Reasoning and Planning with Cooperative Actions for Multiagents Using Answer Set Programming

Tran Cao Son[1] and Chiaki Sakama[2]

[1] Dept. of Computer Science, New Mexico State University, Las Cruces, NM 88003, USA
`tson@cs.nmsu.edu`
[2] Computer and Communication Sciences, Wakayama University, Wakayama 640-8510, Japan
`sakama@sys.wakayama-u.ac.jp`

Abstract. In this paper, we investigate the multiagent planning problem in the presence of cooperative actions and agents, which have their own goals and are willing to cooperate. To this end, we extend the action language $\mathcal{A}$ in [12] to represent and reason about plans with cooperative actions of an individual agent operating in a multiagent environment. We then use the proposed language to formalize the multiagent planning problem and the notion of a joint plan for multiagents in this setting. We discuss a method for computing joint plans using answer set programming and provide arguments for the soundness and completeness of the implementation.

1 Introduction

Cooperative actions are actions of an agent which can be executed only if the agent is operating in a multiagent environment. They can be actions for soliciting something from other agents or actions for setting up some conditions for other agents. They differ from individual actions in that they might affect other agents. Cooperative actions are important not only in situations where multiple agents have to work together to accomplish a common goal but also in situations where each agent has its own goal. This can be seen in the following story, a modified version of the story in [21]:

Example 1. Three new students A, B, and C are moving in a shared apartment and planning to decorate their rooms. Each would like to hang one of their objects on the wall, e.g., A would like to hang a mirror, B a diploma, and C a painting. A and B know how to use either a nail or a screw to complete their job but C knows to use the screw only. A has neither a nail or a screw. B has both. C has only a nail. To use a nail, one will need a hammer. Among three, only B has a hammer.

Do the students have a joint-plan that allows each of them to achieve his/her goal?

Intuitively, we can see that only B can accomplish her job independent of A and C. The three can achieve their goals if B uses the hammer and the nail to hang her diploma then gives A the hammer and C the screw, respectively. C, on the other hand, gives A the nail and uses the screw to hang her painting. A uses the nail (from C) and the hammer (from B) to hang her mirror. Of course, to avoid unpleasant moments, A should ask for the nail (from C) and the hammer (from B) and C should ask for the screw (from B).

However, it is easy to see that if either B or C does not want to give out anything, then only B can achieve her goal. Furthermore, if B decides to use the screw instead of using the nail in hanging her diploma, then C has no way of achieving her goal. □

In the above example, the action of giving a nail, a hammer, or a screw between the students can be considered as cooperative actions. The action of requesting something from others can also be considered as cooperative actions. It is obvious that without some cooperative actions, not all students can achieve their own goals. Even with the cooperative actions at their disposal, the students might still need to coordinate in creating their corresponding plans.

In Example 1, agents (the students) maintain their own local worlds and their actions do generally not affect others' worlds. It should be emphasized that the fact that agents have their own world representation does not exclude the situations in which the worlds of different agents overlap and the execution of one agent's individual actions might affect others as well or the execution of their joint-action.

Example 2. Let us consider A and B who are in one room and studying at their tables. Each of them sits next to a switch which can control the lamp in the room. Flipping either switch will change the status of the light.

Assume that A and B maintain their world representation separately. (They might use the same theory for this purpose but we will not impose this.) Obviously, if A flips the switch next to her, the world in which B is in will also change.

Similarly, if A and B lift a table and place it at different location, their joint-action change the world of both as well. □

In this paper, we will consider multiagent planning problems in which each agent maintains its own representation about the world and its capabilities, which includes individual actions and cooperative actions; and has its own goal. We are mainly interested in the process of creating a joint plan prior to its execution. We will begin by extending the language $\mathcal{A}$ in [12] to allow cooperative actions for a single agent. The semantics of the new language is defined by a transition function which maps pairs of actions and states to states. We then define the multiagent planning problems and the notion of a joint plan for multiagents in presence of cooperative actions. Finally, we discuss a method for computing joint plans using answer set programming [18, 19].

2 An action language with cooperative actions

In this section, we present a language for representing and reasoning about plans for an agent in the multiagent environment with cooperative actions. To this end, we extend the language $\mathcal{A}$ in [12] to allow cooperative actions[1]. In this paper, we consider cooperative actions as actions that an agent would not have if she were in a single agent environment. Specifically, we consider two types of cooperative actions, one that requests the establishment of a condition in an agent's world and another establishes some conditions in the world of another agent. We will assume an arbitrary but fixed set of agent identifiers $\mathcal{AG}$. A planning problem of an agent in $\mathcal{AG}$ is defined over a set of fluents (or state variables) `F`, a set of *individual actions* `A`, and a set of *cooperative*

[1] The choice of $\mathcal{A}$ will be discussed in Section 5.

actions C. We will assume that A always contains a special action `wait` which does not have any effect on the agent's world[2]. Furthermore, we will require that actions in C do not appear in A. This highlights the fact that the cooperative actions are presented due to the presence of other agents.

A *fluent literal* (or *literal*) is either a fluent or its negation. Fluent formulas are propositional formulas constructed from literals and propositional connectives.

2.1 Specifying Individual Actions

A *domain specification* DI over F and A describes the individual actions of an agent and consists of laws of the following form:

$$a \textbf{ causes } l \textbf{ if } \phi \tag{1}$$

$$a \textbf{ executable } \phi \tag{2}$$

where a is an individual action (in A), l is a fluent literal and ϕ is a set of fluent literals.

A law of the form (1), called a *dynamic law*, states that if a is executed when ϕ is true then l becomes true. (2) is an *executability condition* and says that a can be executed only if ϕ is true. The semantics of a domain specification is defined by the notion of *state* and by a *transition function* Φ, that specifies the result of the execution of an action a in a state s.

A set of literals S satisfies a literal l (l holds/is true in S), denoted by $S \models l$, if $l \in S$. For a set of literals ϕ, $S \models \phi$ if $S \models l$ for every $l \in \phi$. A *state* s is a set of fluent literals that is *consistent* and *complete*, i.e., for every $f \in$ F, either $f \in s$ or $\neg f \in s$ but $\{f, \neg f\} \not\subseteq s$. In the following, $\overline{l}$ denotes the negation of l, i.e., if $l = f$ and $f \in$ F, then $\overline{l} = \neg f$; if $l = \neg f$ for some $f \in$ F, then $\overline{l} = f$. For a set of literals S, $\overline{S} = \{\overline{l} \mid l \in S\}$.

An action a is *executable* in a state s if there exists an executability condition (a **executable** ϕ) in DI such that $s \models \phi$.

Let $e_a(s) = \{l \mid \exists(a \textbf{ causes } l \textbf{ if } \phi) \in DI.[s \models \phi]\}$. The result of the execution of a in s is defined by

- $\Phi(a, s) = \textit{fails}$ if a is not executable in s; and
- $\Phi(a, s) = (s \setminus \overline{e_a(s)}) \cup e_a(s)$ if a is executable in s.

A domain specification DI is *consistent* if $\Phi(a, s) \neq \textit{fails}$ holds for every pair of action a and state s such that a is executable in s.

Φ is extended to reason about effect of a sequence of actions as follows.

Definition 1 (Transition function). *Let DI be a domain specification, s be a state, and $\alpha = [a_1; \ldots; a_n]$ be a sequence of actions.*

- $\hat{\Phi}(\alpha, s) = s$ *if* $n = 0$*;*
- $\hat{\Phi}(\alpha, s) = \Phi(a_n, \hat{\Phi}([a_1; \ldots; a_{n-1}], s))$*, otherwise*

where $\Phi(a, \textit{fails}) = \textit{fails}$*.*

An agent can use the transition function to reason about effects of its actions and to planning. An action sequence α is a *plan* achieving a set of literals O from a state I iff O is true in $\hat{\Phi}(\alpha, I)$.

[2] We envision a multiagent environment where agents may have to wait for other agents to finish some actions before they can go on with their course of actions.

Example 3. The domain specification DI_A for A in Example 1 is defined over $\mathtt{F}_A = \{h\_nail, h\_screw, mirror\_on, h\_ham\}$ and $\mathtt{A}_A = \{hw\_nail, hw\_screw\}$ with the set of laws[3]:

hw_nail	**causes**	$mirror\_on$	hw_screw	**causes**	$mirror\_on$
hw_nail	**causes**	$\neg h\_nail$	hw_screw	**causes**	$\neg h\_screw$
hw_nail	**executable**	h_nail, h_ham	hw_screw	**executable**	h_screw

In all of the above, the prefix "hw" stands for "hang with" and "h" stans for "has." □

2.2 Specifying Cooperative Actions

The specification of the set of cooperative actions of an agent, denoted byDC, is defined over C and F and consists of laws of the following form:

$$r \textbf{ requests } \gamma \textbf{ from } \mathcal{A}_\mathtt{i} \textbf{ may\_cause } \phi \textbf{ if } \psi \text{ and} \tag{3}$$

$$p \textbf{ provides } \gamma \textbf{ for } \mathcal{A}_\mathtt{i} \textbf{ causes } \phi \textbf{ if } \psi \tag{4}$$

r and p are action names in C, γ, ϕ, and ψ are sets of literals and $\gamma \subseteq \phi$, and $\mathcal{A}_\mathtt{i}$ is a set of agent identifiers in $\mathcal{AG}$. r is called a *request* for γ and p an *offer* for γ. Since these actions are intended to address other agents, we require that the identifier of the agent having r and/or p does not belong to $\mathcal{A}_\mathtt{i}$. Furthermore, for a request-action, we require that $\bar{\phi} \cap \psi \neq \emptyset$ which indicates that an agent will only request for something that he/she does not have.

Intuitively, (3) represents a set of requests that can be made by the agent; if the agent makes the request for γ (which is the action r) directed to an agent in $\mathcal{A}_\mathtt{i}$ then ϕ might become true. The condition $\gamma \subseteq \phi$ guarantees that requested literals (γ) are true if the request is satisfied (ϕ). Furthermore, the action can only be executed if ψ is true. For this reason, we call $r(\gamma, i)$, $i \in \mathcal{A}_\mathtt{i}$, an instance of a request (3). Similarly, (4) represents the set of offers $p(\gamma, i)$, $i \in \mathcal{A}_\mathtt{i}$. This offer addresses a request made to the agent by establishing γ (for the requestor). This action is similar to the individual actions in A of an agent. The main difference is that they also change the worlds of other agents. It can only be executed if ψ is true and its effects is ϕ.

For simplicity of the presentation, we will assume that each action in C occurs in at most one law of the form (3) or (4). We use *cooperative action* to refer to either a request- or an offer-action. When $\mathcal{A}_\mathtt{i}$ is the set of all other agents, we often omit the part '**from** $\mathcal{A}_\mathtt{i}$' from (3) and '**for** $\mathcal{A}_\mathtt{i}$' from (4).

Example 4. In Example 1, it is reasonable for A to request and/or offer other agents on the literal h_nail. An action for requesting for (offering of) h_nail for A can be specified by

$give\_me\_nail$ **requests** h_nail **from** $\{B, C\}$ **may_cause** h_nail **if** $\neg h\_nail$
get_this_nail **provides** h_nail **for** $\{B, C\}$ **causes** $\neg h\_nail$ **if** h_nail

where $give\_me\_nail$ is a request-action and get_this_nail is an offer-action. If the agent A wants to ask for help, then her set of cooperative actions needs to include the

[3] To simplify the representation, we often write $l_1, \ldots, l_n$ instead of $\{l_1, \ldots, l_n\}$ in describing the domain.

action $give\_me\_nail$. On the other hand, if she wants to help others, then it should include the action get_this_nail. □

Definition 2 (Planning problem with cooperative actions). *A* planning problem with cooperative actions[4] $\mathcal{P}$ *is a tuple* $\langle DI, I, O, DC \rangle$ *where* DI *is a domain specification,* I *is a state representing the initial state,* O *is a set of literals representing the goal, and* DC *is a set of laws of the form (3) and (4).*

Given a planning problem $\mathcal{P} = \langle DI, I, O, DC \rangle$, we need to specify what is a "plan" achieving O in the presence of the cooperative actions. Intuitively, we could consider these actions as the actions of the agent and use the notion of a plan mentioned in the previous subsection. This is, however, not enough since an agent, when executes a request, might or might not receive an offer satisfying his/her request. For example, a request for a nail from A to C might not result in A having the nail because C has already given the nail to B.

We will therefore extend the transition function Φ of the domain specification DI to consider cooperative actions. We will use Φ_D to denote the transition function of $DI \cup DC$. By assuming that cooperative actions are different from the individual actions (i.e., $\mathtt{A} \cap \mathtt{C} = \emptyset$), it suffices to specify what is the result of the execution of a request/offer-action in a given state.

For simplicity of the presentation, we assume that each individual agent executes only one action at a time. The method presents in this paper can be easily extended to the case where individual agents can execute parallel actions.

Let s be a state. We say that an instance $r(\gamma, i)$ of a request-action specified by the law

$$r \textbf{ requests } \gamma \textbf{ from } \mathcal{A}_{\mathrm{i}} \textbf{ may\_cause } \phi \textbf{ if } \psi$$

in DC is *executable* in s if ψ is true in s. Executing the action $r(\gamma, i)$ in s does not guarantee that the agent will obtain ϕ in the resulting state. This is because the agent, whom the request was made to, might not have the capability to establish ϕ for the requestor. We say that the execution of $r(\gamma, i)$ in s might or might not succeed. As such, the result of executing $r(\gamma, i)$ in s is either s, representing the case when the request is not satisfied (by the agent whom the request was made to); or $(s \setminus \overline{\phi}) \cup \phi$, representing the case when the request is satisfied.

Remark 1. Observe that under the assumption that an agent will execute a request-action only when it is necessary (i.e., $\overline{\phi} \cap \psi \neq \emptyset$), we have that $s \neq (s \setminus \overline{\phi}) \cup \phi$ for every instance $r(\gamma, i)$. This allows us to recognize when a request is satisfied.

An instance $p(\gamma, i)$ of an offer-action specified by the law

$$p \textbf{ provides } \gamma \textbf{ for } \mathcal{A}_{\mathrm{i}} \textbf{ causes } \phi \textbf{ if } \psi$$

in DC is *executable* in s if ψ is true in s. The state resulting from executing $p(\gamma, i)$ in s is given by $(s \setminus \overline{\phi}) \cup \phi$.

[4] For simplicity of presentation, we will use planning problem instead of planning problem with cooperative actions whenever no confusion is possible.

Definition 3 (Transition function). *The transition function Φ_D over $DI \cup DC$, a mapping from pairs of actions and states to sets of states, is defined as follows. Let s be a state.*

- *For $a \in \mathtt{A}$, $\Phi_D(a, s) = \{\Phi(a, s)\}$ if $\Phi(a, s) \neq fails$; otherwise, $\Phi_D(a, s) = \emptyset$.*
- *For an instance of an offer-action $p(\gamma, i)$, $\Phi_D(p(\gamma, i), s) = \{(s \setminus \overline{\phi}) \cup \phi\}$ if p is executable in s; otherwise, $\Phi_D(p, s)$=$\emptyset$.*
- *For an instance of a request-action $r(\gamma, i)$, $\Phi_D(r(\gamma, i), s) = \{s, (s \setminus \overline{\phi}) \cup \phi\}$ if $r(\gamma, i)$ is executable in s; otherwise, $\Phi_D(r(\gamma, i), s) = \emptyset$.*

Remark 2. The definition of Φ_D assumes that each cooperative action occurs in only one law of the form (3) or (4). The definition can be extended to remove this restriction by (*i*) defining a set $ec_{r(\gamma,i)}(s)$ (resp. $ec_{p(\gamma,i)}(s)$), similar to the definition of the set of effects of an action $e_a(s)$ and (*ii*) changing the definition accordingly.

The transition function is extended to reason about plans as follows.

Definition 4 (Plan with cooperative actions). *Let $\mathcal{P}$ be a planning problem $\langle DI, I, O, DC \rangle$. We define*

- *A sequence $s_0, a_0, s_1, \dots, a_{n-1}, s_n$, where s_i's are states and a_j's are actions, is a* trajectory *if $s_{i+1} \in \Phi_D(a_i, s_i)$ for $0 \leq i < n$.*
- *A trajectory $s_0, a_0, s_1, \dots, a_{n-1}, s_n$ is a* possible plan *achieving O (or a solution of $\mathcal{P}$) if $s_0 = I$ and $s_n \models O$.*
- *An occurrence of a request $r(\gamma, i) = a_j$ in a trajectory $s_0, a_0, s_1, \dots, a_{n-1}, s_n$ is* satisfied *if $s_{j+1} \neq s_j$; otherwise, the request is said to be unsatisfied.*

Notice that the third item in the above definition is sensible due to Remark 1. A trajectory satisfying the goal O of the planning problem is a solution of $\mathcal{P}$ if all satisfied requests assumed in the trajectory indeed materialized, i.e., for each satisfied $r(\gamma, i)$ in the trajectory, the agent i executes the action $p(\gamma, j)$ (j is the identifier of the agent issuing the request). The topic of coordination between agents will be discussed in the next section.

Example 5. Let $\mathcal{P}_A = \langle DI_A, I_A, O_A, DC_A \rangle$ be the planning problem for A with DI_A (Example 3), $I_A = \{\neg h\_nail, \neg h\_screw, \neg h\_ham, \neg mirror\_on\}$ and $O_A = \{mirror\_on\}$, and DC_A is the set of actions *give_me_nail* and *get_this_nail* whose specifications are given (Example 4) and the two actions

$give\_me\_ham$ **requests** h_ham **from** $\{B, C\}$ **may_cause** h_ham **if** $\neg h\_ham$,
get_this_ham **provides** h_ham **for** $\{B, C\}$ **causes** $\neg h\_ham$ **if** h_ham.

We can easily check the following:

- for $n \leq 2$, the problem has no possible plan.
- for $n = 3$, $\mathcal{P}_A$ has a possible plan which is the following trajectory:
  $s_0^A, give\_me\_nail(h\_nail, C), s_1^A, give\_me\_ham(h\_ham, B), s_2^A, hw\_nail, s_3^A$
  where $s_0^A = \{\neg h\_nail, \neg h\_ham, \neg h\_screw, \neg mirror\_on\}$,
  $s_1^A = \{h\_nail, \neg h\_ham, \neg h\_screw, \neg mirror\_on\}$,
  $s_2^A = \{h\_nail, h\_ham, \neg h\_screw, \neg mirror\_on\}$,
  $s_3^A = \{\neg h\_nail, h\_ham, \neg h\_screw, mirror\_on\}$. □

3 Planning for Multiagents

In a multiagent environment, each agent needs to know her capabilities. She also needs to know from whom she can ask for some favors or to whom she could offer helps. Furthermore, it is also common that groups of agents need to know about their joint capabilities. It is also possible that agents might talk the same language. This can be summarized as follows.

- Each agent has its own planning problem, which is described in the previous section.
- The agent might or might not share the same world representation. By default, the world representation of the agent is local. For example, the three agents in Example 1 can use the same set of fluents and actions; and A has $\neg h\_nail$ in her initial state whereas B has h_nail in hers, yet this is not a contradictory statement about the world since the fluents are local. On the other hand, the two agents in Example 2 share certain features (e.g. the light) and therefore the fluents encoding these features should have the same value in their representations.
- An agent might request another agent to establish certain conditions in her own world. For example, A might request B to establish h_nail to be true for her.
- An agent might execute some actions that change the local world of another agent. For example, B can give A the nail, thus establishing h_nail in the world of A.
- There might be actions that a set of agents should not execute in parallel. For example, two cars– one goes north-south and another east-west– cannot cross an intersection at the same time.
- There might be actions that a set of agents need to execute in parallel. For example, the action of lifting a table by two agents need to be done in parallel.

It turns out that the language developed in the previous section can be extended to represent and reason about plans/actions of agents in a multiagent environment. With the help of the notion of a planning problem with cooperative actions, a multiagent planning problem can be defined as follows.

Definition 5 (Multiagent planning problem). *A* multiagent planning problem $\mathcal{M}$ *is a tuple* $\langle \mathcal{AG}, \{\mathcal{P}_i\}_{i \in \mathcal{AG}}, \mathcal{F}, \mathcal{IC}, \mathcal{C} \rangle$ *where*

- $\mathcal{AG}$ *is a set of agents,*
- $\mathcal{P}_i$ *is a planning problem with cooperative actions for each agent* $i \in \mathcal{AG}$*,*
- $\mathcal{F}$ *is the set of tuples of the form* (i, j, f_i, f_j) *where* $i, j \in \mathcal{AG}$ *and* $f_i \in \mathtt{F}_i$ *and* $f_j \in \mathtt{F}_j$*, and*
- $\mathcal{IC}$ *and* $\mathcal{C}$ *are sets of sets of agent action pairs of the form* (i, a_i) *where* i *is an agent and* a_i *is an action in* $\mathtt{A}_i$*.*

Intuitively, each tuple (i, j, f_i, f_j) indicates that f_i and f_j represent the same state variable in the worlds of two agents i and j and can be changed by either i or j. This mean that they should have the same value in every state of i and j. A set of agent-action pairs $\{(i_1, a_{i_1}), \ldots, (i_k, a_{i_k})\} \in \mathcal{IC}$ indicates that the agents $i_1, \ldots, i_k$ cannot execute the actions $a_{i_1}, \ldots, a_{i_k}$ at the same time. On the other hand, a set of agent-action pairs $\{(i_1, a_{i_1}), \ldots, (i_k, a_{i_k})\} \in \mathcal{C}$ indicates that the agents $i_1, \ldots, i_k$ must execute the actions $a_{i_1}, \ldots, a_{i_k}$ concurrently for their effects to be materialized. The sets $\mathcal{F}$, $\mathcal{IC}$, and $\mathcal{C}$ are called constraints of $\mathcal{M}$.

Example 6. The planning problem in Example 1 can be represented by $\mathcal{M}_1 = \langle \{A, B, C\}, \{\mathcal{P}_A, \mathcal{P}_B, \mathcal{P}_C\}, \emptyset, \emptyset, \emptyset \rangle$ where

- A, B, and C are the students from Example 1;
- $\mathcal{P}_A$ is defined as in Example 5;
- $\mathcal{P}_B = \langle DI_B, I_B, O_B, DC_B \rangle$ where DI_B is defined over $\mathtt{F}_B = \{h\_nail, h\_screw, diploma\_on, h\_ham\}$ and $\mathtt{A}_B = \{hw\_nail, hw\_screw\}$ with the set of laws:
  hw_nail **causes** $diploma\_on$ $\quad$ hw_nail **causes** $\neg h\_nail$
  hw_nail **executable** h_ham, h_nail $\quad$ hw_screw **causes** $diploma\_on$
  hw_screw **causes** $\neg h\_screw$ $\quad$ hw_screw **executable** h_screw
  $I_B = \{h\_nail, h\_screw, h\_ham, \neg diploma\_on\}$ and $O_B = \{diploma\_on\}$, and DC_B contains cooperative actions similar to that in DC_A and DC_C (below).
- $\mathcal{P}_C = \langle DI_C, I_C, O_C, DC_C \rangle$ where DI_C is defined over

$$\mathtt{F}_C = \{h\_nail, h\_screw, painting\_on\}$$

$$\mathtt{A}_C = \{hw\_screw\}$$

  with the set of laws: hw_screw **causes** $painting\_on$
  hw_screw **causes** $\neg h\_screw$ $\quad$ hw_screw **executable** h_screw
  $I_C = \{h\_nail, \neg h\_screw, \neg painting\_on\}$, $O_C = \{painting\_on\}$, and DC_C contains the following laws:

  $give\_me\_screw$ **requests** h_screw **from** $\{A, B\}$ **may_cause** h_screw **if** $\neg h\_screw$
  get_this_screw **provides** h_screw **for** $\{A, B\}$ **causes** $\neg h\_screw$ **if** h_screw □

We now define the notion of a solution for a planning problem.

Definition 6 (Joint plan for multiagents). *Let $\mathcal{M} = \langle \mathcal{AG}, \{\mathcal{P}_i\}_{i \in \mathcal{AG}}, \mathcal{F}, \mathcal{IC}, \mathcal{C} \rangle$ be a multiagent planning problem. For each $i \in \mathcal{AG}$, let $S_i = [s_0^i a_0^i, \ldots, a_{n-1}^i s_n^i]$ be a possible plan of $\mathcal{P}_i$. We say that $\{S_i\}_{i \in \mathcal{AG}}$ is a* joint plan *(or* solution*) of length n for $\mathcal{M}$ if for every $0 \leq k \leq n$:*

- *for each instance of a request $a_k^i = r(\gamma, j)$ that is satisfied in S_i, we have that $a_k^j = p(\gamma, i)$;*
- *for each $(i, j, f_i, f_j) \in \mathcal{F}$, $f_i \in s_k^i$ iff $f_j \in s_k^j$;*
- *for each $S \in \mathcal{IC}$, there exists some $(i, a) \in S$ such that $a_k^i \neq a$; and*
- *for each $S \in \mathcal{C}$, either $\{a \mid (i, a) \in S \text{ and } a = a_k^i\} = \{a \mid (i, a) \in S\}$ or $\{a \mid (i, a) \in S \text{ and } a = a_k^i\} = \emptyset$.*

Intuitively, a joint plan is composed of individual plans which allow the agents to achieve their own goals and satisfy the various constraints of the problem. In the process, agents can help each other in establishing certain conditions. However, if a request of an agent is assumed (by the requestor) to be satisfied within a joint plan then the joint plan must also contain an agent who actually executes an offer action satisfying the request (first item). The second item states that the individual plans must agree with each other on their effects of shared fluents, i.e., it enforces the constraints in $\mathcal{F}$. The third and fourth items make sure that non-parallel and parallel constraints in $\mathcal{IC}$ and $\mathcal{C}$ are maintained by the joint plan.

Example 7. For the multiagent planning problem $\mathcal{M}_1$ from Example 6, We can easily check the following:

- for $n \leq 2$, $\mathcal{M}_1$ has no solution.
- for $n = 3$, it has a solution consisting of the following plans
  - $S_A = [s_0^A,\ give\_me\_nail(h\_nail, C),\ s_1^A,\ give\_me\_ham(h\_ham, B),$
    $s_2^A, hw\_nail,\ s_3^A,\ \texttt{wait},\ s_4^A]$
  - $S_B = [s_0^B,\ hw\_nail,\ s_1^B,\ get\_this\_ham(h\_ham, A),$
    $s_2^B,\ get\_this\_screw(h\_screw, C),\ s_3^B,\ \texttt{wait},\ s_4^B,]$
  - $S_C = [s_0^C,\ get\_this\_nail(h\_nail, A),\ s_1^C,\ \texttt{wait},\ s_2^C,\ give\_me\_screw(h\_screw, B),$
    $s_3^C,\ hw\_screw,\ s_4^C]$

  where all requests are satisfied and the states are uniquely determined by the initial states and the executed actions. □

The joint plan for the agents in Example 7 requires that each agent executes some cooperative actions. It is easy to see that any joint plan for the two agents in the problem $\mathcal{M}_2$ requires that only one agent to flip the switch next to her and other agent to wait.

4 Computing Joint Plans

In this section, we will present different approaches to computing joint plans. Our approaches utilize answer set programming [18, 19], a declarative programming paradigm that has recently emerged from the study of logic programming under the answer set semantics [11].

4.1 Answer Set Semantics of Logic Programs

A logic program Π is a set of rules of the form

$$a_0 \leftarrow a_1, \ldots, a_m, not\ \ a_{m+1}, \ldots, not\ \ a_n \tag{5}$$

where $0 \leq m \leq n$, each a_i is an atom of a propositional language[5] and not represents *negation-as-failure*. A negation as failure literal (or naf-literal) is of the form $not\ \ a$ where a is an atom. For a rule of the form (5), the left (right) hand sides of the rule are called the *head* (*body*), respectively. The head and the body can be empty (but not at the same time). A rule is a *constraint* if its head is empty; it is a *fact* if its body is empty.

Consider a set of ground atoms X. The body of a rule of the form (5) is *satisfied* by X if $\{a_{m+1}, \ldots, a_n\} \cap X = \emptyset$ and $\{a_1, \ldots, a_m\} \subseteq X$. A rule of the form (5) with nonempty head is satisfied by X if either its body is not satisfied by X or $a_0 \in X$. In other words, X satisfies a rule of the form (5) if its head belongs to X whenever X satisfies its body. A constraint is *satisfied* by X if its body is not satisfied by X.

For a set of ground atoms S and a program Π, the *reduct* of Π w.r.t. S, denoted by Π^S, is the program obtained from the set of all ground instances of Π by deleting

1. each rule that has a naf-literal *not* a in its body with $a \in S$, and
2. all naf-literals in the bodies of the remaining rules.

S is an *answer set* of Π if it satisfies the following conditions.

[5] Rules with variables are viewed as a shorthand for the set of its ground instances.

1. If Π does not contain any naf-literal (i.e. $m = n$ in every rule of Π) then S is the smallest set of atoms that satisfies all the rules in Π.
2. If the program Π does contain some naf-literal ($m < n$ in some rule of Π), then S is an answer set of Π if S is the answer set of Π^S. (Note that Π^S does not contain naf-literals, its answer set is defined in the first item.)

A program Π is said to be *consistent* if it has an answer set. Otherwise, it is inconsistent. To make answer set style programming easier, Niemelä et al. [20] introduce a new type of rules, called *cardinality constraint rule* (a special form of the *weight constraint rule*) of the following form: $A_0 \leftarrow A_1, \ldots, A_m, not\ \ A_{m+1}, \ldots, not\ \ A_n$ where each A_i is a *choice atom* of the form $l\{b_1, \ldots, b_k\}u$ with b_j are atoms and l and u are two integers, $l \leq u$; and A_0 can be empty. An atom $l\{b_1, \ldots, b_k\}u$ is said to be true wrt. a set of literals S iff $l \leq |S \cap \{b_1, \ldots, b_k\}| \leq u$. The satisfaction of a rule wrt. a set of atoms is extended in the usual way. Using rules of this type, one can greatly reduce the number of rules of programs in answer set programming. The semantics of logic programs with such rules is given in [20].

4.2 Finding a Possible Plan for One Agent

We will represent each individual problem of each agent $\mathcal{P}_i$ by a logic program. The program will consist of rules describing the effects of actions, the initial knowledge of the agent, and the goal of the agent. Answer set planning [16] refers to the use of answer set programming in planning. This method has been applied to a variety of problems [10, 25]. Let $\mathcal{P} = \langle DI, I, O, DC \rangle$ be a planning problem. We will now describe the program $\Pi(\mathcal{P})$ that encodes $\mathcal{P}$. We adapt the conventional style in logic programming: terms starting with lower-case letter are constant and others are variables. It also has a parameter denoting the maximal length of the plan that the agent considers permissible. The key predicates of $\Pi(\mathcal{P})$ are:

- $h(l, t)$ – fluent literal l holds at the time step t; and
- $o(a, t)$ – action a is executed (by the agent) at the time step t;
- $poss(a, t)$ – action a can be executed at the time step t.

$h(l, t)$ can be extended to define $h(\phi, t)$ for an arbitrary fluent formula ϕ, which states that ϕ holds at the time moment t. In writing the program, we use $h(\{l_1, \ldots, l_k\}, T)$ as a shorthand for $h(l_1, T), \ldots, h(l_k, T)$. In addition, we write $h(ok(r(\gamma, i)), t)$ to denote that the request-action $r(\gamma, i)$ is satisfied at the time step t. The rules of the program is divided into groups:

- *Group 1*: The program contains the following facts:

  $$\{fluent(f) \mid f \in \mathtt{F}\} \cup \{action(a) \mid a \in \mathtt{A}\} \cup$$
  $$\{action(r(\gamma), i) \mid r \text{ occurring in a law of form (3)}, i \in \mathcal{A}_\mathtt{i}\} \cup$$
  $$\{action(p(\gamma), i) \mid p \text{ occurring in a law of form (4)}, i \in \mathcal{A}_\mathtt{i}\}$$

  These facts declare the fluents and the actions of the problem.
- *Group 2*: rules for reasoning about effects of actions. For each action $a \in \mathtt{A}$,
  - if DI contains the law (a **executable** ϕ) then $\Pi(\mathcal{P})$ contains the rules

$$poss(a, T) \leftarrow h(\phi, T) \tag{6}$$
$$\leftarrow o(a, T), not\ \ poss(a, T) \tag{7}$$

- if DI contains the law (a **causes** l **if** ϕ) then $\Pi(\mathcal{P})$ contains the rule

$$h(l, T+1) \leftarrow o(a, T), h(\phi, T) \tag{8}$$

- *Group 3*: rules for reasoning about request-actions. For each statement of the form

  r **requests** γ **from** $\mathcal{A}_\mathbf{i}$ **may_cause** ϕ **if** ψ

  and each $i \in \mathcal{A}_\mathbf{i}$, $\Pi(\mathcal{P})$ contains the rules

$$poss(r(\gamma, i), T) \leftarrow h(\psi, T) \tag{9}$$
$$\leftarrow o(r(\gamma, i), T), not\ \ poss(r(\gamma, i), T) \tag{10}$$
$$0\ \{h(ok(r(\gamma, i)), T+1)\}\ 1 \leftarrow o(r(\gamma, i), T). \tag{11}$$
$$h(\phi, T) \leftarrow h(ok(r(\gamma, i)), T) \tag{12}$$

  where (12) is a shorthand for the collection of rules $\{h(l, T) \leftarrow h(ok(r(\gamma, i)), T) \mid l \in \phi\}$. Observe that atoms of the form $h(ok(\gamma, i), T)$ are used to record the satisfaction of the request $r(\gamma, i)$ and there might be different ways for a condition γ to be satisfied. Hence, (11) and (12) need to be separated even though it looks like they could have been merged into one.
- *Group 4*: rules for reasoning about offer-actions. For each statement of the form

  p **provides** γ **for** $\mathcal{A}_\mathbf{i}$ **causes** ϕ **if** ψ

  and $i \in \mathcal{A}_\mathbf{i}$, $\Pi(\mathcal{P})$ contains the rules

$$poss(p(\gamma, i), T) \leftarrow h(\psi, T) \tag{13}$$
$$\leftarrow o(p(\gamma, i), T), not\ \ poss(p(\gamma, i), T) \tag{14}$$
$$h(\phi, T+1) \leftarrow o(p(\gamma, i), T). \tag{15}$$

  These rules are similar to the rules encoding the effect of individual actions of the agent. The difference between the encoding of a request-action and the encoding of an offer-action lies in that we do not need to introduce an atom of the form $h(ok(p(\gamma, i)), T)$ to record the execution of $p(\gamma, i)$, i.e., effects of offer-actions are deterministic.
- *Group 5*: rules describing the initial state. For each literal $l \in I$, $\Pi(\mathcal{P})$ contains the fact $h(l, 0)$.
- *Group 6*: rules encoding the goal state. For each literal $l \in O$, $\Pi(\mathcal{P})$ contains

$$\leftarrow not\ \ h(l, n). \tag{16}$$

  where n is the desired length of the plan.
- *Group 7*: rules for reasoning by inertial. For each fluent $F \in \mathtt{F}$, $\Pi(\mathcal{P})$ contains

$$h(F, T+1) \leftarrow h(F, T), not\ \ h(\neg F, T+1). \tag{17}$$
$$h(\neg F, T+1) \leftarrow h(\neg F, T), not\ \ h(F, T+1). \tag{18}$$
$$\leftarrow h(F, T), h(\neg F, T). \tag{19}$$

- *Group 8*: rules for generating action occurrences. $\mathit{\Pi}(\mathcal{P})$ contains the rule

$$1\ \{o(A,T) : action(A)\}\ 1 \leftarrow T < n. \tag{20}$$

which states that at any time step, the action must execute one of its actions[6].

Let $\mathcal{P} = \langle DI, I, O, DC\rangle$ be a planning problem and $\mathit{\Pi}(\mathcal{P}, n)$ denote the set of ground rules of $\mathit{\Pi}(\mathcal{P})$ in which the variable T is instantiated with integers between 0 to n. Let M be an answer set of $\mathit{\Pi}(\mathcal{P}, n)$. Let $s_t[M] = \{l \mid l$ is a fluent literal and $h(l,t) \in M\}$. By $\alpha[M]$ we denote the sequence $s_0[M], a_0, s_1[M], \ldots, a_{n-1}, s_n[M]$ where $o(a_i, i) \in M$. We can show the following:

Theorem 1. *Let $\mathcal{P}$ be a planning problem. Then,*

- *for each possible plan α of $\mathcal{P}$ there exists an n and an answer set M of $\mathit{\Pi}(\mathcal{P}, n)$ such that $\alpha = \alpha[M]$;*
- *for each n, if $\mathit{\Pi}(\mathcal{P}, n)$ is inconsistent then $\mathcal{P}$ does not have a solution of length less than or equal to n; and*
- *for each n, if $\mathit{\Pi}(\mathcal{P}, n)$ has an answer set M then $\alpha[M]$ is a solution of $\mathcal{P}$.*

4.3 Compatible Answer Sets and Joint Plan

Individual possible plans can be computed using the program $\mathit{\Pi}(\mathcal{P}_i)$. We will now discuss an approach for combining them to create a plan for all the agents. Intuitively, we need to make sure that if a request is assumed to be satisfied by an agent then there exists an instance of an offer-action matching this request. This can be easily characterized by the notion of a compatible answer sets.

Definition 7 (Compatible answer sets). *Let $\mathcal{M} = \langle \mathcal{AG}, \{\mathcal{P}_i\}_{i\in\mathcal{AG}}, \mathcal{F}, \mathcal{IC}, \mathcal{C}\rangle$ be a multiagent planning problem and $M = \langle M_i\rangle_{i\in\mathcal{AG}}$ be a sequence of answer sets of $\langle \mathit{\Pi}(\mathcal{P}_i, n)\rangle_{i\in\mathcal{AG}}$ where the constant n is fixed. M is a set of* compatible *answer sets if for each $k \leq n$,*

- *for each $i \in \mathcal{AG}$, if $h(ok(r(\gamma, j)), k+1) \in M_i$ then $o(p(\gamma, i), k) \in M_j$;*
- *for each $i \in \mathcal{AG}$, if $o(p(\gamma, j), k) \in M_i$ then $h(ok(r(\gamma, i)), k+1) \in M_j$;*
- *for each (i, j, f_i, f_j) in $\mathcal{F}$, $h(f_i, k) \in M_i$ iff $h(f_j, k) \in M_j$;*
- *for each $S \in \mathcal{IC}$ there exists some $(i, a_i) \in S$ such that $o(a_i, k) \notin M_i$; and*
- *for each $S \in \mathcal{C}$, either $\{a_i | (i, a_i) \in S$ and $o(a_i, k) \in M_i\} = \{a | (i, a) \in S\}$ or $\{a_i | (i, a_i) \in S$ and $o(a_i, k) \in M_i\} = \emptyset$.*

Intuitively, a set of compatible answer sets corresponds to a joint plan (as we will prove in the next theorem) similar to the correspondence between answer sets and plans in the case of a single agent. Observe also that $h(ok(.), T)$ is present only due to the successfulness of a request-action, not an offer-action. The conditions imposed on a set of compatible answer sets make sure that the collection of individual plans extracting from them satisfies the constraints of the planning problem and the requirement that satisfying requests must be matched with offers.

[6] Since we assume that `wait` always belongs to the set of actions of an agent, this is not a strict requirement as it might sound.

Theorem 2. *Let $\mathcal{M} = \langle \mathcal{AG}, \{\mathcal{P}_i\}_{i\in\mathcal{AG}}, \mathcal{F}, \mathcal{IC}\rangle$ be a multiagent planning problem and n be an integer.*

- *if $\langle \Pi(\mathcal{P}_i, n)\rangle_{i\in\mathcal{AG}}$ does not have a set of compatible answer sets then $\mathcal{M}$ does not have a solution with length n.*
- *a sequence of answer sets $M = \langle M_i\rangle_{i\in\mathcal{AG}}$ is compatible iff there exists a solution $S = \langle \alpha_i\rangle_{i\in\mathcal{AG}}$ such that $\alpha[M_i] = \alpha_i$ for every $i \in \mathcal{AG}$.*

Example 8. Let $\mathcal{M}_1$ be the multiagent planning problem from Example 6. We can easily check the following:

- $\{\Pi(\mathcal{P}_i, n)\}_{i\in\{A,B,C\}}$ for $n \leq 2$ does not have compatible answer sets,
- for $n = 3$, the three answer sets M_A, M_B, and M_C of $\Pi(\mathcal{P}_A, 3)$, $\Pi(\mathcal{P}_B, 3)$, and $\Pi(\mathcal{P}_C, 3)$, where
  - M_A contains $o(give\_me\_nail(h\_nail, c), 0)$, $h(ok(give\_me\_nail(h\_nail, c)), 1)$, $o(give\_me\_ham(h\_ham, b), 1)$, $h(ok(give\_me\_ham(h\_ham, b)), 2)$, $o(hw\_nail, 2)$, and $o(\mathtt{wait}, 3)$.
  - M_B contains $o(hw\_nail, 0)$, $o(get\_this\_ham(h\_ham, a), 1)$, $o(get\_this\_screw(h\_screw, c), 2)$, $o(\mathtt{wait}, 3)$; and
  - M_C contains $o(get\_this\_nail(h\_nail, a), 0)$, $o(\mathtt{wait}, 1)$, $o(give\_me\_screw(h\_screw, b), 2)$, $h(ok(give\_me\_screw(h\_screw, b)), 2)$, and $o(hw\_screw, 3)$.

  These answer sets are compatible and correspond to the solution in Example 5. □

The notion of joint plan can be specialized as follows.

Definition 8 (Optimal Joint Plan). *Let $\mathcal{M} = \langle \mathcal{AG}, \{\mathcal{P}_i\}_{i\in\mathcal{AG}}, \mathcal{F}, \mathcal{IC}, \mathcal{C}\rangle$ be a multiagent planning problem and $\{S_i\}_{i\in\mathcal{AG}}$ be a plan for $\mathcal{M}$. We say that $\{S_i\}_{i\in\mathcal{AG}}$ is* optimal *if there exists no unsatisfied request actions in $\{S_i\}_{i\in\mathcal{AG}}$.*

Remark 3. The program $\Pi(\mathcal{P}_i)$ can be easily adapted to generate only optimal plans. Indeed, the only modification that needs to be done is to replace the rule (11) with

$$h(ok(r(\gamma, i)), T + 1) \leftarrow o(r(\gamma, i), T).$$

Intuitively, this rule states that the request $r(\gamma, i)$ is satisfied. Thus, if a joint plan is found it will not contain any unsatisfied requests, i.e., it must be optimal.

Definitions 6 and 7 provide us with a way for computing joint plans of length n for a planning problem $\mathcal{M}$. The process involves (*i*) computing a set $\{M_i\}_{i\in\mathcal{AG}}$ of answer sets, where M_i is an answer set of $\Pi(\mathcal{P}_i, n)$; and (*ii*) checking the compatibility of $\{M_i\}_{i\in\mathcal{AG}}$. In what follows, we discuss a method for doing it. This method computes a joint plan by (*a*) forming a program representing $\mathcal{M}$ from the programs representing the individual plans and the set of constraints in $\mathcal{M}$; and (*b*) extracting joint plan from answer sets of the new program. This method is useful if the planning problem $\mathcal{M}$ is known to an agent or a manager.

4.4 Computing Joint Plans by Answer Set Programming

Let $\mathcal{M} = \langle \mathcal{AG}, \{\mathcal{P}_i\}_{i\in\mathcal{AG}}, \mathcal{F}, \mathcal{IC}, \mathcal{C}\rangle$ be a planning problem. We will define a program $\Pi(\mathcal{M})$ whose answer sets represent the solutions of $\mathcal{M}$. $\mathcal{M}$ is constructed from the programs $\Pi(\mathcal{P}_i)$ for $i \in \mathcal{AG}$ as follows. For each $i \in \mathcal{AG}$, let $\Pi^i(\mathcal{P}_i)$, referred as the *tagged version* of $\Pi(\mathcal{P}_i)$, be the program obtained from $\Pi(\mathcal{P}_i)$ by replacing every literal x in $\Pi(\mathcal{P}_i)$ with the atom x^i (e.g., $action(a)^i$ for $action(a)$, $h(f,t)^i$ for $h(f,t)$, etc.). The program $\Pi(\mathcal{M})$ consists of

- for each $i \in \mathcal{AG}$, the tagged version $\Pi^i(\mathcal{P}_i)$ of $\Pi(\mathcal{P}_i)$;
- for each tuple (i, j, f^i, f^j) in $\mathcal{F}$, the constraints

$$\leftarrow h^i(f^i, T), h^j(\neg f^j, T) \tag{21}$$

$$\leftarrow h^i(\neg f^i, T), h^j(f^j, T) \tag{22}$$

  ensure that shared variables maintain their consistency.
- for each set $S = \{(i_1, a_1), \ldots, (i_k, a_k)\}$ in $\mathcal{C}$, the constraint

$$\leftarrow 0\ \{o^{i_1}(a_1, T), \ldots, o^{i_k}(a_k, T)\}\ k-1 \tag{23}$$

  which makes sure that if a part of S is executed, i.e., $o(i_j, a_j)$ belongs to an answer set, then the whole set S is executed.
- for each set $\{(i_1, a_1), \ldots, (i_k, a_k)\}$ in $\mathcal{IC}$, the constraints

$$\leftarrow o^{i_1}(a_1, T), \ldots, o^{i_k}(a_k, T) \tag{24}$$

  This constraint guarantees that not all the actions $a_1, \ldots, a_k$ are executed at the same time.
- for every pair of instance $r(\gamma, j)$ and $p(\gamma, i)$ of a request-action r (for γ) of an agent i and an offer-action p (for γ) of an agent j, the following constraints

$$\leftarrow o^i(r(\gamma, j), T), h^i(ok(r(\gamma, j)), T+1), not\ \ o^j(p(\gamma, i), T) \tag{25}$$

$$\leftarrow o^j(p(\gamma, i), T), not\ \ o^i(r(\gamma, j), T) \tag{26}$$

$$\leftarrow o^j(p(\gamma, i), T), not\ \ h^i(ok(r(\gamma, j)), T+1) \tag{27}$$

  The first constraint makes sure that if i requests for γ from j and it is satisfied then j does indeed offer the service. The last two rules guarantee the converse.

For a set X of literals in the language of $\Pi(\mathcal{M})$, let $X|_i = \{a \mid a$ is a literal in the language of $\Pi(\mathcal{P}_i)$ and $a^i \in X\}$. We have:

Theorem 3. *Let $\mathcal{M} = \langle \mathcal{AG}, \{\mathcal{P}_i\}_{i \in \mathcal{AG}}, \mathcal{F}, \mathcal{IC}, \mathcal{C} \rangle$ be a multiagent planning problem. M is an answer set of $\Pi(\mathcal{M}, n)$ iff there exists a set of compatible answer sets $\{M_i\}_{i \in \mathcal{AG}}$ such that $M|_i = M_i$.*

The proof of Theorem 3 relies on the Splitting Theorem for logic programs [17]. It is divided into two steps. First, it is proved for program without the constraints (21)-(27). The significance of this proposition is that it allows us to compute the solution of a multiagent planning problem by computing a single answer set of $\mathcal{P}(\mathcal{M})$. Since the problem of determining whether a propositional program has an answer set or not is NP-complete, the following holds.

Corollary 1. *Determining whether a solution of polynomial bounded length of a multiagent planning problem $\mathcal{M}$ exists or not is NP-complete.*

5 Related Works

Multiagent planning could be viewed as a special case of distributed problem solving [9]. In this respect, our work could be viewed as one in the Centralized Planning for Distributed Plans group according to the classification in [9]. This is achieved by the program $\Pi(\mathcal{M})$. Alternatively, the individual plans can also be computed distributedly and coordinated using the program consisting of the constraints (21)-(27) and the tagged versions of the individual answer sets.

Our main goal is to generate a joint plan for the agents before its execution. In this regards, our work differs from many distributed continual planning systems that were discussed in the survey [7] and many papers presented in the recent AAMAS conferences which concentrate on planning and replanning or dealing with unexpected events during the plan execution.

Our approach to generating a joint plan in this paper blends the two components "planning" and "coordination" in the equation

$$\boxed{\text{Multiagent planning} = \text{Planning} + \text{Coordination}}$$

presented in [6] into a single step. Furthermore, we employ a plan representation that allows for the coordination to be done by using time-steps presented in individual plans. This is different from several other systems in which partial order plans are used for plan representation and refinement planning is used for coordination (e.g., [4, 3] or earlier works such as the Partial Global Planning framework).

We use answer set programming [16], a method that has been used for single agent planning [10, 25], in computing the joint plan. The declarativeness and modularity of answer set programming make the process of computing the joint plan fairly simple and simplify the coordination of the plans[7]. Our work is similar to the spirit of that in [8] where an attempt is made to construct joint plan using SAT-based single agent planner. Nevertheless, our use of answer set programming does not require the development of additional algorithms to assemble the final joint plan.

In [2], a language based on PDDL for modeling multiagent planning problems has been proposed that allows for the specification of and reasoning about several features important for multiagent planning and execution such as concurrency, individual and mutual beliefs of agents, planning and acting with incomplete information, communication, continuous events, etc. A special agent, called `env`, is present in all problems for modeling the environment which may act "unpredictably". Our language is less expressive than the above mentioned language as our focus is solely on the generation of a joint plan prior to its execution. On the other hand, the semantics provided in this paper can be used to prove formal properties of plans as well as the correctness of the logic program encoding of multiagent planning problem.

We note that collaborative actions presented in this paper is also suitable for the modeling of multiagent planning with resources. Requesting for a resource and offering a resource can be modeled in a similar fashion to that of asking for and offering of a nail (Example 4). Since our focus is the generation of joint plan before execution, the proposed language is different from the resource logic introduced in [5], whose focus

[7] Recall that this is achieved by simply adding the rules (21)-(27).

was on the plan merging phase. The requesting/offering actions can be seen as special case of *negotiation actions* discussed in [26].

We would like to point out that we use $\mathcal{A}$ because of its simple semantics and its close relationship to PDDL, the language developed for describing planning problems [14]. This means that other extensions or variations of $\mathcal{A}$ (e.g,. $\mathcal{B}$, $\mathcal{C}$ [13], $\mathcal{E}$ [15]) could also be extended to formalize cooperative actions. Observe that there are subtle differences between request actions and non-deterministic actions. First, a cooperative action changes the world of other agents while a non-deterministic action does not. Second, a cooperative action does not change the world of the agent executing this action, while a non-deterministic action does. In this sense, a cooperative action of an agent is like an exogenous action for other agents. Thus, modeling cooperative actions using non-deterministic actions might not be the most natural way.

Finally, we would like to note that an extension of the STRIPS language has been considered for multiagent planning in [1]. In this framework, a multiagent planning problem is formulated as a *single problem* and agent identifiers are attached to the actions, which is different from what we proposed here. As such, the framework in [1] is only appropriate for domains where no privacy among agents is required. This is not an issue in our formulation.

6 Conclusions and Future Works

We extend the action language $\mathcal{A}$ to define a language for representing and reasoning about actions and their effects in presence of cooperative actions between agents. We define the notion of a plan with cooperative actions and use it in formalizing the notion of a joint plan. We use answer set programming to generate joint plans. We introduce the notion of a set of compatible answer sets and provide a translation of a multiagent planning problem to a logic program whose answer sets represent joint plans.

The work so far has focused on the development of a theoretical framework for generating joint plans using answer set programming. The encoding of the examples are available in the technical report version of this paper [24]. Our immediate goal for the future is to investigate the scalability and efficiency of the proposed method. The use of answer set programming allows us to easily incorporate preferences or domain knowledge in the generation of the joint plans [22, 23]. Additionally, we would like to explore the use of more expressive languages (e.g., action languages with constraints and sensing actions) in representing and reasoning about joint-plans of multiagents by addressing various questions mentioned in [2]. This is because the method provides in Section 2 has proved to be very effective in the single agent case (e.g. [25]).

‘

References

1. C. Boutilier and R.I. Brafman. Partial-order planning with concurrent interacting actions. *JAIR*, 14:105–136, 2001.
2. M. Brenner. Planning for Multiagent Environments: From Individual Perceptions to Coordinated Execution. In *Work. on Multiagent Planning & Scheduling, ICAPS*, 80–88. 2005.
3. J. S. Cox and E. H. Durfee. An efficient algorithm for multiagent plan coordination. AAMAS 2005, 828–835.
4. J. S. Cox and E. H. Durfee and T. Bartold. A Distributed Framework for Solving the Multiagent Plan Coordination Problem. In *AAMAS*, pages 821–827. ACM Press, 2005.

5. M. de Weerdt, A. Bos, H. Tonino, and C. Witteveen. A resource logic for multi-agent plan merging. *Ann. Math. Artif. Intell.*, 37(1-2):93–130, 2003.
6. M. de Weerdt, A. ter Mors, and C. Witteveen. Multi-agent planning: An introduction to planning and coordination. In *Handouts of the Euro. Agent Summer School*, 1–32, 2005.
7. M. desJardins, E. H. Durfee, C. L. Ortiz, and M. Wolverton. A survey of research in distributed, continual planning. *AI Magazine*, 20(4):13–22, 1999.
8. Y. Dimopoulos and P. Moraitis. Multi-agent coordination and cooperation through classical planning. In *IEEE/WIC/ACM/IAT*, 398–402. IEEE Comp. Society, 2006.
9. E. Durfee. Distributed Problem Solving and Planning. In *Muliagent Systems (A Modern Approach to Distributed Artificial Intelligence)*, pages 121–164. MIT Press, 1999.
10. T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. Answer Set Planning under Action Costs. *Journal of Artificial Intelligence Research*, 19:25–71, 2003.
11. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Int. Conf. on Logic Programming*, 1070–1080, 1988.
12. M. Gelfond and V. Lifschitz. Representing actions and change by logic programs. *Journal of Logic Programming*, 17(2,3,4):301–323, 1993.
13. M. Gelfond and V. Lifschitz. Action languages. *ETAI*, 3(6), 1998.
14. M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL — the Planning Domain Definition Language. Ver. 1.2. TR1165, Yale, 1998.
15. A. C. Kakas, R. Miller, F. Toni: E-RES. Reasoning about Actions, Events and Observations. In *LPNMR*, 254-266, 2001.
16. V. Lifschitz. Action languages, answer sets and planning. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 357–373. Springer Verlag, 1999.
17. V. Lifschitz and H. Turner. Splitting a logic program. In *ICLP*, 23–38, 1994.
18. V. Marek and M. Truszczyński. Stable models and an alternative logic programming paradigm. In *The Log. Prog. Paradigm: a 25-year Perspective*, 375–398, 1999.
19. I. Niemelä. Logic programming with stable model semantics as a constraint programming paradigm. *AMAI*, 25(3,4):241–273, 1999.
20. I. Niemelä, P. Simons, and T. Soininen. Stable model semantics for weight constraint rules. In *Proc. Logic Programming and NonMonotonic Rreasong*, 315–332, 1999.
21. S. Parsons, C. Sierra, and N. R. Jennings. Agents that reason and negotiate by arguing. *J. of Log. and Comp.*, 8(3):261–292, 1998.
22. T. C. Son, C. Baral, N. Tran, and S. McIlraith. Domain-Dependent Knowledge in Answer Set Planning. *ACM Transactions on Computational Logic*, 7(4), 2006.
23. T. C. Son and E. Pontelli. Planning with Preferences using Logic Programming. *Journal of Theory and Practice of Logic Programming* (TPLP), 6:559–607, 2006.
24. T. C. Son and C.. Sakama. Reasoning and Planning with Cooperative Actions for Multiagents Using Answer Set Programming. Technical Report, 2008.
25. P. H. Tu, T. C. Son, C. Baral. Reasoning and Planning with Sensing Actions, Incomplete Information, and Static Causal Laws using Logic Programming. *TPLP*, 7:1–74, 2006.
26. M. Wooldridge and S. Parsons. Languages for negotiation. In *Proceedings of ECAI*, 2000.