Modeling Multi-Agent Domains in an Action Languages: an Empirical Study Using $\mathcal{C}$

Chitta Baral
Dept. Computer Science & Engineering
Arizona State University
chitta@asu.edu

Tran Cao Son and **Enrico Pontelli**
Dept. Computer Science
New Mexico State University
tson | epontell @cs.nmsu.edu

Abstract

In this paper, we evaluate the expressiveness of the action language $\mathcal{C}$ in modeling multi-agent domains. Our investigation shows that, with minimal extensions, $\mathcal{C}$ can be adapted to model multi-agent domains in a natural way. We also show that the language is suitable for answering various types of queries that are interesting in modeling and analyzing 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 in multi-agent domains. For example, the authors of [Sauro *et al.*, 2006] combine an action logic and a cooperation logic to represent and reason about the capability and the cooperation between agents. [Gerbrandy, 2006] generalizes this framework to consider domains where an agent may control only a part of a proposition. In [van der Hoek *et al.*, 2005], an extension of Alternating-time Temporal Logic (ATL) is developed to facilitate strategic reasoning in multi-agent domains. The work in [Spaan *et al.*, 2006] 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 [Herzig and Troquard, 2006], an extension of Alternating-time Temporal Epistemic Logic (ATEL) is proposed for reasoning about choices.

The rich collection of logics proposed in the literature for formalizing MAS reflects attempts to design solutions to address specific issues in modeling MAS, often justified by a specific application scenario. This makes such logics suitable to address specific subsets of the general features required to model real-world MAS domains. The task of generalizing some of these existing proposals to create a uniform and comprehensive framework for modeling several different aspects of MAS domains is, to the best of our knowledge, still an open problem. Although we do not dispute the possibility of extending several of these existing proposals in various directions, the task does not seem easy. It is worth noticing that, on the other hand, the need for a general language for MAS domains, with a formal and simple semantics, that allows for the verification of plan correctness has been extensively motivated, e.g., [Brenner, 2005].

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. Between the discovery of the frame problem [McCarthy and Hayes, 1969] and 1990, several formalisms for representing and reasoning in dynamic domains have been proposed. Frequently, new proposals have appeared to address shortcomings of previous approaches on specific example domains. For example, the well-known Yale Shooting problem [Hanks and McDermott, 1987] was invented to show that the early solutions to the frame problem fail. A simple solution to the Yale Shooting problem was proposed in [Baker, 1989], yet this solution was shown to fail in the Stolen Car example [Kautz, 1986], etc. Action languages [Gelfond and Lifschitz, 1998] were one of the outcomes of this development, and they have been proved to be very useful ever since.

Action description languages, first introduced in [Gelfond and Lifschitz, 1993] and further refined in [Gelfond and Lifschitz, 1998], 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 offered by several of these languages (e.g., concurrent actions, sensing actions, non-deterministic 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 desiderata in the specification of MAS. 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 answer the above question by investigating whether an action language developed for single agent

domains can be used, with minimal modifications, to model MAS domains with several features of interest. Our starting point is a well-studied and well-understood single agent language action language—the language $\mathcal{C}$ [Gelfond and Lifschitz, 1998]. This language is chosen as it provides a number of features that appear necessary to handle multi-agent domains, such as concurrent interacting actions. The language is employed to formalize 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 extensions that are adequate to model the domains. The resulting action language provides a unifying framework for modeling 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.

Before we continue, let us discuss the desirable features MAS and the assumptions we will make in the rest of this work. In this paper, we consider a multi-agent systems (MAS) domain as an environment in which multiple agents can execute actions to change the environment. We assume that

- Agents can execute actions concurrently;
- Each agent knows its own capabilities;
- 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 to answer in a MAS domain involve

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

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

We would like to note that in the past there have been attempts to use action description languages to formalize multi-agent domains, e.g., [Boutilier and Brafman, 2001]. On the other hand, existing proposals address only some of the properties of the multi-agent scenarios mentioned earlier (e.g., concurrency).

To the best of our knowledge, this is the first investigation of how to adapt a standard singe-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 types of reasoning. In this work we do not deal with the issues of distributed plan generation—an aspect extensively explored in the existing literature. This is certainly an important research topic and worth pursuing but it is outside of the scope of this paper.

The paper is organized as follows. In the next section, we will review the basics of the action language $\mathcal{C}$. Section 3 describes a straightforwards adaptation of $\mathcal{C}$ for multi-agent domains. Each of the following sections (Sections 4–6) shows how minor additions to $\mathcal{C}$ can address several features in representing and reasoning about multi-agent domains. Section 7 presents the query language that can be used with the extended $\mathcal{C}$. Section 8 discusses further aspects of modeling MAS that the proposed extension of $\mathcal{C}$ cannot deal with easily. We conclude in Section 9.

2 Action Language $\mathcal{C}$

The starting point of our investigation is the action language $\mathcal{C}$ [Gelfond and Lifschitz, 1998]—an action description language originally developed to describe single agent domains, where the agent is capable of performing non-deterministic and concurrent actions. In this section we revise the basic definitions of $\mathcal{C}$.

A domain description in $\mathcal{C}$ builds on a language signature $\langle \mathcal{F}, \mathcal{A} \rangle$, where $\mathcal{F}$ is a finite collection of fluent names and $\mathcal{A}$ is a finite collection of 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., ℓ returns true if no action affects it).

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* is described simply by a set of $\mathcal{F}$-literals, such that $I \cap \{f, \neg f \mid f \in \mathcal{F}\} = \emptyset$. Given an interpretation I and a fluent f (literal $\neg f$), we say that $I \models f$ ($I \models \neg f$) if $f \in I$ ($\neg f \in I$). The entailment relation $\models$ can be easily generalized to arbitrarily propositional formulae. 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 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 containing I and closed w.r.t. $\mathcal{SC}$.

A *state* s is a complete interpretation which is closed w.r.t. the static causal laws. Given a state s, a set of actions $A \subseteq \mathcal{A}$, and a collection of dynamic causal laws $\mathcal{DC}$, we define

$$Eff_{\mathcal{DC}}(s, A) = \left\{ \ell \;\middle|\; \begin{array}{l} (\textbf{ caused } \ell \textbf{ if } F \textbf{ after } G) \in \mathcal{DC}, \\ s \cup A \models G, s \models F \end{array} \right\}$$

Let us consider a domain $D = \langle \mathcal{SC}, \mathcal{DC}, \mathcal{IN} \rangle$, where $\mathcal{SC}$ are the static causal laws, $\mathcal{DC}$ are the dynamic causal laws and $\mathcal{IN}$ are the non-inertial axioms. Let us also denote with if the set $if = \{f, \neg f \mid f \in \mathcal{IN} \text{ or } \neg f \in \mathcal{IN}\}$. The semantics of D is given by a transition system (N_D, E_D), where N_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

$$s' = Cl_{\mathcal{SC}}(Eff_{\mathcal{DC}}(s, A) \cup ((s \setminus if) \cap s') \cup (\mathcal{IN} \cap s')).$$

The original $\mathcal{C}$ language supports a query language (called $\mathcal{P}$ in [Gelfond and Lifschitz, 1998]). This language allows queries of the form **necessarily** F **after** $A_1, \ldots, A_k$ where F is a state formula and $A_1, \ldots, A_k$ are 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 $s \models F$.

Let us denote with $State_D$ the set of all possible states for the domain D. 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 $\{\ell \mid$ **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 (N_D, E_D) is the transition system describing the semantics of D. This function can be extended to define Φ^*_D which considers sequences of sets of actions (i.e., plans), where $\Phi^*_D([\,], s) = \{s\}$ and

$$\Phi^*_D([A_1, \ldots, A_n], s) = \bigcup_{s' \in \Phi^*_D([A_1, \ldots, A_{n-1}], s)} \Phi_D(A_n, s').$$

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 will discuss a number of small modifications of $\mathcal{C}$ necessary to enable modeling MAS domains. 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 agents to have knowledge of everything, we only expect the *modeler* to have such knowledge.

We uniquely associate to each agent an element from 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 $\bigcap_{i \in S} \mathcal{F}_i$ could be not empty for some $S \subseteq \mathcal{AG}$. This represents common knowledge between the agents in the group S of agents.

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.

We will now illustrate the use of $\mathcal{C}$ in modeling various examples from the literature.

3.1 The Prison Domain

This domain has been originally presented in [Sauro *et al.*, 2006]. In this example, we have two prison guards, 1 and 2, who control two gates, the inner gate and the outer gate, by operating the four buttons a_1, b_1, a_2, and b_2. Agent 1 controls a_1 and b_1 and 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. In $\mathcal{C}$, this domain can be represented as follows.

The set of agents is $\mathcal{AG} = \{1, 2\}$. For agent 1, we have that

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

Here, in_open and out_open represent the fact that the inner gate and outer gate are open, respectively. The fluent $pressed(X)$ indicates 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

$$\begin{array}{lcl} \mathcal{F}_2 & = & \{in\_open, out\_open, pressed(a_2), pressed(b_2)\} \\ \mathcal{A}_2 & = & \{push(a_2), push(b_2)\} \end{array}$$

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.

Let us now consider the queries that were asked in [Sauro *et al.*, 2006] 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

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

On the other hand, if the outer gate is initially closed, i.e., $\mathcal{I}_2 = \{$ **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 $A \setminus Y \neq \emptyset$.

3.2 The Credit Rating Domain

We will next consider an example from [Gerbrandy, 2006] in which an effect on certain properties of the world cannot be changed by a single agent.

We have two agents, $\mathcal{AG} = \{w, t\}$, denoting the website and the telephone operator respectively. Both agents can set/reset the credit 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 webcustomer (is_web fluent) or not is set only by the website agent w.

The signatures of the two agents are as follows:

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

The domain specification D_{bank} consists of:

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

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 actions within a MAS, focusing on agents' capabilities and actions interaction. This is in itself not a big surprise and has been discussed by [Boutilier and Brafman, 2001]. We will now present a small extension of $\mathcal{C}$ that facilitates strategic reasoning. 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 set of 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.

4.1 The Rocket Domain

This domain was originally proposed in [van der Hoek *et al.*, 2005]. We have a rocket, a cargo, and three 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 (resp. unloaded) into the rocket by 1 and 3 (resp. 2 and 3). Agent 3 can refill the rocket if the tank is not full.

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. The fluent in_rocket says that the cargo is inside the rocket and $tank\_full$ states that the tank is full. We will also have a non-inertial fluent literal, $moving$, denoting that the rocket is in the state of moving. The signatures for the agents can be defined as follows.

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

This domain has a special feature—there are priorities among actions. The domain states that $load$ 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 action.

To account for the priority of actions, we define Pr_D as follows:

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

It is easy to see that Pr_D 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** $rocket(london), in\_rocket$
non_inertial $\neg moving$

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

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

We can show the following

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

5 Reasoning with Agent Knowledge

In this section, we will consider some examples from [Spaan *et al.*, 2006; Herzig and Troquard, 2006] which address another aspect of modeling MAS, i.e., the exchange of knowledge between agents and the reasoning in presence of incomplete knowledge. We will show that using $\mathcal{C}$, we can model this aspect without the introduction of additional features.

5.1 Heaven and Hell Domain

This example has been drawn from [Spaan *et al.*, 2006]. This example has 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 knowledge of where the heaven and hell are located. Neither 1 nor 2 know where heaven/hell is, but, by visiting the priest, they can receive the information that tells them where heaven is. 1 and 2 want to meet in heaven.

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

$$\begin{array}{lcl} \mathcal{F}_1 & = & \{heaven_1^2, heaven_1^3, hell_1^2, hell_1^3, at_1^k\} \\ \mathcal{A}_1 & = & \{m_1(k,h), ask_1\} \\ \mathcal{F}_2 & = & \{heaven_2^2, heaven_2^3, hell_2^2, hell_2^3, at_2^k\} \\ \mathcal{A}_2 & = & \{m_2(k,h), ask_2\} \\ \mathcal{F}_p & = & \{heaven_p^2, heaven_p^3, hell_p^2, hell_p^3\} \\ \mathcal{A}_p & = & \emptyset \end{array}$$

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

caused $heaven_1^j$ **if** $heaven_p^j$ **after** ask_1 $\quad (j \in \{2,3\})$
caused $heaven_2^j$ **if** $heaven_p^j$ **after** ask_2 $\quad (j \in \{2,3\})$
caused at_i^j **if** at_i^k **after** $m_i(k,j) (i \in \{1,2,p\}, j,k \in \{1,2,3\})$
caused $\neg at_i^j$ **if** at_i^k $\quad (i \in \{1,2,p\}, j,k \in \{1,2,3\}, j \neq k)$
caused $hell_i^j$ **if** $heaven_i^k$ $\quad (i \in \{1,2,p\}, j,k \in \{2,3\}, j \neq k)$
caused $heaven_i^j$ **if** $hell_i^k$ $\quad (i \in \{1,2,p\}, j,k \in \{2,3\}, j \neq k)$
caused $\neg hell_i^j$ **if** $heaven_i^j$ $\quad (i \in \{1,2,p\}, j \in \{2,3\})$
caused $\neg heaven_i^j$ **if** $hell_i^j$ $\quad (i \in \{1,2,p\}, j \in \{2,3\})$

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** $\neg hell_1^j$ **initially** $\neg heaven_2^j$
initially $\neg hell_2^j$

We can show that

$$(D_{heav}, \mathcal{I}_5) \models \textbf{necessarily } at_1^2 \wedge heaven_1^2 \textbf{ after } \{ask_1\}, \{m_1(1,2)\}.$$

5.2 Blind Agents & Lamp Domain

This next example is drawn from [Herzig and Troquard, 2006]. There are two blind agents and two switches; the light is on only when both switches are in the same position. Agent 1 can toggle the first switch and agent 2 the second one.

$$\begin{array}{lcl} \mathcal{F}_1 & = & \{switch\_on_1, switch\_off_1, on\} \\ \mathcal{A}_1 & = & \{t_1\} \\ \mathcal{F}_2 & = & \{switch\_on_2, switch\_off_2, on\} \\ \mathcal{A}_2 & = & \{t_2\} \end{array}$$

Intuitively, $switch\_on_i$ (resp. $switch\_off_i$) represents the fact that i knows that switch i is at the *on* (resp. off) position. The domain D_{blind} is composed of the laws:

caused $\neg switch\_on_i$ **if** $switch\_off_i$
caused $\neg switch\_off_i$ **if** $switch\_on_i$
caused on **if** $switch\_on_1 \wedge switch\_on_2$
caused on **if** $switch\_off_1 \wedge switch\_off_2$
caused $switch\_on_i$ **if** $switch\_off_i$ **after** t_i
caused $switch\_off_i$ **if** $switch\_on_i$ **after** t_i

Let $\mathcal{I} = \{$ **initially** $switch\_on_1$, **initially** $switch\_off_2$, **initially** $\neg on\}$. We can show that

$$(D_{blind}, \mathcal{I}) \models \textbf{necessarily } on \textbf{ after } t_2$$

On the other hand, if $\mathcal{I} = \{$ **initially** $\neg on\}$, we can show that there is no plan α (i.e., a *conformant plan*) such that

$$(D_{blind}, \mathcal{I}) \not\models \textbf{necessarily } on \textbf{ after } \alpha$$

6 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 numeric 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:

$$value(F, \alpha) = \max \left\{ s(F) \;\middle|\; \begin{array}{l} s \in \Phi^*(\alpha, s_0), \\ s_0 \textit{ is an initial state w.r.t. } \mathcal{I}, D \end{array} \right\}$$

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:

- $(D, \mathcal{I}) \models \textbf{necessarily } \varphi \textbf{ after } A_1, \ldots, A_n$
- for every other plan $B_1, \ldots, B_m$ such that

  $$(D, \mathcal{I}) \models \textbf{necessarily } \varphi \textbf{ after } B_1, \ldots, B_m$$

  we have that

  $$value(F, [A_1, \ldots, A_n]) \geq value(F, [B_1, \ldots, B_m]).$$

6.1 Social Laws Domain

The following example has been derived from [Boella and van der Torre, 2005]. 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

$$\begin{array}{lcl} \mathcal{F}_0 & = & \{st_0, reward\} \\ \mathcal{A}_0 & = & \{play\_0, play\_not\_0\} \\ \mathcal{F}_1 & = & \{st_1, reward_1\} \\ \mathcal{A}_1 & & \{play\_1, play\_not\_1\} \\ \mathcal{F}_2 & = & \{st_2, reward_2\} \\ \mathcal{A}_2 & = & \{play\_2, play\_not\_2\} \end{array}$$

The domain specification D_{gam} consists of:

caused st_0 **after** $play\_0$
caused $\neg st_0$ **after** $play\_not\_0$
caused st_1 **after** $play\_1$
caused $\neg st_1$ **after** $play\_not\_1$
caused st_2 **after** $play\_2$
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

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

7 Reasoning and Properties

In this section we discuss various types of reasoning that are directly enabled by the semantics of $\mathcal{C}$.

7.1 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 $X \subseteq \mathcal{AG}$. The intuition is to validate whether the group of agents X can guarantee that φ is satisfied.

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 $X \neq \mathcal{AG}$, 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 $\textbf{can}^n_g\ X$ **do** φ (n: not interference, g: availability of all knowledge).

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

- for each $1 \leq i \leq k$ 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** $A_1, \ldots, A_k$

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 $\textbf{can}^n_l\ X$ **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 $\textbf{can}^n_l$ type of queries can be formalized as follows: $(D, \mathcal{I}) \models \textbf{can}^n_l\ X$ **do** φ if there is a sequence of sets of actions $A_1, \ldots, A_k$ such that:

- for each $1 \leq i \leq k$ we have that $A_i \subseteq \bigcup_{j \in X} \mathcal{A}_j$
- $(D, proj(\mathcal{I}, X)) \models$ **necessarily** φ **after** $A_1, \ldots, A_k$ (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 $\textbf{can}^i\ X$ **do** φ, 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: $(D, \mathcal{I}) \models \textbf{can}^i\ X$ **do** φ if there is a sequence of sets of actions $A_1, \ldots, A_k$ such that:

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

7.2 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 in accomplishing a goal. Formally, agent i is redundant w.r.t. a state formula φ and an initial state $\mathcal{I}$ if

$$(D, \mathcal{I}) \models \textbf{can } X \setminus \{i\} \textbf{ do } \varphi.$$

The "level" of necessity can be refined, by adopting different levels of **can** (e.g., $\textbf{can}^n_l$ 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 $A_1, \ldots, A_k$, such that $A_j \cap \mathcal{A}_i = \emptyset$ for all $1 \leq j \leq k$, we have that it is not the case that $(D, \mathcal{I}) \models$ **necessarily** φ **after** $A_1, \ldots, A_k$.

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

7.3 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

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

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

8 Discussion

This section discusses an aspect of modeling MAS that cannot be easily dealt with in $\mathcal{C}$, i.e., representing and reasoning about knowledge of agents. In the domains 5.1 and 5.2, we use two different fluents to model the knowledge of an agent about properties of the world, similar to the approach in [Palacios and Geffner, 2007; Son and Baral, 2001]. 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.

Example 1 (Muddy Children, [Fagin *et al.*, 1995])**.** 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.

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. We use m_i as a fluent denoting 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 $\mathbf{K}_2 m_2$.

9 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 [Son *et al.*, 2006]). The results presented in this paper identify several interesting features that are necessary for modeling MAS, and they show how such features can be encoded in $\mathcal{C}$—either directly or with simple extensions of the action language. We also report on different forms of reasoning that are naturally supported by the proposed language.

The natural next steps in this line of work consist of *(1)* exploring how more complex domains can be captured, especially domains requiring complex reasoning about knowledge of other agents (as discussed in Sect. 8); *(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.

References

[Baker, 1989] A. Baker. A simple solution to the Yale Shooting Problem. In R. Brachman, H. Levesque, and R. Reiter, editors, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 11–20. Morgan Kaufmann, 1989.

[Boella and van der Torre, 2005] Guido Boella and Leendert W. N. van der Torre. Enforceable social laws. In Frank Dignum, Virginia Dignum, Sven Koenig, Sarit Kraus, Munindar P. Singh, and Michael Wooldridge, editors, *4rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005), July 25-29, 2005, Utrecht, The Netherlands*, pages 682–689. ACM, 2005.

[Boutilier and Brafman, 2001] Craig Boutilier and Ronen I. Brafman. Partial-order planning with concurrent interacting actions. *J. Artif. Intell. Res. (JAIR)*, 14:105–136, 2001.

[Brenner, 2005] M. Brenner. Planning for Multiagent Environments: From Individual Perceptions to Coordinated Execution. In *Proceedings of Workshop on Multiagent Planning and Scheduling, ICAPS*, pages 80–88. 2005.

[Fagin *et al.*, 1995] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about Knowledge*. MIT press, 1995.

[Gelfond and Lifschitz, 1993] M. Gelfond and V. Lifschitz. Representing actions and change by logic programs. *Journal of Logic Programming*, 17(2,3,4):301–323, 1993.

[Gelfond and Lifschitz, 1998] M. Gelfond and V. Lifschitz. Action languages. *ETAI*, 3(6), 1998.

[Gerbrandy, 2006] Jelle Gerbrandy. Logics of propositional control. In Hideyuki Nakashima, Michael P. Wellman, Gerhard Weiss, and Peter Stone, editors, *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), Hakodate, Japan, May 8-12, 2006*, pages 193–200. ACM, 2006.

[Hanks and McDermott, 1987] S. Hanks and D. McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33(3):379–412, 1987.

[Herzig and Troquard, 2006] Andreas Herzig and Nicolas Troquard. Knowing how to play: uniform choices in logics of agency. In Hideyuki Nakashima, Michael P. Wellman, Gerhard Weiss, and Peter Stone, editors, *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), Hakodate, Japan, May 8-12, 2006*, pages 209–216, 2006.

[Kautz, 1986] H. Kautz. The logic of persistence. In *Proceedings of AAAI-86*, pages 401–405, 1986.

[McCarthy and Hayes, 1969] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, Edinburgh, 1969.

[Palacios and Geffner, 2007] H. Palacios and H. Geffner. From Conformant into Classical Planning: Efficient Translations that may be Complete Too. In *Proceedings of the 17th International Conference on Planning and Scheduling*, 2007.

[Sauro *et al.*, 2006] Luigi Sauro, Jelle Gerbrandy, Wiebe van der Hoek, and Michael Wooldridge. Reasoning about action and cooperation. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 185–192, New York, NY, USA, 2006. ACM.

[Son and Baral, 2001] T.C. Son and C. Baral. Formalizing sensing actions - a transition function based approach. *Artificial Intelligence*, 125(1-2):19–91, January 2001.

[Son *et al.*, 2006] Tran Cao Son, Chitta Baral, Nam Tran, and Sheila McIlraith. Domain-dependent knowledge in answer set planning. *ACM Trans. Comput. Logic*, 7(4):613–657, 2006.

[Spaan *et al.*, 2006] Matthijs T. J. Spaan, Geoffrey J. Gordon, and Nikos A. Vlassis. Decentralized planning under uncertainty for teams of communicating agents. In Hideyuki Nakashima, Michael P. Wellman, Gerhard Weiss, and Peter Stone, editors, *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), Hakodate, Japan, May 8-12, 2006*, pages 249–256, 2006.

[van der Hoek *et al.*, 2005] Wiebe van der Hoek, Wojciech Jamroga, and Michael Wooldridge. A logic for strategic reasoning. In Frank Dignum, Virginia Dignum, Sven Koenig, Sarit Kraus, Munindar P. Singh, and Michael Wooldridge, editors, *4rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005), July 25-29, 2005, Utrecht, The Netherlands*, pages 157–164. ACM, 2005.