Modeling Cooperative Multi-Agent Systems

Gregory Gelfond[1] and Richard Watson[2]

Texas Tech University

Abstract. Current work in the application of answer-set programming for the development of reasoning agents has largely focused on single-agent domains. A substantial body of research in this area has been compiled describing a detailed methodology for their development based upon a theoretical foundation. In this paper we present an extension of the current body of work towards the domain of cooperative multi-agent systems. The fundamental notion of an agent is extended to enable communication between agents through the introduction of special named sets of fluents known as requests. Once that has been done, we define the concepts of an agent's local and global perspectives and their respective diagrams which serve as the theoretical foundation of this work. Having established this baseline, we present an axiomatization of multi-agent communication. Finally a series of results detailing some of the properties of the framework are presented.

1 Introduction

Current work in the application of answer-set programming for the development of reasoning agents has largely focused on single-agent domains. A substantial body of research in this area has been compiled describing a detailed methodology for their development based upon a theoretical foundation.

The goal of this work is to present an overview of an extension to the existing frameworks dealing with single-agent domains towards domains containing multiple cooperating agents.

We begin by extending the principle notion of an agent. Previous work on single-agent systems has largely abstracted out the means by which an agent might communicate with the outside world. In order to represent communication between agents the definition of an agent has been extended via the addition of named sets of fluents known as *requests*. Requests provide a *language for communication* between the agents that may be present in a given system. Depending on the particular system being represented, requests may be used in the formation of *messages* which the agents may pass between themselves. Building on our redefinition of concept of an agent, we define the notions of a *collaborative multi-agent system* and a special class of actions known as *message passing* actions. Depending on the particular task at hand, an agent reasons about the effects of such actions from either a *local perspective*, or a *global perspective*. These modes of reasoning are then described by an artifact called the *communication module*, which uses a combination of action languages and logic programming under the answers-set semantics as its foundation. The particular languages used are the action language $\mathcal{AL}$, and the answer-set programming language CR-Prolog.

The rest of the introduction is structured as follows: a brief overview of the action language $\mathcal{AL}$ will be presented, followed by a description of the syntax and semantics of the relevant portions of CR-Prolog. Finally, a description of the current framework for modeling single-agent systems will be given.

1.1 The Action Language $\mathcal{AL}$

Briefly stated, action languages are a class of declarative languages used for describing the effects of actions. They have a simple syntax and semantics, and yet are powerful enough to represent many complex reasoning domains [**?**]. Collections of statements in an action language are termed *action descriptions*, and define transition diagrams whose nodes correspond to possible *states of the world*, and whose arcs are labeled by actions. An arc $\langle \sigma, a, \acute{\sigma} \rangle$ states that if an action a occurs in state σ, the resulting state will be $\acute{\sigma}$. In addition to specifying the effects of actions, it is necessary to specify what is left *unchanged* by the occurrences of actions. This is known as the *frame problem* [**?**], and its solution lies in an elegant and precise representation of the *inertia axiom*.

Action descriptions of $\mathcal{AL}$ are comprised of collections of *dynamic causal laws*, *static causal laws*, and *impossibility conditions*. Dynamic causal laws are statements of the form:

$$a \textbf{ causes } f \textbf{ if } l_0, \ldots, l_n$$

where a is an action and f and $l_0, \ldots, l_n$ are fluent literals. Laws of this form are read as "action a causes f to be true if $l_0, \ldots, l_n$." Static causal laws have the form:

$$\textbf{caused } f \textbf{ if } l_0, \ldots, l_n$$

where f and $l_0, \ldots, l_n$ are fluent literals. Static causal laws (also known as *state constraints*) are read as: "f is true whenever $l_0, \ldots, l_n$ are true." Unlike dynamic causal laws, state constraints define properties of states, rather than the direct effects of an action. They may be used however to specify the indirect effects of actions. Lastly, impossibility conditions (also known as *executability conditions*) have the form:

$$a \textbf{ impossible if } l_0, \ldots, l_n$$

where as before, a is an action and $l_0, \ldots, l_n$ are fluent literals. Rules such as this are used to state that "action a may not occur if $l_0, \ldots, l_n$ are true." From the standpoint of the transition system, such rules specify that an outgoing arc labeled by a may not originate in a state that satisfies $l_0, \ldots, l_n$.

The semantics of an action description of $\mathcal{AL}$ is given by its transition diagram. For a detailed description of the semantics of $\mathcal{AL}$, the reader is referred to [**?**].

1.2 CR-Prolog

CR-Prolog is an extension of the logic programming language A-Prolog developed by Michael Gelfond and Vladimir Lifschitz in [?] which introduces the notion of *consistency-restoring rules* and the ability to assign *preferences* over them. When taken together these new constructs allow for a more graceful handling of planning and diagnosis. For a complete description of the syntax and semantics of CR-Prolog the reader is referred to [?]. What follows is a description (taken in part from the one presented in [?] with permission of the author) of the relevant language constructs. A program of this subset of CR-Prolog is a collection of rules of the following form:

$$l_0 \; or \; \ldots \; or \; l_k \leftarrow l_{k+1}, \ldots, l_m, not\; l_{m+1}, \ldots, not\; l_n$$

and

$$r : \; l_0 \; or \; \ldots \; or \; l_k \overset{+}{\leftarrow} l_{k+1}, \ldots, l_m, not\; l_{m+1}, \ldots, not\; l_n$$

where $l_0 \ldots l_n$ are literals, *not* is *negation-as-failure* (also known as *default negation*), and r is the name of a rule. Rules of the first form are termed *regular rules*, and are read as: "if one has reason to believe in $l_{k+1}, \ldots, l_m$, and no reason to believe in $l_{m+1}, \ldots, l_n$, then one must believe in one element of $l_0 \ldots l_k$." Rules of the second form are called *consistency restoring rules*, (also known as *cr-rules*), and are read as "if one has reason to believe in $l_{k+1}, \ldots, l_m$, and no reason to believe in $l_{m+1}, \ldots, l_n$, then one may possibly believe in one element of $l_0 \ldots l_k$." In addition, there is an underlying assumption that such rules are used as little as possible.

Now that we have introduced the syntax of the language, a brief description of its semantics is given. It should be noted that this discussion assumes that the reader is familiar with the semantics of A-Prolog. Given a CR-Prolog program Π, we denote the set of regular rules of Π by Π^r. Similarly, the set of cr-rules of Π is denoted by Π^{cr}. In addition, let $\alpha(r)$ denote the regular rule obtain from the cr-rule by replacing the symbol $\overset{+}{\leftarrow}$ with $\leftarrow$. α is extended in a similar vein to apply to sets of cr-rules.

Definition 1 (abductive support). Given a CR-Prolog program Π, a minimal (with respect to set theoretic inclusion) set R of cr-rules of Π, such that $\Pi^r \cup \alpha(R)$ is consistent (i.e. has an answer set) is called an *abductive support* of Π.

Definition 2 (answer set of a CR-Prolog program). Given a CR-Prolog program Π, a set of literals, A, is called an *answer set* of Π, if it is an answer set of a regular program $\Pi^r \cup \alpha(R)$ for some abductive support R of Π.

2 The Multi-Agent Framework

2.1 Systems of Agents

When reasoning about systems which may consist of multiple agents, we typically operate under the assumption that the agents are able to communicate amongst themselves. Agents in such systems may ask other agents of the domain to perform various tasks. In order to model the ability of agents to communicate, the definition of an agent must be extended. This definition is extended by the introduction of *requests*, which present a *language for communication* between the agents in a given system. Depending on the nature of the domain being represented, these requests may be used in the formation of *messages* which the agents may pass between themselves.

Definition 3 (agent). An *agent*, α, is defined as a 4-tuple $\langle F, A, R, D\rangle$ where:

- ▹ F is a set of *fluents*.
- ▹ A is a set of *elementary actions*.
- ▹ R is a collection of named sets of fluents from F known as *requests*.
- ▹ D is an action description in the language of $\mathcal{AL}$ with signature $\Sigma = F \cup A$.

Given an agent α, F_α denotes the set of α's fluents, A_α denotes the set of α's actions, etc.

Having defined the notion of an agent, we now introduce the concepts of a *multi-agent system* and a *collaborative multi-agent system*.

Definition 4 (multi-agent system). A *multi-agent system*, M, is defined as a finite, non-empty set of agents such that:

$$\forall \alpha, \beta \in M, F_\alpha \cap F_\beta = \emptyset \wedge A_\alpha \cap A_\beta = \emptyset.$$

The conditions placed upon a multi-agent system seem restrictive and are discussed in greater detail in [**?**].

Definition 5 (collaborative multi-agent system). A *collaborative multi-agent system*, M, is defined as a multi-agent system, combined with the 3-tuple $\langle \mathcal{C}, \mathcal{S}, \mathcal{M}\rangle$ where:

- ▹ $\mathcal{C}$ is a function which given an agent α and a request $r \in R_\alpha$, returns a set of agents known as the *clients* of r.
- ▹ $\mathcal{S}$ is a function which given an agent α and a request $r \in R_\alpha$, returns a set of agents known as the *servers* of r.
- ▹ $\mathcal{M}$ is a set of ordered pairs of the form $\langle r, \beta\rangle$ known as *messages* such that $\beta \in M$, $r \in R_\alpha$ for some $\alpha \neq \beta \in M$, and $\beta \in \mathcal{S}(\alpha, r)$

In addition to the above, $\mathcal{C}$, $\mathcal{S}$, and $\mathcal{M}$ must satisfy the following properties:

- ▹ If $\beta \in \mathcal{C}(\alpha, r)$ then $\forall \gamma \in M$, $\beta \notin \mathcal{S}(\gamma, r)$.
- ▹ If $\beta \in \mathcal{C}(\alpha, r)$ then $\alpha \in \mathcal{S}(\beta, r)$.
- ▹ If $\beta \in \mathcal{S}(\alpha, r)$ then $\alpha \in \mathcal{C}(\beta, r)$.

2.2 Agent Communication

An agent's knowledge defines a pair of transition diagrams known as its *local* and *global* diagrams which are used by the agent to reason from what are termed its *local* and *global* perspectives. These perspectives differ on how actions which involve message passing are handled. These perspectives are defined by what is termed *communication module* which is split into two sub-modules: the *local module*, C_{local}, and *global module*, C_{global}. C_{local} is defined by an action description in the language $\mathcal{AL}$, while C_{global} is defined by a logic program in CR-Prolog.

The Local Perspective Consider the following scenario: "John wants to prepare for a trip. In order to do so he must pack his bags and obtain a ticket. John has a secretary that is able to obtain a ticket for him." John's reasoning about this domain could be described as follows:

- ▷ John is ready when he has his tickets and is packed.
- ▷ Packing bags causes them to be packed.
- ▷ Having his secretary purchase tickets for him causes him to have tickets.

The first two statements can be captured by the following statements of $\mathcal{AL}$:

$$\textbf{caused}\ ready\ \textbf{if}\ packed,\ ticket.$$
$$pack\ \textbf{causes}\ packed.$$

How could we represent the final statement? When we represent agents we only represent their knowledge of the domain. John is not concerned with how his secretary obtains the tickets. Rather, he only knows that when asked, the secretary obtains the tickets for him. This line of reasoning can be captured by the following dynamic causal law:

$$ask\ \textbf{causes}\ ticket.$$

We call this *reasoning from a local perspective*. When reasoning in this perspective an agent abstracts out all of the details concerning the actions other agents may take. The sole focus is how his requests affect his own model of the world. This approach is generalized through the use of named sets of fluents called *requests*. To make this law more general, and therefore applicable to other commands that John may at some point issue to his secretary, we introduce into our representation the request $buy(ticket, john)$ which contains the fluent $ticket$. Once this is accomplished, we represent this final statement via the following dynamic causal law:

$$send(Request)\ \textbf{causes}\ Fluent\ \textbf{if}\ Fluent \in Request.$$

This is precisely what the local communication module, C_{local} accomplishes. The representation of John in our framework is given below:

- ▷ $F_{john} = \{ticket, packed, ready\}$.

- ▷ $A_{john} = \{pack\}$.
- ▷ $R_{john} = \{buy(ticket, john) = \{ticket\}\}$.
- ▷ $D_{john} = \left\{ \begin{array}{l} \textbf{caused}\ ready\ \textbf{if}\ packed,\ ticket. \\ pack\ \textbf{causes}\ packed. \end{array} \right\}$

and the causal law:

$$send(Request)\ \textbf{causes}\ Fluent\ \textbf{if}\ Fluent \in Request.$$

is part of C_{local}.

When taken together D_{john} and C_{local} define what is known as John's *local diagram*.

Definition 6 (local diagram). Given an agent α, α's *local diagram*, $T_{local}(\alpha)$, is the diagram defined by the action description $D_\alpha \cup C_{local}$.

Example 1. Using our previous description of John, $D_{john} \cup C_{local}$ is as follows:

$$D_{john} \cup C_{local} = \left\{ \begin{array}{l} \textbf{caused}\ ready\ \textbf{if}\ packed,\ ticket. \\ pack\ \textbf{causes}\ packed. \\ send(Request)\ \textbf{causes}\ Fluent\ \textbf{if}\ Fluent \in Request. \end{array} \right\}$$

Which gives the following local diagram for John:

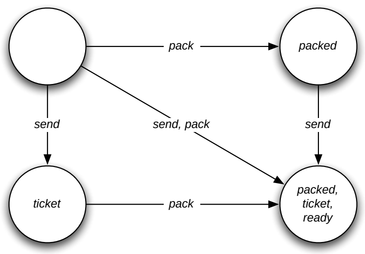

Fig. 1. local diagram for agent John

The Global Perspective As we mentioned previously, an agent's local perspective is used by the agent to reason about his own actions. However, it is at times necessary for an agent to reason about how his actions may *actually play out*. It is highly unlikely that John's secretary is capable of obtaining his tickets instantly. Consequently John knows that he must wait for some unspecified period of time, depending on how many other tasks his secretary has to perform. His new model is described as follows:

- ▷ John is ready when he has his tickets and is packed.

▷ Packing bags causes them to be packed.
▷ Issuing a request causes that request to become pending.
▷ If his request is satisfied, its corresponding effect is satisfied.
▷ Until his request is satisfied John has to wait.

Again, John is not concerned with the actual actions that his secretary may perform to obtain the tickets. He does however need to understand that he must wait for her to get them. This type of reasoning is termed *reasoning from a global perspective*. With the exception of the first and last statements, the above rules are not expressible in $\mathcal{AL}$. They are however easily represented in A-Prolog (and hence in CR-Prolog), and their representation comprises the module C_{global}.

Taken together, D_{john} and C_{global} define what is known as John's *global diagram*.

Definition 7 (global diagram). Given an agent α, α's *global diagram*, $T_{global}(\alpha)$, is the transition diagram defined by the following logic program:

$$\Pi_{global}(\alpha) = \Pi_{\alpha} \cup \Pi_{inertia} \cup \Pi_{effects} \cup \Pi_{default} \cup C_{global}.$$

Where Π_{α} is a logic program representation of the agent; $\Pi_{inertia}$ is a logic program describing inertia; $\Pi_{effects}$ is a logic program describing the general effects of actions; and $\Pi_{default}$ is a logic program characterizing the behavior of default fluents. An in depth presentation of these modules is given in [**?**].

Example 2. Using our previous description of John, a high level presentation of $D_{john} \cup C_{global}$ is given below:

$$D_{john} \cup C_{global} = \left\{ \begin{array}{l} \textbf{caused } ready \textbf{ if } packed,\ ticket. \\ pack \textbf{ causes } packed. \\ send(Request) \textbf{ causes } pending(Request). \\ pending(Request) \textbf{ triggers } wait(Request). \\ wait(Request) \textbf{ causes } satisfied(Request) \textbf{ or } \neg satisfied(Request). \\ \textbf{caused } Fluent \textbf{ if } satisfied(Request),\ Fluent \in Request. \end{array} \right\}$$

Which describes the global diagram presented in figure 2 (note that the notation has been simplified somewhat).

The System Perspective Every agent of a multi-agent system has its own local and global perspectives of the domain that it uses to reason about the effects of its actions. If we take a step back however, and look at the system of agents as a whole, the global perspectives of the agents may be combined to form what is termed the *system perspective*, characterized by the *system diagram* defined as follows:

Definition 8 (system diagram of a collaborative multi-agent system). Given a collaborative multi-agent system, M, M's *system diagram*, $T_{system}(M)$, is the transition diagram defined by the following logic program:

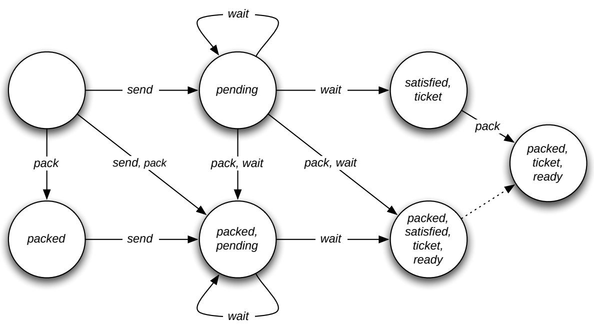

Fig. 2. global diagram for agent John

$$\Pi_{system}(M) = (\bigcup_{\alpha \in M} \Pi_\alpha) \cup \Pi_{inertia} \cup \Pi_{effects} \cup \Pi_{default} \cup C_{global}.$$

The system diagram is used to describe the behavior of a multi-agent system as a whole, showing the interactions of all the agents of the system. This diagram is used primarily in diagnosis (where it plays the same role as the actual transition diagram mentioned in [**?**]), and in analyzing the properties of a given multi-agent system.

2.3 The Multi-Agent Loop

When describing single-agent systems, an agent's behavior is characterized by what has been termed the agent-loop [**?**] (also known as the observe-think-act loop). Essentially, the agent loops through the following steps:

- ▷ observe the state of the world
- ▷ if these observations fail to match up with the agent's expectations, identify the reason for the discrepancy
- ▷ select a goal
- ▷ generate a sequence of actions to achieve the goal
- ▷ execute the first element of the sequence

In our framework for multi-agent systems the structure of the loop remains unchanged. What does change is how the agent performs the steps of *planning* and *diagnosis*. This is where local and global perspectives play come into play.

In the single-agent version of the loop, planning is reduced to finding a path from the agent's current/initial state to a goal state in the agent's transition diagram. This basic template is observed in our framework as well, except that path generated is from the agent's local diagram, rather than the diagram described by the agent's action description.

Example 3. Recall that our previous description of John, yielded the local diagram described in figure 1. Suppose that John has the goal of becoming *ready*. During the planning phase of the multi-agent loop John could generate the following trajectory, π as a possible plan:

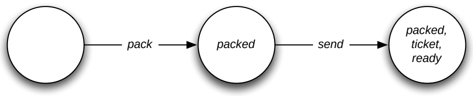

Fig. 3. possible plan, π, for achieving *ready*

Once a plan has been generated, the agent uses its global diagram to generate a sequence of actions describing how the execution of its plan may play out in the presence of other agents. This sequence forms the agent's expectations concerning the results of its actions, and is captured by the notion of an *expansion of a path*.

Definition 9 (expansion of a transition). Let $\tau = \langle \sigma, a, \acute{\sigma} \rangle$ be a transition in $T_{local}(\alpha)$ for some agent α. The *expansion* of τ in $T_{global}(\alpha)$, denoted by $\acute{\tau}$, is defined as follows:

- ▷ If $a \cap A_\alpha = a$, then $\acute{\tau}$ is a transition $\langle \delta, \acute{a}, \acute{\delta} \rangle \in T_{global}(\alpha)$ where $\sigma \subseteq \delta$, $\acute{\sigma} \subseteq \acute{\delta}$, and $a \subseteq \acute{a}$, where $\acute{a} \setminus a = \emptyset$, or $\acute{a} \setminus a$ only contains actions of type *wait*.
- ▷ If a contains an action of type *send* whose corresponding message is m, $\acute{\tau}$ is a path in $T_{global}(\alpha)$ of the form:

$$\langle \delta, \acute{a}, \delta_0, a_0, \ldots, a_n, \acute{\delta} \rangle$$

where $a_0, \ldots, a_n$ are actions of type *wait* whose corresponding message is m, and $\sigma \subseteq \delta$, $\acute{\sigma} \subseteq \acute{\delta}$, and $a \subseteq \acute{a}$.

Definition 10 (expansion of a path). Let $\pi = \langle \sigma_1, a_1, \sigma_2, a_2, \ldots, a_n, \sigma_n \rangle$ be a path in $T_{local}(\alpha)$ for some agent α. The *expansion* of π in $T_{global}(\alpha)$, is any path $\acute{\pi} = \beta \circ \gamma \in T_{global}(\alpha)$ such that:

- ▷ β is an expansion of $\langle \sigma_1, a_1, \sigma_2 \rangle$ whose final state is δ_2
- ▷ γ is an expansion of $\langle \sigma_2, a_2, \ldots, a_n, \sigma_n \rangle$ whose initial state is δ_2

Example 4. Given the possible plan from example 3, John could use the following expansion to form his expectations of how the plan might actually be realized:

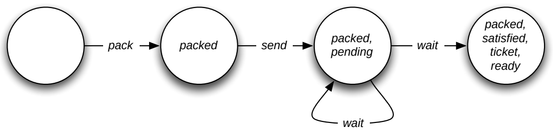

Fig. 4. an expansion of π from example 3

3 Properties of the Framework

In this section we present several properties of multi-agent systems that are reflected in various characteristics of an agent's local and global diagrams.

3.1 Feasibility of Plans

Once an agent selects a goal, he then uses his local perspective to determine a plan which will achieve it. Subsequently, he uses his global perspective to determine how the plan may play out in the domain by determining its possible expansions. Is this always possible however? Do paths in an agent's local diagram always have expansions in the global diagram? In this section we present some results showing that this is the case, that the paths of an agent's local diagram are guaranteed to have at least one expansion in the agent's global diagram. For proofs of the subsequent theorems the reader is referred to [**?**].

We begin by with a result stating that the states of an agent's local diagram may be extended to obtain states in the agent's global diagram.

Lemma 1. Let α be an agent, and let σ be a state of $T_{local}(\alpha)$. Then there exists a state $\delta \in T_{global}(\alpha)$ such that $\sigma \subseteq \delta$.

A similar pair of results specifying that the arcs of an agent's local diagram may be extended to the global diagram are given below:

Lemma 2. Let α be an agent, ϵ be a single elementary action such that $type(\epsilon) \neq send$, and let $\tau = \langle \sigma, \epsilon, \acute{\sigma} \rangle$ be a transition in $T_{local}(\alpha)$. Then for any state $\delta \in T_{global}(\alpha)$ such that $\sigma \subseteq \delta$, there exists an expansion $\acute{\tau} = \langle \delta, \acute{\epsilon}, \acute{\delta} \rangle \in T_{global}(\alpha)$ of τ.

Lemma 3. Let α be an agent, a be a single elementary action such that $type(a) = send$, and let $\tau = \langle \sigma, a, \acute{\sigma} \rangle$ be a transition in $T_{local}(\alpha)$. There exists an expansion $\acute{\tau} = \langle \delta, a, \delta_0, a_0, \ldots, a_n, \acute{\delta} \rangle \in T_{global}(\alpha)$ of τ, where $a_0, \ldots, a_n$ are actions of type *wait* whose message corresponds to that of a, and $\sigma \subseteq \delta$, and $\acute{\sigma} \subseteq \acute{\delta}$.

Lastly, the relationship between the paths of an agent's local and global diagrams is given by the following theorem.

Theorem 1. Let α be an agent, and let π be a path whose arcs are labeled by elementary actions in $T_{local}(\alpha)$. Then there exists an expansion $\acute{\pi}$ of π in $T_{global}(\alpha)$.

4 Conclusions and Future Work

In this work we presented a general overview of a framework for reasoning about cooperative multi-agent systems. We began by extending the fundamental notion of an agent to facilitate communication via the introduction of requests. We then introduced the notions of an agent's local and global perspectives and their respective diagrams. These form the basis of the framework by providing a foundation upon which an agent perceives the world around him. Together these diagrams are used by an agent for planning, and reasoning about their execution. In addition we introduced the system diagram, which is used as a means of modeling the "actual state of the world" for the purposes of diagnosis and reasoning about the system at large. Having introduced the basic framework we then presented an axiomatization of reasoning about agent communication. An example detailing a simple system was presented followed by a series of results describing the relationship between the paths in an agent's local diagram and those of it's global diagram.