Reasoning about Multi-agent Domains Using Action Language $\mathcal{C}$: A Preliminary Study

Chitta Baral[1], Tran Cao Son[2], and Enrico Pontelli[2]

[1] Dept. Computer Science & Engineering, Arizona State University
chitta@asu.edu
[2] Dept. Computer Science, New Mexico State University
{tson,epontell}@cs.nmsu.edu

Abstract. This paper investigates the use of action languages, originally developed for representing and reasoning about single-agent domains, in modeling multi-agent domains. We use the action language $\mathcal{C}$ and show that minimal extensions are sufficient to capture several multi-agent domains from the literature. The paper also exposes some limitations of action languages in modeling a specific set of features in multi-agent domains.

1 Introduction and Motivation

Representing and reasoning in multi-agent domains are two of the most active research areas in *multi-agent system (MAS)* research. The literature in this area is extensive, and it provides a plethora of logics for representing and reasoning about various aspects of MAS domains. For example, the authors of [24] combine an action logic and a cooperation logic to represent and reason about the capabilities and the forms of cooperation between agents. The work in [16] generalizes this framework to consider domains where an agent may control only parts of propositions and to reason about strategies of agents. In [31], an extension of Alternating-time Temporal Logic is developed to facilitate strategic reasoning in multi-agent domains. The work in [30] suggests that decentralized partially observable Markov decision processes could be used to represent multi-agent domains, and discusses the usefulness of agent communication in multi-agent planning. In [18], an extension of Alternating-time Temporal Epistemic Logic is proposed for reasoning about choices. Several other works (e.g., [12,32]) discuss the problem of reasoning about knowledge in MAS.

Even though a large number of logics have been proposed in the literature for formalizing MAS, several of them have been designed to specifically focus on particular aspects of the problem of modeling MAS, often justified by a specific application scenario. This makes them suitable to address specific subsets of the general features required to model real-world MAS domains. Several of these logics are quite complex and require modelers that are transitioning from work on single agents to adopt a very different modeling perspective.

The task of generalizing some of these existing proposals to create a uniform and comprehensive framework for modeling different aspects of MAS domains is, to the best of our knowledge, still an open problem. Although we do not dispute the possibility

J. Dix, M. Fisher, and P. Novák (Eds.): CLIMA X, LNAI 6214, pp. 46–63, 2010.
© Springer-Verlag Berlin Heidelberg 2010

of extending these existing proposals in various directions, the task does not seem easy. On the other hand, the need for a general language for MAS domains, with a formal and simple semantics that allows the verification of plan correctness, has been extensively motivated (e.g., [8]).

The state of affairs in formalizing multi-agent systems reflects the same trend that occurred in the early nineties, regarding the formalization of *single agent* domains. Since the discovery of the frame problem [22], several formalisms for representing and reasoning about dynamic domains have been proposed. Often, the new formalisms responded to the need to address shortcomings of the previously proposed formalisms within specific sample domains. For example, the well-known Yale Shooting problem [17] was invented to show that the earlier solutions to the frame problem were not satisfactory. A simple solution to the Yale Shooting problem, proposed by [2], was then shown not to work well with the Stolen Car example [20], etc. Action languages [15] have been one of the outcomes of this development, and they have been proved to be very useful ever since.

Action description languages, first introduced in [14] and further refined in [15], are formal models used to describe dynamic domains, by focusing on the representation of effects of actions. Traditional action languages (e.g., $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$) have mostly focused on domains involving a single agent. In spite of different features and several differences between these action languages (e.g., concurrent actions, sensing actions, nondeterministic behavior), there is a general consensus on what are the essential components of an action description language in single agent domains. In particular, an action specification focuses on the *direct effects* of each action on the state of the world; the semantics of the language takes care of all the other aspects concerning the evolution of the world (e.g., the ramification problem).

The analogy between the development of several formalisms for single agent domains and the development of several logics for formalizing multi-agent systems indicates the need for, and the usefulness of, a formalism capable of dealing with multiple desired features in multi-agent systems. A natural question that arises is whether single agent action languages can be adapted to describe MAS. *This is the main question that we explore in this paper.*

In this paper, we attempt to answer the above question by investigating whether an action language developed for single agent domains can be used, with minimal modifications, to model interesting MAS domains. Our starting point is a well-studied and well-understood single agent action language—the language $\mathcal{C}$ [15]. We chose this language because it already provides a number of features that are necessary to handle multi-agent domains, such as concurrent interacting actions. The language is used to formalize a number of examples drawn from the multi-agent literature, describing different types of problems that can arise when dealing with multiple agents. Whenever necessary, we identify weaknesses of $\mathcal{C}$ and introduce simple extensions that are adequate to model these domains. The resulting action language provides a unifying framework for modeling several features of multi-agent domains. The language can be used as a foundation for different forms of reasoning in multi-agent domains (e.g., projection, validation of plans), which are formalized in the form of a query language. We expect that further development in this language will be needed to capture additional aspects such as agents' knowledge about other agents' knowledge. We will discuss them in the future.

We would like to note that, in the past, there have been other attempts to use action description languages to formalize multi-agent domains, e.g., [6]. On the other hand, the existing proposals address only some of the properties of the multi-agent scenarios that we deem to be relevant (e.g., focus only on concurrency).

Before we continue, let us discuss the desired features and the assumptions that we place on the target multi-agent systems. In this paper, we consider MAS domains as environments in which multiple agents can execute actions to modify the overall state of the world. We assume that

- Agents can execute actions concurrently;
- Each agent knows its own capabilities—but they may be unaware of the global effect of their actions;
- Actions executed by different agents can interact;
- Agents can communicate to exchange knowledge; and
- Knowledge can be private to an agent or shared among groups of agents.

The questions that we are interested in answering in a MAS domain involve

- *hypothetical reasoning*, e.g., what happens if agent A executes the action a; what happens if agent A executes a_1 while B executes b_1 at the same time; etc.
- *planning/capability*, e.g., can a specified group of agents achieve a certain goal from a given state of the world.

Variations of the above types of questions will also be considered. For example, what happens if the agents do not have complete information, if the agents do not cooperate, if the agents have preferences, etc.

To the best of our knowledge, this is the first investigation of how to adapt a single agent action language to meet the needs of MAS domains. It is also important to stress that the goal of this work is to create a framework for *modeling* MAS domains, with a query language that enables plan validation and various forms of reasoning. In this work, we do not deal with the issues of distributed plan generation—an aspect extensively explored in the literature. This is certainly an important research topic and worth pursuing but it is outside of the scope of this paper. We consider the work presented in this paper a necessary precondition to the exploration of distributed MAS solutions.

The paper is organized as follows. Section 2 reviews the basics of the action language $\mathcal{C}$. Section 3 describes a straightforward adaptation of $\mathcal{C}$ for MAS. The following sections (Sects. 4–5) show how minor additions to $\mathcal{C}$ can address several necessary features in representation and reasoning about MAS domains. Sect. 6 presents a query language that can be used with the extended $\mathcal{C}$. Sect. 7 discusses further aspects of MAS that the proposed extension of $\mathcal{C}$ cannot easily deal with. Sect. 8 presents the discussion and some conclusions.

2 Action Language $\mathcal{C}$

The starting point of our investigation is the action language $\mathcal{C}$ [15]—an action description language originally developed to describe single agent domains, where the agent is capable of performing non-deterministic and concurrent actions. Let us review a slight adaptation of the language $\mathcal{C}$.

A domain description in $\mathcal{C}$ builds on a language signature $\langle \mathcal{F}, \mathcal{A} \rangle$, where $\mathcal{F} \cap \mathcal{A} = \emptyset$ and $\mathcal{F}$ (resp. $\mathcal{A}$) is a finite collection of fluent (resp. action) names. Both the elements of $\mathcal{F}$ and $\mathcal{A}$ are viewed as propositional variables, and they can be used in formulae constructed using the traditional propositional operators. A propositional formula over $\mathcal{F} \cup \mathcal{A}$ is referred to simply as a *formula*, while a propositional formula over $\mathcal{F}$ is referred to as a *state formula*. A fluent literal is of the form f or $\neg f$ for any $f \in \mathcal{F}$.

A domain description D in $\mathcal{C}$ is a finite collection of axioms of the following forms:

caused ℓ **if** F	*(static causal law)*
caused ℓ **if** F **after** G	*(dynamic causa laws)*

where ℓ is a fluent literal, F is a state formula, while G is a formula. The language also allows the ability to declare properties of fluents; in particular **non_inertial** ℓ declares that the fluent literal ℓ is to be treated as a non-inertial literal, i.e., the frame axiom is not applicable to ℓ.

A problem specification is obtained by adding an initial state description $\mathcal{I}$ to a domain D, composed of axioms of the form **initially** ℓ, where ℓ is a fluent literal.

The semantics of the language can be summarized using the following concepts. An *interpretation* I is a set of fluent literals, such that $\{f, \neg f\} \not\subseteq I$ for every $f \in \mathcal{F}$. Given an interpretation I and a fluent literal ℓ, we say that I satisfies ℓ, denoted by $I \models \ell$, if $\ell \in I$. The entailment relation $\models$ is extended to define the entailment $I \models F$ where F is a state formula in the usual way. An interpretation I is *complete* if, for each $f \in \mathcal{F}$, we have that $f \in I$ or $\neg f \in I$. An interpretation I is *closed* w.r.t. a set of static causal laws $\mathcal{SC}$ if, for each static causal law **caused** ℓ **if** F, if $I \models F$ then $\ell \in I$. Given an interpretation I and a set of static causal laws $\mathcal{SC}$, we denote with $Cl_{\mathcal{SC}}(I)$ the smallest set of literals that contains I and that is closed w.r.t. $\mathcal{SC}$. Given a domain description D, a *state* s in D is a complete interpretation which is closed w.r.t. the set of static causal laws in D.

The notions of interpretation and entailment over the language of $\mathcal{F} \cup \mathcal{A}$ are defined in a similar way.

Given a state s, a set of actions $A \subseteq \mathcal{A}$, and a collection of dynamic causal laws $\mathcal{DC}$, we define

$$\mathit{Eff}_{\mathcal{DC}}(s, A) = \left\{ \ell \mid (\textbf{ caused } \ell \textbf{ if } F \textbf{ after } G) \in \mathcal{DC}, s \dot{\cup} A \models G, s \models F \right\}$$

where $s \dot{\cup} A$ stands for $s \cup A \cup \{\neg a \mid a \in \mathcal{A} \setminus A\}$.

Let $D = \langle \mathcal{SC}, \mathcal{DC}, \mathcal{IN} \rangle$ be a domain, where $\mathcal{SC}$ are the static causal laws, $\mathcal{DC}$ are the dynamic causal laws and $\mathcal{IN}$ are the non-inertial axioms. The semantics of D is given by a transition system $(State_D, E_D)$, where $State_D$ is the set of all states and the transitions in E_D are of the form $\langle s, A, s' \rangle$, where s, s' are states, $A \subseteq \mathcal{A}$, and s' satisfies the property

$$s' = Cl_{\mathcal{SC}}(\mathit{Eff}_{\mathcal{DC}}(s, A) \cup ((s \setminus IFL) \cap s') \cup (\mathcal{IN} \cap s'))$$

where $IFL = \{f, \neg f \mid f \in \mathcal{IN} \text{ or } \neg f \in \mathcal{IN}\}$.

The original $\mathcal{C}$ language supports a query language (called $\mathcal{P}$ in [15]). This language allows queries of the form **necessarily** F **after** $A_1, \ldots, A_k$, where F is a state formula

and $A_1, \ldots, A_k$ is a sequence of sets of actions (called a *plan*). Intuitively, the query asks whether each state s reached after executing $A_1, \ldots, A_k$ from the initial state has the property $s \models F$.

Formally, an initial state s_0 w.r.t. an initial state description $\mathcal{I}$ and a domain D is an element of $State_D$ such that $\{\ell \mid \textbf{initially } \ell \in \mathcal{I}\} \subseteq s_0$. The transition function $\Phi_D : 2^{\mathcal{A}} \times State_D \rightarrow 2^{State_D}$ is defined as $\Phi_D(A, s) = \{s' \mid \langle s, A, s' \rangle \in E_D\}$, where $(State_D, E_D)$ is the transition system describing the semantics of D. This function can be extended to define Φ^*_D, which considers plans, where $\Phi^*_D([\,], s) = \{s\}$ and

$$\Phi^*_D([A_1, \ldots, A_n], s) = \begin{cases} \emptyset & \text{if } \Phi^*_D([A_1, \ldots, A_{n-1}], s) = \emptyset \vee \\ & \quad \exists s' \in \Phi^*_D([A_1, \ldots, A_{n-1}], s).[\Phi_D(A_n, s') = \emptyset] \\ \bigcup_{s' \in \Phi^*_D([A_1, \ldots, A_{n-1}], s)} \Phi_D(A_n, s') & \text{otherwise} \end{cases}$$

Let us consider an action domain D and an initial state description $\mathcal{I}$. A query **necessarily** F **after** $A_1, \ldots, A_k$ is *entailed* by $(D, \mathcal{I})$, denoted by

$$(D, \mathcal{I}) \models \textbf{necessarily } F \textbf{ after } A_1, \ldots, A_k$$

if for every s_0 initial state w.r.t. $\mathcal{I}$, we have that $\Phi^*_D([A_1, \ldots, A_k], s_0) \neq \emptyset$, and for each $s \in \Phi^*_D([A_1, \ldots, A_k], s_0)$ we have that $s \models F$.

3 $\mathcal{C}$ for Multi-agent Domains

In this section, we explore how far one of the most popular action languages developed for single agent domains, $\mathcal{C}$, can be used and adapted for multi-agent domains. We will discuss a number of incremental small modifications of $\mathcal{C}$ necessary to enable modeling MAS domains. We expect that similar modifications can be applied to other single-agent action languages with similar basic characteristics. We will describe each domain from the perspective of someone (the modeler) who has knowledge of everything, including the capabilities and knowledge of each agent. Note that this is *only a modeling perspective*—it does not mean that we expect individual agents to have knowledge of everything, we only expect the *modeler* to have such knowledge.

We associate to each agent an element of a set of *agent identifiers*, $\mathcal{AG}$. We will describe a MAS domain over a set of signatures $\langle \mathcal{F}_i, \mathcal{A}_i \rangle$ for each $i \in \mathcal{AG}$, with the assumption that $\mathcal{A}_i \cap \mathcal{A}_j = \emptyset$ for $i \neq j$. Observe that $\bigcap_{i \in S} \mathcal{F}_i$ may be not empty for some $S \subseteq \mathcal{AG}$. This represents the fact that fluents in $\bigcap_{i \in S} \mathcal{F}_i$ are relevant to all the agents in S.

The result is a $\mathcal{C}$ domain over the signature $\langle \bigcup_{i=1}^n \mathcal{F}_i, \bigcup_{i=1}^n \mathcal{A}_i \rangle$. We will require the following condition to be met: if **caused** ℓ **if** F **after** G is a dynamic law and $a \in \mathcal{A}_i$ appears in G, then the literal ℓ belongs to $\mathcal{F}_i$. This condition summarizes the fact that agents are aware of the direct effects of their actions. Observe that on the other hand, an agent might not know all the consequences of his own actions. For example, a deaf agent bumping into a wall might not be aware of the fact that his action causes noise observable by other agents. These global effects are captured by the modeler, through the use of static causal laws.

The next two sub-sections illustrate applications of the language in modeling cooperative multi-agent systems. In particular, we demonstrate how the language is already sufficiently expressive to model simple forms of cooperation between agents even though these application scenarios were not part of the original design of $\mathcal{C}$.

3.1 The Prison Domain

This domain has been originally presented in [24]. In this example, we have two prison guards, 1 and 2, who control two gates, the inner gate and the outer gate, by operating four buttons a_1, b_1, a_2, and b_2. Agent 1 controls a_1 and b_1, while agent 2 controls a_2 and b_2. If either a_1 or a_2 is pressed, then the state of the inner gate is toggled. The outer gate, on the other hand, toggles only if both b_1 and b_2 are pressed.

The problem is introduced to motivate the design of a logic for reasoning about the ability of agents to cooperate. Observe that neither of the agents can individually change the state of the outer gate. On the other hand, individual agents' actions can affect the state of the inner gate.

In $\mathcal{C}$, this domain can be represented as follows. The set of agents is $\mathcal{AG} = \{1, 2\}$. For agent 1, we have:

$$\mathcal{F}_1 = \{in\_open, out\_open, pressed(a_1), pressed(b_1)\}.$$

Here, in_open and out_open represent the fact that the inner gate and outer gate are open respectively. $pressed(X)$ says that the button X is pressed where $X \in \{a_1, b_1\}$. We have $\mathcal{A}_1 = \{push(a_1), push(b_1)\}$. This indicates that guard 1 can push buttons a_1 and b_1. Similarly, for agent 2, we have that

$$\mathcal{F}_2 = \{in\_open, out\_open, pressed(a_2), pressed(b_2)\} \quad \mathcal{A}_2 = \{push(a_2), push(b_2)\}$$

We assume that the buttons do not stay pressed—thus, $pressed(X)$, for $X \in \{a_1, b_1, a_2, b_2\}$, is a non-inertial fluent with the default value *false*.
The domain specification (D_{prison}) contains:

non_inertial $\neg pressed(X)$
caused $pressed(X)$ **after** $push(X)$
caused in_open **if** $pressed(a_1), \neg in\_open$
caused in_open **if** $pressed(a_2), \neg in\_open$
caused $\neg in\_open$ **if** $pressed(a_1), in\_open$
caused $\neg in\_open$ **if** $pressed(a_2), in\_open$
caused out_open **if** $pressed(b_1), pressed(b_2), \neg out\_open$
caused $\neg out\_open$ **if** $pressed(b_1), pressed(b_2), out\_open$

where $X \in \{a_1, b_1, a_2, b_2\}$. The first statement declares that $pressed(X)$ is non-inertial and has *false* as its default value. The second statement describes the effect of the action $push(X)$. The remaining laws are static causal laws describing relationships between properties of the environment.

The dynamic causal laws are "local" to each agent, i.e., they involve fluents that are local to that particular agent; in particular, one can observe that each agent can achieve

certain effects (e.g., opening/closing the inner gate) disregarding what the other agent is doing (just as if it was operating as a single agent in the environment). On the other hand, if we focus on a single agent in the domain (e.g., agent 1), then such agent will possibly see exogenous events (e.g., the value of the fluent in_open being changed by the other agent). On the other hand, the collective effects of actions performed by different agents are captured through "global" static causal laws. These are laws that the modeler introduces and they do not "belong" to any specific agent.

Let us now consider the queries that were asked in [24] and see how they can be answered by using the domain specification D_{prison}. In the first situation, both gates are closed, 1 presses a_1 and b_1, and 2 presses b_2. The question is whether the gates are open or not after the execution of these actions.

The initial situation is specified by the initial state description $\mathcal{I}_1$ containing

$$\mathcal{I}_1 = \{ \textbf{ initially } \neg in\_open, \quad \textbf{initially } \neg out\_open \}$$

In this situation, there is only one initial state $s_0 = \{\neg \ell \mid \ell \in \mathcal{F}_1 \cup \mathcal{F}_2\}$. We can show that

$$(D_{prison}, \mathcal{I}_1) \models \textbf{ necessarily } out\_open \wedge in\_open \textbf{ after } \{push(a_1), push(b_1), push(b_2)\}$$

If the outer gate is initially closed, i.e., $\mathcal{I}_2 = \{ \textbf{ initially } \neg out\_open\}$, then the set of actions $A = \{push(b_1), push(b_2)\}$ is both necessary and sufficient to open it:

$$(D_{prison}, \mathcal{I}_2) \models \textbf{ necessarily } out\_open \textbf{ after } X$$
$$(D_{prison}, \mathcal{I}_2) \models \textbf{ necessarily } \neg out\_open \textbf{ after } Y$$

where $A \subseteq X$ and $A \setminus Y \neq \emptyset$. Observe that the above entailment correspond to the environment logic entailment in [24].

3.2 The Credit Rating Domain

We will next consider an example from [16]; in this example, we have a property of the world that cannot be changed by a single agent. The example has been designed to motivate the use of logic of propositional control to model situations where different agents have different levels of control over fluents.

We have two agents, $\mathcal{AG} = \{w, t\}$, denoting the website and the telephone operator, respectively. Both agents can set/reset the credit rating of a customer. The credit rating can only be set to be ok (i.e., the fluent $credit\_ok$ set to *true*) if both agents agree. Whether the customer is a web customer (is_web fluent) or not can be set only by the website agent w. The signatures of the two agents are as follows:

$$\mathcal{F}_w = \{is\_web, credit\_ok\} \qquad \mathcal{A}_w = \left\{ \begin{array}{l} set\_web, reset\_web, \\ set\_credit(w), reset\_credit(w) \end{array} \right\}$$
$$\mathcal{F}_t = \{credit\_ok\} \qquad \mathcal{A}_t = \{set\_credit(t), reset\_credit(t)\}$$

The domain specification D_{bank} consists of:

caused is_web **after** set_web
caused $\neg is\_web$ **after** $reset\_web$
caused $\neg credit\_ok$ **after** $reset\_credit(w)$
caused $\neg credit\_ok$ **after** $reset\_credit(t)$
caused $credit\_ok$ **after** $set\_credit(w) \wedge set\_credit(t)$

We can show that

$$(D_{bank}, \mathcal{I}_3) \models \textbf{ necessarily } credit\_ok \textbf{ after } \{set\_credit(w), set\_credit(t)\}$$

where $\mathcal{I}_3 = \{$ **initially** $\neg\ell \mid \ell \in \mathcal{F}_w \cup \mathcal{F}_t\}$. This entailment also holds if $\mathcal{I}_3 = \emptyset$.

4 Adding Priority between Actions

The previous examples show that $\mathcal{C}$ is sufficiently expressive to model the basic aspects of agents executing cooperative actions within a MAS, focusing on capabilities of the agents and action interactions. This is not a big surprise, as discussed in [6]. We will now present a small extension of $\mathcal{C}$ that allows for the encodings of competitive behavior between agents, i.e., situations where actions of some agents can defeat the effects of other agents.

To make this possible, for each domain specification D, we assume the presence of a function $Pr_D : 2^{\mathcal{A}} \rightarrow 2^{\mathcal{A}}$. Intuitively, $Pr_D(A)$ denotes the actions whose effects will be accounted for when A is executed. This function allows, for example, to prioritize certain sets of actions. The new transition function $\Phi_{D,P}$ will be modified as follows:

$$\Phi_{D,P}(A, s) = \Phi_D(Pr_D(A), s)$$

where Φ_D is defined as in the previous section. Observe that if there is no competition among agents in D then Pr_D is simply the identity function.

4.1 The Rocket Domain

This domain was originally proposed in [31]. It was invented to motivate the development of a logic for reasoning about strategies of agents. This aspect will not be addressed by our formalization of this example as $\mathcal{C}$ lacks this capability. Nevertheless, the encoding is sufficient for determining the state of the world after the execution of actions by the agents.

We have a rocket, a cargo, and the agents 1, 2, and 3. The rocket or the cargo are either in $london$ or $paris$. The rocket can be moved by 1 and 2 between the two locations. The cargo can be loaded (unloaded) into the rocket by 1 and 3 (2 and 3). Agent 3 can refill the rocket if the tank is not full.

There are some constraints that limit the effects of the actions. They are:

- If 1 or 2 moves the rocket, the cargo cannot be loaded or unloaded;
- If two agents load/unload the cargo at the same time, the effect is the same as if it were loaded/unloaded by one agent.
- If one agent loads the cargo and another one unloads the cargo at the same time, the effect is that the cargo is loaded.

We will use the fluents $rocket(london)$ and $rocket(paris)$ to denote the location of the rocket. Likewise, $cargo(london)$ and $cargo(paris)$ denote the location of the cargo. in_rocket says that the cargo is inside the rocket and $tank\_full$ states that the tank is full. The signatures for the agents can be defined as follows.

$$\mathcal{F}_1 = \left\{ \begin{array}{l} in\_rocket, rocket(london), rocket(paris), \\ cargo(london), cargo(paris) \end{array} \right\}$$
$$\mathcal{A}_1 = \{ load(1), unload(1), move(1) \}$$
$$\mathcal{F}_2 = \left\{ \begin{array}{l} in\_rocket, rocket(london), rocket(paris), \\ cargo(london), cargo(paris) \end{array} \right\}$$
$$\mathcal{A}_2 = \{ unload(2), move(2) \}$$
$$\mathcal{F}_3 = \left\{ \begin{array}{l} in\_rocket, rocket(london), rocket(paris), \\ cargo(london), cargo(paris), tank\_full \end{array} \right\}$$
$$\mathcal{A}_3 = \{ load(3), refill \}$$

The constraints on the effects of actions induce priorities among the actions. The action $load$ or $unload$ will have no effect if $move$ is executed. The effects of two $load$ actions is the same as that of a single $load$ action. Likewise, two $unload$ actions have the same result as one $unload$ action. Finally, $load$ has a higher priority than $unload$.

To account for action priorities and the voting mechanism, we define $Pr_{D_{rocket}}$:

- $Pr_{D_{rocket}}(X) = \{move(a)\}$ if $\exists a.\ move(a) \in X$.
- $Pr_{D_{rocket}}(X) = \{load(a)\}$ if $move(x) \notin X$ for every $x \in \{1,2,3\}$ and $load(a) \in X$.
- $Pr_{D_{rocket}}(X) = \{unload(a)\}$ if $move(x) \notin X$ and $load(x) \notin X$ for every $x \in \{1,2,3\}$ and $unload(a) \in X$.
- $Pr_{D_{rocket}}(X) = X$ otherwise.

It is easy to see that $Pr_{D_{rocket}}$ defines priorities among the actions: if the rocket is moving then load/unload are ignored; load has higher priority than unload; etc. The domain specification consists of the following laws:

caused in_rocket **after** $load(i)$ $(i \in \{1,3\})$
caused $\neg in\_rocket$ **after** $unload(i)$ $(i \in \{1,2\})$
caused $tank\_full$ **if** $\neg tank\_full$ **after** $refill$
caused $\neg tank\_full$ **if** $tank\_full$ **after** $move(i)$ $(i \in \{1,2\})$
caused $rocket(london)$ **if** $rocket(paris), tank\_full$ **after** $move(i)$ $(i \in \{1,2\})$
caused $rocket(paris)$ **if** $rocket(london), tank\_full$ **after** $move(i)$ $(i \in \{1,2\})$
caused $cargo(paris)$ **if** $rocket(paris), in\_rocket$
caused $cargo(london)$ **if** $rocket(london), in\_rocket$

Let $\mathcal{I}_4$ consist of the following facts:

initially $tank\_full$ **initially** $rocket(paris)$
initially $cargo(london)$ **initially** $\neg in\_rocket$

We can show the following

$$(D_{rocket}, \mathcal{I}_4) \models \textbf{necessarily } cargo(paris) \textbf{ after } \{move(1)\}, \{load(3)\}, \{refill\}, \{move(3)\}.$$

Observe that without the priority function $Pr_{D_{rocket}}$, for every state s,

$$\Phi_{D_{rocket}}(\{load(1), unload(2)\}, s) = \emptyset,$$

i.e., the concurrent execution of the $load$ and $unload$ actions is unsuccessful.

5 Adding Reward Strategies

The next example illustrates the need to handle numbers and optimization to represent reward mechanisms. The extension of $\mathcal{C}$ is simply the introduction of *numerical fluents*—i.e., fluents that, instead of being simply true or false, have a numerical value. For this purpose, we introduce a new variant of the necessity query

$$\textbf{necessarily max } F \textbf{ for } \varphi \textbf{ after } A_1, \ldots, A_n$$

where F is a numerical expressions involving only numerical fluents, φ is a state formula, and $A_1, \ldots, A_n$ is a plan. Given a domain specification D and an initial state description $\mathcal{I}$, we can define for each fluent numerical expression F and plan α:

$$value(F, \alpha) = \max \{s(F) \mid s \in \Phi^*(\alpha, s_0), s_0 \textit{ is an initial state w.r.t. } \mathcal{I}, D\}$$

where $s(F)$ denotes the value of the expression F in state s. This allows us to define the following notion of entailment of a query:

$$(D, \mathcal{I}) \models \textbf{necessarily max } F \textbf{ for } \varphi \textbf{ after } A_1, \ldots, A_n$$

if:

- $(D, \mathcal{I}) \models$ **necessarily** φ **after** $A_1, \ldots, A_n$
- for every other plan $B_1, \ldots, B_m$ such that $(D, \mathcal{I}) \models$ **necessarily** φ **after** $B_1, \ldots, B_m$ we have that $value(F, [A_1, \ldots, A_n]) \geq value(F, [B_1, \ldots, B_m])$.

The following example has been derived from [5] where it is used to illustrate the coordination among agents to obtain the highest possible payoff. There are three agents. Agent 0 is a normative system that can play one of two strategies—either st_0 or $\neg st_0$. Agent 1 plays a strategy st_1, while agent 2 plays the strategy st_2. The reward system is described in the following tables (the first is for st_0 and the second one is for $\neg st_0$).

st_0	st_1	$\neg st_1$
st_2	$1, 1$	$0, 0$
$\neg st_2$	$0, 0$	$-1, -1$

$\neg st_0$	st_1	$\neg st_1$
st_2	$1, 1$	$0, 0$
$\neg st_2$	$0, 0$	$1, 1$

The signatures used by the agents are

$\mathcal{F}_0 = \{st_0, reward\}$ $\quad \mathcal{F}_1 = \{st_1, reward_1\}$ $\quad \mathcal{F}_2 = \{st_2, reward_2\}$
$\mathcal{A}_0 = \{play\_0, play\_not\_0\}$ $\quad \mathcal{A}_1 \quad \{play\_1, play\_not\_1\}$ $\quad \mathcal{A}_2 = \{play\_2, play\_not\_2\}$

The domain specification D_{gam} consists of:

caused st_0 **after** $play\_0$ $\quad$ **caused** $\neg st_0$ **after** $play\_not\_0$
caused st_1 **after** $play\_1$ $\quad$ **caused** $\neg st_1$ **after** $play\_not\_1$
caused st_2 **after** $play\_2$ $\quad$ **caused** $\neg st_2$ **after** $play\_not\_2$
caused $reward\_1 = 1$ **if** $\neg st_0 \wedge st_1 \wedge st_2$
caused $reward\_2 = 1$ **if** $\neg st_0 \wedge st_1 \wedge st_2$
caused $reward\_1 = 0$ **if** $\neg st_0 \wedge st_1 \wedge \neg st_2$
caused $reward\_2 = 0$ **if** $\neg st_0 \wedge st_1 \wedge \neg st_2$
...
caused $reward = a + b$ **if** $reward_1 = a \wedge reward_2 = b$

Assuming that $\mathcal{I} = \{$ **initially** $st_0\}$ we can show that

$$(D_{game}, \mathcal{I}) \models \textbf{ necessarily max } reward \textbf{ after } \{play_1, play_2\}.$$

6 Reasoning and Properties

In this section we discuss various types of reasoning that are directly enabled by the semantics of $\mathcal{C}$ that can be useful in reasoning about MAS. Recall that we assume that the action theories are developed from the perspective of a modeler who has the view of the complete MAS.

6.1 Capability Queries

Let us explore another range of queries, that are aimed at capturing the capabilities of agents. We will use the generic form **can** X **do** φ, where φ is a state formula and $X \subseteq \mathcal{AG}$ where $\mathcal{AG}$ is the set of agent identifiers of the domain. The intuition is to validate whether the group of agents X can guarantee that φ is achieved.

If $X = \mathcal{AG}$ then the semantics of the capability query is simply expressed as $(D, \mathcal{I}) \models$ **can** X **do** φ iff $\exists k.\ \exists A_1, \ldots, A_k$ such that

$$(D, \mathcal{I}) \models \textbf{ necessarily } \varphi \textbf{ after } A_1, \ldots, A_k.$$

If $X \neq \{1, \ldots, n\}$, then we can envision different variants of this query.

Capability query with non-interference and complete knowledge: Intuitively, the goal is to verify whether the agents X can achieve φ when operating in an environment that includes *all* the agents, but the agents $\mathcal{AG} \setminus X$ are simply providing their knowledge and not performing actions or interfering. We will denote this type of queries as $\textbf{can}^n_k\ X$ **do** φ (n: not interference, k: availability of all knowledge).

The semantics of this type of queries can be formalized as follows: $(D, \mathcal{I}) \models \textbf{can}^n_k\ X$ **do** φ if there is a sequence of sets of actions $A_1, \ldots, A_m$ with the following properties:

- for each $1 \leq i \leq m$ we have that $A_i \subseteq \bigcup_{j \in X} \mathcal{A}_j$ (we perform only actions of agents in X)
- $(D, \mathcal{I}) \models$ **necessarily** φ **after** $A_1, \ldots, A_m$

Capability query with non-interference and projected knowledge: Intuitively, the query with projected knowledge assumes that not only the other agents ($\mathcal{AG} \setminus X$) are passive, but they also are not willing to provide knowledge to the active agents. We will denote this type of queries as $\textbf{can}^n_{\neg k}\ X$ **do** φ.

Let us refer to the *projection* of $\mathcal{I}$ w.r.t. X (denoted by $proj(\mathcal{I}, X)$) as the set of all the **initially** declarations that build on fluents of $\bigcup_{j \in X} \mathcal{F}_j$. The semantics of $\textbf{can}^n_{\neg k}$ type of queries can be formalized as follows: $(D, \mathcal{I}) \models \textbf{can}^n_{\neg k}\ X$ **do** φ if there is a sequence of sets of actions $A_1, \ldots, A_m$ such that:

- for each $1 \leq i \leq m$ we have that $A_i \subseteq \bigcup_{j \in X} \mathcal{A}_j$
- $(D, proj(\mathcal{I}, X)) \models$ **necessarily** φ **after** $A_1, \ldots, A_m$ (i.e., the objective will be reached irrespective of the initial configuration of the other agents)

Capability query with interference: The final version of capability query takes into account the possible interference from other agents in the system. Intuitively, the query with interference, denoted by $\mathbf{can}^i\ X\ \mathbf{do}\ \varphi$, implies that the agents X will be able to accomplish X in spite of other actions performed by the other agents.

The semantics is as follows: $(D,\mathcal{I}) \models \mathbf{can}^i\ X\ \mathbf{do}\ \varphi$ if there is a sequence of sets of actions $A_1,\ldots,A_m$ such that:

- for each $1 \leq i \leq m$ we have that $A_i \subseteq \bigcup_{j\in X} \mathcal{A}_j$
- for each sequence of sets of actions $B_1,\ldots,B_m$, where $\bigcup_{j=1}^{m} B_j \subseteq \bigcup_{j\notin X} \mathcal{A}_j$, we have that $(D,\mathcal{I}) \models$ **necessarily** φ **after** $(A_1 \cup B_1),\ldots,(A_m \cup B_m)$.

6.2 Inferring Properties of the Theory

The form of queries explored above allows us to investigate some basic properties of a multi-agent action domain.

Agent Redundancy: agent redundancy is a property of $(D,\mathcal{I})$ which indicates the ability to remove an agent to accomplish a goal. Formally, agent i is redundant w.r.t. a state formula φ and an initial state $\mathcal{I}$ if $(D,\mathcal{I}) \models$ **can** $X \setminus \{i\}$ **do** φ. The "level" of necessity can be refined, by adopting different levels of **can** (e.g., $\mathbf{can}^n_{\neg k}$ implies that the knowledge of agent i is not required); it is also possible to strengthen it by enabling the condition to be satisfied for *any* $\mathcal{I}$.

Agent Necessity: agent necessity is symmetrical to redundancy—it denotes the inability to accomplish a property φ if an agent is excluded. Agent i is necessary w.r.t. φ and $(D,\mathcal{I})$ if for all sequences of sets of actions $A_1,\ldots,A_m$, such that for all $1 \leq j \leq m$ $A_j \cap \mathcal{A}_i = \emptyset$, we have that it is not the case that

$$(D,\mathcal{I}) \models \textbf{necessarily}\ \varphi\ \textbf{after}\ A_1,\ldots,A_m.$$

We can also define different degrees of necessity, depending on whether the knowledge of i is available (or it should be removed from I) and whether i can interfere.

6.3 Compositionality

The formalization of multi-agent systems in $\mathcal{C}$ enables exploring the effects of composing domains; this is an important property, that allows us to model dynamic MAS systems (e.g., where new agents can join an existing coalition).

Let D_1, D_2 be two domains and let us indicate with $\langle \mathcal{F}_i^1, \mathcal{A}_i^1 \rangle_{i\in\mathcal{AG}_1}$ and $\langle \mathcal{F}_i^2, \mathcal{A}_i^2 \rangle_{i\in\mathcal{AG}_2}$ the agent signatures of D_1 and D_2. We assume that all actions sets are disjoint, while we allow $(\bigcup_{i\in\mathcal{AG}_1} \mathcal{F}_i^1) \cap (\bigcup_{i\in\mathcal{AG}_2} \mathcal{F}_i^2) \neq \emptyset$.

We define the two instances $(D_1,\mathcal{I}_1)$ and $(D_2,\mathcal{I}_2)$ to be *composable* w.r.t. a state formula φ if $(D_1,\mathcal{I}_1) \models$ **can** $\mathcal{AG}_1$ **do** φ or $(D_2,\mathcal{I}_2) \models$ **can** $\mathcal{AG}_2$ **do** φ implies

$$(D_1 \cup D_2, \mathcal{I}_1 \cup \mathcal{I}_2) \models \mathbf{can}\ \mathcal{AG}_1 \cup \mathcal{AG}_2\ \mathbf{do}\ \varphi$$

Two instances are composable if they are composable w.r.t. all formulae φ. Domains D_1, D_2 are composable if all the instances $(D_1,\mathcal{I}_1)$ and $(D_2,\mathcal{I}_2)$ are composable.

7 Reasoning with Agent Knowledge

In this section, we will consider some examples from [12,30,18] which address another aspect of modeling MAS, i.e., the exchange of knowledge between agents and the reasoning in presence of incomplete knowledge. The examples illustrate the limitation of $\mathcal{C}$ as a language for multi-agent domains and the inadequacy of modeling MAS from the perspective of an omniscient modeler.

7.1 Heaven and Hell Domain: The Modeler's Perspective

This example has been drawn from [30], where it is used to motivate the introduction of decentralized POMDP and its use in multi-agent planning. The following formalization does not consider the rewards obtained by the agents after the execution of a particular plan.

In this domain, there are two agents 1 and 2, a priest p, and three rooms r_1, r_2, r_3. Each of the two rooms r_2 and r_3 is either heaven or hell. If r_2 is heaven then r_3 is hell and vice versa. The priest has the information where heaven/hell is located. The agents 1 and 2 do not know where heaven/hell is; but, by visiting the priest, they can receive the information that tells them where heaven is. 1 and 2 can also exchange their knowledge about the location of $heaven$. 1 and 2 want to meet in heaven.

The signatures for the three agents are as follows ($k, h \in \{1, 2, 3\}$):

$$\begin{array}{ll} \mathcal{F}_1 = \{heaven_1^2, heaven_1^3, at_1^k\} & \mathcal{A}_1 = \{m_1(k,h), ask_1^2, ask_1^p\} \\ \mathcal{F}_2 = \{heaven_2^2, heaven_2^3, at_2^k\} & \mathcal{A}_2 = \{m_2(k,h), ask_2^1, ask_2^p\} \\ \mathcal{F}_p = \{heaven_p^2, heaven_p^3\} & \mathcal{A}_p = \emptyset \end{array}$$

Intuitively, $heaven_i^j$ denotes that i knows that $heaven$ is in the room j and at_i^j denotes that i is at the room j. ask_i^j is an action whose execution will allow i to know where $heaven$ is if j knows where $heaven$ is. On the other hand, $m_i(k, h)$ encodes the action of moving i from the room k to the room h.

Observe that the fact that i does not know the location of $heaven$ is encoded by the formula $\neg heaven_i^2 \wedge \neg heaven_i^3$.

The domain specification D_{hh} contains the following laws:

$$\begin{array}{ll} \textbf{caused } heaven_1^j \textbf{ if } heaven_x^j \textbf{ after } ask_1^x & (j \in \{2,3\}, x \in \{2,p\}) \\ \textbf{caused } heaven_2^j \textbf{ if } heaven_x^j \textbf{ after } ask_2^x & (j \in \{2,3\}, x \in \{1,p\}) \\ \textbf{caused } at_i^j \textbf{ if } at_i^k \textbf{ after } m_i(k,j) & (i \in \{1,2,p\}, j,k \in \{1,2,3\}) \\ \textbf{caused } \neg at_i^j \textbf{ if } at_i^k & (i \in \{1,2,p\}, j,k \in \{1,2,3\}, j \neq k) \\ \textbf{caused } \neg heaven_i^2 \textbf{ if } heaven_i^3 & (i \in \{1,2,p\}, j \in \{2,3\}) \\ \textbf{caused } \neg heaven_i^3 \textbf{ if } heaven_i^2 & (i \in \{1,2,p\}, j \in \{2,3\}) \end{array}$$

The first two laws indicate that if 1 (or 2) asks 2 or p (or 1 or p) for the location of $heaven$, then 1 (or 2) will know where $heaven$ is if $2/p$ (or $1/p$) has this information. The third law encodes the effect of moving between rooms by the agents. The fourth law represents the static law indicating that one person can be at one place at a time.

Let us consider an instance that has initial state described by $\mathcal{I}_5$ ($j \in \{2,3\}$):

initially at_1^1 **initially** at_2^2 **initially** $heaven_p^2$
initially $\neg heaven_1^j$ **initially** $\neg heaven_2^j$

We can show that

$$(D_{hh}, \mathcal{I}_5) \models \textbf{necessarily } at_1^2 \wedge at_2^2 \textbf{ after } \{ask_1^p\}, \{m_1(1,2)\}$$

7.2 Heaven and Hell: The Agent's Perspective

The previous encoding of the domain has been developed considering the perspective of a domain modeler, who has complete knowledge about the world and all the agents. This perspective is reasonable in the domains encountered in the previous sections. Nevertheless, this perspective makes a difference when the behavior of one agent depends on knowledge that is not immediately available, e.g., agent 1 does not know where $heaven$ is and needs to acquire this information through knowledge exchanges with other agents. The model developed in the previous subsection is adequate for certain reasoning tasks (e.g., plan validation) but it is weak when it comes to tasks like planning.

An alternative model can be devised by looking at the problem from the perspective of each individual agent (not from a central modeler). This can be captured through an adaptation of the notion of sensing actions discussed in [25,26]. Intuitively, a sensing action allows for an agent to establish the truth value of unknown fluents. A sensing action a can be specified by laws of the form

$$\textbf{determines } l_1, \ldots, l_k \textbf{ if } F \textbf{ after } a$$

where $l_1, \ldots, l_k$ are fluent literals, F is a state formula, and a is a sensing action. Intuitively, a can be executed only when F is true and after its execution, one of $l_1, \ldots, l_k$ is set to true and all the others are set to false. The semantics of $\mathcal{C}$ extended with sensing actions can be defined in a similar fashion as in [26] and is omitted here for lack of space. It suffices to say that the semantics of the language should now account for different possibilities of the multi-agent systems due to incomplete information of the individual agents.

The signatures for the three agents are as follows ($k, h \in \{1,2,3\}$):

$\mathcal{F}_1 = \{heaven_1^2, heaven_1^3, ok_1^2, ok_1^p, at_1^k\}$ $\mathcal{A}_1 = \{m_1(k,h), ask_1^2, ask_1^p, know?_1^2, know?_1^p\}$
$\mathcal{F}_2 = \{heaven_2^2, heaven_2^3, ok_2^1, ok_2^p, at_2^k\}$ $\mathcal{A}_2 = \{m_2(k,h), ask_2^1, ask_2^p, know?_2^1, know?_2^p\}$
$\mathcal{F}_p = \{heaven_p^2, heaven_p^3\}$ $\mathcal{A}_p = \emptyset$

Intuitively, the fluent ok_y^x denotes the fact that agent y knows that agent x knows the location of heaven. The initial state for 1 is given by $I_5^1 = \{$ **initially** at_1^1, **initially** $ok_1^p\}$. Similarly, the initial state for 2 is $I_5^2 = \{$ **initially** at_2^2, **initially** $ok_2^p\}$, and for p is $I_5^p = \{$ **initially** $heaven_p^2\}$. The domain specification D_1 for 1 include the last four statements of D_{hh} and the following sensing action specifications:

determines $heaven_1^2, heaven_1^3$ **if** ok_1^x **after** ask_1^x $(x \in \{2,p\})$
determines $ok_1^x, \neg ok_1^x$ **after** $know?_1^x$ $(x \in \{2,p\})$

The domain specification D_2 for 2 is similar. The domain specification D_p consists of only the last two static laws of D_{hh}. Let $D'_{hh} = D_1 \cup D_2 \cup D_p$ and $I'_5 = I^1_5 \cup I^2_5 \cup I^p_5$, we can show that

$$(D'_{hh}, I'_5) \models \textbf{necessarily } heaven^2_1 \wedge heaven^2_2 \textbf{ after } \{ask^p_1\}, \{know?^1_2\}, \{ask^1_2\}.$$

7.3 Beyond $\mathcal{C}$ with Sensing Actions

This subsection discusses an aspect of modeling MAS that cannot be easily dealt with in $\mathcal{C}$, even with sensing actions, i.e., representing and reasoning about knowledge of agents. In Section 7.1, we use two different fluents to model the knowledge of an agent about properties of the world, similar to the approach in [26]. This approach is adequate for several situations. Nevertheless, the same approach could become quite cumbersome if complex reasoning about knowledge of other agents is involved.

Let us consider the well known *Muddy Children* problem [12]. Two children are playing outside the house. Their father comes and tells them that at least one of them has mud on his/her forehead. He then repeatedly asks "do you know whether your forehead is muddy or not?". The first time, both answer "no" and the second time, both say 'yes'. It is known that the father and the children can see and hear each other.

The representation of this domain in $\mathcal{C}$ is possible, but it would require a large number of fluents (that describe the knowledge of each child, the knowledge of each child about the other child, etc.) as well as a formalization of the axioms necessary to express how knowledge should be manipulated, similar to the fluents ok^j_i in the previous example.

A more effective approach is to introduce explicit knowledge operators (with manipulation axioms implicit in their semantics—e.g., as operators in a S5 modal logic) and use them to describe agents state. Let us consider a set of modal operators $\mathbf{K}_i$, one for each agent. A formula such as $\mathbf{K}_i\varphi$ denotes that agent i knows property φ. Knowledge operators can be nested; in particular, $\mathbf{K^*}_G\psi$ denotes all formulae with arbitrary nesting of $\mathbf{K}_G$ operators (G being a set of agents).

In our example, let us denote the children with 1 and 2, m_i as a fluent to denote whether i is muddy or not. The initial state of the world can then be described as follows:

$$\textbf{initially } m_1 \wedge m_2 \tag{1}$$

$$\textbf{initially } \neg\mathbf{K}_i m_i \wedge \neg\mathbf{K}_i \neg m_i \tag{2}$$

$$\textbf{initially } \mathbf{K^*}(m_1 \vee m_2) \tag{3}$$

$$\textbf{initially } \mathbf{K^*}_{\{1,2\}\setminus\{i\}} m_i \tag{4}$$

$$\textbf{initially } \mathbf{K^*}(\mathbf{K^*}_{\{1,2\}\setminus\{i\}} m_i \vee \mathbf{K^*}_{\{1,2\}\setminus\{i\}} \neg m_i) \tag{5}$$

where $i \in \{1, 2\}$. (1) states that all the children are muddy. (2) says that i does not know whether he/she is muddy. (3) encodes the fact that the children share the common knowledge that at least one of them is muddy. (4) captures the fact that each child can see the other child. Finally, (5) represents the common knowledge that each child knows the muddy status of the other one.

The actions used in this domain would enable agents to gain knowledge; e.g., the 'no' answer of child 1 allows child 2 to learn $\mathbf{K}_1(\neg\mathbf{K}_1 m_1 \wedge \neg\mathbf{K}_1 \neg m_1)$. This, together

with the initial knowledge, would be sufficient for 2 to conclude $\mathbf{K}_2 m_2$. A discussion of how these inferences occur can be found, for example, in [12].

8 Discussion and Conclusion

In this paper, we presented an investigation of the use of the $\mathcal{C}$ action language to model MAS domains. $\mathcal{C}$, as several other action languages, is interesting as it provides well studied foundations for knowledge representation and for performing several types of reasoning tasks. Furthermore, the literature provides a rich infrastructure for the implementation of action languages (e.g., through translational techniques [27]). The results presented in this paper identify several interesting features that are necessary for modeling MAS, and they show how many of these features can be encoded in $\mathcal{C}$—either directly or with simple extensions of the action language. We also report challenging domains for $\mathcal{C}$.

There have been many agent programming languages such as the BDI agent programming AgentSpeak [23], (as implemented in Jason [4]), JADE [3] (and its extension Jadex [7]), ConGolog [10], IMPACT [1], 3APL [9], GOAL [19]. A good comparison of many of these languages can be found in [21].

We would like to stress that the paper does not introduce a new agent "programming language", in the style of languages mentioned above. Rather, we bring an action language perspective, where the concern is on succinctly and naturally specifying the transition between worlds due to actions. Thus our focus is how to extend actions languages to the multi-agent domain in a way to capture various aspects of multi-agent reasoning. The issues of implementation and integration in a distributed environment are interesting, but outside of the scope of this paper. To draw an analogy, what we propose in this paper is analogous to the role of situation calculus or PDDL in the description of single-agent domains, which describe the domains without providing implementation constructs for composing programs, as in Golog/ConGolog or GOAL. As such, our proposal could provide the underlying representation formalism for the development of an agent programming language; on the other hand, it could be directly used as input to a reasoning system, e.g., a planner [8]. Our emphasis in the representation is exclusively on the description of effects of actions; this distinguishes our approach from other logic-based formalisms, such as those built on MetateM [13].

Although our proposal is not an agent programming language, it is still interesting to analyze it according to the twelve dimensions discussed in [11] and used in [21];

1. *Purpose of use:* the language is designed for formalization and verification of MAS.
2. *Time:* the language does not have explicit references to time.
3. *Sensing:* the language supports sensing actions.
4. *Concurrency:* our proposed language enables the description of concurrent and interacting actions.
5. *Nondeterminism:* the language naturally supports nondeterminism.
6. *Agent knowledge:* our language allows for the description of agents with incomplete knowledge and can be extended to handle uncertainty.
7. *Communication:* this criteria is not applicable to our language.

8. *Team working:* the language could be used for describing interaction between agents including coordination [28] and negotiation [29].
9. *Heterogeneity and knowledge sharing:* the language does not force the agents to use the same ontology.
10. *Programming style:* this criteria is not applicable to our language since it is not an agent programming language.
11. *Modularity:* our language does not provide any explicit mechanism for modularizing the knowledge bases.
12. *Semantics:* our proposal has a clear defined semantics, which is based on the transition system between states.

The natural next steps in this line of work consist of *(1)* exploring the necessary extensions required for a more natural representation and reasoning about knowledge of agents in MAS domains (see Sect. 7); *(2)* adapting the more advanced forms of reasoning and implementation proposed for $\mathcal{C}$ to the case of MAS domains; *(3)* investigating the use of the proposed extension of $\mathcal{C}$ in formalizing distributed systems.

Acknowledgement. The last two authors are partially supported by the NSF grants IIS-0812267, CBET-0754525, CNS-0220590, and CREST-0420407.

References

1. Subrahmanian, V.S., Bonatti, P., Dix, J., Eiter, T., Kraus, S., Ozcan, F., Ross, R.: Heterogeneous Agent Systems: Theory and Implementation. MIT Press, Cambridge (2000)
2. Baker, A.: A simple solution to the Yale Shooting Problem. In: KRR, pp. 11–20 (1989)
3. Bellifemine, F.L., Caire, G., Greenwood, D.: Developing Multi-Agent Systems with JADE. J. Wiley & Sons, Chichester (2007)
4. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming Multi-agent Systems in AgentSpeak using Jason. J. Wiley and Sons, Chichester (2007)
5. Boella, G., van der Torre, L.: Enforceable social laws. In: AAMAS 2005, pp. 682–689. ACM, New York (2005)
6. Boutilier, C., Brafman, R.I.: Partial-order planning with concurrent interacting actions. J. Artif. Intell. Res (JAIR) 14, 105–136 (2001)
7. Braubach, L., Pokahr, A., Lamersdorf, W.: Jadex: a BDI-Agent System Combining Middleware and Reasoning. In: Software Agent-based Applications, Platforms and Development Kits. Springer, Heidelberg (2005)
8. Brenner, M.: Planning for Multi-agent Environments: From Individual Perceptions to Coordinated Execution. In: Work. on Multi-agent Planning and Scheduling, ICAPS, pp. 80–88 (2005)
9. Dastani, M., Dignum, F., Meyer, J.J.: 3APL: A Programming Language for Cognitive Agents. ERCIM News, European Research Consortium for Informatics and Mathematics, Special issue on Cognitive Systems (53) (2003)
10. De Giacomo, G., Lespèrance, Y., Levesque, H.J.: ConGolog, a concurrent programming language based on the situation calculus. Artificial Intelligence 121(1–2), 109–169 (2000)
11. Jennings, N., Sycara, K., Wooldridge, M.: A roadmap of agent research and development. Autonomous Agents and Multi-Agent Systems 1, 7–38 (1998)
12. Fagin, R., Halpern, J., Moses, Y., Vardi, M.: Reasoning about Knowledge. MIT Press, Cambridge (1995)

13. Fisher, M.: A survey of Concurrent METATEM – the language and its applications. In: Gabbay, D.M., Ohlbach, H.J. (eds.) ICTL 1994. LNCS (LNAI), vol. 827, pp. 480–505. Springer, Heidelberg (1994)
14. Gelfond, M., Lifschitz, V.: Representing actions and change by logic programs. Journal of Logic Programming 17(2,3,4), 301–323 (1993)
15. Gelfond, M., Lifschitz, V.: Action languages. ETAI 3(6) (1998)
16. Gerbrandy, J.: Logics of propositional control. In: AAMAS 2006, pp. 193–200. ACM, New York (2006)
17. Hanks, S., McDermott, D.: Nonmonotonic logic and temporal projection. Artificial Intelligence 33(3), 379–412 (1987)
18. Herzig, A., Troquard, N.: Knowing how to play: uniform choices in logics of agency. In: AAMAS 2006, pp. 209–216 (2006)
19. de Boer, F.S., Hindriks, K.V., van der Hoek, W., Ch, J.-J.: Meyer. A verification framework for agent programming with declarative goals. Journal of Applied Logic 5, 277–302 (2005)
20. Kautz, H.: The logic of persistence. In: Proceedings of AAAI 1986, pp. 401–405. AAAI Press, Menlo Park (1986)
21. Mascardi, V., Martelli, M., Sterling, L.: Logic-Based Specification Languages for Intelligent Software Agents. Theory and Practice of Logic Programming 4(4), 495–537
22. McCarthy, J., Hayes, P.: Some philosophical problems from the standpoint of artificial intelligence. Machine Intelligence 4, 463–502 (1969)
23. Rao, A.S.: AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In: Perram, J., Van de Velde, W. (eds.) MAAMAW 1996. LNCS, vol. 1038, pp. 42–55. Springer, Heidelberg (1996)
24. Sauro, L., Gerbrandy, J., van der Hoek, W., Wooldridge, M.: Reasoning about action and cooperation. In: AAMAS 2006, pp. 185–192. ACM Press, New York (2006)
25. Scherl, R., Levesque, H.: Knowledge, action, and the frame problem. Artificial Intelligence 144(1-2) (2003)
26. Son, T.C., Baral, C.: Formalizing sensing actions - a transition function based approach. Artificial Intelligence 125(1-2), 19–91 (2001)
27. Son, T.C., Baral, C., Tran, N., McIlraith, S.: Domain-dependent knowledge in answer set planning. ACM Trans. Comput. Logic 7(4), 613–657 (2006)
28. Son, T.C., Sakama, C.: Reasoning and Planning with Cooperative Actions for Multiagents Using Answer Set Programming. In: Baldoni, M., Bentahar, J., van Riemsdijk, M.B., Lloyd, J. (eds.) DALT 2009. LNCS, vol. 5948, pp. 208–227. Springer, Heidelberg (2010)
29. Son, T.C., Pontelli, E., Sakama, C.: Logic Programming for Multiagent Planning with Negotiation. In: Hill, P.M., Warren, D.S. (eds.) Logic Programming. LNCS, vol. 5649, pp. 99–114. Springer, Heidelberg (2009)
30. Spaan, M., Gordon, G.J., Vlassis, N.A.: Decentralized planning under uncertainty for teams of communicating agents. In: AAMAS 2006, pp. 249–256 (2006)
31. van der Hoek, W., Jamroga, W., Wooldridge, M.: A logic for strategic reasoning, pp. 157–164. ACM, New York (2005)
32. van Ditmarsch, H.P., van der Hoek, W., Kooi, B.P.: Concurrent Dynamic Epistemic Logic for MAS. In: AAMAS (2003)