Specifying Communicative Multi-Agent Systems*

Steven Shapiro[1], Yves Lespérance[2], and Hector J. Levesque[1]

[1] Department of Computer Science
University of Toronto
Toronto, ON, Canada M5S 3G4
e-mail: {steven,hector}@ai.toronto.edu
[2] Department of Computer Science
Glendon College, York University
Toronto, ON, Canada M4N 3M6
e-mail: lesperan@yorku.ca

Abstract. In this paper, we describe a framework for specifying communicative multi-agent systems, using a theory of action based on the situation calculus to describe the effects of actions on the world and on the mental states of agents; and the concurrent, logic programming language ConGolog to specify the actions performed by each agent. Since ConGolog has a well-defined semantics, the specifications can be used to reason about the behavior of individual agents and the system as a whole. We extend the work presented in [7] to allow the specifications to mention agents' goals explicitly. The framework presented here allows the behavior of different agents to be specified at different levels of abstraction, using a rich set of programming language constructs. As an example, we specify a meeting scheduler multi-agent system.

1 Introduction

Many agent theorists (some examples are [1, 14, 19]) follow a similar methodology. They present a framework for representing the mental states of agents and the physical states of the world and axiomatize the relationships between the various components (beliefs, goals, intentions, etc.) of the agents' mental states. They then prove success theorems which state that agents will achieve their intentions under certain conditions (e.g., the agents are committed to their intentions, they possess the ability to achieve them, etc.). While this methodology can lead to some interesting theories, the facts that one can conclude from them are quite weak. One may be able to surmise that the agent will eventually achieve its intentions, but not when or by what means. It is not at all clear that these theories can be used to represent and reason about the actual behavior of agents in a detailed way, especially in the context of *complex* multi-agent systems with communicating agents.

On the other hand, there are formalisms for representing and reasoning about concurrent processes, e.g., [2, 5]. However, the representations of these formalisms are at a

* This research received financial support from the Information Technology Research Centre (Ontario, Canada), the Institute for Robotics and Intelligent Systems (Canada), and the Natural Sciences and Engineering Research Council (Canada). We thank Wayne Wobcke for helpful comments on an earlier version of this paper.

low level. They typically do not allow the specification of agents in terms of their mental states, nor do they explicitly represent actions in terms of their effects on the world.

In this paper, we explore a middle ground. We use the situation calculus [10] with Reiter's solution to the frame problem [15]—enhanced with predicates to describe agents' knowledge [18] and goals—to formally, perspicuously, and systematically describe the effects of actions on the world and the mental states of agents. We add INFORM and REQUESTUNLESS actions as primitive situation calculus actions to model inter-agent communication. We then use the notation of the concurrent, logic programming language ConGolog [4] to specify the behavior of agents. Since ConGolog has a well-defined semantics, the specifications can be used to reason about the behavior of individual agents and the system as a whole. Since ConGolog is based on the situation calculus, it is easy to add the capacity to specify the behavior of agents in terms of their mental states. In addition, ConGolog offers a wide variety of programming-language constructs (e.g., nondeterministic choice of actions, nondeterministic iteration, waiting for arbitrary first-order formulae to hold, concurrent execution with different priorities, and interrupts), which make it easy to specify a wide range of complex behaviors.

This paper extends the work presented in [7]. In that paper, the agents' mental states consisted only of knowledge. We have added the capacity to specify agents' goals. This addition allows us to dispense with the explicit representation of message queues for agents introduced in [7], yielding a cleaner account of communicative actions. We reformulate the meeting scheduler application described in [7] in this enhanced framework.

In the next section, we describe the situation calculus. In Sect. 3, we make time explicit in the situation calculus, and in Sect. 4, we add knowledge and goals to the situation calculus. In Sect. 5, we present ConGolog, and in Sect. 6, the meeting scheduler application is described. Finally, we conclude and discuss future work.

2 Situation Calculus

2.1 Theory of Action

Our action theory is based on an extended version of the situation calculus [15], a predicate calculus dialect for representing dynamically changing worlds. In this formalism, the world is taken to be in a certain situation. That situation can only change as a result of an agent performing an action. The term $do(a, s)$ represents the situation that results from the agent of action a executing a in situation s. For example, the formula $\mathrm{ON}(\mathrm{A}, \mathrm{B}, do(\mathrm{PUTON}(agt, \mathrm{A}, \mathrm{B}), s))$ could mean that A is on B in the situation resulting from *agt* putting A on B in s. Predicates and function symbols whose value may change from situation to situation (and whose last argument is a situation) are called *fluents*. There is also a special constant, S_0, used to denote the actual[1] initial situation.

An action is specified by first stating the conditions under which it can be performed by means of a *precondition axiom*. For example,[2]

$$Poss(\mathrm{PICKUP}(agt, x), s) \Leftrightarrow \forall z \neg \mathrm{HOLDING}(agt, z, s) \wedge \mathrm{NEXTTO}(agt, x, s)$$

[1] There can be alternative initial situations, see Sect. 4.

[2] We adopt the convention that unbound variables in a formula are universally quantified in the widest scope.

means that it is possible for *agt* to pick up an object x in situation s iff *agt* is not holding anything and it is standing next to x in s.

We adopt Reiter's solution to the frame problem [15],[3] which shows how one can transform axioms defining the effects of actions on fluents into a set of *successor state axioms*, one for each fluent, that imply the effect axioms as well as all the frame axioms. Successor state axioms have the following form, for each fluent, R:

$$\begin{array}{l} Poss(a,s) \Rightarrow \\ \quad [R(\boldsymbol{x}, do(a,s)) \Leftrightarrow \gamma_R^+(\boldsymbol{x},a,s) \vee (R(\boldsymbol{x},s) \wedge \neg\gamma_R^-(\boldsymbol{x},a,s))], \end{array}$$

where $\gamma_R^+(\boldsymbol{x},a,s)$ ($\gamma_R^-(\boldsymbol{x},a,s)$, respectively) denotes a formula which defines the conditions under which R changes truth value to becomes true (false, respectively) after a is performed in s. γ_R^+ and γ_R^- will usually contain conditions of the form: $a = t$, where t is term and a is the unbound action variable, since more than one action can change the truth value of a fluent (see for example the successor state axiom for ATMEETING in Sect. 6). This necessitates unique names axioms for actions, so that different action terms denote different actions. We assume that we have the following axioms:
for distinct action types A and A',

$$A(\boldsymbol{x}) \neq A'(\boldsymbol{y}),$$

and

$$A(x_1, \ldots, x_n) = A(y_1, \ldots, y_n) \Rightarrow x_1 = y_1 \wedge \ldots \wedge x_n = y_n.$$

2.2 Foundational Axioms

We need some domain-independent axioms to characterize the set of situations and to define an ordering over situations [16]. To characterize the set of situations, we start with a set of initial situations (including S_0) with no predecessors, i.e., we have the following axioms:

$$\begin{array}{l} \textsc{Init}(S_0), \\ \textsc{Init}(s) \Leftrightarrow \neg\exists a, s'(s = do(a,s')). \end{array}$$

We then define the set of situations to be the least set containing the initial situations and closed under the application of *do* to an action and a situation. We do not present this second-order axiom, but a suitable one is given in [6]. We also need a unique names axiom for situations:

$$do(a_1, s_1) = do(a_2, s_2) \Rightarrow a_1 = a_2 \wedge s_2 = s_2.$$

It will be useful to define an ordering on situations. $s < s'$ means that there is a possible sequence of actions that can be performed starting in situation s and which results in situation s'. We define $<$ using two axioms adapted from [16]:

$$\begin{array}{l} \textsc{Init}(s) \Rightarrow \forall s'(\neg s' < s), \\ s < do(a,s') \Leftrightarrow Poss(a,s') \wedge s \leq s', \end{array}$$

where $s \leq s' \stackrel{\text{def}}{=} s < s' \vee s = s'$.

[3] For treatments of the ramification and qualification problems compatible with our approach see [9, 11].

2.3 Formulae as Terms

We will need to have terms that represent formulae for two reasons. Firstly, we have actions (INFORM, REQUESTUNLESS) which take formulae as arguments. Secondly, we quantify over ConGolog programs when defining their semantics (although we do not give the semantics here). As we will see in Sect. 5, programs can contain first-order formulae, thus quantifying over programs can involve quantifying over formulae as well.

In [3], an encoding of formulae as first-order terms, called pseudo-formulae, is given, which we adopt and slightly extend to handle path formulae (see Sect. 3) and quantification over situations and paths. The encoding involves defining constants (pseudo-variables) denoting variables, and functions that correspond to situation calculus predicates and functions, and logical operators, as well as defining substitution for pseudo-variables, etc. Pseudo-formulae are related to actual formulae using the predicate HOLDS. We omit the details of the encoding and simply assume that we have the necessary axioms. However, to simplify the notation, we use actual formulae as arguments of actions and in programs. The reader should keep in mind that these formulae should really be pseudo-formulae.

3 Time

In our example application, we will need to model time: what has happened, what will happen, the actual time at a situation, and periods of time. In the situation calculus, the *do* function structures the situations as a tree (or a forest when there are several possible initial situations, as is the case here). Each situation has a linear past and a branching future. The past of a situation is well defined. It is simply the (unique) sequence of situations from the initial situation (the root of the tree) to the current situation. **Previously**(a, ϕ, s) is true if the action a has occurred in the past of s and ϕ holds after the occurrence of a:

$$\begin{aligned}&\textbf{Previously}(a, \phi, s) \stackrel{\text{def}}{=} \exists s' \, (do(a, s') \leq s \wedge \phi[do(a, s')]),\\ &\textbf{Previously}(a, s) \stackrel{\text{def}}{=} \textbf{Previously}(a, \textbf{True}, s).\end{aligned}$$

To model the future of a situation, we need a way to select one future among the many possible futures. We do this using *action selection functions* (ASFs), which are functions from situations to actions. Given a starting situation, an ASF defines an infinite path of situations. For each situation in the path, the ASF selects the action that is to be performed next to yield the next situation in the path. We define the predicate **OnPath** to mean that situation s' is in the situation sequence defined by ASF σ and situation s:

$$\textbf{OnPath}(\sigma, s, s') \stackrel{\text{def}}{=} s \leq s' \wedge \forall a, s^* (s < do(a, s^*) \leq s' \Rightarrow \sigma(s^*) = a).$$

We can use **OnPath** to define predicates that talk about the future. For example, we could define **Eventually**(ψ, σ, s) to mean that eventually ψ will hold along the path defined by σ starting at s:[4]

$$\textbf{Eventually}(\psi, \sigma, s) \stackrel{\text{def}}{=} \exists s^* (\textbf{OnPath}(\sigma, s, s^*) \wedge \psi[\sigma, s^*]).$$

[4] We use ψ to denote a formula whose fluents may contain two placeholders sit and asf instead of a situation argument and an ASF argument (respectively), e.g., **Next**$(a, \mathsf{asf}, \mathsf{sit})$, which means

We also introduce a functional fluent $\text{TIME}(s)$ which maps a situation into the current time at a situation. We assume that all actions in the domain increment TIME by $\text{DURATION}(a)$, the amount of time it takes to perform the action a.

Since meetings take place over periods of time, we also model periods of time. A *period* is a pair of times. The first and second elements of the pair are denoted by $\text{START}(\mathit{period})$ and $\text{END}(\mathit{period})$, respectively. We use the predicate $\mathbf{During}(\mathit{period}, \phi, \sigma, s)$ to stipulate that ϕ[5] holds throughout *period* on the path defined by σ and s:

$$
\begin{aligned}
&\mathbf{During}(\mathit{period}, \phi, \sigma, s) \stackrel{\text{def}}{=} \\
&\quad \exists s', s''(\mathbf{OnPath}(\sigma, \mathit{root}(s), s') \wedge \mathbf{OnPath}(\sigma, \mathit{root}(s), s'') \wedge \\
&\qquad\quad \text{TIME}(s') \leq \text{START}(\mathit{period}) < \text{TIME}(\mathit{do}(\sigma(s'), s')) \wedge \\
&\qquad\quad \text{TIME}(s'') < \text{END}(\mathit{period}) \leq \text{TIME}(\mathit{do}(\sigma(s''), s'')) \wedge \\
&\qquad\quad \forall s^*(s' \leq s^* \leq \mathit{do}(\sigma(s''), s'') \Rightarrow \phi[s^*])),
\end{aligned}
$$

where $\mathit{root}(s)$ is the initial situation, s_i, such that $s_i \leq s$. Since actions are of extended duration, the endpoints of a time period might fall between two adjacent situations on the path. Therefore, the definition stipulates that ϕ holds starting at the latest situation on the path (s') whose time is earlier than or equal to the start of the period until the earliest situation on the path ($\mathit{do}(\sigma(s''), s'')$) whose time is later than or equal to the end of period. Note that in this definition and elsewhere, we overload the $<$ and $\leq$ operators, using them to relate times as well as situations. For other approaches to adding time to the situation calculus, see [13, 17].

4 Mental Attitudes

4.1 Knowledge

Moore [12] showed how agents' knowledge could be modelled in the situation calculus by adapting the possible worlds model of knowledge to the situation calculus. We adopt his approach, using the notation of [18]. $K(\mathit{agt}, s', s)$ will be used to denote that in situation s, *agt* thinks that it could be in situation s'. We call s' a *K-alternative situation* for *agt* in s. $\mathbf{Know}(\mathit{agt}, \phi, s)$ is then defined to be $\forall s'(K(\mathit{agt}, s', s) \Rightarrow \phi[s'])$, and $\mathbf{KWhether}(\mathit{agt}, \phi, s)$ is defined as $\mathbf{Know}(agt, \phi, s) \vee \mathbf{Know}(agt, \neg\phi, s)$.

Scherl and Levesque [18] show how to obtain a successor state axiom for K that completely specifies how knowledge is affected by actions. In their framework, the knowledge-producing actions were performed by the agent itself, i.e., sensing actions. In this paper, we model communicative actions, which are actions that affect the mental state of an agent other than the one performing the action. However, Scherl and

that a is the next action to be performed. The placeholders get replaced by an ASF and a situation by an outer construct, in this case **Eventually**. $\psi[\sigma, s]$ is the formula that results from replacing asf with σ and sit with s. Where the intended meaning is clear, we suppress the placeholders, e.g., $\mathbf{Next}(a)$.

[5] We use ϕ to denote a formula that may contain the placeholder sit, but not asf. Again, we suppress the placeholder where possible. $\phi[s]$ is the formula that results from substituting s for sit in ϕ.

Levesque's successor state axiom for K is easily adapted to handle communicative actions. Here is the axiom we would use for a domain where the only knowledge-producing action is INFORM:[6]

$$
\begin{aligned}
&\mathit{Poss}(a, s) \Rightarrow \\
&\quad [K(\mathit{agt}, s'', \mathit{do}(a, s)) \Leftrightarrow \\
&\quad\; \exists s' (K(\mathit{agt}, s', s) \wedge s'' = \mathit{do}(a, s') \wedge \mathit{Poss}(a, s') \wedge \\
&\qquad\quad \forall \mathit{informer}, \phi (a = \text{INFORM}(\mathit{informer}, \mathit{agt}, \phi) \Rightarrow \phi[s']))]
\end{aligned}
$$

First note that for any action other than an INFORM action directed towards *agt*, the specification ensures that the only change in knowledge that occurs in moving from s to $\mathit{do}(a, s)$ is that it is known (by all agents) that the action a has been successfully performed. This is true because the K-alternative situations to $\mathit{do}(a, s)$ are the situations that result from doing a in a K-alternative situation to s where a is possible. Note that unlike [7], we assume that all actions are public, i.e., every agent is aware of the actions performed by all other agents. For the action $\text{INFORM}(\mathit{informer}, \mathit{agt}, \phi)$, the idea is that in moving from s to $\mathit{do}(\text{INFORM}(\mathit{informer}, \mathit{agt}, \phi), s)$, *agt* not only knows that the action has been performed (as above), but it also knows that ϕ. In this case, the K-alternative situations to $\mathit{do}(a, s)$ for *agt* are the situations that result from performing a in a K-alternative situation to s where a is possible, except the ones where ϕ is false.

Scherl and Levesque [18] showed that one could place constraints on the K-relation in the initial situation[7], and the constraints would continue to hold after any (possible) sequence of actions because the successor state axiom for K preserves these constraints. Since we are modelling knowledge (i.e., true beliefs), we constrain K to be reflexive.

4.2 Goals

We extend the framework described in [7] by modelling the goals of agents. The behavior of agents will be specified, in part, in terms of their own goals and what they know about the goals of other agents. Following Cohen and Levesque [1], we characterize the goals of an agent by specifying the paths in which all its goals (both maintenance goals and achievement goals) are achieved. Our approach differs slightly from Cohen and Levesque's in that our accessibility relation, $H(\mathit{agt}, \sigma, s', s)$, is used to state that in situation s, the path defined by σ and s' is consistent with what *agt* wants. We think of these paths as defining what the agent wants independently of what the agent knows. Cohen and Levesque restricted the G-accessible worlds (their version of H) to be a subset of the B-accessible worlds (their version of K). The end result is the same, however, as we define the goals of the agent to be the formulae true in the H-accessible paths that start in a K-accessible situation:

$$\mathbf{Goal}(\mathit{agt}, \psi, s) \stackrel{\text{def}}{=} \forall \sigma, s' (K(\mathit{agt}, s', s) \wedge H(\mathit{agt}, \sigma, s', s) \Rightarrow \psi[\sigma, s'])$$

[6] In our example system, we have no need for sensing actions, but they could easily be added if necessary.

[7] In particular, they show that K can be constrained to be reflexive, symmetric, transitive, and/or Euclidean.

In the meeting scheduler application, the personal agents will need to be able to revise their goals. Unfortunately, we do not have a general method to handle goal revision. Instead, we assume that requests are made using a conditional request action, which we call REQUESTUNLESS. When *requester* performs the REQUESTUNLESS(*requester*, *agt*, ψ, ϕ) action, it is requesting that *agt* adopt the goal that ψ, unless ϕ obtains. We call ψ the *conditional goal* of the request and ϕ the *condition* of the request. The idea is that once the request is made (and *agt* is willing to serve it), *agt* acquires the disjunctive goal that $\psi \vee \phi$. If *agt* comes to know that ϕ does not hold, then it will have the definite goal that ψ (because agents' goals are a superset of what they know). If instead *agt* comes to know that ϕ, then it can consider the conditional goal to have been cancelled since the disjunctive goal is already satisfied. Thus, after making a REQUESTUNLESS, *requester* can cancel the request by informing *agt* that ϕ holds. In the meeting scheduler application, an agent will not adopt $\neg\psi$ as a goal after receiving a request that ψ unless ϕ, therefore the agent will not adopt ϕ as a goal.

There is a potential problem if an agent gets a request for ψ_1 unless ϕ_1, followed by a request for ψ_2 unless ϕ_2 before it knows whether ϕ_1 holds, and where the first conditional goal ψ_1 is incompatible with the new conditional goal ψ_2. If the agent adopts both goals and then finds out that both $\neg\phi_1$ and $\neg\phi_2$, then its goals will be inconsistent. To avoid this problem, we prevent agents from accepting a request involving a conditional goal that may conflict with an existing conditional goal, such as when the agent does not know that the condition associated with the existing conditional goal holds. We use the predicate POSSIBLECONFLICT to express that there was a previous request that might conflict with a request for ψ:

$$
\begin{aligned}
&\textsc{PossibleConflict}(agt, \psi, s) \stackrel{\text{def}}{=} \\
&\quad \exists requester, \psi', \phi' (\mathbf{Previously}(\textsc{RequestUnless}(requester, agt, \psi', \phi'), s) \wedge \\
&\qquad\qquad\qquad \mathbf{Goal}(agt, \psi' \vee \phi', s) \wedge \mathbf{Goal}(agt, \psi' \Rightarrow \neg\psi, s) \wedge \\
&\qquad\qquad\qquad \neg\mathbf{Know}(agt, \phi', s))
\end{aligned}
$$

The successor state axiom for H has the same form as the one for K:

$$
\begin{aligned}
&Poss(a, s) \Rightarrow \\
&\quad [H(agt, \sigma, s'', do(a, s)) \Leftrightarrow \\
&\quad \exists s' (H(agt, \sigma, s', s) \wedge s'' = do(\sigma(s'), s') \wedge \\
&\qquad \forall requester, \psi, \phi (a = \textsc{RequestUnless}(requester, agt, \psi, \phi) \wedge \\
&\qquad\qquad\qquad \textsc{Serves}(agt, requester, \psi) \wedge \\
&\qquad\qquad\qquad \neg\textsc{PossibleConflict}(agt, \psi, s) \Rightarrow (\psi \vee \phi)[\sigma, s']))]
\end{aligned}
$$

If the action, a, is not a REQUESTUNLESS action directed towards *agt*, then the paths (represented by σ, s'') accessible at $do(a, s)$ are the same as the ones that were accessible at s (represented by σ, s'), but the current situation (s'') is advanced by one action along the path ($do(\sigma(s'), s')$). If a is a REQUESTUNLESS action directed towards *agt*, *agt* is willing to serve the request, and there is no possible conflict with a previous request, then only those paths that also satisfy $\psi \vee \phi$ are accessible in $do(a, s)$. The SERVES predicate is used to characterize which requesters and requests the agent is willing to serve. For simplicity, SERVES is not a fluent and we omit the axioms that define it. H is constrained to be initially non-empty, i.e., the agent's wishes are initially consistent.

5 ConGolog

We have just presented a framework in which one can systematically and perspicuously describe the effects of actions on the world and on the mental states of multiple, communicating agents. In order to describe a multi-agent system, we must also specify what actions the agents perform.

We specify the behavior of agents with the notation of the logic programming language ConGolog [4], the concurrent version of Golog [8]. While versions of both Golog and ConGolog have been implemented, we are mainly interested here in the potential for using ConGolog as a specification language. A ConGolog program[8] is composed of a sequence of procedure declarations, followed by a complex action. Complex actions are composed of various constructs:

a,	primitive action
$\phi?$,	wait for a condition
$\delta_1 ; \delta_2$,	sequence
$\delta_1 \mid \delta_2$,	nondeterministic choice between actions
δ^*,	nondeterministic iteration
if ϕ **then** δ_1 **else** δ_2,	conditional
for $x \in \Sigma$ **do** δ,	for loop
while ϕ **do** δ,	while loop
$\delta_1 \parallel \delta_2$,	concurrency with equal priority
$\delta_1 \rangle\!\rangle\, \delta_2$,	concurrency with δ_1 at a higher priority
$\langle\, \boldsymbol{x} : \phi \rightarrow \delta\, \rangle$,	interrupt
$\beta(\boldsymbol{p})$,	procedure call.

a denotes a situation calculus action, as described above. The ConGolog specification can be for a single agent or multiple agents, depending on whether the primitive actions contain an argument for the agent of the action. ϕ denotes a situation calculus formula with the situation argument of its fluents suppressed. δ, δ_1, and δ_2 stand for complex actions, Σ is a set, $\boldsymbol{x}$ is a set of variables, β is a procedure name, and $\boldsymbol{p}$ denotes the actual parameters to the procedure. These constructs are mostly self-explanatory. Intuitively, the interrupts work as follows. Whenever $\exists \boldsymbol{x}.\phi$ becomes true then δ is executed with the bindings of $\boldsymbol{x}$ that satisfied ϕ.

Procedures are defined with the following syntax: **proc** $\beta(\boldsymbol{y})\ \delta$, where β is the procedure name, $\boldsymbol{y}$ denotes the formal parameters to the procedure, and δ is a complex action. The semantics of ConGolog programs are defined using the *Do* predicate (see [4] for details). Informally, $Do(\rho, s, s')$ holds if situation s' is a legal terminating situation of program ρ starting in situation s.

6 Specification of a Meeting Scheduler Multi-Agent System

In our example system, we have meeting organizer agents, which are trying to schedule meetings with personal agents, which manage the schedules of their (human) owners. To

[8] We retain the term 'program' even though it is not our intention to execute the programs directly.

simplify the system, we assume that the personal agents have the authority to schedule meetings on behalf of their owners without consulting them. They simply inform their owners of a pending meeting fifteen minutes before it starts.

A significant feature of our approach is that different agents can be modelled at different levels of detail. The program for the personal agents mentions both their knowledge and their goals, whereas the program of the meeting organizer agents only mentions what they know. In other contexts, we might find it useful to model some agents without referring to their mental state at all.

In order to specify our multi-agent system, we must first axiomatize the actions and fluents of our domain. The actions are INFORM and REQUESTUNLESS which were previously described, and

- $\text{GOTOMEETING}(\textit{user}, \textit{chair})$: (the human) *user* goes to a meeting chaired by (the human) *chair*,
- $\text{LEAVEMEETING}(\textit{user}, \textit{chair})$: *user* leaves the meeting chaired by *chair*, and
- TICK: the time increases by one minute.

The fluents are K, H, and TIME, which were previously described, and

- $\text{ATMEETING}(\textit{user}, \textit{chair}, s)$: *user* is at a meeting chaired by *chair* in s.

In addition, we have two non-fluent functions: $\text{PAG}(\textit{user})$, whose value is the personal agent of *user*, and $\text{DURATION}(a)$, which maps an action to its duration. For simplicity, we omit the axioms that specify these functions.

Here are the axioms that specify the actions and fluents (other than the successor state axioms for K and H which were stated previously):

Precondition axioms:

$$\begin{aligned}
&\textit{Poss}(\text{INFORM}(\textit{informer}, \textit{agt}, \phi), s) \Leftrightarrow \mathbf{Know}(\textit{informer}, \phi, s)\\
&\textit{Poss}(\text{REQUESTUNLESS}(\textit{requester}, \textit{agt}, \psi, \phi), s) \Leftrightarrow \mathbf{Goal}(\textit{requester}, \psi \vee \phi, s)\\
&\textit{Poss}(\text{GOTOMEETING}(\textit{user}, \textit{chair}), s) \Leftrightarrow\\
&\qquad \neg\exists \textit{chair}' (\text{ATMEETING}(\textit{user}, \textit{chair}', s))\\
&\textit{Poss}(\text{LEAVEMEETING}(\textit{user}, \textit{chair}), s) \Leftrightarrow \text{ATMEETING}(\textit{user}, \textit{chair}, s)\\
&\textit{Poss}(\text{TICK}, s)
\end{aligned}$$

Successor state axioms:

$$\begin{aligned}
&\textit{Poss}(a, s) \Rightarrow\\
&\quad [\text{ATMEETING}(\textit{user}, \textit{chair}, \textit{do}(a, s)) \Leftrightarrow\\
&\qquad a = \text{GOTOMEETING}(\textit{user}, \textit{chair}) \vee\\
&\qquad (\text{ATMEETING}(\textit{user}, \textit{chair}, s) \wedge a \neq \text{LEAVEMEETING}(\textit{user}, \textit{chair}))]\\
&\textit{Poss}(a, s) \Rightarrow\\
&\quad [\text{TIME}(\textit{do}(a, s)) = t \Leftrightarrow \text{TIME}(s) = t' \wedge t = t' + \text{DURATION}(a)]
\end{aligned}$$

Depending on the properties to be proven about the system, it will often be necessary to stipulate some conditions about the initial state of the system. This is accomplished by adding axioms that specify what holds in the initial situation, S_0, for example:

$\textbf{proc}$ ORGANIZEMEETING(*oa*, *chair*, *Participants*, *period*)
  $\textbf{for}$ $p \in$ *Participants* $\textbf{do}$
    REQUESTUNLESS(*oa*, PAG(p), $\textbf{During}$(*period*, ATMEETING(p, *chair*)),
        SOMEONEDECLINED(*chair*, *chair*, *Participants*));
  $\textbf{KWhether}$(*oa*, SOMEONEDECLINED(*period*, *chair*, *Participants*))?;
  $\textbf{for}$ $p \in$ *Participants* $\textbf{do}$
    INFORMWHETHER(*oa*, PAG(p), SOMEONEDECLINED(*period*, *chair*, *Participants*));
  INFORMWHETHER(*oa*, *chair*, SOMEONEDECLINED(*period*, *chair*, *Participants*));
endProc,
where:

$$\text{SomeoneDeclined}(period, chair, Participants, s) \stackrel{\text{def}}{=} \exists p \in Participants(\text{DeclinedRequest}(\text{pag}(p), \mathbf{During}(period, \text{AtMeeting}(p, chair)), s)),$$

and where:

$$\text{DeclinedRequest}(agt, \psi, s) \stackrel{\text{def}}{=} \exists requester, \phi(\mathbf{Previously}(\text{requestUnless}(requester, agt, \psi, \phi), \neg\mathbf{Goal}(agt, \psi \vee \phi), s))$$

$\textbf{proc}$ INFORMWHETHER(*agt*, *agt'*, ϕ)
  $\textbf{Know}(agt, \phi)?;\ \text{inform}(agt, agt', \phi)\ |$
  $\textbf{Know}(agt, \neg\phi)?;\ \text{inform}(agt, agt', \neg\phi)\ |$
  $\neg\textbf{KWhether}(agt, \phi)?;\ \text{inform}(agt, agt', \neg\textbf{KWhether}(agt, \phi))$
endProc

Fig. 1. Procedure run by the meeting organizer agents.

Initial state axioms:

$$\mathbf{Know}(agt, \text{time} = 9{:}00 \wedge \neg\text{AtMeeting}(user, chair), S_0)$$
$$\text{pag}(user') = agt \Rightarrow \neg\mathbf{Goal}(agt, \mathbf{During}(period, \text{AtMeeting}(user, chair)), S_0)$$

We are now ready to use ConGolog to define the behavior of the agents. We start with the meeting organizer agents. The procedure that defines the behavior of these agents is given in Fig. 1. Their task is to organize a meeting for a given period and set of participants on behalf of the chair of the meeting. They do this by asking the personal agent of each participant to have their owner meet during the given period unless someone declines the meeting. Someone declines the meeting if following a conditional request to a participant's personal agent—such that the conditional goal of the request is that the participant be at the meeting—the agent has not adopted the corresponding goal, i.e., the disjunction of the conditional goal and the condition of the request. The participants' agents will then reply whether they have declined the meeting. In our example, a personal agent declines the meeting if it has a possible scheduling conflict. The meeting organizer waits until it knows whether one of the participants' agents declined the meeting. This will happen when either someone has declined the meeting or everyone

```
proc MANAGESCHEDULE(pa, user)
  ⟨period, chair :
    Goal(pa, During(period, ATMEETING(user, chair))) ∧
    Know[pa, ¬Previously(INFORM(pa, user,
                 Goal(pa, During(period, ATMEETING(user, chair))))) ∧
             START(period) − :15 ≤ TIME ≤ START(period)] →
                 INFORM(pa, user,
                       Goal(pa, During(period, ATMEETING(user, chair)))) ⟩
  ⟫
  ⟨oa, period, Participants, chair :
    Know[pa, Previously(REQUESTUNLESS(oa, pa,
                 During(period, ATMEETING(user, chair)),
                 SOMEONEDECLINED(period, chair, Participants))) ∧
             ¬Previously(INFORMWHETHER(pa, oa,
                   DECLINEDREQUEST(pa,
                       During(period, ATMEETING(user, chair)))))] →
                         INFORMWHETHER(pa, oa,
                           DECLINEDREQUEST(pa, During(period,
                                          ATMEETING(user, chair)))) ⟩
endProc
```

Fig. 2. Procedure run by the personal agents.

has agreed to it. Once the organizer knows the answer, it informs all the participants' agents and the chair whether someone has declined.

The procedure that specifies the behavior of the personal agents is given in Fig. 2. Its arguments are the personal agent and the user, i.e., the owner of the personal agent. The procedure is defined with two interrupts, the first running at higher priority than the second. The first interrupt fires when the agent has the goal that the user be at a meeting that starts in less than fifteen minutes, and the agent knows that it has not previously informed the user to go to the meeting. The action taken by the agent is to inform the user that it wants the user to go the meeting.

The second interrupt handles meeting requests; it fires when the agent knows that an organizer agent has requested a meeting, and it knows that it has not yet replied to the request. The action taken is to inform the organizer whether it declines the meeting request.

A complete meeting scheduler system is defined by composing instances of the two agent procedures in parallel, thereby modelling the behavior of several agents acting independently.[9] We also need to compose the nondeterministic iteration of the tick action in parallel (at a lower priority) with the calls to the agent procedures to allow time to pass when the agents are not acting. For example, let the program ρ consist of the three

[9] The model is only approximate, since the *Do* relation defines interleaved executions of parallel actions, whereas one would expect that in practice different agents will be running on different processors allowing them to act simultaneously.

procedure definitions given in Figs. 1 and 2, followed by:

$$
\begin{aligned}
&[\text{MANAGESCHEDULE}(\text{PA}_1, \text{USER}_1) \parallel \\
&\text{MANAGESCHEDULE}(\text{PA}_2, \text{USER}_2) \parallel \\
&\text{MANAGESCHEDULE}(\text{PA}_3, \text{USER}_3) \parallel \\
&\text{ORGANIZEMEETING}(\text{OA}_1, \text{USER}_1, \{\text{USER}_1, \text{USER}_3\}, \text{12:00–2:00}) \parallel \\
&\text{ORGANIZEMEETING}(\text{OA}_2, \text{USER}_2, \{\text{USER}_2, \text{USER}_3\}, \text{1:30–2:45})] \\
&\rangle\!\rangle\ \text{TICK}^*
\end{aligned}
$$

A sequence of actions $\boldsymbol{a}$ that satisfies:

$$Axioms \models Do(\rho, S_0, do(\boldsymbol{a}, S_0)),$$

where *Axioms* are the axioms mentioned above and suitable initial state axioms, will represent a possible evolution of the system.

In this example, the meeting organizers will both try to obtain USER_3's agreement for meetings that overlap; there will thus be two types of execution sequences, depending on who obtains this agreement.

7 Discussion and Future Work

We have stressed that we are using ConGolog programs as specifications of agent behavior that can be used to prove properties of multi-agent systems rather than as implementations of agents. However, since ConGolog is a programming language, we could simulate the behavior of a system by running its program. A ConGolog interpreter has been implemented. Unfortunately, the interpreter does not handle queries about goals or knowledge. However, if the interpreter had access to a first-order logic theorem prover (since our definitions of **Know** and **Goal** are first-order), the program defined in Sect. 6 could be run (though not very efficiently).

If we did run the program, we would have a simulation of the multi-agent system. What we really want, however, is a method of transforming the program into a set of programs, one for each agent, that could be executed on different processors to yield a true multi-agent system that implements the original specification. Since the program is composed of separate procedures for each agent, this should not be too hard to accomplish. However, once again, we would have to find a way of dealing with queries about the mental states of agents, or find a transformation that could produce single-agent programs that do not refer to mental states. It would be interesting to identify a large class of multi-agent ConGolog programs that could be decomposed into single-agent ConGolog programs, and show that properties that were proven about the multi-agent program ought still to hold for a system composed of the agents running the single-agent programs on separate processors.

The multi-agent meeting scheduler application described above is very simplistic. We would like to expand it by modelling more complex communicative interactions between the agents, and between the agents and their owners. In order to accomplish this, we would have to develop a more sophisticated theory of communicative interaction than the one presented in Sect. 4. Also, we want to develop a method for handling goal (and belief) revision in our framework.

References

1. Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.
2. Michael Fisher. Towards a semantics for Concurrent METATEM. In Michael Fisher and Richard Owens, editors, *Executable Modal and Temporal Logics*. Springer-Verlag: Heidelberg, Germany, 1995.
3. Giuseppe De Giacomo. Concurrency (notes). Unpublished manuscript, 1996.
4. Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. Reasoning about concurrent execution, prioritized interrupts, and exogenous actions in the situation calculus. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1221–1226, Yokohama, Japan, 1997.
5. C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall Int., 1985.
6. Gerhard Lakemeyer and Hector J. Levesque. AOL: a logic of acting, sensing, knowing, and only knowing. In *Proceedings of Knowledge Representation and Reasoning (KR&R-98)*, 1998. To appear.
7. Yves Lespérance, Hector J. Levesque, Fangzhen Lin, Daniel Marcu, Raymond Reiter, and Richard B. Scherl. Foundations of a logical approach to agent programming. In Michael Wooldridge, Jörg P. Müller, and Milind Tambe, editors, *Intelligent Agents II: Agent Theories, Architectures, and Languages (LNAI Volume 1037)*, pages 331–346. Springer-Verlag: Heidelberg, Germany, 1996.
8. Hector J. Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin, and Richard B. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31:59–84, 1997.
9. Fangzhen Lin and Raymond Reiter. State constraints revisited. *Journal of Logic and Computation*, 4(5):655–678, 1994.
10. John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In Bernard Meltzer and Donald Michie, editors, *Machine Intelligence 4*. Edinburgh University Press, 1969.
11. Sheila A. McIlraith. *Towards a Formal Account of Diagnostic Problem Solving*. PhD thesis, Department of Computer Science, University of Toronto, Toronto, ON, 1997.
12. Robert C. Moore. A formal theory of knowledge and action. In Jerry R. Hobbs and Robert C. Moore, editors, *Formal Theories of the Common Sense World*, pages 319–358. Ablex Publishing, Norwood, NJ, 1985.
13. Javier Pinto and Raymond Reiter. Adding a time line to the situation calculus. In *The Second Symposium on Logical Formalizations of Commonsense Reasoning*, pages 172–177, Nagoya, Japan, 1993.
14. Anand S. Rao and Michael P. Georgeff. Modeling rational agents within a BDI-architecture. In Richard Fikes and Eric Sandewall, editors, *Proceedings of Knowledge Representation and Reasoning (KR&R-91)*, pages 473–484. Morgan Kaufmann Publishers: San Mateo, CA, April 1991.
15. Raymond Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, San Diego, CA, 1991.
16. Raymond Reiter. Proving properties of states in the situation calculus. *Artificial Intelligence*, pages 337–351, December 1993.
17. Raymond Reiter. Natural actions, concurrency and continuous time in the situation calculus. In *Proceedings of Knowledge Representation and Reasoning (KR&R-96)*, pages 2–13, 1996.

18. Richard B. Scherl and Hector J. Levesque. The frame problem and knowledge-producing actions. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 689–695, Washington, DC, July 1993. AAAI Press/The MIT Press.
19. Munindar P. Singh. *Multiagent Systems: A Theoretical Framework for Intentions, Know-How, and Communications (LNAI Volume 799).* Springer-Verlag: Heidelberg, Germany, 1994.