Logic Programming for Multiagent Planning with Negotiation

Tran Cao Son[1], Enrico Pontelli[1], and Chiaki Sakama[2]

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

Abstract. Multiagent planning deals with the problem of generating plans for multiple agents. It requires formalizing ways for the agents to interact and cooperate, in order to achieve their goals. One way for the agents to interact is through negotiations. Integration of negotiation in multiagent planning has not been extensively investigated and a systematic way for this task has yet to be found. We develop a generic model for negotiation in dynamic environments and apply it to generate joint-plans with negotiation for multiple agents. We identify the minimal requirements for such a model and propose a general scheme for one-to-one negotiations. This model of negotiation is instantiated to deal with dynamic knowledge of planning agents. We demonstrate how logic programming can be employed as a uniform platform to support both planning and negotiation, providing an ideal testbed for experimenting with multiagent planning with negotiations.

1 Introduction

Negotiation and planning are two important tasks that autonomous agents are frequently engaged in during their existence. Theories of *negotiation* have been developed to provide the agents with strategies and methods for doing negotiation (e.g., [1, 4, 13, 15, 19, 20, 17]). On the other hand *planning* research gears towards developing systems and algorithms that allow agents with a way to select appropriate courses of actions to achieve their individual goals (e.g., [11]). It is interesting to observe that there has been limited connection between these research communities. On the other hand, in many practical situations, there is a need for intelligent agents to negotiate when they are planning, especially in multiagent systems. This can be seen in the following simple example.

Example 1 (From [18]). Two home builder agents, A and B, need to hang a mirror (A's job) and a picture (B's job). A can use a screw with a screwdriver to hang the mirror. B can only use a nail and a hammer. Initially, A has a screwdriver and can buy a nail, while B has a screw and a hammer. A and B cannot achieve their goals independently.

Let us now consider the following conversation between A and B: **(1)** $\underline{A \text{ to } B}$: can you give me your screw? **(2)** $\underline{B \text{ to } A}$: yes, but only if you give me a nail; **(3)** $\underline{A \text{ to } B}$: ok, but wait for me to buy some nails. Thereafter, A buys a nail and exchanges it for the screw with B. Both can then achieve their goals independently. □

This story illustrates two important issues that agents are facing in a multiagent environment. Besides the planning capabilities, agents may also need to negotiate in order to achieve their goals—where negotiation becomes a form of cooperation between agents.

In this paper, we are interested in the problem of predicting whether a group of agents, each with her own planning problem to solve, can achieve their independent goals. In the process, the agents might need to negotiate with each others to place themselves in a position where they can achieve their goals.

We start by proposing a *generic model of negotiation*. This model is novel with respect to existing formalisms (e.g., [1, 4, 13, 17, 19, 20]). Our focus is on agents in dynamic environments, whereas most of the other formalisms concentrate on agents in static environments. One of the main objectives is in predicting/generating successful negotiations within the context of the agents achieving their planning goals. The planning process is driven by the goals of all agents, while negotiations are only a means for agents to achieve their goals. This is different from other models of negotiation (e.g., [1, 13, 17, 19, 20, 24]), where the focus is on providing the receiver with explanations about negotiations or on the development of languages for logic-based negotiations.

We instantiate the proposed model of negotiation to the context of *multiagent planning* and define two different notions of planning with negotiations. The first notion views complete negotiations as individual steps during planning, while the second one allows the interleaving of steps of negotiations and action executions. 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 (e.g., [6]), which concentrate on planning and replanning or deal with unexpected events during plan execution. More significantly, we explore the use of negotiation as a means for agents to cooperate.

A key contribution of this work is to demonstrate how logic programming allows a direct and modular encoding of both negotiation and multiagent planning. To the best of our knowledge, this logic programming based solution is the first attempt to deal with negotiation in multiagent planning. The declarative nature of logic programming allows us to provide a compositional solution to the problem, by combining two orthogonally developed logic programs—one describing the planning problem and one describing the negotiation process.

In the past, multiagent planning has been considered using refinement planning (e.g., [2, 12, 5]) but without negotiation. In [14], negotiation has been integrated with planning and control operations in the cycle theories, to create an agent architecture and to ensure that the agents can achieve their goals. However, negotiation is used mainly to ensure that the execution of a given plan is successful, e.g., to acquire necessary resources for the execution of a plan—in particular, the authors do not investigate the integration of negotiation within the planning phase. Negotiation using logic programming has been investigated by others (e.g., [4]), with a focus on the principles of negotiation and building new proposals, when the current one is not acceptable. Our characterization is in similar spirit to this approach.

2 Background: Answer Set Planning

In this section, we review the language $\mathcal{A}$ [9] for representing and reasoning about actions and plans in single-agent domains. To simplify the notations in the rest of the

paper, we will assume that the discussion in this section is associated to an agent i. We assume the reader to be familiar with the basic concepts of answer set programming.

A *planning problem for* i is defined over a set of *fluents* $\mathtt{F}_i$ and a set of *actions* $\mathtt{A}_i$. We assume that $\mathtt{A}_i$ contains a special action `noop`, which does not have any effect on the agent's world. A *fluent literal* is either a fluent $f \in \mathtt{F}_i$ or its negation $\neg f$. A *domain specification* D_i over $\mathtt{F}_i$ and $\mathtt{A}_i$ describes the actions of an agent and consists of laws of the following forms:[1] (a **causes** ℓ **if** φ) (if $\varphi = true$ then the **if** part will be omitted) and (a **executable** φ), where a is an individual action (in $\mathtt{A}_i$), ℓ is a fluent literal and φ is a set of fluent literals (interpreted as a conjunction). The first law is a *dynamic law*, and states that if a is executed when φ is true then ℓ becomes true. The second law is an *executability condition* and it states that a can be executed only if φ is true.

The semantics of a domain specification D_i is defined by the notion of *state* and by a *transition function* Φ_{D_i}, that specifies the result of the execution of an action a in a state s. A set of literals s satisfies a fluent literal ℓ, denoted by $s \models \ell$, if $\ell \in s$. For a set of fluent literals ϕ, $s \models \phi$ if $s \models \ell$ for every $\ell \in \phi$. A *state* s is a set of fluent literals that is *consistent*—i.e., for each fluent $f \in \mathtt{F}_i$ we have that $\{f, \neg f\} \not\subseteq s$—and *complete*—i.e., for every f either $f \in s$ or $\neg f \in s$. In the following, we use $\overline{\ell}$ to denote the complement literal of ℓ, i.e., if $\ell = f$ for some $f \in \mathtt{F}_i$, then $\overline{\ell} = \neg f$; if $\ell = \neg f$ for some $f \in \mathtt{F}_i$, then $\overline{\ell} = f$. For a set of literals S, $\overline{S} = \{\overline{\ell} \mid \ell \in S\}$.

An action a is *executable* in a state s if there exists an executability condition of the form a **executable** φ in D_i such that $s \models \varphi$. Let

$$e_a(s) = \{\ell \mid \exists (a \textbf{ causes } \ell \textbf{ if } \phi) \in D_i . [s \models \phi]\}$$

The result of the execution of a in s is defined by $\Phi_{D_i}(a, s) = fails$, if a is not executable in s, and $\Phi_{D_i}(a, s) = s \cup e_a(s) \setminus \overline{e_a(s)}$ if a is executable in s. The function Φ_{D_i} can be extended to reason about the effects of a sequence of actions:

Definition 1. *Let D_i be a domain specification, s be a state, and $\alpha = [a_1; \ldots; a_n]$ be a sequence of actions. We define $\hat{\Phi}_{D_i}(\alpha, s) = s$ if $n = 0$, and $\hat{\Phi}_{D_i}(\alpha, s) = \Phi_{D_i}(a_n, \hat{\Phi}_{D_i}([a_1; \ldots; a_{n-1}], s))$, otherwise. Observe that $\Phi_{D_i}(a, fails) = fails$.*

An agent can use the transition function to reason about effects of its actions and to perform planning. A *planning problem* is a tuple $\langle D_i, I_i, O_i \rangle$ where D_i is a domain specification, I_i is a state describing the initial configuration of the world for i, and O_i is a set of literals representing the desired goal.

Definition 2. *Let $\mathcal{P}_i = \langle D_i, I_i, O_i \rangle$ be a planning problem. An action sequence α is a* plan *for $\mathcal{P}_i$ iff O_i is true in $\hat{\Phi}_{D_i}(\alpha, I_i)$.*

Example 2. The domain specification D_A for A in Ex. 1 is defined over $\mathtt{F}_A$ = $\{h\_nail$, h_screw, $mirror\_on$, h_hammer, $h\_screwdriver\}$ and $\mathtt{A}_A = \{hw\_screw, buy\_nail\}$, with the set of laws:

buy_nail	**causes**	h_nail	hw_screw	**causes**	$mirror\_on$
hw_screw	**causes**	$\neg h\_screw$	hw_screw	**executable**	$h\_screw, h\_screwdriver$

The domain specification of B is defined over $\mathtt{A}_B = \{hw\_nail\}$ and $\mathtt{F}_B$ = $\{h\_nail$, h_screw, $picture\_on$, h_hammer, $h\_screwdriver\}$, with the set of laws:

hw_nail	**causes**	$picture\_on$	hw_nail	**causes**	$\neg h\_nail$
hw_nail	**executable**	h_nail, h_hammer			

[1] Originally, $\mathcal{A}$ did not include a **executable** φ. It was later introduced by the creator of $\mathcal{A}$.

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

Answer set planning (e.g., [16, 23]) refers to approaches to planning using logic programming with answer set semantics [8]. In these approaches, a planning problem is translated into a logic program, whose answer sets correspond one-to-one to the solutions of the original problem. As with the action language $\mathcal{A}$, answer set planning approaches have mainly focused on solving single agent planning problems. An exception is [10], dealing with multiagent systems supporting message-based coordination.

Let $\mathcal{P}_i = \langle D_i, I_i, O_i \rangle$ be a planning problem of agent i. We will now describe the logic program $\Pi^n(\mathcal{P}_i)$ that encodes $\mathcal{P}_i$. Let us denote with n the maximal length of a plan. The key predicates of $\Pi^n(\mathcal{P}_i)$ are:

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

$h(i, \ell, t)$ can be extended to define $h(i, \varphi, t)$ for an arbitrary fluent formula φ, which states that φ holds at the time step t. We use $h(i, \{l_1, \ldots, l_k\}, T)$ as a shorthand for $h(i, l_1, T), \ldots, h(i, l_k, T)$. In all the program rules, T denotes a time step, ranging from 0 to n. $\Pi^n(\mathcal{P}_i)$ is defined as follows:

- *Rules for declaring fluents and actions of an agent i:* For each fluent $f \in \mathtt{F}_i$ and action $a \in \mathtt{A}_i$, $\Pi^n(\mathcal{P}_i)$ contains facts of the form $fluent(i, f)$ and $action(i, a)$.
- *Rules for reasoning about effects of actions:* For each action $a \in \mathtt{A}_i$,
  - if D_i contains the law a **executable** φ then $\Pi^n(\mathcal{P}_i)$ contains the rules

$$poss(i, a, T) \leftarrow h(i, \varphi, T). \tag{1}$$
$$\leftarrow o(i, a, T), not\ \ poss(i, a, T). \tag{2}$$

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

$$h(i, l, T+1) \leftarrow o(i, a, T), h(i, \varphi, T). \tag{3}$$

- *Rules describing the initial state and the goal state:* For each literal $\ell \in I_i$ and for each $\ell' \in O_i$, $\Pi^n(\mathcal{P}_i)$ contains the rules

$$h(i, \ell, 0) \leftarrow \qquad\qquad \leftarrow not\ \ h(i, \ell', n).$$

- *Rules for encoding inertia:* For each fluent $f \in \mathtt{F}_i$, $\Pi^n(\mathcal{P}_i)$ contains the rules

$$h(i, f, T+1) \leftarrow h(i, f, T), not\ \ h(i, \neg f, T+1). \tag{4}$$
$$h(i, \neg f, T+1) \leftarrow h(i, \neg f, T), not\ \ h(i, f, T+1). \tag{5}$$
$$\leftarrow h(i, f, T), h(i, \neg f, T). \tag{6}$$

- *Rules for generating action occurrences:* $\Pi^n(\mathcal{P}_i)$ contains the rule

$$1\ \{o(i, A, T) : action(i, A)\}\ 1 \leftarrow T < n. \tag{7}$$

which states that, at any time step, the agent must execute one of its actions.

The following theorem can be proved.

Theorem 1. *The program $\Pi^n(\mathcal{P}_i)$ is consistent iff $\mathcal{P}_i$ has a plan of length n.*

Let $\mathcal{P}_A$ be the planning problem for A from Example 1. We can easily check that for every n, $\Pi^n(\mathcal{P}_A)$ is inconsistent. Likewise, $\Pi^n(\mathcal{P}_B)$ is inconsistent.

3 Multiagent Planning and Answer Set Planning

In this paper, we are interested in the planning problem in multiagent environments. We focus on situations where each agent has her own planning problem, and the agents are loosely connected—i.e., they might or might not use the same language in their representations (e.g., they might use different names to describe the same property). Furthermore, there are group actions that should be executed together for their effects to take place. Likewise, there are actions that cannot be executed by a group at the same time. Let us start with some preliminary definitions.

Definition 3. *Let $\{\mathcal{P}_i\}_{i\in\mathcal{AG}}$ be a set of planning problems of agents in $\mathcal{AG}$. A* tagged-*fluent is of the form $f[i]$ where f is a fluent in $\mathcal{P}_i$. A* tagged-*formula over $\mathcal{AG}$ is a formula constructible from the set of tagged fluents.*

We will call a sequence of states $S = \langle s_i\rangle_{i\in\mathcal{AG}}$ a *combined state* between agents in $\mathcal{AG}$. Given a tagged-formula φ over $\mathcal{AG}$ and a combined state $S = \langle s_i\rangle_{i\in\mathcal{AG}}$, the truth value of φ in $\langle s_i\rangle_{i\in\mathcal{AG}}$ is determined as follows: if φ is a tagged-fluent $f[i]$ (resp. the negation of a tagged-fluent $\neg f[i]$) then φ is true in S if f is true (resp. false) in s_i; the truth value of a complex formula is computed in the usual way.

Definition 4. *A* multiagent planning problem *$\mathcal{M}$ is a tuple $\langle\mathcal{AG}, \{\mathcal{P}_i\}_{i\in\mathcal{AG}}, \mathcal{F}, \mathcal{NC}, \mathcal{C}\rangle$ where* (i) *$\mathcal{AG}$ is a set of agents,* (ii) *$\mathcal{P}_i$ is a planning problem for each agent $i \in \mathcal{AG}$,* (iii) *$\mathcal{F}$ is the set of tagged-formulas over $\mathcal{AG}$, and* (iv) *$\mathcal{NC}$ and $\mathcal{C}$ are sets of sets of pairs (i, a^i) where i is an agent and a^i is an action in $\mathtt{A}_i$.*

Intuitively, $\mathcal{F}$ is a set of constraints on the combined states, $\mathcal{NC}$ is the set of non-concurrent actions, and $\mathcal{C}$ is the set of concurrent actions within $\mathcal{AG}$. For a multiagent planning problem $\mathcal{M}$, a joint-action sequence of length k of agents in $\mathcal{AG}$ is a sequence $\langle\alpha_i\rangle_{i\in\mathcal{AG}}$ where, for each $i \in \mathcal{AG}$, $\alpha_i = [a_0^i, \ldots, a_k^i]$ is a sequence of actions in D_i, executed by i at the time steps $0, 1, \ldots, k$.

Definition 5. *A joint-action sequence $\langle\alpha_i\rangle_{i\in\mathcal{AG}}$ of length k is said to be* compatible *if, for every l, $0 \le l \le k$, the following conditions are satisfied:*

- *For each tagged-formula $\varphi \in \mathcal{F}$, φ is true in $\langle\widehat{\Phi}_{D_i}(\alpha_i[l], I_i)\rangle_{i\in\mathcal{AG}}$ where $\alpha_x[l]$ denotes the action sequence $[a_0^x, \ldots, a_l^x]$.*
- *For each $S \in \mathcal{NC}$, there exists some $(i, a) \in S$ such that $a_l^i \neq a$.*
- *For each $S \in \mathcal{C}$, either $\{a \mid (i, a) \in S \text{ and } a = a_l^i\} = \{a \mid (i, a) \in S\}$ or $\{a \mid (i, a) \in S \text{ and } a = a_l^i\} = \emptyset$.*

Intuitively, a joint-action sequence is compatible if no constraint in $\mathcal{M}$ is violated. The first item indicates that the combined states must not violate the constraints in $\mathcal{F}$, i.e., the individual sequence of actions must agree with each other on their effects in shared environment. The second and third items make sure that non-concurrent and concurrent action constraints in $\mathcal{NC}$ and $\mathcal{C}$ are maintained by the joint-action sequence.

Definition 6. *Let $\mathcal{M} = \langle\mathcal{AG}, \{\mathcal{P}_i\}_{i\in\mathcal{AG}}, \mathcal{F}, \mathcal{NC}, \mathcal{C}\rangle$ be a multiagent planning problem. A joint-action sequence of length n, $\langle\alpha_i\rangle_{i\in\mathcal{AG}}$, is a* joint plan *of length n for $\mathcal{M}$ if $\langle\alpha_i\rangle_{i\in\mathcal{AG}}$ is compatible and for each $i \in \mathcal{AG}$, α_i is a plan of $\mathcal{P}_i$.*

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.

Example 3. The story in Example 1 can be formalized as a multiagent planning $\mathcal{M}_{AB} = \langle\{A, B\}, \{\mathcal{P}_A, \mathcal{P}_B\}, \emptyset, \emptyset, \emptyset\rangle$ where $\mathcal{P}_A = \langle D_A, I_A, O_A\rangle$ and $\mathcal{P}_B = \langle D_B, I_B, O_B\rangle$ with D_A and D_B are given in Example 2 and

- $I_A = \{\neg h\_nail, \neg mirror\_on, h\_screwdriver, \neg h\_screw, \neg h\_hammer\}$,
- $O_A = \{mirror\_on\}$,
- $I_B = \{\neg h\_nail, \neg picture\_on, h\_screw, \neg h\_screwdriver, h\_hammer\}$, and
- $O_B = \{picture\_on\}$.

We can easily show that $\mathcal{M}_{AB}$ has no solution. □

Answer set planning can be easily extended to compute solutions of multiagent planning problems. This is achieved by defining a program $\Pi^n(\mathcal{M})$, which consists of the rules of $\Pi^n(\mathcal{P}_i)$ along with rules enforcing the constraints in $\mathcal{F}$, $\mathcal{NC}$, and $\mathcal{C}$:

- For each tagged-formula φ in $\mathcal{F}$, a set of rules defining an atom $h(tagged, n_\varphi, T)$, where n_φ is a unique name assigned to φ. Due to lack of space, we omit the set of rules defining this atom (that can be found in [22]). To make sure that the formula is satisfied by the combined state at each time point, we add to $\Pi^n(\mathcal{M})$ the constraint:
$$\leftarrow not\ \ h(tagged, n_\varphi, T).$$
- For each set $\{(i_1, a_1), \ldots, (i_k, a_k)\}$ in $\mathcal{C}$, the constraint
$$\leftarrow 0\ \{o(i_1, a_1, T), \ldots, o(i_k, a_k, T)\}\ k - 1.$$
which makes sure that if a part of S is executed, i.e., $o(i_j, a_j, T)$ 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 $\mathcal{NC}$, the constraints
$$\leftarrow o(i_1, a_1, T), \ldots, o(i_k, a_k, T).$$
This guarantees that not all actions $a_1, \ldots, a_k$ are executed at the same time.

We can extend Th. 1 and prove that the program $\Pi^n(\mathcal{M})$ is consistent iff $\mathcal{M}$ has a solution of length n.

4 Negotiations Between Agents in Dynamic Environments

We consider one-to-one negotiations between agents in a dynamic environment and assume that the negotiation is related to the world representation of each agent. Each agent maintains her own world representation and has her own means to affect the world. Exchanges between agents can be characterized by logical formulae constructible from their representation languages. The acceptance by an agent i of an exchange coming from agent j will affect i's state of the world, and possibly that of j as well.

We assume that each agent i uses her own representation language $\mathcal{L}_i$ and a knowledge base KB_i, whose set of models $\mathcal{W}_i$ represents the acceptable states that she could be in. We assume that there exists a relation $\mathcal{R}_{i,j} \subseteq \mathcal{W}_i \times \mathcal{W}_j$ which encodes the set of *compatible* models between agents i and j—i.e., a pair $(w_i, w_j) \in \mathcal{R}_{i,j}$ is a possible combined state of the agents i and j. We will assume that $\mathcal{R}_{j,i}$ is the inverse relation of $\mathcal{R}_{i,j}$. We will not worry about how $\mathcal{R}_{i,j}$ could be defined or what properties it should have. For example, if the agents in Example 1 have only one hammer, then any pair of possible worlds between them should indicate that exactly one of them has the hammer.

Since negotiation entails an exchange between agents, and the agents can potentially rely on distinct languages, we introduce partial functions $\rho_{i,j}$, called *language matching* functions, that map formulae of $\mathcal{L}_i$ to formulae of $\mathcal{L}_j$. We will assume that the these functions are unambiguous w.r.t. equivalence of formulas, i.e., $\rho_{i,j}(\varphi) = \rho_{i,j}(\psi)$ if

$\varphi \equiv \psi$. In the case of planning agents, the function $\rho_{A,B}$ could be used to map fluents of A to corresponding fluents of B—e.g., $\rho_{A,B}(f) = g$ states that when A refers to f, it will mean g for B. We require that for each pair of agents i and j there are two functions $\rho_{i,j}$ and $\rho_{j,i}$ that are used by the agents in their communications. As in the case of the compatible relation $\mathcal{R}$, we will not worry about how ρ is defined. Before we continue, let us illustrate these notions using the agents A and B from Ex. 1.

Example 4. A and B can use the languages constructible from $\mathtt{F_A}$ and $\mathtt{F_B}$ as $\mathcal{L}_A$ and $\mathcal{L}_B$ respectively. Thus, we have that $\mathcal{W}_A$ (resp. $\mathcal{W}_B$) is the set of possible states of A (resp. B). Since there are no constraints on the combined states, we have that $\mathcal{R}_{A,B} = \mathcal{W}_A \times \mathcal{W}_B$. The language matching functions between A and B are identities. □

A negotiation originates from an agent i (*originator*), who is trying to have agent j (*recipient*) to establish for her a property (ψ). In turn, i may have to agree to establish φ for j. Such an exchange is captured by the notion of a conditional proposal.

Definition 7. *A* conditional proposal *(or, simply,* proposal*) from i to j has the form $\varphi \overset{i,j}{\Rightarrow} \psi$, where φ and ψ are formulae in $\mathcal{L}_i$ s.t. $\rho_{i,j}(\varphi)$ and $\rho_{i,j}(\psi)$ are both defined.*

A proposal $\varphi \overset{i,j}{\Rightarrow} \psi$ says that i is willing to establish φ for j (i.e., j can consider that $\rho_{i,j}(\varphi)$ is true in her state) and, in exchange, i requires j to establish $\rho_{i,j}(\psi)$ for her. For example, the conditional proposal $h\_nail \overset{A,B}{\Rightarrow} h\_screw$ from A to B in Ex. 1 states that A wants to exchange her $nail$ for B's screw. A can make this offer if she has a nail, and it will have a screw in the resulting state after the proposal is accepted by B.

Agents will negotiate by exchanging proposals. Each agent has her own way of evaluating and assimilating proposals within her knowledge base. We will assume that each agent i is associated with three functions, $RPre_i$, $RPost_i$, and $OPost_i$, which map models and proposals to sets of models. The use of separate originators/receivers functions allows us to formalize various types of negotiations, e.g., asymmetric ones.

$OPost_i$ describes the possible states i will be in if her proposal is accepted. $RPre_i$ and $RPost_i$ represent the conditions for i to consider a received proposal and the consequences of accepting it. These functions satisfy the following conditions:

- $RPre_i(w, \varphi \overset{j,i}{\Rightarrow} \psi) \subseteq \mathcal{W}_i$ and for each $w' \in RPre_i(w, \varphi \overset{j,i}{\Rightarrow} \psi)$, $w' \models \rho_{j,i}(\psi)$.
- $RPost_i(w, \varphi \overset{j,i}{\Rightarrow} \psi) \subseteq \mathcal{W}_i$.
- $OPost_i(w, \varphi \overset{i,j}{\Rightarrow} \psi) \subseteq \mathcal{W}_i$ and for each $w' \in OPost_i(w, \varphi \overset{i,j}{\Rightarrow} \psi)$, $w' \models \psi$.

The condition on $RPre_i$ indicates that if i wants to accept the proposal $\varphi \overset{j,i}{\Rightarrow} \psi$ then she should (somehow) have $\rho_{j,i}(\psi)$ to satisfy the proposal. $OPost_i$ requires that if i made the proposal $\varphi \overset{i,j}{\Rightarrow} \psi$ then she should have ψ if the proposal is accepted by j. Finally, for all functions, it is required that the agent considers only acceptable states.

Example 5. Consider the agents in Example 4. A possible definition for $RPre_A$ w.r.t. the proposal $h\_screw \overset{B,A}{\Rightarrow} h\_nail$ is

$$RPre_A(w, h\_screw \overset{B,A}{\Rightarrow} h\_nail) = \begin{cases} \{w\} & \text{if } w \models h\_nail \\ \emptyset & \text{otherwise} \end{cases}$$

(i.e., to accept the proposal, A should have a nail). A possible definition of $RPost_A$

w.r.t. the proposal $h\_screw \overset{B,A}{\Rightarrow} h\_nail$ is

$$RPost_A(w, h\_screw \overset{B,A}{\Rightarrow} h\_nail) = \begin{cases} \{w' \mid w' = w \setminus \{\neg h\_screw, h\_nail\} \cup \\ \qquad \{h\_screw, \neg h\_nail\}\} \text{ if } w \models h\_nail \\ \emptyset \quad \text{otherwise} \end{cases}$$

A possible definition of $OPost_B$ w.r.t. the proposal $h\_screw \overset{B,A}{\Rightarrow} h\_nail$ is given next:

$$OPost_B(w, h\_screw \overset{B,A}{\Rightarrow} h\_nail) = \begin{cases} \{w' \mid w' = w \setminus \{h\_screw, \neg h\_nail\} \cup \\ \qquad \{\neg h\_screw, h\_nail\}\} \text{ if } w \models h\_screw \\ \emptyset \quad \text{otherwise} \end{cases}$$

Let us define when an agent can make a proposal and what she can do if a proposal was made to her. If a proposal $\varphi \overset{i,j}{\Rightarrow} \psi$ is made, j can either accept the proposal, reject it, or continue with the negotiation. First, if j were to accept the proposal, then $RPre_j(w, \varphi \overset{i,j}{\Rightarrow} \psi)$ should not be empty, since this set tells j that she can satisfy the request of i (e.g., she has the formula that is being requested by i). Furthermore, $RPost_j(w, \varphi \overset{i,j}{\Rightarrow} \psi)$ should not be empty as this set indicates that the consequence of accepting the proposal is acceptable to j. Otherwise, j can make a counter proposal or reject the offer. A counter proposal can only be made if j thinks that she can offer $\rho_{i,j}(\psi)$ to i in exchange for φ', i.e., $\rho_{j,i}(\varphi') \overset{i,j}{\Rightarrow} \psi$ should be a possible proposal. If j cannot accept the proposal and cannot make a counter proposal to i, then she will reject the proposal. We formulate these notions in the next definitions—where $\alpha = \varphi \overset{i,j}{\Rightarrow} \psi$ is a proposal from i to j and w_i and w_j are the current states of i and j, respectively.

Definition 8. *α is acceptable to both i and j w.r.t. w_i and w_j if*

- *$w_i \models \varphi$ and $OPost_i(w_i, \alpha) \neq \emptyset$ (α is O-acceptable w.r.t. w_i).*
- *$RPre_j(w_j, \alpha) \neq \emptyset$ and $RPost_j(w_j, \alpha) \neq \emptyset$ (α is R-acceptable w.r.t. w_j).*

Observe that the definitions of $RPre_j$ and $RPost_j$ take care of converting the formulae in the proposal (that are in the language of i) to the local language of j.

Definition 9. *α is R-negotiable w.r.t. w_j if α is not R-acceptable w.r.t. w_j, and there is some φ' (in the language of j) s.t. $RPre_j(w_j, \beta) \neq \emptyset$ and $RPost_j(w_j, \beta) \neq \emptyset$ where $\beta = \rho_{j,i}(\varphi') \overset{i,j}{\Rightarrow} \psi$. α is R-rejectable if it is not R-acceptable and not R-negotiable.*

Definition 10. *α is O-negotiable w.r.t. w_i if α is not O-acceptable w.r.t. w_i, and there exists some φ' such that $w_i \models \varphi'$ and $OPost_i(w_i, \varphi' \overset{i,j}{\Rightarrow} \psi) \neq \emptyset$. α is O-rejectable if it is not O-acceptable and not O-negotiable.*

Example 6. Let us assume that $RPre_B$, $RPost_B$, and $OPost_B$ are defined similarly to $RPre_A$, $RPost_A$, and $OPost_A$ in Example 5. Given two states $w_A = I_A$ and $w_B = I_B$ (Example 3) of A and B, we can easily check the following: **(a)** $h\_nail \overset{A,B}{\Rightarrow} h\_screw$ is O-negotiable in w_A; **(b)** $true \overset{A,B}{\Rightarrow} h\_screw$ is O-acceptable w.r.t. w_A and R-acceptable to B; **(c)** $h\_nail \overset{A,B}{\Rightarrow} h\_screw$ is R-acceptable in w_B. □

We will next define the notion of a negotiation. An *exchange* between i and j is either a formula in $\mathcal{L}_i$ or a *critique*, which is either `accept` or `reject`.

Definition 11. *Let i and j be two agents and w_i and w_j be their current states. A sequence of exchanges $m_0, \ldots, m_n, \ldots$ is a* (i,j)-negotiation for ψ w.r.t. w_i and w_j *if*

- *for every k, $m_{2k} \overset{i,j}{\Rightarrow} \psi$ is O-negotiable w.r.t. w_i and $m_{2k+1} \overset{i,j}{\Rightarrow} \psi$ is R-negotiable w.r.t. w_j;*
- *if m_n is a critique then the sequence is finite and*
  - *if $m_n =$ `accept` then $m_{n-1} \overset{i,j}{\Rightarrow} \psi$ is acceptable w.r.t. w_i and w_j.*
  - *if $m_n =$ `reject` then $m_{n-1} \overset{i,j}{\Rightarrow} \psi$ is O-rejectable w.r.t. w_i if n is even or $m_{n-1} \overset{i,j}{\Rightarrow} \psi$ is R-rejectable w.r.t. w_j if n is odd.*

Since acceptance of a proposal leads two agents to possibly change states, states compatibility becomes an issue:

Definition 12. *Let i and j be two agents and w_i and w_j be their states. A finite* (i,j)-negotiation $(m_0, \ldots, m_n)$ for ψ w.r.t. w_i and w_j is *practical* if $m_n =$ `accept` and there exists a pair $(w_i', w_j') \in \mathcal{R}_{i,j}$ such that $w_i' \in OPost_i(w_i, m_{n-1} \overset{i,j}{\Rightarrow} \psi)$ and $w_j' \in RPost_j(w_j, m_{n-1} \overset{i,j}{\Rightarrow} \psi)$. We say that $m_{n-1} \overset{i,j}{\Rightarrow} \psi$ is the *outcome* of the negotiation.

Definition 13. *A (i,j)-negotiation $m_0, \ldots$ for ψ w.r.t. w_i and w_j is non-repeating if, for every pair of $k \neq t$, m_k is not logically equivalent to m_t (i.e., $\not\models m_k \Leftrightarrow m_t$).*

The following theorem is an immediate consequence of the finiteness of the languages of i and j and the definition of non-repeating negotiation.

Theorem 2. *Any non-repeating (i,j)-negotiation for ψ w.r.t. w_i and w_j is finite.*

5 Integration of Negotiation in Multiagent Planning

In this section, we will integrate the proposed method for negotiation in planning in presence of multiple agents. We have seen (e.g., Ex. 3) that a planning problem may not have a solution (e.g., agents A and B cannot achieve their goals). It is easy to see that, if A purchases a nail and exchanges it with B for a screw, then both A and B can achieve their goals. Thus, negotiation can provide the interaction between multiple agents necessary to achieve success. In order for the negotiation to be used during planning, we will need to instantiate our model of negotiation to the case of multiagent planning.

5.1 Negotiation in Multiagent Planning

Let us consider a multiagent planning problem $\mathcal{M} = \langle \mathcal{AG}, \{\mathcal{P}_i\}_{i \in \mathcal{AG}}, \mathcal{F}, \mathcal{NC}, \mathcal{C} \rangle$. In this case, the language $\mathcal{L}_i$ for negotiation used by agent i is the propositional language built using the set of fluents $\mathtt{F}_i$ in D_i. The set $\mathcal{W}_i$ of permissible states of i is the set of all possible states in $\mathcal{P}_i$.[2] In this paper, we are concerned with the case where each successful (i,j)-negotiation with outcome $\varphi \overset{i,j}{\Rightarrow} \psi$, where φ and ψ are conjunctions of literals, will result in (*i*) agent i having ψ and $\overline{\varphi}$ in the next state; and (*ii*) agent j having φ and $\overline{\psi}$ in the next state (recall that $\overline{\psi}$ denotes the $\{\overline{\ell} \mid \ell \in \psi\}$). In order to accept a proposal $\varphi \overset{i,j}{\Rightarrow} \psi$, an agent j should have ψ. This means that the functions $RPre_j$, $RPost_j$, and $OPost_i$ are defined as follows.

[2] This set could exclude some interpretations, e.g., because of the $\mathcal{F}$ constraints of Def. 4.

$$RPre_j(w, \varphi \overset{i,j}{\Rightarrow} \psi) = \begin{cases} \{w\} & \text{if } w \models \rho_{i,j}(\psi) \\ \emptyset & \text{otherwise} \end{cases}$$

$$RPost_j(w, \varphi \overset{i,j}{\Rightarrow} \psi) = \begin{cases} \{w \cup e \setminus \overline{e}\} & \text{if } w \models \rho_{i,j}(\psi) \\ \emptyset & \text{otherwise} \end{cases}$$

where $e = \rho_{i,j}(\varphi) \cup \overline{\rho_{i,j}(\psi)}$. Furthermore,

$$OPost_i(w, \varphi \overset{i,j}{\Rightarrow} \psi) = \begin{cases} \{w \cup e' \setminus \overline{e'}\} & \text{if } w \models \varphi \\ \emptyset & \text{otherwise} \end{cases}$$

where $e' = \psi \cup \overline{\varphi}$. The model compatibility relation $\mathcal{R}_{i,j}$ consists of (s, s') if there exists a combined state $\langle w_t \rangle_{t \in \mathcal{AG}}$ such that $s = w_i$, $s' = w_j$, and the formulas in $\mathcal{F}$ are satisfied by $\langle w_t \rangle_{t \in \mathcal{AG}}$. As there is no explicit requirement on the languages used in formalizing $\mathcal{M}$, we will keep assuming the existence of a language matching function ρ. In our examples, ρ will simply correspond to the identity function.

5.2 Planning with Non-Interleaved Negotiations

The first approach we consider is the case where agents participating in a negotiation are prevented from performing any other activities until the negotiation is complete.

Let us assume that each finite length negotiation between any two distinct agents in $\mathcal{AG}$ is assigned a unique name, and let us denote with $N_{i,j}$ the set of the names of all finite (i, j)-negotiations. A *joint-action sequence with negotiation* of length k for the agents in $\mathcal{AG}$ is a sequence $\langle \alpha_i \rangle_{i \in \mathcal{AG}}$ where, for each $i \in \mathcal{AG}$, $\alpha_i = [a_0^i, \ldots, a_k^i]$ and, for each $0 \leq l \leq k$, a_l^i is either an action in D_i or an element of $N_{i,j} \cup N_{j,i}$. A joint-action sequence $\langle \alpha_i \rangle_{i \in \mathcal{AG}}$ of length k is said to be *compatible* if: *(i)* It is compatible w.r.t. Definition 5, and *(ii)* If $a_l^i \in N_{i,j} \cup N_{j,i}$ then $a_l^j = a_l^i$. This is illustrated next.

Example 7. Consider the multiagent planning problem $\mathcal{M}_{AB}$ in Ex. 3, where $\mathcal{M}_{AB}$ has no solution. It is easy to see that the following *joint-action sequence* can achieve the goals of both A and B: *(1)* A buys a nail. *(2)* A proposes to B to exchange a screw for a nail. *(3)* B accepts the proposal of A and the exchange is made. *(4)* A hangs the mirror with her screwdriver and the screw. *(5)* B hangs the picture with the nail and the hammer. This can be represented as $\langle \alpha_i \rangle_{i \in \{A,B\}}$, where $\alpha_A = [buy\_nail, N_1, hw\_screw]$ and $\alpha_B = [\texttt{noop}, N_1, hw\_nail]$, where $N_1 = h\_nail \overset{A,B}{\Rightarrow} h\_screw, \texttt{accept}$. □

Let us extend the definition of the transition function Φ to encompass negotiations.

Definition 14. *Let $N \in N_{i,j}$ be an (i, j)-negotiation and s_i and s_j be the states of i and j, respectively. We define $\Phi(N, s_i)$ and $\Phi(N, s_j)$ as follows.*

- *If N ends with* `reject`*, then $\Phi(N, s_i) = s_i$ and $\Phi(N, s_j) = s_j$.*
- *If N ends with* `accept` *and the outcome is $\varphi \overset{i,j}{\Rightarrow} \psi$, then $\Phi(N, s_i) = OPost_i(s_i, \varphi \overset{i,j}{\Rightarrow} \psi)$ and $\Phi(N, s_j) = RPost_j(s_j, \varphi \overset{i,j}{\Rightarrow} \psi)$.* [3]

[3] Note that we slightly abuse the notation since $OPost$ and $RPost$ return singleton sets.

The function $\widehat{\Phi}$ can be extended to the case of sequences of actions with negotiations. Definition 6 can then be used to define the notion of a joint-plan with negotiation for multi-agent planning problems. For example, it is easy to see that $\langle \alpha_i \rangle_{i \in \{A,B\}}$ (Example 7) is a joint-plan with negotiation for A and B in the problem $\mathcal{M}_{AB}$.

5.3 Planning with Interleaved Negotiations

A joint-plan with negotiation as defined in the previous subsection does not consider the case where agents may align themselves to make a proposal acceptable. For example, if B makes the proposal $h\_screw \overset{B,A}{\Rightarrow} h\_nail$ to A in the initial state, A—by virtue of having no nail—will reject it. On the other hand, A can accept the proposal after it purchases a nail, i.e., the following could be considered a joint-plan for A and B:

$$\alpha_A = [\texttt{noop}, buy\_nail, \texttt{accept}, hw\_screw]$$
$$\alpha_B = [h\_screw \overset{B,A}{\Rightarrow} h\_nail, \texttt{noop}, \texttt{noop}, hw\_nail]$$

In the above joint-plan, individual exchanges of a negotiation act like individual actions. To accommodate this, we introduce *negotiation actions* of the following forms:

a. $starts(i, j, \varphi, \psi)$: i starts a negotiation with j, by making the proposal $\varphi \overset{i,j}{\Rightarrow} \psi$;
b. $proposes(i, j, \varphi, \psi)$: φ is a non-critique exchange in an (i, j)-negotiation for ψ;
c. $accepts(i, j, \varphi, \psi)$: i and j accept the proposal $\varphi \overset{i,j}{\Rightarrow} \psi$;
d. $rejects(i, j)$: i and j reject the last exchange made and terminate the negotiation.

These actions are referred to as (i, j)-*negotiation actions*.

The notion of a compatible joint-action sequence has to be modified to account for negotiation actions. Different views may lead to different definitions of a joint-action with negotiation actions. In the following, we will require the following:

- at any time, one agent is engaged in at most one negotiation; and
- agents must finish one negotiation before they can start a new one.

We extend the definition of transition function to account for the negotiation actions:

$$\Phi_{D_i}(starts(i, j, \varphi, \psi), w) = w$$
$$\Phi_{D_j}(starts(i, j, \varphi, \psi), w) = \textit{fails}$$
$$\Phi_{D_x}(proposes(i, j, \varphi, \psi), w) = w$$
$$\Phi_{D_x}(rejects(i, j), w) = w$$
$$\Phi_{D_i}(accepts(i, j, \varphi, \psi), w) = OPost_i(w, \varphi \overset{i,j}{\Rightarrow} \psi)$$
$$\Phi_{D_j}(accepts(i, j, \varphi, \psi), w) = RPost_j(w, \varphi \overset{i,j}{\Rightarrow} \psi)$$

where $x \in \{i, j\}$. Let $\alpha_i = [a_1^i, \ldots, a_k^i]$ and $\alpha_j = [a_1^j, \ldots, a_k^j]$ be two action sequences, containing ordinary actions from D_i and D_j and/or negotiation actions. Let $\alpha_i \oplus \alpha_j = [C_1, \ldots, C_k]$ where C_l is the set of negotiation actions among $\{a_l^i, a_l^j\}$. Let $Ag(C_l) = \{x \mid a_l^x \in C_l\}$. We say that (α_i, α_j) is *syntactically correct* if

- C_l is either an empty set or a singleton, i.e., $|C_l| \leq 1$;
- for each $1 \leq l < l' \leq k$ s.t. $C_l \neq \emptyset$ and $C_{l'} \neq \emptyset$, if $C_{l+1} = \cdots = C_{l'-1} = \emptyset$ then $Ag(C_l) \cup Ag(C_{l'}) = \{i, j\}$.

The pair (α_i, α_j) is (i, j)-*syntactically correct* if it is syntactically correct and

- either no (i, j)-negotiation action occurs in $\alpha_i \oplus \alpha_j$, or
- for each (i, j)-negotiation action a_x^i or a_x^j in $\alpha_i \oplus \alpha_j$ there exists $l \neq l'$ such that **(a)** $l \leq x \leq l'$, **(b)** $C_l = \{a_l^i\} = \{starts(i, j, \varphi_l, \psi)\}$, **(c)** for every $l < t < l'$, $C_t = \emptyset$

or $C_t = \{proposes(i, j, \varphi_t, \psi)\}$, and **(d)** either $a^i_{l'} = a^j_{l'} = accepts(i, j, \varphi_{l'}, \psi)$ or $a^i_{l'} = a^j_{l'} = rejects(i, j)$.

We say a joint-action sequence $\langle \alpha_i \rangle_{i \in \mathcal{AG}}$ of length k, where each a^i_l is an action in $\mathcal{P}_i$ or one of the negotiation actions, is *compatible* if:

- it satisfies the conditions in Definition 5; and
- for every $i \neq j$, (α_i, α_j) is (i, j)-syntactically correct and, for every C_t in $\alpha_i \oplus \alpha_j$ containing an (i, j)-negotiation action a we have that:
  - if $a = starts(i, j, \varphi, \psi)$ then $\varphi \overset{i,j}{\Rightarrow} \psi$ is O-negotiable w.r.t. $\hat{\Phi}_{D_i}(\alpha_i[t-1], I_i)$;
  - if $a = proposes(i, j, \varphi, \psi)$ and $Ag(C_t) = i$ then $\varphi \overset{i,j}{\Rightarrow} \psi$ is O-negotiable w.r.t. $\hat{\Phi}_{D_i}(\alpha_i[t-1], I_i)$;
  - if $a = proposes(i, j, \varphi, \psi)$ and $Ag(C_t) = j$ then $\varphi \overset{i,j}{\Rightarrow} \psi$ is R-negotiable w.r.t. $\hat{\Phi}_{D_j}(\alpha_j[t-1], I_j)$;
  - if $a = accepts(i, j, \varphi, \psi)$ then $\varphi \overset{i,j}{\Rightarrow} \psi$ is acceptable w.r.t. $\hat{\Phi}_{D_i}(\alpha_i[t-1], I_i)$ and $\hat{\Phi}_{D_j}(\alpha_j[t-1], I_j)$;
  - if $a = rejects(i, j)$ and $proposes(i, j, \varphi, \psi)$ or $starts(i, j, \varphi, \psi)$ is the last occurrence of an (i, j)-negotiation action in $\alpha_i \oplus \alpha_j$ before t, then $\varphi \overset{i,j}{\Rightarrow} \psi$ is O-rejectable w.r.t. $\hat{\Phi}_{D_i}(\alpha_i[t-1], I_i)$ or R-rejectable w.r.t. $\hat{\Phi}_{D_j}(\alpha_j[t-1], I_j)$.

Joint-plans with negotiation are defined accordingly.

5.4 Computing Plan with Negotiation Using Logic Programming

In the rest of this section, we will present a set of rules that, when added to $\Pi^n(\mathcal{M})$, will generate joint-plans with negotiation. We refer to the new program as $\Gamma^n(\mathcal{M})$. Due to lack of space, we omit some of the more technical details and we make use of a rather informal logic programming syntax. In the following rules, i, j, and k denote possible agents. As in the previous discussion, we will consider the case where formulae involved in negotiations are composed only of sets of literals. We assume the existence of negotiation formula atoms in the program `formula_name`$(.)$, naming the possible set of literals. The composition of the actual formula φ can be described as a collection of facts `in_formula`(φ, ℓ) for each literal $\ell \in \varphi$ (with a slight abuse of notation, we will use φ as the name of the formula itself). We also assume that the language matching functions are identities. For each agent i, we introduce

$$NA_i = \left\{ \begin{array}{l} starts(i, j, \varphi, \psi), proposes(i, j, \varphi, \psi), \\ accepts(i, j, \varphi, \psi), rejects(i, j) \end{array} \middle| \; j \in \mathcal{AG}, \varphi, \psi \textit{ are fomulas names} \right\}.$$

A predicate `na`(i, a) is used to identify the elements $a \in NA_i$. Since the construction of an exchange requires hypothetical reasoning, we assume a predicate $hyp\_h(i, \varphi, \psi, \ell, T)$ that is true if ℓ is true in $OPost_i(w, \varphi \overset{i,j}{\Rightarrow} \psi)$, where w is the state for i described by $h(i, \cdot, T)$. The definition of hyp_h is straightforward. This allows us to describe states that are not acceptable; in our case:

$$\texttt{bad}(i, \varphi, \psi, T) \leftarrow fluent(i, f), hyp\_h(i, \varphi, \psi, f), hyp\_h(i, \varphi, \psi, neg(f))$$

The rules used to describe the effects of negotiation can be summarized as follows.

• *Generation rules:* The rule for generating action occurrences is expanded to:

$$1\{o(i, a, T) : \texttt{na}(i, a), o(i, A, T) : action(i, A)\}1 \leftarrow agent(i), time(T), T < n$$

• *Negotiation rules:* These rules control the ability to perform steps of negotiation; the predicate `wait` is used to indicate that it is not the agent's turn to respond to a negotiation, allowing to enforce the exchange protocol described earlier. Each non-critique exchange will prompt `wait` to become true; each generated exchange will also invalidate the `wait` of the other party. In the following, $x \in \{i, j\}$ and $\bar{i} = j$ and $\bar{j} = i$. The variable T denotes the time parameter.

$$
\begin{array}{ll}
\%\% \quad \textit{i starts} & \\
h(i, wait(i,j,\varphi,\psi), T+1) & \leftarrow o(i, starts(i,j,\varphi,\psi), T). \\
h(j, neg(wait(i,j,\varphi,\psi)), T+1) & \leftarrow o(i, starts(i,j,\varphi,\psi), T). \\
\%\% \quad \textit{x exchanges} & \\
h(x, wait(i,j,\varphi,\psi), T+1) & \leftarrow o(x, proposes(i,j,\varphi,\psi), T). \\
h(\bar{x}, neg(wait(i,j,\varphi,\psi)), T+1) & \leftarrow o(x, proposes(i,j,\varphi,\psi), T). \\
\%\% \quad \textit{Suspend waiting on termination} & \\
h(x, neg(wait(i,j,\varphi,\psi)), T+1) & \leftarrow o(x, accepts(i,j,\varphi,\psi), T). \\
h(x, neg(wait(i,j,\varphi,\psi)), T+1) & \leftarrow o(x, rejects(i,j), T).
\end{array}
$$

A successful completion of a negotiation will be achieved when an acceptable exchange is reached. The `bad` predicate will be used to validate acceptability:

$$
\begin{array}{l}
\texttt{acceptable}(i,j,T) \leftarrow h(i, wait(i,j,\varphi,\psi), T), h(i,\varphi,T), not\ \ bad(i,\varphi,\psi,T). \\
\texttt{acceptable}(i,j,T) \leftarrow h(j, wait(i,j,\varphi,\psi), T), h(j,\psi,T), not\ \ bad(j,\psi,\varphi,T).
\end{array}
$$

If the negotiation is not acceptable, then an attempt to generate a new exchange is made. We use a $repeated$ predicate to avoid repeated exchanges (the code is simple but tedious, and omitted).

$$
\begin{array}{ll}
\texttt{valid\_proposal}(i,j,\varphi',T) \leftarrow & h(i, wait(i,j,\varphi,\psi), T), formula\_name(\varphi'), \\
& not\ \ repeated(i,j,\varphi',T), h(i,\varphi',T), not\ \ bad(i,\varphi',\psi,T). \\
\texttt{valid\_proposal}(i,j,\varphi',T) \leftarrow & h(j, wait(i,j,\varphi,\psi), T), formula\_name(\varphi'), \\
& not\ \ repeated(i,j,\varphi',T), h(j,\psi,T), not\ \ bad(j,\psi,\varphi',T). \\
\texttt{negotiable}(i,j,T) \leftarrow & h(i, wait(i,j,\varphi,\psi), T), not\ \texttt{acceptable}(i,j,T), \\
& \texttt{valid\_proposal}(i,j,\varphi',T). \\
\texttt{negotiable}(i,j,T) \leftarrow & h(j, wait(i,j,\varphi,\psi), T), not\ \texttt{acceptable}(i,j,T), \\
& \texttt{valid\_proposal}(i,j,\varphi',T).
\end{array}
$$

Using the above characterizations of what is acceptable and negotiable, we can introduce constraints that will avoid generation of unsuitable negotiation actions:

$$
\begin{array}{ll}
\%\% \quad \textit{Protocol: actions must be done in exchange way} & \\
\quad \leftarrow o(x, proposes(i,j,\varphi',\psi'), T), h(x, neg(wait(i,j,\varphi,\psi)), T). & \\
\quad \leftarrow o(x, accepts(i,j,\varphi',\psi'), T), h(x, neg(wait(i,j,\varphi,\psi)), T). & \\
\quad \leftarrow o(x, rejects(i,j)), h(x, neg(wait(i,j,\varphi,\psi)), T). & \\
\%\% \quad \textit{Ensure only valid actions are performed} & \\
\quad \leftarrow o(i, starts(i,j,\varphi,\psi), T), not\ \ h(i,\varphi,T). & \\
\quad \leftarrow o(i, proposes(i,j,\varphi,\psi), T), not\ \texttt{valid\_proposal}(i,j,\varphi,T). & \\
\quad \leftarrow o(j, proposes(i,j,\varphi,\psi), T), not\ \texttt{valid\_proposal}(i,j,\varphi,T). & \\
\quad \leftarrow o(i, accepts(i,j,\varphi,\psi), T), not\ \texttt{acceptable}(i,j,T). & \\
\quad \leftarrow o(j, accepts(i,j,\varphi,\psi), T), not\ \texttt{acceptable}(i,j,T). & \\
\quad \leftarrow o(i, rejects(i,j), T), not\ \ negotiable(i,j,T), not\ \ acceptable(i,j,T). &
\end{array}
$$

Finally, we introduce rules to describe the state changes produced by a negotiation:

$$h(i,\ell,T+1) \leftarrow o(i, accepts(i,j,\varphi,\psi), T), \texttt{in\_formula}(\psi,\ell).$$

$h(i, \bar{\ell}, T+1) \leftarrow o(i, accepts(i, j, \varphi, \psi), T), \texttt{in\_formula}(\varphi, \ell).$
$h(j, \ell, T+1) \leftarrow o(j, accepts(i, j, \varphi, \psi), T), \texttt{in\_formula}(\varphi, \ell).$
$h(j, \bar{\ell}, T+1) \leftarrow o(j, accepts(i, j, \varphi, \psi), T), \texttt{in\_formula}(\psi, \ell).$

Theorem 3. *For a multiagent planning problem $\mathcal{M}$ and an integer n,*

- *if I is an answer set of $\Gamma^n(\mathcal{M})$ and $\alpha_i = [a_0^i, \ldots, a_{n-1}^i]$ such that $o(i, a_t^i, t) \in I$, then $\langle \alpha_i \rangle_{i \in \mathcal{AG}}$ is a joint-plan.*
- *if $\langle \alpha_i \rangle_{i \in \mathcal{AG}}$ where $\alpha_i = [a_0^i, \ldots, a_{n-1}^i]$ is a joint-plan of length n, then there is an answer set I of $\Gamma^n(\mathcal{M})$ such that $o(i, a_t^i, t) \in I$ for $i \in \mathcal{AG}$ and $0 \leq t \leq n-1$.*

Observe that the program $\Gamma^n(\mathcal{M})$ can be easily modified to generate plans with non-interleaved negotiations by adding constraints forbidding an occurrence of an ordinary action for an agent i when the fluent $wait(i, j, ., .)$ or $wait(j, i, ., .)$ is true.

6 Discussion and Conclusions

In this paper, we presented a preliminary investigation of the use of logic programming technology to address the composite problem of multiagent planning and negotiation. We developed a generic model of negotiation, suitable for dynamic environments, and instantiate it to the case of multiple planning agents with independent goals. We defined different notions of planning with negotiations. We illustrated how logic programming provides an elegant and modular encoding of the different aspects of the problem. Observe that the use of logic programming allows for a simple integration between planning and negotiation. The generation of an answer set satisfying the goal of the planning problem drives the generation of any negotiation that needs to occur between agents.

This preliminary work offers several directions for future research. Part of our discussion relies on the use of identity function as a language matching functions. This restriction was imposed for the sake of simplicity and it should be lifted to allow more complex scenarios. For example, in planning with resources, two agents—one using British pounds and one using Dollars—would need matching functions that convert between the two currencies. Other examples include agents using ontologies with different granularities; e.g., agent i uses $has\_nail\_brand\_x(foo)$ to describe nail named foo, while agent j uses separate predicates to describe individual properties of foo (e.g., $is\_nail(foo)$ and $is\_brand\_x(foo)$). Thus, $\rho_{i,j}$ applied to $has\_nail\_brand\_x(foo)$ should result in the formula $is\_nail(foo) \wedge is\_brand\_x(foo) \wedge has(foo)$. The language matching functions are domain-dependent but still expressible in logic programming.

The proposed model of negotiation is grounded and used in the most simple way. It is suitable for negotiations involving exchanges of consumable resources, such as the nail or the screw. Non-consumable resources may require different definitions of the functions $OPost$, $RPre$, and $RPost$. In practice, there could be situations in which the owner of a resource does not lose it after an exchange has happened. For example, a student, agreeing to give another student a ride to school in exchange for the solution of a homework, does not lose her car after the exchange.

There are also situations in which an agent may need to take into consideration what other agents offer before deciding to accept or reject an offer. For example, a student with a car without gasoline could agree to drive some friends to school if the friends give her enough money to buy gasoline. In this case, the student has to take into consideration what was offered before accepting the offer.

The implementation in planning assumes that agents negotiate on sets of literals. This is not a limitation of the general planning framework, but the consequence of the language restrictions of several answer set solvers. This restriction could be lifted by adopting a more general logic programming framework e.g., [3, 7].

Further generalizations include the use of negotiation models involving groups of agents, preferences, more expressive action languages (e.g., with static causal laws, concurrent actions), and more complex planning scenarios (e.g., joint-goals).

References

1. L. Amgoud, Y. Dimopoulos, and P. Moraitis. A unified and general framework for argumentation-based negotiation. *AAMAS*, pages 1018–1025. ACM Press, 2006.
2. C. Boutilier and R. I. Brafman. Partial-order planning with concurrent interacting actions. *J. Artif. Intell. Res.*, 14:105–136, 2001.
3. L. Castro et al. XASP: Answer Set Programming with XSB and Smodels, 2007.
4. W. Chen, M. Zhang, and N. Foo. Repeated negotiation of logic programs. *Workshop on Nonmonotonic Reasoning, Action and Change*, 2006.
5. J. S. Cox and E. H. Durfee. An efficient algorithm for multiagent plan coordination. *AAMAS*, pages 828–835. ACM, 2005.
6. 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.
7. O. El-Khatib, E. Pontelli, and T. Son. ASP-Prolog: A System for Reasoning about Answer Set Programs in Prolog. *PADL*, pages 148–162. Springer, 2004.
8. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Int. Conf. on Logic Programming*, 1070–1080, 1988.
9. M. Gelfond and V. Lifschitz. Representing actions and change by logic programs. *JLP*, 17(2,3,4):301–323, 1993.
10. G. Gelfond and R. Watson. Modeling Cooperative Multi-Agent Systems, ASP-2007.
11. M. Ghallab, D. Nau, and P. Traverso. *Automated Planning*. Morgan Kaufmann, CA, 2004.
12. J. S. Cox and E. H. Durfee and T. Bartold. A Distributed Framework for Solving the Multi-agent Plan Coordination Problem. In *AAMAS*, pages 821–827. ACM Press, 2005.
13. A. Kakas and P. Moraitis. Adaptive agent negotiation via argumentation. *AAMAS*, pages 384–391, ACM Press, 2006.
14. A. Kakas, P. Torroni, and N. Demetriou. Agent planning, negotiation and control of operation. *ECAI*, pages 28–32. IOS Press, 2004.
15. S. Kraus. Negotiation and cooperation in multi-agent environments. *AIJ*, 94(1-2), 1997.
16. V. Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1–2):39–54, 2002.
17. I. Rahwan et al. Argumentation-based negotiation. *Knowledge Engineering Review*, 18:343–375, 2003.
18. S. Parsons et al. Agents that reason and negotiate by arguing. *JLC*, 8(3):261–292, 1998.
19. F. Sadri, F. Toni, and P. Torroni. An abductive logic programming architecture for negotiating agents. *JELIA*, pages 419–431. Springer Verlag, 2002.
20. C. Sakama and K. Inoue. Negotiation by abduction and relaxation. *AAMAS*, pages 1018–1025. ACM Press, 2007.
21. P. Simons, N. Niemelä, and T. Soininen. Extending and Implementing the Stable Model Semantics. *Artificial Intelligence*, 138(1–2):181–234, 2002.
22. T. C. Son, C. Baral, N. Tran, and S. McIlraith. Domain-dependent knowledge in answer set planning. *ACM Trans. Comput. Logic*, 7(4):613–657, 2006.
23. V. Subrahmanian and C. Zaniolo. Relating stable models and AI planning domains. In *Proceedings of the International Conference on Logic Programming*, pages 233–247, 1995.
24. M. Wooldridge and S. Parsons. Languages for negotiation. In *Proceedings of ECAI*, 2000.