsf{Var}$ and $t \in \mathsf{T}$ (cf. [12]).

Then we can define the notion of an *answer*: a substitution θ is an answer for φ relative to a $\sigma \subseteq \mathcal{L}$ iff $\sigma \models \varphi\theta$ where $\varphi\theta$ denotes the formula obtained by simultaneous substitution of variables X in φ by t if $X/t \in \theta$. The function Free is a mapping from expressions to the set of free variables occurring in that expression. A *sentence* in $\mathcal{L}$ is a formula φ without free variables, i.e. $\mathsf{Free}(\varphi) = \varnothing$. For example, $\mathcal{L}$ might be a first-order language and $\models$ the usual consequence operator for first-order logic. The logical language is the agent's means for representing its knowledge. The choice of the knowledge representation language thus is not fixed by the programming language, but the programmer is free to choose the logical language he prefers. However, for ease of exposition, we assume that the language is a first-order predicate language.

Definition 2.1 *(goals and basic actions)*
Let $\mathcal{B}$ be a set of *basic action symbols* with typical elements b, b', $\mathcal{G}$ be a set of goal variables with typical elements G, G', and $\mathcal{L}$ be a *first-order language*. The set of *goals* $\mathcal{L}^g$ is defined by:

1. if $b \in \mathcal{B}$ and $t_1, \ldots, t_n \in \mathsf{T}$, then $b(t_1, \ldots, t_n) \in \mathcal{L}^g$,

2. $\mathsf{At} \subseteq \mathcal{L}^g$,

3. if $\varphi \in \mathcal{L}$, then $\varphi? \in \mathcal{L}^g$, and

4. if $\pi_1, \pi_2 \in \mathcal{L}^g$, then $(\pi_1;\ \pi_2), (\pi_1 + \pi_2), (\pi_1 \| \pi_2) \in \mathcal{L}^g$,

5. $\mathcal{G} \subseteq \mathcal{L}^g$.

The *basic actions* $b(\vec{t})$, *simple (achievement) goals* $P(\vec{t})$, and the *test goals* $\varphi?$ are the *basic goals* in the agent language. Basic actions are the basic means to achieve the goals of the agent. In the mental life of an agent, basic actions cause changes in the agent's beliefs. Therefore, basic actions are defined as *update actions* or *update operators* on the belief base of an agent. For example, the action $\mathsf{ins}(\mathit{meet}(\mathit{MeetId}, \mathit{Time}, \mathit{Length}, \mathit{Location}, \mathit{Att}))$ inserts the fact that a meeting *MeetId* will take place at time *Time* of duration *Length* at location *Location* with attendants *Att* in the belief base. The action $\mathsf{del}(\mathit{meet}(\mathit{MeetId}, \mathit{Time}, \mathit{Length}, \mathit{Location}, \mathit{Att}))$ deletes a scheduled meeting from the belief base. The atomic formulas $P(\vec{t})$ of the logical language are also used as simple (achievement) goals. They are useful to specify more abstract goals to achieve a certain state and thus provide an abstraction mechanism. The last type of basic goal, a test goal $\varphi?$, is a means for introspection for the agent. Test goals can be used to inspect the belief base of the agent to see if a proposition φ follows from it.

The complex goals are compositions of basic goals, constructed by using the programming constructs for sequential composition, nondeterministic choice, and parallel composition which are the well-known programming constructs from imperative programming. This allows the specification of a sequence of goals $\pi_1;\ \pi_2$, disjunctive goals $\pi_1 + \pi_2$ by means of nondeterministic choice, i.e. do either π_1 or π_2, and conjunctive goals $\pi_1 \| \pi_2$ by means of parallel composition, i.e. do both π_1 and π_2.

The goal variables in the definition of the set of goals should be thought of as *placeholders* for goals and not as actual goals. They are used to give the agent reflective capabilities related to its goals. We refer the reader to [10] for a discussion and examples of these capabilities. The notion of a substitution and the function Free are extended to also apply to goals and goal variables. $\mathsf{Free}(\pi)$ returns the set of all free variables in a goal, and a substitution θ now may also include bindings G/π where G is a goal variable and π is an arbitrary goal.

Definition 2.2 *(practical reasoning rules)*
The set of *rules* $\mathcal{L}^p$ is defined by:

6. if $\pi, \pi' \in \mathcal{L}^g$ and $\varphi \in \mathcal{L}$, then $\pi \leftarrow \varphi \mid \pi' \in \mathcal{L}^p$,

7. if $\pi \in \mathcal{L}^g$ and $\varphi \in \mathcal{L}$, then $\pi \leftarrow \varphi \ \mid, \leftarrow \varphi \mid \pi \in \mathcal{L}^p$.

Given a rule $\pi \leftarrow \varphi \mid \pi'$, π is called the *head* of the rule, φ is called the *guard* of the rule, and π' is called the *body* of the rule. Note that we allow rules with empty heads and empty bodies.

An agent consists of a unique name, a set of initial beliefs, a set of initial goals, and a set of rules. We assume a set of names $\mathcal{A}$ with typical elements a, b.

Definition 2.3 *(intelligent agent)*
An *intelligent agent* is a tuple $\langle a, \Pi_0, \sigma_0, \Gamma\rangle$ where

- a is the *name* of the agent,
- $\Pi_0 \subseteq \mathcal{L}^g$ is a set of *initial goals*,
- $\sigma_0 \subseteq \mathcal{L}$ is a set of sentences from $\mathcal{L}$, also called *initial beliefs*,
- $\Gamma \subseteq \mathcal{L}^p$ is a set of *practical reasoning rules*, also called the *rule base*.

A *mental state* is a pair $\langle \Pi, \sigma\rangle$, where $\Pi \subseteq \mathcal{L}^g$ and $\sigma \subseteq \mathcal{L}$, consisting of the goals and beliefs at a particular moment during execution. Π is called the *goal base*, σ is called the *belief base*. The goal and belief base are the only components of an agent that change during execution of the agent. The PR-base of the agent is fixed.

The *operational semantics* of the agent language is defined by a Plotkin-style transition system ([14]). Formally, a transition system is a deductive system which allows to *derive* the transitions of an agent. A transition corresponds to a one-step computation. We will also call a computation step an *action* which is the more natural term in the context of agents. A transition system consists of a set of *transition rules* where each transition rule consists of zero or more premisses and a conclusion. The set of transition rules associated with each programming construct of the language is a specification of the meaning of that construct. A transition system is an inductive definition of a transition relation $\longrightarrow$.

We use the notion of a labelled transition system. Labels are used to distinguish different types of transitions. These distinctions serve several purposes. In this paper, labels are primarily used because they allow an elegant modelling of communication. A distinction is made between transitions labelled with τ and transitions labelled with communication labels. The label τ is associated with *completed* actions of the agent, while the communication labels are associated with *attempts* to communicate (cf. also [13]). Later in the paper another type of label is introduced to define the semantics of the post operator for posting goals.

The transition relation associated with 3APL is a relation defined at several levels of the agent system: (i) at the level of a goal of the agent, (ii) at the level of an agent, and (iii) at the level of a multi-agent system.

The agent level in the language 3APL consists of the mental state of the agent, that is, its goal and belief base. At this level, an agent may execute an action by choosing a plan from its goal base and selecting an action to execute from this plan. An agent may pick any plan in its current goal base that is enabled (can be executed), execute it, and update its mental state accordingly. Alternatively, an agent may select a practical reasoning rule which is applicable to a goal in its goal base and apply the rule to the goal. The latter mechanism supplies the agent with reflective capabilities concerning its goals.

In the transition rule below, l denotes a label associated with the transition. The θ denotes a substitution or an *answer* that is computed by transforming goal π_i to goal π_i'. The answer θ is associated with transitions at the goal level. The bindings in the answer are passed to the remaining goal. This provides for a parameter mechanism in 3APL. The set of variables V is used in the transition rule for PR-rule application and its purpose is explained when this transition rule is introduced. Note that no substitution is associated with a transition at the agent level: there is no parameter passing at this level.

Definition 2.4 *(agent level)*
Let $V = \mathsf{Free}(\pi_i)$ and let θ be a substitution.

$$\frac{\langle \pi_i, \sigma \rangle_V \xrightarrow{l}_\theta \langle \pi_i', \sigma' \rangle}{\langle \{\pi_0, \ldots, \pi_{i-1}, \pi_i, \pi_{i+1}, \ldots, \pi_n\}, \sigma \rangle \xrightarrow{l} \langle \{\pi_0, \ldots \pi_{i-1}, \pi_i', \pi_{i+1}, \ldots, \pi_n\}, \sigma' \rangle}$$

An agent is a multi-threaded entity. This means that an agent may have multiple goals at the same time it is attempting to achieve. Transition rule 2.4 reduces a computation step at the *agent level* in the conclusion of the rule to a computation step of one of the goals of the agent (which occurs in the premiss of the rule). The premiss of the rule transforms a single goal from the goal base and updates the belief base accordingly. The other transition rules in this section deal with transitions at the *goal level* associated with a pair $\langle \pi, \sigma \rangle$ where π is a goal and σ is a belief base.

A basic action is an update operator on the belief base. Given a set of beliefs a basic action transforms this set into another set of beliefs, which is made formal in the execution rule for basic actions. We assume a specification of the update semantics of basic actions is given by a *transition function* $\mathcal{T}$. A test $\varphi?$ is used to derive information from the belief base. A test retrieves an answer θ which can be passed on to other parts of the goal of an agent. A test $\varphi?$ can be executed only if an instance of φ can be derived from the belief base. Moreover, it does not modify the belief base of an agent. The symbol E is used to denote (successful) termination in the transition system. We identify $E;\ \pi$, $E\|\pi$ and $\pi\|E$ with π. We also identify a goal base Π and $\Pi \cup \{E\}$.

Definition 2.5 *(execution rules for basic actions and tests)*
Let θ be a substitution restricted to the free variables of φ.

$$\frac{\mathcal{T}(b(t_1,\ldots,t_n),\sigma)=\sigma'}{\langle b(t_1,\ldots,t_n),\sigma\rangle_V \xrightarrow{\tau}_{\varnothing} \langle E,\sigma'\rangle} \qquad \frac{\sigma \models \varphi\theta}{\langle \varphi?,\sigma\rangle_V \xrightarrow{\tau}_{\theta} \langle E,\sigma\rangle}$$

Definition 2.6 *(execution rule for sequential composition)*

$$\frac{\langle \pi_1,\sigma\rangle_V \xrightarrow{l}_{\theta} \langle \pi_1',\sigma'\rangle}{\langle \pi_1;\ \pi_2,\sigma\rangle_V \xrightarrow{l}_{\theta} \langle \pi_1';\ \pi_2\theta,\sigma'\rangle}$$

Definition 2.7 *(execution rule for nondeterministic choice)*

$$\frac{\langle \pi_1,\sigma\rangle_V \xrightarrow{l}_{\theta} \langle \pi_1',\sigma'\rangle}{\langle \pi_1+\pi_2,\sigma\rangle_V \xrightarrow{l}_{\theta} \langle \pi_1',\sigma'\rangle} \qquad \frac{\langle \pi_2,\sigma\rangle_V \xrightarrow{l}_{\theta} \langle \pi_2',\sigma'\rangle}{\langle \pi_1+\pi_2,\sigma\rangle_V \xrightarrow{l}_{\theta} \langle \pi_2',\sigma'\rangle}$$

Definition 2.8 *(execution rule for parallel composition)*

$$\frac{\langle \pi_1,\sigma\rangle_V \xrightarrow{l}_{\theta} \langle \pi_1',\sigma'\rangle}{\langle \pi_1\|\pi_2,\sigma\rangle_V \xrightarrow{l}_{\theta} \langle \pi_1'\|\pi_2\theta,\sigma'\rangle} \qquad \frac{\langle \pi_2,\sigma\rangle_V \xrightarrow{l}_{\theta} \langle \pi_2',\sigma'\rangle}{\langle \pi_1\|\pi_2,\sigma\rangle_V \xrightarrow{l}_{\theta} \langle \pi_1\theta\|\pi_2',\sigma'\rangle}$$

The transition rules of definitions 2.6, 2.7 and 2.8 are fairly standard (cf. [14], [9]). The transition rules presented thus far deal only with the execution of a goal. In these transition rules, a goal is transformed by the *execution* of either a basic action or test. Another goal transformation method is goal *revision* by means of applying a rule to a goal. In that case, the goal is deleted and the body of the rule is substituted for the old goal.

Definition 2.9 *(application of rules)*
Let θ be a most general unifier for π, π', and γ be a substitution restricted to the free variables in φ.

$$\frac{\pi' \leftarrow \varphi \mid \pi'' \in' \Gamma \text{ and } \sigma \models \varphi\theta\gamma}{\langle \pi,\sigma\rangle_V \xrightarrow{\tau}_{\theta\gamma} \langle \pi''\theta\gamma,\sigma\rangle} \qquad \frac{\pi' \leftarrow \varphi \mid \in' \Gamma \text{ and } \sigma \models \varphi\theta\gamma}{\langle \pi,\sigma\rangle_V \xrightarrow{\tau}_{\theta\gamma} \langle E,\sigma\rangle}$$

$$\frac{\leftarrow \varphi \mid \pi \in' \Gamma \text{ and } \sigma \models \varphi\theta}{\langle \Pi,\sigma\rangle \xrightarrow{\tau} \langle \Pi \cup \{\pi\theta\},\sigma\rangle}$$

In the transition rules for PR-rule application, we have to make sure that no undesired interference of variables in the PR-rule and the goal to which the rule is applied occurs. Technically, this is taken care of by the set V of free variables (which is introduced in the transition rule 2.4 for the agent level). The free variables in a practical reasoning rule in the set Γ must be renamed such that the rule contains no occurrences of any variables in V. We use an accent $\in'$ to refer to this renaming. Further details can be found in [9].

2.2 Multiagent Systems

A multi-agent system is a system of multiple agents. Each agent has a unique identity which distinguishes it from other agents. Agents may differ in various respects from each other. They may have different expertises, different responsibilities and different tasks.

Definition 2.10 *(multiagent system)*
A *multiagent system* is a finite set $\mathcal{M} = \{A_1, \ldots, A_n\}$ of agents $A_1, \ldots, A_n$ such that any two different agents have different names.

The parallel execution of a multi-agent system *without communication* is modelled by interleaving. A multi-agent system can, similar to an agent, be viewed as a multi-threaded system. The transition rule which defines a computation step of a multi-agent system is very similar to the rule which defines agent execution. One of the agents is selected for execution and the multi-agent system is updated accordingly. Note that there is no label associated with the transition of the multi-agent system itself. There is no need to distinguish any transitions at the multi-agent level.

Definition 2.11 *(parallel execution of a multi-agent system)*
Let $\{A_1, A_2, \ldots, A_n\}$ be a multi-agent system.

$$\frac{A_i \xrightarrow{\tau} A_i' \text{ for some } i, 0 < i \leq n}{\{A_1, \ldots, A_i, \ldots, A_n\} \longrightarrow \{A_1, \ldots, A_i', \ldots, A_n\}}$$

3 Communication

At the multi-agent level, a new concern of how to manage the interaction of the diverse agents arises. One of the ways to manage the interaction is by means of an *agent communication language*. Communication involves two (or more) agents which interact by exchanging messages. The exchange of messages between intelligent agents concerns their beliefs and goals. In the agent communication language we propose, the messages that are exchanged between agents are sentences from a shared logical language. It is the same language which the agents use to represent their knowledge and was introduced earlier. In the context of communication, this language is also called the *content language*. (The requirement that agents need to share a content language may be generalised to the requirement that a translation mechanism between different content languages is available; cf. [19].)

We introduce two pairs of communication primitives. The semantics of these primitives is based on two types of reasoning: *deduction* and *abduction*. The first set of communication primitives, tell and ask, is used to exchange information. The reasoning involved in this type of communication is that of deduction. Deduction serves to derive information from a received message. The second set of communication primitives, req and offer, is used to make requests and to propose offers, respectively. The type of reasoning involved in this type of communication is that of abduction. Abduction serves to explain facts and to find plans to fulfil requests.

Our communication primitives are *synchronous* communication primitives. Synchronous communication of messages involves the participation of both agents at the same time. This means that the agents have to synchronise their communicative actions. Such a simultaneous communicative act of two agents is called a *handshake*. A handshake is an atomic act, in the sense that it cannot be interrupted by other actions. Although synchronous communication primitives may seem to have a number of disadvantages compared with asynchronous communication primitives, these limitations are more apparent than real. Asynchronous communication can be simulated by synchronous communication by using buffers, for example. Also, it is quite natural at the agent level to have synchronous communication.

The use of synchronous communication between agents implies that a question and an answer, or a request and an offer, are exchanged simultaneously by the agents involved in the communication. We call the agent that tells something to or requests something from another agent the *sender*, and the other agent that replies with an answer or an offer is called the *receiver*. In the sequel, we will see that this is a natural convention corresponding with the computational burden on the receiver to 'decode' the message received.

3.1 Information Exchange

We first introduce two communication primitives for the exchange of information. The semantics of these primitives is based on deduction. The exchange of information between two agents involves the sending of a piece of information by one of the agents and the receiving of that information by the other agent. Both parties must be willing to take their respective roles. In particular, we can view the role of the receiving agent as the agent asking for some information and only willing to accept messages which inform it about its question. The burden is on the receiving agent to *deduce* by using the received information the (unknown) information it asked for. Only if it is able to derive an answer to its question from the message sent, the information will be accepted (and a handshake occurs).

It is important to make clear that the first two transition rules below for the tell and ask primitive only specify *virtual* or *attempted* computation steps. Actual communication between agents only occurs when the labels associated with the virtual steps match and a handshake occurs. The precise conditions for actual communication are specified in the third transition rule below.

The communicative action $\mathsf{tell}(a, \varphi)$ is an action to send the message φ to agent a. We require that φ is a sentence *when* $\mathsf{tell}(a, \varphi)$ *is executed*, since it does not make sense to send indefinite information to another agent when an attempt is made to inform that agent. That is, any free variables occurring in a $\mathsf{tell}(a, \varphi)$ action in a plan of an agent must have been instantiated by parameter passing before the formula φ can be communicated. The empty substitution $\varnothing$ is associated with the transition of tell since the sender receives no information. The transition is labelled with $a!_i\varphi$ to indicate that an attempt of *sending* of information φ to agent a is being made. The exclamation mark ! indicates that a message is being sent, while the subscript i indicates the type of communication, namely information exchange.

Definition 3.1 *(rule for* tell*)*

$$\frac{\varphi \text{ is closed}}{\langle \mathsf{tell}(a, \varphi), \sigma\rangle_V \xrightarrow{a!_i\varphi}_{\varnothing} \langle E, \sigma\rangle}$$

The communicative action $\mathsf{ask}(a, \psi)$ is an action of posing a question ψ to agent a. In this case, the question ψ does not have to be a sentence, but may contain free variables. The free variables indicate what the question is about. An answer to the question is a substitution θ which *grounds* ψ. The requirement that θ grounds ψ means θ is a *complete* answer. The answer θ is the information the agent *deduces* from the message φ of the sender a and its own beliefs σ (there may be more than one possible answer, in which case the agent has to select one). The answer is deduced by computing which θ satisfy $\sigma \cup \varphi \models \psi\theta$. The computational burden thus is on the asking agent. In the transition rule for ask, the question mark ? in the label indicates that the asking agent is the receiving agent. The subscript i in the label indicates that the communication concerns information exchange.

Definition 3.2 *(rule for* ask*)*
Let θ be a substitution restricted to the free variables of ψ.

$$\frac{\psi\theta \text{ is closed}}{\langle \mathsf{ask}(a, \psi), \sigma\rangle_V \xrightarrow{a?_i\psi\theta}_{\theta} \langle E, \sigma\rangle}$$

Remark 3.3 By relaxing the requirement that θ grounds the question ψ and only requiring that θ provides an answer for question ψ, the possibility of providing only *partial* answers would be allowed. From a programming perspective, however, we believe that partial answers only complicate the programming task. Moreover, similar, though not quite the same behaviour, can be obtained by existentially quantifying variables in the question.

In Figure 1, the communication of information between two agents is illustrated. The sending agent A, which is the agent using the communicative action $\mathsf{tell}(B, \varphi)$, communicates to agent B the proposition φ. When this proposition is received by agent B and that agent has a matching $\mathsf{ask}(A, \psi)$ indicating that it is willing to communicate with A, agent B uses the received information to compute an answer to its question. The received

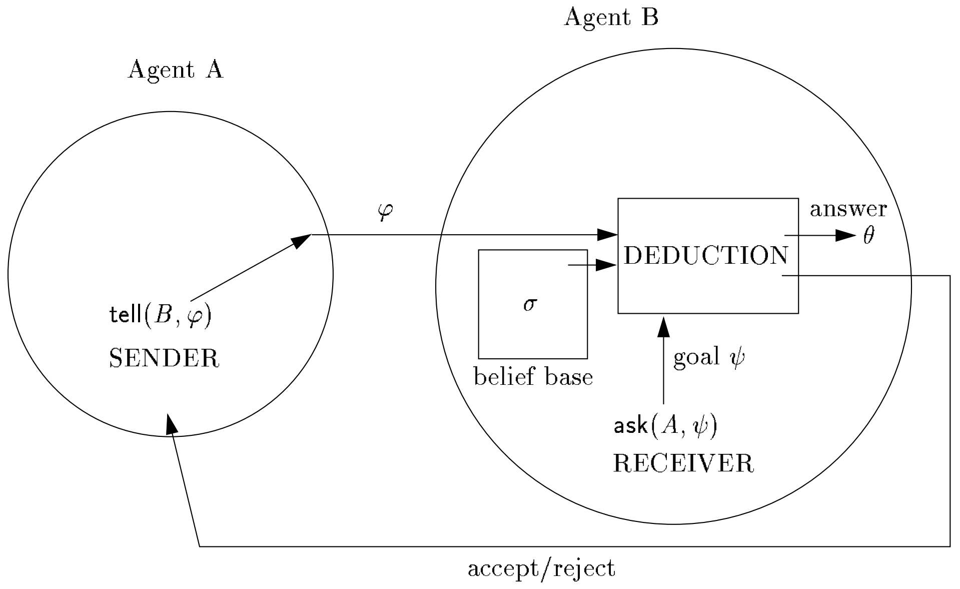

Figure 1: Information exchange by means of deduction

proposition φ and the current belief base of agent B are inputs for a deductive system, like for example Prolog, and the deductive system is given the question ψ as a *goal*, which means it should attempt to compute an *answer* for ψ. In case the deductive system succeeds, it outputs an answer θ. The answer is used in further computations of agent B. The deductive system also indicates whether it succeeded or failed. This information is (implicitly) communicated to agent A (an accept is sent in case of success, a reject in case of a failure).

An act of simultaneous information exchange between two agents (a handshake) occurs when both agents address each other and the sending agent provides the information from which the receiving agent is able to deduce an answer to the question it posed to the sending agent.

Notation 3.4 If $\mathcal{M}$ is a set of agents and A is an agent, the notation $\mathcal{M}, A$ is used to denote the union of $\mathcal{M} \cup \{A\}$.

Definition 3.5 *(rule for synchronous information exchange)*
Let $A = \langle a, \Pi_a, \sigma_a, \Gamma_a \rangle$ and $B = \langle b, \Pi_b, \sigma_b, \Gamma_b \rangle$ be two agents such that $a \neq b$, and let $\mathcal{M}$ be a (possibly empty) multi-agent system such that $A \notin \mathcal{M}, B \notin \mathcal{M}$.

$$\frac{A \xrightarrow{b!_i\varphi} A' \ , \ B \xrightarrow{a?_i\psi} B' \text{ and } \sigma_b \cup \varphi \models \psi}{\mathcal{M}, A, B \longrightarrow \mathcal{M}, A', B'}$$

Example 3.6 As a simple example, consider two agents John and Roger which are going to a meeting to discuss a paper. Three simple questions agent Roger could ask agent John concerning this meeting are "Where do we meet?", "Who do we meet?" and "When do we meet?". The corresponding questions can be asked in the agent language, respectively, by:

$$\mathsf{ask}(john, meet(paper, 10:00Fri, 1:00, Location, [john, roger, mark]))$$
$$\mathsf{ask}(john, meet(paper, 10:00Fri, 1:00, amsterdam, Att))$$
$$\mathsf{ask}(john, meet(paper, Time, 1:00, amsterdam, [john, roger, mark]))$$

In response to each of these three questions John might send the following message to Roger. This message provides Roger with the information that the meeting will take place 10am on Friday in Amsterdam together with John and Mark.

$$\mathsf{tell}(roger, meet(paper, 10:00Fri, 1:00, amsterdam, [john, roger, mark]))$$

Notice that we do not require sending agents to be *honest* when they tell something to another agent. One reason for this is that we want to keep the semantics of our primitives as simple as possible. Another reason is that from the perspective of speech act theory one could argue that sincerity conditions concerning the mental state of the agent should not be part of the (semantic) definition of speech acts (cf. [2]). For example, if agent a is lying to agent b that φ is the case, agent a can still be described as informing or telling agent b that φ. Moreover, it is possible to define a communication primitive inform by using the primitive tell which only succeeds in case the sending agent believes what it tells.

Example 3.7 *(honest information exchange)*
An honest communication primitive inform is defined by:

$$\mathsf{inform}(a, \varphi) \equiv \varphi?;\ \mathsf{tell}(a, \varphi)$$

Although the inform action first performs a check on the belief base to see whether or not the message φ is believed, there is no guarantee that the agent still believes φ at the moment of sending the message. The reason is that an agent is multi-threaded and the defined action inform is not atomic. As a consequence, some other goal of the agent may have interferred and caused a change in the belief base of the agent *after* performing φ? but before communication by means of tell has taken place. However, if no other goal of the agent interrupts the execution of the defined primitive inform by changing the belief of the agent that φ is the case, then the primitve inform succeeds only if the agent believes what it tells.

4 Requests

The second pair of communication primitives allow agents to make requests and offers in reply to requests. The semantics of these primitives is based on abduction. We distinguish two types of requests. A request to *establish* a state of affairs and a request to *explain* an observation. It is expected from the receiving agent to make an offer which satisfies the request.

Abduction is sometimes paraphrased as reasoning from an observation to an explanation, or alternatively, from an effect to a cause. In a communication setting, where more than one agent is involved, abduction can be used to specify the semantics of requests and offers made in reply. As in the case of information exchange, we introduce two new communication primitives req and offer. The primitive req corresponds to a *request* and the primitive offer corresponds to an *offer*. The basic idea for the semantics of these primitives is that abductive reasoning can be used by the offering agent to provide a cause that *would* establish an effect which is desired by the requesting agent. Abduction reasoning thus not only can be used to explain the past ([18]), but can also be used to provide a cause that would establish a state of affairs.

More formally, abduction can be characterised as follows. Given a *background theory* T, a set of conjectures or *hypotheses* H, and an *observation* o, the *abductive problem* is to find an (instance of a) $h \in H$ such that $T \cup h \models o$ and $T \cup h$ is consistent (cf. [15], [6]; this is the strong notion of abduction). A number of requirements can be formulated concerning the hypothesis h. In particular, the hypothesis is required to be a most specific hypothesis which explains o. The explanation h is said to *cover* o.

The communicative action $\mathsf{req}(a, \varphi)$ is a request to agent a to cover φ in the formal sense outlined above. The request should be definite, i.e. we do not allow the occurrence of free variables in *φ when $\mathsf{req}(a, \varphi)$ is executed.* That is, any free variables occurring in a request must have been instantiated by parameter passing before the request can be communicated. The requesting agent is not automatically informed of the offer the receiving agent a makes, which explains the empty substitution $\varnothing$. In particular, the sending agent cannot infer that its request (to establish something) will be carried out. This depends on the goals of the receiving agent. It is (implicitly) acknowledged, however, whether or not the receiving agent is able to make an acceptable offer. The symbol $!_r$ in the label associated with the transition distinguishes requesting from information exchange. As before the exclamation mark ! indicates that a message is being sent, while the subscript r indicates the type of communication, namely requesting.

Definition 4.1 *(rule for* req*)*

$$\frac{\varphi \text{ closed}}{\langle \mathsf{req}(a, \varphi), \sigma \rangle_V \xrightarrow{a!_r\varphi}_{\varnothing} \langle E, \sigma \rangle}$$

The communicative action $\mathsf{offer}(a, \psi)$ is an action to make an offer ψ in reply to a requesting agent a. ψ may contain free variables which indicate the range of offers the agent is prepared to make. That is, by offering ψ implicitly any instance of the free variables in ψ is offered. The offering agent needs to *abduce* suitable instances for these free variables such that the request φ of agent a is covered. That is, the offering agent needs to compute a substitution θ such that $\sigma \cup \psi\theta \models \varphi$ where σ is the belief base of that agent. The computational burden thus is on the offering agent. In the terminology introduced above, the belief base of the offering agent corresponds to the background theory in the corresponding abductive problem; the request φ corresponds to the observation in the abductive problem; and ψ is the hypothesis which the offering agent may use to cover the request. There may be more than one instantiation of the free variables in the offer ψ which satisfies the request of agent a (this is a general characteristic of abduction). In that case, the offering agent is free to choose according to his own preferences. The condition that $\psi\theta$ is closed in the transition rule below corresponds to the requirement that the agent should make the most specific offer it is able to make. The question mark ? in the label indicates that the offering agent is the receiving agent. The subscript r indicates that the communication involves a request.

Definition 4.2 *(rule for* offer*)*
Let θ be a substitution restricted to the free variables of ψ.

$$\frac{\psi\theta \text{ is closed}}{\langle \mathsf{offer}(a,\psi),\sigma\rangle_V \xrightarrow{a?_r\psi\theta}_\theta \langle E,\sigma\rangle}$$

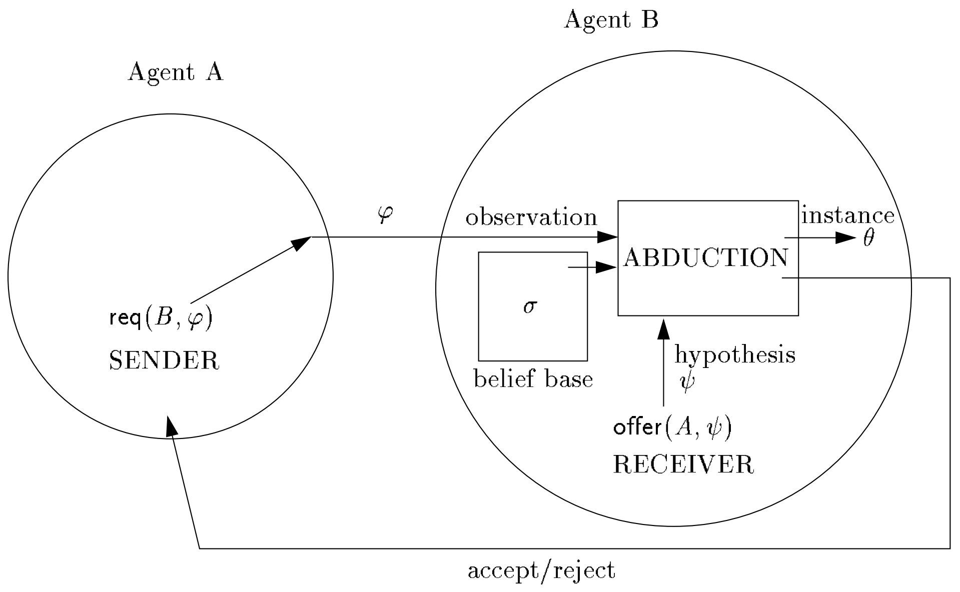

Figure 2: Requesting and offering by means of abduction

In figure 2, the requesting and offering between two agents is illustrated. The sending agent A, which is the agent using the communicative action $\mathsf{req}(B,\varphi)$, communicates to agent B the proposition φ. When this proposition is received by agent B and that agent has a matching $\mathsf{offer}(A,\psi)$ indicating that it is willing to communicate with A, agent B uses the received information to compute a most specifc *instance* of its offer. Recall that an abductive problem consisted of a background theory T, a hypothesis h and observation o. To compute the instance, agent B uses an *abductive system.* The received proposition

φ is inputted as the *observation* into this system. The current belief base of agent B is inputted as the *background theory*. And the abductive system is asked to compute an instance of *hypothesis* ψ. In case the abductive system succeeds, it outputs an instance θ. The instance θ is used in further computations of agent B. The abductive system also indicates whether it succeeded or failed. This information is communicated to agent A (accept in case of success, reject in case of a failure).

A handshake between a requesting agent a and an offering agent b occurs when both agents address each other and agent b is able to abduce a specific offer ψ which covers the request φ of agent a. The offering agent should also believe that its offer is consistent given what it believes to be true. This precludes the agent from offering too strong offers like the absurd offer $\bot$ (false) which would satisfy any request. The semantics of the primitives req and offer does not specify that the offering agent is *capable* of actually achieving the offer which is made in reply to a request to establish some state of affairs. This is analogous to the fact that there is no honesty condition associated with the tell primitive. Note that an agent which requests an explanation of some fact is implicitly informed of a failure of the offering agent to offer an *explanation* since in that case the communication does not succeed.

Definition 4.3 *(rule for synchronous requests)*
Let $A = \langle a, \Pi_a, \sigma_a, \Gamma_a \rangle$ and $B = \langle b, \Pi_b, \sigma_b, \Gamma_b \rangle$ be two agents such that $a \neq b$, and let $\mathcal{M}$ be a (possibly empty) multi-agent system such that $A \notin \mathcal{M}, B \notin \mathcal{M}$.

$$\frac{A \xrightarrow{b!_r\varphi} A' \ , \ B \xrightarrow{a?_r\psi} B' \ , \sigma_b \not\models \neg\psi \text{ and } \sigma_b \cup \psi \models \varphi}{\mathcal{M}, A, B \longrightarrow \mathcal{M}, A', B'}$$

Example 4.4 As an example, consider again the meeting example concerning agents John and Roger. A request of agent John for a meeting to discuss a paper with agent Roger at 10 am next Friday in Amsterdam would be the following.

$\mathsf{req}(roger, meet(paper, 10:00Fri, 1:00, amsterdam, [roger, john]))$

Agent Roger might make the following offer to meet John *together with Mark* at *any place* 10 am next Friday:

$\mathsf{offer}(john, meet(paper, 10:00Fri, 1:00, AnyPlace, [john, roger, mark]))$

Given just this offer, it is not possible to find an instance of it that covers John's request. However, if agent Roger believes that a meeting with John, Mark and himself implies that a meeting is held with John and himself, then this is possible. This belief is expressed by the following clause, where *subset* is the usual subset relation:

$$\begin{aligned}&(meet(MeetId, Time, Length, Location, SetAtt) \wedge subset(SubAtt, SetAtt))\\ &\qquad \rightarrow meet(MeetId, Time, Length, Location, SubAtt)\end{aligned}$$

If this belief is a belief of Roger, it is possible for Roger to abduce that offering a meeting in Amsterdam will fulfil the request of John.

The example illustrates the use of variables in the offer of the offering agent. On the part of the offering agent a variable in an offer ψ is an offer to satisfy *any specific request* of the requesting agent concerning this parameter in the offer. In the example, the agent offers to meet at any requested place. It does not mean, however, that the agent offers to achieve *every* instance of its offer. That is, it does not offer the universal closure $\forall\psi$ to the requesting agent. And neither would the requesting agent be satisfied by such an offer in the meeting example (it does not even make sense to meet at *all possible* places at a particular time). The requesting agent would like the offering agent to meet its request as concrete as possible. This requires that the offering agent abduces a *most specific instance* of its offer which satisfies the needs of the requesting agent.

Example 4.5 Agent Roger may update its agenda accordingly after making the offer to John. The following goal would do. Note that the offer action calculates a binding for the free variable *AnyPlace* which is passed onto the ins action.

```
offer($john, meet(paper, 10:00Fri, 1:00, AnyPlace, [john, roger, mark]$);
ins($meet(paper, 10:00Fri, 1:00, AnyPlace, [john, roger, mark])$)
```

Suppose that Roger did not only offer to meet at any place but also at any time, but only would meet its offer if the time of the meeting requested would be later than 11 am. The following goal provides a corresponding plan in the agent language:

```
offer($john, meet(MeetId, AnyTime, AnyLength, AnyPlace, [john, roger, mark]$);
IF $AnyTime >= 11:00$
THEN begin
          ins($meet(MeetId, AnyTime, AnyLength, AnyPlace, [john, roger, mark])$);
          tell($john, confirm(AnyTime)$)
     end
ELSE tell($john, disconfirm$)
```

The plan of Roger is to agree to the proposal for a meeting if the meeting time is later than 11am and otherwise to refuse the meeting proposal. In both cases, Roger is decent enough to inform John of his decision.

Remark 4.6 By using the offer(a, ψ) primitive, the set of hypotheses associated with the abductive problem in the semantics of synchronous requests is a singleton $\{\psi\}$. It is, however, possible for communicating agents to use abductive problems in their full generality. That is, it is possible to implement an action offer(a, H) where H is a set of hypotheses, and the associated abductive problem is to find an instance of precisely one hyposthesis that covers the request made by another agent. The more general communicative action is implemented by

$$\mathsf{offer}(a, \{h_1, \ldots, h_n) \equiv \mathsf{offer}(a, h_1) + \ldots + \mathsf{offer}(a, h_n)\}$$

Example 4.7 As an example of a request for an explanation, consider the use of the request primitive for asking *why*-questions. For example, agent Roger might wonder why John will go to Amsterdam next Friday. Since scheduled activities are reasons for agents to be at certain locations, the request can be formulated by:

$$\mathsf{req}(john, meet(\{course, paper, workshop\}, 11:00Fri, amsterdam, [john])$$

In this example, we introduced a new notation for formulating requests. The expression $\{course, paper, workshop\}$ is a notational device denoting the possible explanations agent John allows or thinks are possible. Such a request is *not* equivalent to a disjunctive request made up of the three disjuncts corresponding to the alternatives. Aa disjunctive request might trivially be explained by the disjunct itself, but the request of John is a request to explain *one* of the alternatives indicated. It is equivalent, however, to the following nondeterministic goal which is a request to either explain that John has a course to teach, a paper to discuss, or a workshop.

$$\begin{aligned}&\mathsf{req}(john, meet(course, 11:00Fri, amsterdam, [john]) +\\&\mathsf{req}(john, agenda(paper, 11:00Fri, amsterdam, [john]) +\\&\mathsf{req}(john, agenda(workshop, 11:00Fri, amsterdam, [john])\end{aligned}$$

Agent John might simply offer one of the three alternatives as an explanation. For example, agent John could explain that it will go to Amsterdam because of a course it has to teach there:

$$\mathsf{offer}(roger, meet(course, 11:00Fri, amsterdam, [john]))$$

The explanation, however, still needs to be told to Roger. The receiving agent is only induced to explain the observation of the other agent. The agent could communicate how it explains the observation by means of tell.

The use of nondeterministic choice to implement the notational device introduced in this example is analogous to the use of choice to generalise an offer to a set of hypotheses in the previous remark. The same notational device used in this example for a requesting agent therefore can be used to make several offers.

Deductive and abductive semantics for communication primitives Because of two properties of first-order logic, deduction and abduction are closely related to each other. It is an easy consequence of the deduction theorem and contraposition, that $\sigma \cup \varphi \models \psi$ iff $\sigma \cup \neg\psi \models \neg\varphi$ (cf. also [5]). However, the primitives cannot be defined in terms of each other because of the consistency requirement in the semantics of the req and offer primitives. The differences between the sets of primitives become even bigger if logics are used in which either the deduction theorem or contraposition fails. For example, in logic programming negation is interpreted as negation as failure, and contraposition fails.

5 Meeting Scheduling

One of the interesting metaphores introduced in agent research, is to view programs as intelligent agents acting on our behalf. For example, intelligent agents may perform routine and tedious information processing activies in organisations on behalf of their users. One of the applications one might consider using agents for is that of meeting scheduling. Although meeting scheduling is a complex problem, some solutions have been proposed in the literature (cf. [17]). One of the solutions proposed, is to implement distributed meeting scheduling by using the multi-stage negotiation protocol ([4]). We show how to implement the multi-stage negotiation protocol in 3APL. First, we design a set of plans for two-agent meeting scheduling where only two agents need to agree on a meeting time. Then we generalise this solution to multi-agent meeting scheduling for an arbitrary number of agents.

The basic idea of the multi-stage negotiation protocol is to allow agents to negotiate for several rounds to reach an agreement. One agent is designated as the *host agent*; it is the agent who is trying to arrange the meeting with the *invitees* (a subset of the other agents). Every stage of the negotiation protocol consists of two phases. The first phase is the *negotiation phase* in which proposals and counterproposals are exchanged. In the negotiation phase, the host starts by proposing a meeting time to all the invitees for the meeting. In reply the invitees either indicate *acceptance* of the meeting proposal or they *counterpropose* a *later* meeting time. The second phase is the *evaluation phase* in which the *host* evaluates the counterproposals and selects a best proposal.

To guarantee that the negotiation will come to a conclusion, we make a number of assumptions. First of all, we assume that all attendants of a meeting including host and invitees always share a free slot of appropriate length in their agenda where the meeting can be scheduled. Furthermore, the strategy of the host during the negotiation is to request the invitees to counterpropose with the earliest meeting time consistent with their constraints and preferences. We also assume that all scheduled meetings can be identified by unique identifiers and that a new label which is different from all labels used to refer to meetings in the multi-agent system is associated with the meeting that is being negotiated. The latter requirement allows agents to identify the meeting which is negotiated and differentiate it from already scheduled meetings.

More formally, the condition that all agents have a free slot at the same time of any duration can be specified by (any free variables are implicitly universally quantified):

$$\exists\, PosTime \cdot \forall\, a \in \mathcal{A} \cdot PosTime \geq T \wedge \neg \mathsf{B}_a(\neg meet(MeetId, PosTime, L, Att))$$

The B operator is a modality for belief, indicating that the agent believes the proposition following the operator. (We do not provide the formal semantics of this operator, which remains for future work.) Note also that we dropped the argument corresponding to the location of the scheduled activity. We assume that locations are not negotiated which justifies leaving out the location argument in the sequel.

5.1 Integrity Constraints Associated with Meetings

A number of integrity constraints need to be satisfied for a set of scheduled meetings to be coherent. Therefore, we suppose that the integrity constraints listed below are part of the belief bases of all agents. Of course, agents may have more integrity constraints associated with meetings, in particular concerning the preferences of their users. Below, free variables are implicitly universally quantified.

Constraint 1: Meeting Identifiers refer to unique meetings:

$$\begin{aligned}&(meet(MeetId, T_1, L_1, Att1) \wedge meet(MeetId, T_2, L_2, Att2))\\&\qquad\qquad \rightarrow (T_1 = T_2 \wedge L_1 = L_2 \wedge Att1 = Att2)\end{aligned}$$

Constraint 2: The attendants of two overlapping meetings are disjoint sets of people:

$$\begin{aligned}&(meet(MeetId1, T_1, L_1, Att1) \wedge meet(MeetId2, T_2, L_2, Att2) \wedge\\&T_1 \leq T_2 < T_1 + L_1 \wedge MeetId1 \neq MeetId2)\\&\qquad\qquad\qquad \rightarrow Att1 \cap Att2 = \emptyset\end{aligned}$$

Constraint 3: A meeting takes longer than 0:00 but less than 8:00 hours:

$$meet(MeetId, T, L, Att) \rightarrow 0:00 < L < 8:00$$

Constraint 4: There are no meetings scheduled between 18:00 and 8:00:

$$meet(MeetId, T, L, Att) \rightarrow (T \geq 8:00 \wedge T + L \leq 18:00)$$

5.2 Two-agent Meeting Scheduling

We first design a negotiation protocol for negotiation between two agents. In two-agent meeting scheduling, one of the agents is the host and the other agent is the invitee. The two-agent case is substantially simpler than the case where more than two agents are allowed. The main reason is that in the latter case the host has to evaluate the proposals of more than one agent and has to select the *best* proposal each round of the negotiation. Because the host has to deal with more than one invitee, the synchronisation of rounds of the negotiation is more complex.

The negotiation proceeds as follows. Each round the host requests the invitee to accept the proposed meeting time or to counterpropose the earliest possible meeting time consistent with the constraints of the invitee. After the counterproposal has been made, the host evaluates it. In case an agreement has been reached, the negotiation is ended and a confirmation of the agreement is communicated to the invitee. In case an agreement has not yet been reached, the host starts a new round in the negotation and counterproposes with the earliest possible meeting time relative to the meeting time which was proposed by the invitee and which is consistent with the host's integrity constraints. We assume that a unique meeting identifier is available to the host, which has not been used to refer to any other meeting to identify the meeting which is being negotiated.

To formalise the request for an earliest possible meeting, we introduce the predicate $epmeet(MeetId, PosTime, L, Att, T)$. This predicate states that time $PosTime$ is the earliest time after time T such that the meeting $MeetId$ of Length L and attendants Att can be scheduled. $epmeet(MeetId, PosTime, L, Att, T)$ holds if it is not consistent with the agent's beliefs that $meet(MeetId, T', L, Att)$ holds for a time T' between T and $PosTime$, but the meeting can be scheduled at time $PosTime$. Formally, $epmeet(MeetId, PosTime, L, Att, T)$ is defined as follows:

$$\begin{aligned} & epmeet(MeetId, PosTime, L, Att, T) \leftrightarrow \\ & \quad meet(MeetId, PosTime, L, Att) \wedge PosTime \geq T \wedge \\ & \quad\quad (\forall T' \cdot T \leq T' < PosTime \rightarrow \neg meet(MeetId, T', L, Att)) \end{aligned}$$

The host invites another agent to meet at a time T acceptable to the host of the meeting by means of the goal

invite($Invitee, MeetId, T, L, Att$)

The plan (rule) to achieve this goal is given next. The first two steps in the plan (the body of the rule below) which consist of the request and offer actions constitute the *negotiation phase* of the plan. In this phase, the host first requests the invitee to meet at the earliest possible time after time T that is consistent with its constraints. The invitee in response calculates a counterproposal which it communicates to the host. In reply, the host offers a meeting time T' which is the earliest possible meeting time at which the host is able to meet and which is later or equal to the time proposed by the invitee. The latter part of the plan is called the *evaluation phase* of the plan. In this phase the host evaluates the proposal of the invitee. If the time T' that is proposed by the host is identical to the original time T requested by the host, an agreement has been reached. In that case, the host confirms that an agreement has been reached. In the other case, the host starts a new round of negotiation with a request to meet at time T'.

```
invite($Invitee, MeetId, T, L, Att$) $\leftarrow$ true |
    req($Invitee, \exists T1 \cdot epmeet(MeetId, T1, L, Att, T)$);
    offer($Invitee, meet(MeetId, T', L, Att)$);
    IF $T = T'$
        THEN tell($Invitee, confirm(MeetId, T)$)
        ELSE invite($Invitee, MeetId, T', L, Att$)
```

The reply($Host$) goal is used by the invitee to negotiate with the host. The invitee first calculates an offer it counterproposes in reply to the request of the host to meet at the earliest time possible. After a counterproposal has been established, the invitee replies to the host with a request to accept its counterproposal. In case the host confirms that an agreement has been reached, the invitee accepts the confirmation (this is implemented by the ask branch of the nondeterministic plan for reply).

```
reply(Host) ← true |
    begin
        offer(Host, meet(MeetId, U, L, Att));
        req(Host, ∃ U1 · epmeet(MeetId, U1, L, Att, U));
        reply(Host)
    end+
    ask(Host, confirm(MeetId, U))
```

Note that the host as well as the invitee only make offers for meeting times which are consistent with their constraints and preferences. For example, if an agent already has scheduled to attend another meeting at time T, the agent will not offer this time in reply to a request since it violates constraint 2. The consistency condition present in the abduction semantics of offer and req enforces the integrity constraints automatically.

5.3 Using tell and ask for Meeting Scheduling

In the previous section, a protocol for two-agent meeting scheduling was designed by using the primitives req and offer. The question we address in this section, is whether or not there is a just as natural solution for two-agent meeting scheduling by using the tell and ask primitives. We argue that there is not. We do so by comparing a few attempts to provide a solution to two-agent meeting scheduling by using tell and ask. These preliminary attempts also illustrate some of the differences between the sets of communication primitives. Finally, we provide an implementation of the multi-stage negotiation protocol by using the tell and ask primitives and discuss some of the disadvantages of this implementation.

The strategy we will use is to try and find particalur tell and ask actions which can serve to replace the req and offer in the plans provided in the previous section. So, we are looking for tell and ask actions such that the replacement of all requests req and all offers offer in the implementation of the previous section by the tell and ask actions respectively results in a correct implementation of the protocol.

It seems that there is no intuitive way of implementing the negotiation by replacing all requests req by ask and replacing offer with tell. The reason is that the host is supposed to make the first proposal in the negotiation protocol presented and the invitee does not have the relevant information concerning meeting times which are acceptable to the host to make a first proposal.

First Attempt: The first and most naive attempt would be to try and implement the requests of the agents by

$$\mathsf{tell}(a, \mathit{meet}(\mathit{MeetId}, T, L, \mathit{Att}))$$

and the offers of the agents with

$$\mathsf{ask}(a, \mathit{epmeet}(\mathit{MeetId}, T1, L, \mathit{Att}, T))$$

where the agent name a is substituted appropriately by either the host name *Host* or the invitee name *Invitee*. The main problem with this solution is that the proposition $meet(MeetId, T, L, Att)$ which the agents communicate may be inconsistent with the belief base of the receiving agent! There is no consistency check included in the semantics of the **tell** and **ask** primitives. Moreover, the meeting time proposed by the host will be accepted immediately since $T1 = T$ always provides an answer to the question asked.

Second Attempt: The second attempt to implement the negotiation between host and invitee is to introduce a new predicate $proposal(MeetId, T, L, Att)$ which is not used for other purposes and implement the protocol by using this predicate. The basic idea of the protocol is that the receiving agent either accepts or proposes a counterproposal, so we introduce a new predicate

$$count\_prop(MeetId, T, L, Att)$$

which the receiving agent will use to derive a counterproposal given a proposal of the other agent. To be able to do this, some logical connection between these predicates must be provided. Intuitively, the conditions stating when a counterproposal can be derived from a proposal should be provided in the antecedent of a clause with consequent $count\_prop(MeetId, T, L, Att)$. For example, the following clause seems both intuitive and plausible:

$$\begin{array}{l} (proposal(MeetId, T, L, Att) \wedge \\ (\forall\, T1 \cdot T \leq T1 < T' \rightarrow \neg meet(MeetId, T1, L, Att))) \\ \qquad\qquad \rightarrow count\_prop(MeetId, T', L, Att) \end{array}$$

The main problem with this clause, however, is, as in the first attempt, that it is possible to prove that a proposal $proposal(MeetId, T, L, Att)$ is accepted right away. It is easy to derive the counterproposal $count\_prop(MeetId, T, L, Att)$.

A Solution: A solution can be given with the **tell** and **ask** primitives by rearranging a number of things. First, the integrity constraints should be removed from the belief bases of the agents. To be able to provide a solution which only uses the **tell** and **ask** primitives, in the rest of this section we consider the integrity constraints no longer to be part of the belief bases of the agents.

The integrity constraints, however, will show up in a slightly changed format as the questions asked (which corresponds to making an offer in the negotiation) by the agents to derive counterproposals. As in the second attempt above, the solution makes use of the predicates $proposal(MeetId, T, L, Att)$ and $count\_prop(MeetId, T', L, Att)$. All the requests in the previous section can then be replaced by

$$\mathsf{tell}(a, proposal(MeetId, T, L, Att))$$

where a is substituted appropriately by either the host name *Host* or the invitee name *Invitee*. The *proposal* and *count_prop* predicates are logically related as follows:

$$proposal(MeetId, T, L, Att) \wedge T' \geq T \rightarrow count\_prop(MeetId, T', L, Att)$$

Thus a counterproposal is a proposal which proposes a meeting time later than or equal to that of the proposal received. The implication codes part of the strategy that agents use to make counterproposals. These conditions for a counterproposal are obviously too weak, since a counterproposal may still violate some of the integrity constraints associated with meetings. To remedy this, the agents include all these constraints in their questions which replace the offers in the previous section:

$$\begin{array}{l}
\mathsf{ask}(proposal(MeetId, T, L, Att) \wedge \\
\quad count\_prop(MeetId, T', L, Att) \wedge \\
\quad (\forall\, T1 \cdot T \leq T1 < T' \rightarrow \neg meet(MeetId, T1, L, Att)) \wedge \\
\quad (\forall\, MeetId1, T1, L1, Att1 \cdot (meet(MeetId1, T1, L1, Att1) \wedge \\
\qquad\qquad T' \leq T1 < T' + L) \rightarrow Att1 \cap Att = \varnothing) \wedge \\
\quad 0:00 < L < 8:00 \wedge \\
\quad T' \geq 8:00 \wedge T' + L \leq 18:00 \\
)
\end{array}$$

As can easily be checked, the three latter conjuncts correspond to the integrity constraints (2), (3), and (4) introduced earlier. Also note that the constraint $0:00 < L < 8:00$ can be dropped. Although this solution seems to work, the main disadvantage is that the integrity constraints have to be explicitly provided everywhere in the plans of agents and a more general approach to deal with integrity constraints is sacrificed for a reduction of the number of communication primitives. The simplicity of the solution given in terms of req and offer is also lost. We believe this shows that the new communication primitives provide expressivity lacking in the tell and ask primitives and are therefore a new and powerfull extension to the agent language.

5.4 Multi-agent Meeting Scheduling

Meeting scheduling in the case of more than two agents is considerably harder than for two agents. The host of the meeting now has to contact all the invitees to the meeting, and has to find a common meeting time for all of these agents. It has to figure out such a meeting time from the offers for meeting times the invited agents make.

The plan for the host in the meeting negotiation in the previous section has to be changed in such a way that this plan can deal with any number of agents. We suppose the host has a *list of agents* which are the invitees for the meeting. The host has to negotiate with each of these agents about the meeting time. Each of these individual negotations can be viewed as a subgoal of the host's goal of negotiating a meeting time with the group of invitees. Accordingly, in the host agent, we create a number of subgoals corresponding to the number of invitees. For this purpose, we introduce the facility of posting a goal and semaphores to implement the posting operation first. Semaphores will also be used in the plans for negotiation for multi-agent scheduling.

Binary Semaphores in 3APL: A *semaphore* is a facility well-known from concurrent programming ([1]). One of the problems which arises in concurrent programming is how to manage different concurrent processes which access some shared object. The part of a concurrent program which accesses the shared object is called a *critical section*. Semaphores are used to guarantee mutual exclusion of critical sections. Interferences between critical sections which share some object is prevented in this way. A semaphore has two associated operators, the $\mathbf{P}(s)$ and $\mathbf{V}(s)$ operators. The $\mathbf{P}(s)$ operator decreases the semaphore s until some lower bound is reached. The $\mathbf{P}(s)$ acts like a guard and allows a process to enter its critical section only if the lower bound on the semaphore has not yet been reached. The $\mathbf{V}(s)$ semaphore increments the semaphore, thereby indicating that other processes may use the shared resources again. In particular, a binary semaphore allows that at most one process is executing its critical section at a time. We remark that binary semaphores can be used to implement arbitrary semaphores (cf. [1]).

In this paper, we only need *binary* semaphores. It is not to difficult to implement such semaphores given the synchronous communication primitives introduced earlier. A new agent sem_a is introduced for each agent a to keep track of the semaphores used by agent a. Such an agent is called a *semaphore agent for a* and has name sem_a. A semaphore agent sem_a has one goal corresponding to each binary semaphore s used by its corresponding agent a. Although the informal explanation of a semaphore above explains a semaphore as being raised and lowered, we do not implement the semaphore by a variable which is increased and decreased. Instead, we use a proposition symbol p and the tell and ask primitives to implement a binary semaphore p. The informal explanation is the more natural way to understand and think about semaphores, however.

The semaphore agent sem_a continually executes a goal $semaphore_p$. The following plan rule provides a plan which implements this goal and is incorporated in the rule base of the agent sem_a:

```
$semaphore_p \leftarrow \mathsf{true}$ |
    $\mathsf{tell}(a, p)$;
    $\mathsf{ask}(a, p)$;
    $semaphore_p$
```

The $\mathbf{P}(p)$ and $\mathbf{V}(p)$ operators which are used by agent a are defined by:

$$\mathbf{P}(p) \equiv \mathsf{ask}(sem_a, p)$$

$$\mathbf{V}(p) \equiv \mathsf{tell}(sem_a, p)$$

Creating Goals / Posting Goals in 3APL: In this paragraph, we show how an operator $\mathsf{post}(\pi)$ to post a new goal π can be implemented in 3APL by means of condition-action rules. These rules are tailor-made for creating new goals. First, we introduce a new transition rule which defines the intended semantics of the post construct. The posting of a goal at the *goal level* must be accounted for at the *agent level* since this goal must be added to the goal base of the agent. One way to do this is to store the information that a particular goal has been posted in a label associated with a goal level transition. For this purpose, a new label $\mathsf{post}(\pi)$ is introduced which indicates that a goal has been posted. At the agent level, this information is taken into account. As a consequence, we also need to change the semantics of the agent level slightly.

Definition 5.1 *(rule for* post*)*

$$\frac{}{\langle \mathsf{post}(\pi), \sigma\rangle_V \xrightarrow{\mathsf{post}(\pi)}_{\varnothing} \langle E, \sigma\rangle}$$

Definition 5.2 *(agent execution)*
Let $V = \mathsf{Free}(\pi_i)$ and let θ be a substitution.

$$\frac{\langle \pi_i, \sigma\rangle_V \xrightarrow{l}_\theta \langle \pi_i', \sigma'\rangle, l \neq \mathsf{post}(\pi)}{\langle \{\pi_0, \ldots, \pi_{i-1}, \pi_i, \pi_{i+1}, \ldots, \pi_n\}, \sigma\rangle \xrightarrow{l} \langle \{\pi_0, \ldots \pi_{i-1}, \pi_i', \pi_{i+1}, \ldots, \pi_n\}, \sigma'\rangle}$$

$$\frac{\langle \pi_i, \sigma\rangle_V \xrightarrow{\mathsf{post}(\pi)}_\theta \langle \pi_i', \sigma'\rangle}{\langle \{\pi_0, \ldots, \pi_{i-1}, \pi_i, \pi_{i+1}, \ldots, \pi_n\}, \sigma\rangle \xrightarrow{\tau} \langle \{\pi_0, \ldots \pi_{i-1}, \pi_i', \pi_{i+1}, \ldots, \pi_n\} \cup \{\pi\}, \sigma'\rangle}$$

Implementing post with condition-action rules The basic idea is to use condition-action rules and the ins and del basic actions for inserting and deleting a proposition to implement the post operator. The idea is to create for a $\mathsf{post}(\pi)$ goal a condition-action rule

(1) $\leftarrow \mathit{create}_\pi \mid \mathsf{del}(\mathit{create}_\pi);\ \pi$

and replace the $\mathsf{post}(\pi)$ goal with

(2) $\mathbf{P}(s_\pi);\ \mathsf{ins}(\mathit{create}_\pi);\ \neg \mathit{create}_\pi?;\ \mathbf{V}(s_\pi)$

Informally, the goal (2), which replaces the $\mathsf{post}(\pi)$ goal, inserts a predicate create_π in the belief base of the agent which triggers the condition of the condition-action rule (1). After inserting the trigger create_π in the belief base, the goal (2) waits until the trigger is removed again (this is the function of the test $\neg \mathit{create}_\pi?$). The condition-action rule will fire because of the fact that the trigger condition is inserted. The rule creates a goal $\mathsf{del}(\mathit{create}_\pi);\ \pi$, which consists of deleting the trigger condition and inserting the posted goal into the goalbase after that. The goal (2) is a critical section and a binary semaphore s_π is used to guarantee mutual exclusion. Only after the proposition create_π has been removed from the belief base again the semaphore can be decreased by the $\mathbf{V}$ operator. Although the condition-action rule may fire more than once before the trigger condition create_π has been removed, this does not give rise to multiple copies of the same goal because of the *set interpretation* of the goal base. That is, the goal base is a set of goals and so never contains more than one occurrence of a goal.

Remark 5.3 We make a few remarks about the update semantics of the action ins. We assume that the ins(φ) only succeeds if the proposition φ is a sentence, i.e. contains no free variables, and is consistent with the integrity constraints. If ins(φ) can be executed successfully, then the belief base of an agent is updated in such a way that φ is implied by the new belief base. The integrity constraints thus may never be violated by an action ins(ϕ), however, other beliefs in the belief base may be removed to be able to obtain a new belief base which implies ϕ and is consistent. We assume that the changes made to the belief base are *minimal* in a some suitable sense (cf. [8]). Moreover, in the sequel we assume that all predicates $P(t_1, \ldots, t_n)$ which are used have an associated integrity constraint

$$(\forall x_1, \ldots, x_n, y_1, \ldots, y_n \cdot (P(x_1, \ldots, x_n) \wedge P(y_1, \ldots, y_n)) \rightarrow x_1 = y_1 \wedge \ldots \wedge x_n = y_n)$$

This constraint implies that for ins($P(t_1, \ldots, t_n)$) to succeed a current belief about P has to be removed, and only after removing this belief $P(t_1, \ldots, t_n)$ can be inserted in the belief base.

Multi-agent Meeting Scheduling We will now design plans for multi-agent negotiation about meeting times. We use the **post** operator introduced in the previous paragraphs to create the subgoals of the host to negotiate about the meeting time with each of the invitees individually.

A number of predicates are used to separate the different rounds in the negotiation and to be able to conclude that an agreement has been reached. First, the predicate $count(List, N)$ returns the number N of items in the list $List$. The strategy that the agents use is the same as that used by the agents in the two-agent case. Each agent proposes the earliest time possible to schedule the meeting. Each round the host requests all invitees to respond to his proposal for a meeting time. A predicate $invitee(MeetId, N)$ is introduced to keep track of the number of invitees that have responded so far in a round to the proposal of the host. If the number is the same as the number of invitees, this indicates that the host may start a new round in the negotiation. A predicate $disagree(MeetId, N)$ is used to count the number of agents in a round which disagree with the proposal of the host. If this number is 0, an agreement has been reached, and the negotiation is finished. A predicate $bestproposal(MeetId, Time, Length)$ is used to store the best proposal evaluated by the host so far in the negotiation. The best proposal is that proposal which is the latest time proposed so far.

The plans below are used by the host to initiate and to control the negotiation process. The first plan to initiate the negotiation initialises the predicates discussed above, and creates subgoals to negotiate with each of the invitees individually. After initialising, the first round of the negotiation is started.

```
initiate_negotiation($MeetId, T, L, Att$) $\leftarrow$ true |
    $count(Att - Host, N)?$;
    ins($bestproposal(MeetId, 0, 0)$);
    ins($disagree(MeetId, N)$);
    ins($invitee(MeetId, 0)$);
    initiate_indiv_negotiation($Att - Host, MeetId, T, L, Att$);
    negotiation($MeetId, Att$)
```

The negotiation plan, the second plan, waits until all invitees have responded to the requests of the host (dealt with by the subgoals to negotiate with each invitee individually). It then checks whether an agreement has been reached (in that case $disagree(MeetId, 0)$ holds) and what the best proposal so far is. In case an agreement has been reached, negotiation is finished and the meeting can be scheduled. In the other case, a new round is started and the predicates *invitee* and *disagree* are initialised for this new round.

```
negotiation($MeetId, Att$) $\leftarrow$ true |
    $count(Att - Host, N)?$;
    $invitee(MeetId, N)?$;
    IF $disagree(MeetId, 0)$
    THEN begin
            $bestproposal(MeetId, T, L)?$;
            schedule_meeting($MeetId, T, L, Att$);
        end
    ELSE begin
            ins($disagree(MeetId, N)$);
            ins($invitee(MeetId, 0)$);
            negotiation($MeetId, Att$)
        end
```

The next two plans create the subgoals of the host to negotiate with each of the invitees individually. The **post** operator is used for this purpose. We use the expression $List - Item$ to denote the list obtained from removing *Item* from *List*. Below, the variable *Att* denotes a list of agents.

```
initiate_indiv_negotiation($Invitees, MeetId, T, L, Att$)
  $\leftarrow$ $Invitees = [Invitee, Rest]$ |
    post(invite($Invitee, MeetId, T, L, Att$));
    initiate_indiv_negotiation($Rest, MeetId, T, L, Att$)

initiate_indiv_negotiation($Invitees, MeetId, T, L, Att$) $\leftarrow$ $Invitees = []$ |
```

The subgoals created to negotiate with the invitees individually are `invite` goals similar to the two-agent case, but some changes have been made. In particular, the evaluation phase of the plan for `invite` goals in which the offers of the invitees are evaluated has to be revised in order to consider all the different offers which the invitees have made. The negotiation phase has not been changed. The simple strategy the host uses is to select the *latest* offer compatible with the host's constraints as the 'winning' offer. All the subgoals will then communicate this winning offer to the invitees and request to accept it as the new meeting time.

The subgoals of the host which negotiate with the invitees have to communicate with each other about the offers they receive from the invitees. The communication between the subgoals is achieved by means of the belief base and the predicate *bestproposal*. Each of the subgoals compares its received offer with the latest proposal for a meeting time stored in the belief base of the agent as $bestproposal(MeetId, T, L)$. It also raises a counter $invitee(MeetId, N)$, to indicate that it did receive an offer and is waiting for the next round of the negotiation. The evaluation phase is a critical section in the plans and mutual exclusion needs to be guaranteed between the different subgoals since in these sections of the plans the predicate *bestproposal* is updated. *Binary* semaphores are used to implement the mutual exclusion of the subgoals. The $\mathbf{P}(bp)$ operator increments the semaphore, and the $\mathbf{V}(bp)$ operator decreases the semaphore.

```
invite($Invitee, MeetId, T, L, Att$) $\leftarrow$ true |
    req($Invitee, \exists\, T1 \cdot epmeet(MeetId, T1, L, Att, T)$);
    offer($Invitee, meet(MeetId, T', L)$);
    $\mathbf{P}(bp)$;
        $bestproposal(MeetId, PropTime, L)?$;
        IF $PropTime < T'$ THEN ins($bestproposal(MeetId, T', L)$);
        $disagree(MeetId, M)?$;
        IF $T = T'$ THEN ins($disagree(MeetId, M-1)$);
        $invitee(MeetId, N)?$;
        ins($invitee(MeetId, N+1)$);
    $\mathbf{V}(bp)$;
    * wait until every agent has replied and a new round begins
    * invitee(MeetId, 0) set by negotiation plan to start a new round
    $invitee(MeetId, 0)?$;
    IF $disagree(MeetId, 0)$
        THEN tell($Invitee, confirm(MeetId, T')$)
        ELSE invite($Invitee, MeetId, T', L, Att$)
```

The plans for the `reply` goals of the invitees do not have to be changed. They are the same as in the two-agent case.

```
reply($Host$) ← true |
    begin
        offer($Host$, $meet(MeetId, T, L, Att)$);
        req($Host$, $\exists\, T1 \cdot epmeet(MeetId, T1, L, Att, T)$);
        reply($Host$)
    end+
    ask($Host$, $confirm(MeetId, T)$)
```

6 Approaches Based on Speech Act Theory

The main approaches to agent level communication in the literature like KQML ([11]) or ACL of FIPA ([7]) are based on speech act theory ([16]). Before we discuss these other approaches, we therefore first provide a short summary to introduce the most important concepts of speech act theory. One of the basic insights of speech act theory is that the conveying of a message not only consists in making a statement, but in a more broader sense constitutes a communicative action: making a promise, begging, informing, etc. are all different kinds of communicative actions.

In speech act theory, communicative actions are decomposed into locutionary, illocutionary and perlocutionary acts. A locutionary act refers to what the speaker says; the illocutionary act refers to what the speaker does by saying such-and-such; and the perlocutionary act refers to how the speaker affects the hearer by saying such-and-such. For example, a speaker may say that it is six o'clock (locutionary act); by saying so the speaker may be requesting the hearer to leave the shop (illocutionary act); and the effect on the hearer may be that (s)he actually leaves the shop (perlocutionary act).

By performing an illocutionary act, a speaker expresses mental attitudes: beliefs and desires. Accordingly, one can classify types of illocutionary acts in terms of expressed attitudes. Many classifications of speech acts have been proposed. The one we summarise here is based on that of Bach and Harnish ([2]). They distinguish four main classes of speech acts: (i) *constatives*, like assertives and predictives, which express belief, (ii) *directives*, like requestives and questions, which express a wish concerning some prospective action by the hearer, (iii) *commissives*, like promises and offers, which express an intention and belief that the speech act obligates the speaker to do something, and (iv) *acknowledgements*, like apologise and greet, which express feelings regarding the hearer.

Note that although a speech act is a means of expressing attitutes, the speaker does not actually have to have the expressed attitudes. As Bach and Harnish ([2], p. 39) write:

> To *express* an attitude in uttering something is [...] to (..)intend that the hearer take one's utterance as a reason to believe one has the attitude. The speaker need not have the attitude expressed, and the hearer need not form a corresponding attitude. The speaker's having the attitude expressed is the mark of sincerity. If the hearer forms a corresponding attitude that the speaker intended him to form, the speaker has achieved a perlocutionary effect in addition to illocutionary uptake.

In a very loose sense, we can classify our primitives as belonging to one of the classes outlined above. The tell primitive, for example, can be classified as a kind of constative. The req and ask can be classified as directives. And the offer can be classified as a commissive. Thus, three of the main classes are covered. The communication primitives, however, should not be taken to correspond to any speech act in particular. The tell primitive, for example, does not correspond to informing only, or suggesting only. The communication primitives can be used for many different purposes. The tell can be used both to assert and to predict; the offer primitive can both be used to promise and to make an offer. So, the informal meaning associated with the names of the communication primitives should not be confused with the formal semantics based on deduction and abduction. As was illustrated in section 5.3, the tell and ask primitives can even be used to make offers and requests, by coding a number of things in the content language.

No sincerity conditions concerning the actual mental state of the speaker have been included in the semantics of the communication primitives. For example, it is not required that the agent using tell actually believes what it tells. Also, no *perlocutionary effects* are incorporated in the semantics of the primitives. The way an agent deals with a particular message of another agent is not specified in the semantics of the communication primitives (and is not part of the speech act per se, cf. the quote from [2] above). For example, the agent receiving a proposition communicated by tell does not automatically update its belief base with this proposition.

Our approach is based on the view that agent level communication should be based on a well-defined and clear semantics and at the same time should be kept as simple as possible. Therefore, we take a somewhat different perspective as to how speech act theory could be of use in the design and development of agents. Our view is that speech act theory may provide a useful set of *abstract descriptions* of communicative actions for agents which can be used to *specify* and *verify* the behaviour of agents. Speech acts thus would play a role in the *proof theory* associated with the agent programming language. In the associated programming logic, the speech acts could be derived just like, for example, the mental attitude of belief would be derived from the program text and its semantics. Instead of *defining* a set of communication primitives corresponding to all kinds of subtle distinctions related to speech acts, we propose to *derive* what kind of speech act has been performed by an agent from the characteristics of the content proposition and the mental state of the agent. Alternatively, a formal specification in the logic may specify that an agent should perform a type of speech act. In that case it is up to the programmer to satisfy the specification [1]. We think speech act theory should not be viewed as a repository of all kinds of different communicative acts for which computational equivalents should be provided in an agent communication language, as is done in KQML or ACL.

The view outlined has a number of advantages. First of all, the agent communication language is kept simple. This prevents all kinds of confusions on the part of the program-

[1]For a similar remark cf. [11], where it is suggested that the propositional content conditions should be taken care of by the programmer; and indeed, it is hard to see how these conditions could be guaranteed in any other way in KQML-like languages. Cf. also the discussion later in this section.

mer as to what kind of communication primitive it should use in a particular case. Second, the idea to derive speech acts from the semantics of the agent (communication) language seems to provide for a more principled approach than the approach which is used to provide semantics for KQML or FIPA. It prevents unnecesary complications in the semantics and prevents the tendency to incorporate 'everything' into the semantics of speech acts. The different concerns which arise in the context of communicating agents are not clearly separated in the approaches mentioned, we think (although this is more true of KQML than of FIPA).

For example, in the paper [11] an attempt is made to incorporate various aspects of illocutionary acts *and* conversation policies into the semantics of speech acts. There are several disadvantages with such an approach. On the one hand, including conversation policies in the semantics provides a burden on the programmer, who has to deal with these policies. For example, we think that a requirement to reply in one of several ways to a communicative action of another agent should not be part of the semantics of a speech act. On the other hand, there is a methodological reason. If conversation policies are integrated into the semantics of speech acts, it is difficult to see where to draw the line. For example, is the Contract Net Protocol a conversation policy or is it a protocol which is not part of the semantics of speech acts? In our view, it is better to separate these issues as we have done in this paper.

As another illustration of the problems created by providing a semantics for computational equivalents of speech acts, consider the different conditions of propositional content associated with distinct speech acts. For example, a promise must make reference to the future, and cannot make reference to the past since this does not make sense. On the other hand, an explanation for a given fact should not make reference to the future but refer to the past. In this respect, a KQML or FIPA performative might become self-defeating and this is even more true if belief and goal modalities are allowed in the content language (cf. also [3]). For example, an agent might inform another agent that $\varphi \wedge \mathsf{B}\neg\varphi$ where B is a belief modality. Since a precondition on informing is that an agent believes what it informs another agent of, we have $\mathsf{B}(\varphi \wedge \mathsf{B}\neg\varphi)$ which together with $(\mathsf{BB}\neg\varphi) \rightarrow \mathsf{B}\neg\varphi$ implies a contradiction (in most belief logics). In our approach, we would not encounter such difficulties. For example, consider the action $\mathsf{offer}(do(a,t))$ where $do(a,t)$ is a proposition expressing that action a happens at time t. Depending on whether the time parameter t refers to the past respectively to the future we could *describe* $\mathsf{offer}(do(a,t))$ as an explanation or a promise (a similar point can be made for the sincerity conditions, i.e. the conditions on the mental state of the agent and other components of illocutionary acts).

Moreover, some of the speech acts provided in KQML seem to have arisen more from a computational concern than from a clear and formal perspective on speech acts. For example, the performative **stream-all** is used to ask an agent to provide all known answers to a question *in a particular format*. Another performative **ask-all** is introduced which provides for yet another format of response. A similar point can be made regarding the idea to introduce specific communication primitives to communicate with a so-called facilitator. A facilitator is a special agent which is introduced to manage particular aspects of the communication between agents, and a subset of the communication primitives supplied in

KQML is used only to deal with this facilitator. In our view, these features should not be included in the semantics of an agent communication language.

Finally, the specification of the semantics of the communication primitives in KQML and FIPA in a modal logic including operators for belief, intention, uncertainty, etc. poses a problem for integrating these communication languages into agent frameworks, i.e. agent programming languages or architectures. The problem is that the semantics is quite abstract and it is hard to see how to *ground* the modal semantics in an arbitrary agent framework. To put it in another way, how do we verify that an agent in a particular agent framework actually does have the required mental attitudes, etc. corresponding to the modal semantics of the operators? In general, this is a hard problem. The problem is sometimes referred to as the *gap between theory and practice*. Our work does not suffer from the same problems since it has been particularly aimed at bridging this gap (cf. [9]). The communication primitives we propose naturally fit into the agent language 3APL.

7 Conclusion

We illustrated and discussed the difference of the two sets of communication primitives. The semantics of tell and ask is based on deductive reasoning. These primitives are most naturally used for information exchange. The semantics of req and offer is based on abductive reasoning. These primitives are most naturally used to abduce feasible offers to satisfy requests. Both deduction and abduction are well-understood, and provide a clear and expressive semantics for communication primitives.

We illustrated the use of the new communication primitives by an extended example concerning meeting scheduling agents. We implemented a multi-stage negotiation protocol, both for the case where only two agents are involved in the negotiation and the case where an arbitrary number of agents are involved in the negotiation. The implementation is both natural and concise. The latter illustrates the expressive power of the communication primitives, while the former shows that the agent programming language 3APL is a natural means for programming personal assistants. Furthermore, we implemented binary semaphores by using the tell and ask primitives, and a new operator for posting goals. These were used in the implementation of a multi-stage negotiation protocol for an arbitrary number of agents.

The new communication primitives are integrated into the agent language 3APL. The semantics of the primitives fits in with the other primitives of the language naturally. This approach differs in numerous ways from other approaches based on designing a 'universal' communication language.

We did not try to provide computational equivalents of speech acts, but instead defined semantics for communication primitives which is both simple and intuitive. We argued that this approach is more natural and precludes a number of problems, while maintaining that speech act theory might be useful in the proof theory of the agent language. We believe speech act theory should not be seen as a repository of all kinds of different speech acts for which computational equivalents should be defined. Such an approach is unwieldy and too many subtle differences between computational primitives are introduced to be useful for programming agents.

References

[1] Gregory R. Andrews. *Concurrent Programming: Principles and Practice.* The Benjamin/Cummings Publishing Company, 1991.

[2] Kent Bach and Robert M. Harnish. *Linguistic Communication and Speech Acts.* The MIT Press, 1979.

[3] P.R. Cohen and H.J. Levesque. Communicative Actions for Artificial Agents. In *Proceedings of the International Conference on Multi-Agent Systems.* AAAI Press, 1995.

[4] Susan E. Conry, Robert A. Meyer, and Victor R. Lesser. Multistage Negotiation in Distributed Planning. In Alan H. Bond and Les Gasser, editors, *Readings in Distributed Artificial Intelligence*, pages 367–384. Morgan Kaufman, 1988.

[5] Luca Console, Danielle Theseider Dupre, and Pietro Torasso. On the relationship between abduction and deduction. *Journal of Logic and Computation*, 1(5):661–690, 1991.

[6] Luca Console and Pietro Torasso. A Spectrum of Logical Definitions of Model-Based Diagnosis. In W. Hamscher et al., editor, *Readings in Model-based Diagnosis*, pages 133–141. Morgan Kaufman, 1992.

[7] FIPA. FIPA 97 Specification, Part 2, Agent Communication Language, October 1998.

[8] P. Gärdenfors. *Knowledge in flux: Modelling the dynamics of epistemic states.* Bradford books, MIT, Cambridge, 1988.

[9] Koen V. Hindriks, Frank S. de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer. Formal Semantics for an Abstract Agent Programming Language. In Munindar P. Singh, Anand Rao, and Michael J. Wooldridge, editors, *Intelligent Agents IV (LNAI 1365)*, pages 215–229. Springer-Verlag, 1998.

[10] Koen V. Hindriks, Frank S. de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer. Control Structures of Rule-Based Agent Languages. In J. P. Müller, M. P. Singh, and A. S. Rao, editors, *Intelligent Agents V (LNAI 1555)*, pages 381–396. Springer-Verlag, 1999.

[11] Yannis Labrou and Tim Finin. A semantics approach for KQML - a general purpose communication language for software agents. In *Third International Conference on Information and Knowledge Management (CIKM'94).* 1994.

[12] J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1987.

[13] Robin Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[14] G. Plotkin. A structural approach to operational semantics. Technical report, Aarhus University, Computer Science Department, 1981.

[15] David Poole. Explanation and prediction: An architecture for default and abductive reasoning. *Computational Intelligence*, 5(1), 1989.

[16] John R. Searle. *Speech acts.* Cambridge University Press, 1969.

[17] Sandip Sen and Edmund Durfee. A Contracting Model for Flexible Distributed Scheduling. *Annals of Operations Research*, 65:195–222, 1996.

[18] Murray Shanahan. Prediction is Deduction but Explanation is Abduction. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'89)*, pages 1055–1060, 1989.

[19] Rogier M. van Eijk, Frank S. de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer. Systems of communicating agents. In Henri Prade, editor, *Proceedings of 13th biennial European Conference on Artificial Intelligence (ECAI'98)*, pages 293–297. John Wiley and Sons, 1998.

[20] Rogier M. van Eijk, Frank S. de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer. Information-Passing and Belief Revision in Multi-Agent Systems. In J. P. M. Müller, M. P. Singh, and A. S. Rao, editors, *Intelligent Agents V (LNAI 1555)*, pages 29–45. Springer-Verlag, 1999.