ASP-Prolog for Negotiation Among Dishonest Agents

Ngoc-Hieu Nguyen[1], Tran Cao Son[1], Enrico Pontelli[1], and Chiaki Sakama[2]

[1] New Mexico State University, nhieu|tson|epontell@cs.nmsu.edu
[2] Wakayama University, sakama@sys.wakayama-u.ac.jp

Abstract. This paper describes a platform to develop negotiating agents, whose knowledge and rules of behavior are represented as Abductive Logic Programs. The platform implements a flexible negotiation framework. Negotiating agents can operate with multiple goals and incomplete knowledge, and dynamically modify their goals depending on the progress of the negotiation exchanges. Differently from other frameworks, agents can operate dishonestly, by generating false statements or statements that are not substantiated by the agents' knowledge. The proposed platform has been implemented using the ASP-Prolog platform.

1 Introduction

Real-world interactions among agents often require the ability to perform *negotiation*—i.e., perform sequences of offers/counter-offers to reach a consensus about possible exchanges of knowledge and/or goods. The issue of modeling negotiation in multi-agent systems is an important area of research that has received significant attention, e.g., [1, 5, 6, 9, 11, 14]. Nevertheless, of the many frameworks and theories proposed to model negotiation among agents, relatively few are capable of handling *incomplete information* (e.g., [3, 13]) and even fewer have considered the case of agents that are lying or misleading other agents during the negotiation process [15].

We have recently [12] developed a theoretical framework that enables modeling negotiation among two agents, where each agent can introduce deception in the generation of offers/counter-offers, and he has incomplete knowledge about the other agent and preferences on the choice of possible counter-offers. The proposal builds on the use of *Abductive Logic Programming (ALP) with preferences* [10] to encode the knowledge bases of the agents and their strategies for negotiation (including strategies to introduce deception). It enables to model interactions such as the one in the following example.

Example 1. A buyer agent and a seller agent negotiate the purchase/sale of a camera. The buyer starts the negotiation by indicating the desire to purchase a camera produced by maker C, at the lowest price and of good quality. The seller responds that the model A is produced by C and has good quality, and it is sold at discounted price to students—in this case, the seller may mislead the buyer, by stating knowledge about the quality of product A that he/she may not have. The buyer could honestly respond that he is not a student. The seller may attempt to draw the attention on product B, a camera produced by maker D, of good quality, sold at discounted priced if paid in cash—in this case, the seller is lying, as he/she knows that B is a poor quality product. The buyer is aware of the poor quality, and refuses to accept the offer to purchase B at the discounted price.

One more time, the seller focuses on B by offering a further discount if the buyer joins the seller's mailing list. At that point, the buyer accepts the offer (but he/she really has no intention to subscribe to the mailing list, thus he/she lies about it). □

In this paper, we describe a platform that implements a variant of ALP necessary to support the negotiation process, and its use in modeling agents that can negotiate in presence of incomplete knowledge, deception, and preferences. The framework is realized using a combination of answer set programming and Prolog, as provided by the ASP-Prolog system [8].

2 Background

2.1 Abductive Logic Programming (ALP), Preferences, and Disinformation

We use an extension of abductive logic programs defined in [10]. An *abductive program* is a pair (P^r, P^a) where P^r and P^a are disjunctive logic programs. The rules in P^a are referred to as *abducibles*. A set of literals S is a *belief set* of (P^r, P^a) if S is an answer set of $P^r \cup E$ for some $E \subseteq P^a$. (P^r, P^a) is *consistent* if it has a belief set; otherwise, it is inconsistent. For a program (P^r, P^a) and a set of rules X, $P \cup^r X$ (resp. $P \cup^a X$) denotes the abductive program $(P^r \cup X, P^a)$ (resp. $(P^r, P^a \cup X)$). Given a set of literals S, we denote $S^\neg = \{ \neg\ell \mid \ell \in S \}$ ($\neg\neg a$ represents the atom a).

For the discussion in the following sections, we associate a name n_r to each rule r and freely use the name to represent the rule.[1] When multiple sets of abducible rules can be used to generate belief sets of a program, a *preference* relation among abducibles, in the form of $prefer(n_1, n_2)$, can be introduced, allowing us to define a preference relation among belief sets. It is assumed that $prefer$ is a transitive and asymmetric relation among abducibles of a program. The relation $prefer$ is also extended to define a preference relation among sets of abducible rules as follows: for $Q_1, Q_2 \subseteq P^a$, Q_1 is preferred to Q_2 if either (*i*) $Q_1 \subseteq Q_2$ or (*ii*) there exist $n_1 \in Q_1 \setminus Q_2$ and $n_2 \in Q_2 \setminus Q_1$ such that $prefer(n_1, n_2)$ holds and there exists no $n_3 \in Q_2 \setminus Q_1$ such that $prefer(n_3, n_1)$ holds. In turn, this allows the comparison of belief sets of a program $P = (P^r, P^a)$; if S_1 (resp. S_2) is a belief set of P obtained from $P^r \cup Q_1$ (resp. $P^r \cup Q_2$), then S_1 is *preferred* to S_2 if Q_1 is preferred to Q_2. A belief set S of P is a *most preferred* belief set if there is no belief set S' of P that is preferred to S.

Example 2. Let us consider a seller agent s whose sale knowledge is encoded by an abductive program with preferences $P_s = (P_s^r, P_s^a)$ as follows:

- P_s^r consists of the rules:

$$
\begin{array}{rl}
senior\_customer \leftarrow & age \geq 65 \\
student\_customer \leftarrow & student \\
\neg quality_B \leftarrow & product_B \\
maker_C \leftarrow & product_A \\
maker_D \leftarrow & product_B \\
bargain \leftarrow & product_B \\
sale \leftarrow & product_A, \mathbf{price_1} \\
sale \leftarrow & product_B, \mathbf{price_2}
\end{array}
\qquad
\begin{array}{rl}
product_A \leftarrow & \\
product_B \leftarrow & \\
prefer(n_1, n_i) \leftarrow & (\text{for } i \in \{2,3,4,5\}) \\
prefer(n_i, n_5) \leftarrow & (\text{for } i \in \{2,3,4\}) \\
\leftarrow & high\_pr, low\_pr \\
\leftarrow & high\_pr, lowest\_pr \\
\leftarrow & low\_pr, lowest\_pr \\
\leftarrow & not\ sale
\end{array}
$$

[1] We omit the rule names when not needed in the discussion.

where $\mathbf{price_1} \in \{\, high\_pr,\, low\_pr \,\}$ and $\mathbf{price_2} \in \{\, low\_pr, lowest\_pr,\, high\_pr\}$. P_s^r defines various types of customers and some features of the products. It also states the sales conditions and the preferences among the abducible rules of the seller. The constraints on the prices indicate that the seller sells a product for only one price.

- $P_s^a = H_s \cup R_s$ where $H_s = \{\, age \geq 65,\ \ student,\ \ cash,\ \ mail\_list \,\}$ and R_s consists of the following rules

$$
\begin{array}{ll}
n_1 : high\_pr \leftarrow & \\
n_2 : low\_pr \ \leftarrow senior\_customer & n_4 : low\_pr \ \ \leftarrow bargain,\, cash \\
n_3 : low\_pr \ \leftarrow student\_customer & n_5 : lowest\_pr \leftarrow mail\_list,\, cash
\end{array}
$$

Intuitively, each belief set of s represents one possible way to sell a product. It is easy to see that any belief set of P_s must contain at least one of the literal $high\_pr$, low_pr, or $lowest\_pr$; the most preferred belief set of P_s contains $high\_pr$. □

Dishonest agents are those who use intentionally false or inaccurate information. For an abductive program $P = (P^r, P^a)$, a pair of disjoint sets of literals (L, B) is called *disinformation* w.r.t. P if

- L represents *lies* [7], literals that are known to be false—i.e., $\forall l \in L$, $\neg l$ belongs to every belief set of P.
- B represents *bullshit (BS)* [4], literals that are unknown—i.e., $\forall l \in B$, neither l nor $\neg l$ belongs to any belief set of P.

Given a disinformation (L, B) w.r.t. a program $P = (P^r, P^a)$, we define

$$
\begin{array}{l}
I = \{\, r \mid r \in P^r \text{ and } head(r) \cap L^{\neg} \neq \emptyset\}, \\
\Phi = \{\, prefer(n_i, n_j) \mid n_i \in I \text{ and } n_j \in P^a\} \ \cup \\
\qquad \{\, prefer(n_b, n_l) \mid n_b \in B \text{ and } n_l \in L\} \ \cup \\
\qquad \{\, prefer(n_j, n_t) \mid n_j \in P^a \cup I \text{ and } n_t \in (L \cup B)\}.
\end{array}
$$

Φ represents that (i) any rule from P^r is preferred to hypotheses in P^a, (ii) bullshit is preferred to lies, and (iii) honest information is preferred to dishonest one. The abductive program $\delta(P, L, B) = (P^r \setminus I \cup \Phi, P^a \cup I \cup L \cup B)$ is called the *Abductive Logic program with Disinformation (ALD-program)* (L, B) w.r.t. P.

Example 3. Assume that s from Example 2 can claim that both products A and B are of good quality, if needed to make a sale—i.e., s would lie about $quality_B$ and BS about $quality_A$. This is described by the disinformation: $(L_s, B_s) = (\{quality_B\}, \{quality_A\})$.

The ALD-program (L_s, B_s) w.r.t. P_s is $\delta(P_s, L_s, B_s) = (\delta P_s^r, \delta P_s^a)$ where δP_s^a is equal to $P_s^a \cup L_s \cup B_s$ plus the rule n_6 defined as $\neg quality_B \leftarrow product_B$, and $\delta P_s^r = P_s^r \setminus \{\neg quality_B \leftarrow product_B\}$ plus the preferences

- $prefer(n_6, n_i)$ for $i = 1, \ldots, 5$ and $prefer(n_6, n_h)$ where $n_h \in H_s$ (each fact in H_s is considered as a rule with the same name);
- $prefer(n_h, n_t)$ and $prefer(n_j, n_t)$ for $j = 1, \ldots, 6$, $n_h \in H_s$ and $n_t \in (L_s \cup B_s)$;
- $prefer(n_b, n_l)$ for $n_b \in B$ and $n_l \in L$. □

Because $\delta(P, L, B)$ is an abductive program, belief sets of $\delta(P, L, B)$ and the preferences among them are defined as before. For a set of belief sets Σ of $\delta(P, L, B)$, we say that $M \in \Sigma$ is a most preferred belief set of Σ if there exists no $M' \in \Sigma$ such that M' is preferred to M.

2.2 Negotiation Knowledge Base (n-KB)

A *negotiation knowledge base (n-KB)* is a tuple $\langle P, L, B, H, N^{\prec} \rangle$ where P is an abductive program encoding the agent's beliefs and rules of conduct; (L, B) is a disinformation with respect to P, representing possibly false information that the agent may use in order to obtain a deal; H is a set of assumptions about the other party; $N^{\prec}$ is a set of literals whose members represent goals (issues) that he would like to negotiate, along with a strict partial order $\prec$ among them.

Example 4. Continuing with the previous example, assume that the n-KB of the agent s be $K_s = \langle P_s, L_s, B_s, H_s, N_s^{\prec} \rangle$ where P_s, H_s, and (L_s, B_s) are given in Examples 2 and 3, respectively, and $N_s^{\prec}$ is the set of possible prices that the seller is willing to negotiate about: $N_s^{\prec} = \{\, high\_pr,\ low\_pr,\ lowest\_pr \,\}$ with $lowest\_pr \prec low\_pr \prec high\_pr$. This indicates that the seller prefers $high\_pr$ over low_pr and $lowest\_pr$. □

Example 5. An n-KB $K_b = \langle P_b, L_b, B_b, H_b, N_b^{\prec} \rangle$ with $P_b = (P_b^r, P_b^a)$ for the buyer b:

- P_b^r consists of

$$
\begin{array}{ll}
purchase \leftarrow product_X,\ quality_X,\ \mathbf{price_3} & prefer(n_1, n_2) \leftarrow \\
\leftarrow\ not\ purchase & prefer(n_2, n_3) \leftarrow \\
\neg student \leftarrow & prefer(n_1, n_3) \leftarrow \\
\leftarrow low\_pr, lowest\_pr & cash \leftarrow
\end{array}
$$

  where $X \in \{A, B\}$ and $\mathbf{price_3} \in \{\, low\_pr,\ lowest\_pr \,\}$.
- $H_b = \{\, quality_A,\ quality_B,\ maker_C,\ maker_D,\ product_A, product_B \}$.
- $N_b^{\prec} = \{\, low\_pr,\ lowest\_pr \,\}$ with $\prec = \{\, low\_pr \prec lowest\_pr \,\}$.
- $P_b^a = H_b \cup R_b$ where R_b consists of

  $n_1 : lowest\_pr \leftarrow maker_C \quad n_2 : lowest\_pr \leftarrow maker_D \quad n_3 : low\_pr \leftarrow maker_C$

The set H_b represents properties of products that the buyer needs to check and $N_b^{\prec}$ specifies that the buyer prefers to pay the lowest price. Suppose that the buyer does not want to be on the seller's mailing list but could pretend to join it if it works to his/her advantage. S/he decides to use the disinformation $(L_b, B_b) = (\emptyset, \{mail\_list\})$ w.r.t. P_b. Thus, he/she will use the ALD-program $\delta(P_b, L_b, B_b)$ in his/her negotiation. □

2.3 Negotiations using n-KBs

In this section, we review the definitions of negotiation among dishonest agents using n-KBs [12]. We consider negotiations involving two agents a and b, whose n-KBs are K_a and K_b. We assume that K_a and K_b share the same language and the set of assumptions in K_a is disjoint from the set of assumptions in K_b. A negotiation contains several rounds, where the two agents alternate in generating proposals to the other agent.

Intuitively, for an agent a, a proposal is a tuple $\langle G, S, R \rangle$ stating that the goal of a is to negotiate to achieve G. The proposal is supported by a belief set M. By making the proposal, a indicates assumptions that she has made about the receiver of the proposal (the set S), as well as information about her/himself (R) that the receiver of the proposal will have to respect. Formally, a *proposal* for $G \subseteq N$ w.r.t. $K = \langle P, L, B, H, N^{\prec} \rangle$ is a tuple $\gamma = \langle G, S, R \rangle$ where $\delta(P, L, B) \cup^r Goal(G)$[2] has a belief set M such that

[2] $Goal(G) = \{\leftarrow not\ \ell \mid \ell \in G\}$.

$S = M \cap H$, and $R \subseteq M \setminus H$. We refer to G, S, R, and M as the *goal*, *assumptions*, *conditions*, and *support* of $\langle G, S, R\rangle$, respectively. The proposal is *honest* if $M \cap (L \cup B) = \emptyset$; it is *deceptive* if $M \cap L \neq \emptyset$; and it is *unreliable* if $M \cap B \neq \emptyset$.

Let $\gamma = \langle G, S, R\rangle$ be a proposal from an agent b to an agent a, the latter described by the n-KB $K_a = \langle P, L, B, H, N^{\preceq}\rangle$. Let $\delta Q = \delta(P, L, B) \cup^r Goal(G)$.

- γ is *acceptable* w.r.t. K if δQ has a belief set M such that $S \subseteq M$ and $M \cap H \subseteq R \cap H$. We say that γ is acceptable *without disinformation* if $M \cap (L \cup B) = \emptyset$, *with disinformation*, otherwise.
- γ is *rejectable* if δQ is inconsistent.
- γ is *negotiable*, otherwise.

Example 6. For K_s and K_b from Examples 4 and 5:
$\langle\{high\_pr\}, \{product_A\}, \emptyset\rangle$ is acceptable without disinformation w.r.t. K_s, as $\delta(P_s \cup^r Goal(\{high\_pr\}), L_s, B_s)$ has a disinformation-free belief set $\{high\_pr, product_A\} \subseteq M$.
$\langle\{high\_pr\}, \{product_A, quality_A\}, \emptyset\rangle$ is acceptable with disinformation w.r.t. K_s as $\delta(P_s \cup^r Goal(\{high\_pr\}), L_s, B_s)$ has a belief set M containing $high\_pr$, $product_A$, and each belief set with this property contains $quality_A$.
$\langle\{low\_pr\}, \{product_B, maker_D, quality_B\}, \emptyset\rangle$ is a negotiable proposal w.r.t. K_s since $\delta(P_s \cup^r Goal(\{low\_pr\}), L_s, B_s)$ has a belief set containing its assumptions but requires at least one of the sets $\{student\}$, $\{age \geq 65\}$, or $\{cash\}$.
$\langle\{high\_pr\}, \emptyset, \{product_A, maker_C, quality_A\}\rangle$ is a rejectable proposal w.r.t. K_b because $\delta(P_b \cup^r Goal(\{high\_pr\}), L_b, B_b)$ has no belief set containing $high\_pr$. □

The receiver of a proposal generates a response. Let K_a be the n-KB of agent a and $\gamma_b = \langle G, S, R\rangle$ be a proposal by b w.r.t. its n-KB K_b. A *response* to γ_b by a is: *(i)* A proposal $\gamma_a = \langle G', S', R'\rangle$ w.r.t. K_a; **or** *(ii)* $\langle\top, \emptyset, \emptyset\rangle$, denoting *acceptance of the proposal* if γ_b is acceptable w.r.t. K_a; **or** *(iii)* $\langle\bot, \emptyset, \emptyset\rangle$, denoting *rejection of the proposal*. Intuitively, a negotiation is a series of responses between two agents, who, in alternation, take into consideration the other agent's response and put forward a new response; this can be either accept, reject, or a new proposal that may involve explanations of why the latest proposal (of the other agent) was not acceptable.

Formally, a *negotiation* between a and b, starting with a, is a possible infinite sequence of proposals $\omega_1, \ldots, \omega_i, \ldots$ where $\omega_i = \langle G_i, S_i, F_i\rangle$, ω_{2k+1} is a proposal w.r.t. K_a ($k \geq 0$), ω_{2k} is a proposal w.r.t. K_b ($k \geq 1$), and ω_{i+1} is a response to ω_i for every $i \geq 1$. A negotiation *ends* at i if $\omega_i = \langle\top, \emptyset, \emptyset\rangle$ or $\omega_i = \langle\bot, \emptyset, \emptyset\rangle$. When $G_i \neq G_{i+2}$, we say that a *goal change* has occurred for the agent who proposes ω_i.

Example 7. Consider the seller s and the buyer b agents (Examples 4 and 5).

$b_1 : \langle\{low\_pr\}, \{product_A, quality_A, maker_C\}, \emptyset\rangle$
$s_1 : \langle\{low\_pr\}, \{student\}, \{product_A, quality_A, maker_C\}\rangle$
$b_2 : \langle\{low\_pr\}, \{product_A, quality_A, maker_C\}, \{\neg student\}\rangle$
$s_2 : \langle\{low\_pr\}, \{cash\}, \{product_B, maker_D, quality_B\}\rangle$
$b_3 : \langle\{lowest\_pr\}, \{product_B, maker_D, quality_B\}, \{cash\}\rangle$
$s_3 : \langle\{lowest\_pr\}, \{cash, mail\_list\}, \{product_B, maker_D, quality_B\}\rangle$
$b_4 : \langle\top, \emptyset, \emptyset\rangle$.

The seller bullshits in s_1 and lies in s_2. The buyer lies in b_4. A goal change has occurred at b_3 (for the buyer) and s_3 (for the seller). □

By definition, a negotiation can be infinite. Nevertheless, if agents do not repeat their responses then a negotiation is finite. Given two agents, a negotiation tree encodes all possible negotiations among them. In order to achieve their goals, agents employ *negotiation strategies* to construct their responses, given their n-KBs and a proposal. Detailed discussions on these notions are given in [12]. We present here one strategy. Let $\gamma_b = \langle G, S, R\rangle$ be a proposal by b w.r.t. K_b. A *conscious response* to γ_b by a is

(*i*) A proposal $\gamma_a = \langle G', S', R'\rangle$ w.r.t. K_a with a support M such that $G \preceq G'$, $R \cap H \subseteq S'$, and $S^{\neg} \cap M \subseteq R'$ where M is the support of γ_a, if γ_b is not rejectable w.r.t. K_a; **or**

(*ii*) A proposal $\gamma_a = \langle G', S', R'\rangle$ w.r.t. K_a with a support M such that $G \not\preceq G'$ and $S^{\neg} \cap M \subseteq R'$ where M is a support of γ_a if γ_b is rejectable w.r.t. K_a; **or**

(*iii*) $\langle \top, \emptyset, \emptyset\rangle$, denoting *acceptance of the proposal*, if γ_b is acceptable w.r.t. K_a; **or**

(*iv*) $\langle \bot, \emptyset, \emptyset\rangle$, denoting *rejection of the proposal*.

If the proposal $\langle G, S, R\rangle$ is acceptable, the agent can accept it (case (*iii*)) or negotiate for better options (case (*i*)). If the proposal is negotiable, he can attempt to get a better option (case (*i*)). If the proposal is rejectable, the agent can negotiate for something that is not as good as the current goal (case (*ii*)). In any case, the agent can stop with a rejection (case (*iv*)). An agent generates a new proposal whose goal depends on the goal of the original one, whose assumptions cover the conditions in the original proposal ($R \cap H \subseteq S'$), and whose conditions identify all incorrect assumptions in the original one ($S^{\neg} \cap M \subseteq R'$). An agent who decides to consider preferable proposals, requires that the support for the new proposal is preferred to any support for accepting γ_b.

Example 8. The proposal $\gamma_1 = \langle\{low\_pr\}, \{cash\}, \{product_A, quality_A, maker_C\}\rangle$ ("[s]: I can sell you the $product_A$, made by $maker_C$, and has good quality for low_pr if you pay in cash") is acceptable w.r.t. K_b. The buyer can also respond with $\langle\{lowest\_pr\}, \{product_A, quality_A, maker_C\}, \{cash\}\rangle$ ("[b]: Can I get the $lowest\_pr$?").
The proposal $\gamma_2 = \langle\{low\_pr\}, \{product_A, maker_C\}, \emptyset\rangle$ ("[b]: Can I have $product_A$ from $maker_C$ for low_pr?") is not acceptable but negotiable w.r.t. K_s, since low_pr requires additional assumptions (e.g., $student$ or $age \geq 65$). The seller responds $\langle\{low\_pr\}, \{student\}, \{product_A, maker_C\}\rangle$ ("[s]: Yes, if you are a student.").

The proposal $\gamma_3 = \langle\{high\_pr\}, \emptyset, \{product_A, quality_A, maker_C\}\rangle$ ("[s]: I can sell $product_A$, made by $maker_C$, has good quality, at $high\_pr$") is rejectable w.r.t. K_b, as no rule in K_b derives $high\_pr$. The buyer can reject this proposal or weaken the goal: $\langle\{low\_pr\}, \{product_A, quality_A, maker_C\}, \emptyset\rangle$ ("[b]: Can I get it for low_pr?"). □

We define an agent a with the n-KB $K = \langle P, L, B, H, N^{\prec}\rangle$ to be an *adaptive* agent if it imports the information received during the negotiation into his/her n-KB and keeps this information for the next round of negotiation. Furthermore, an adaptive agent prefers to accept a proposal if a better outcome cannot be achieved. Formally, a is adaptive if for every negotiation $\omega_1, \ldots$, whenever a responds to a proposal $\omega_i = \langle G, S, R\rangle$,

- a responds with $\langle G', S', R'\rangle$, which is a conscious response to $\langle G, S, R\rangle$, and if $\langle G, S, R\rangle$ is acceptable w.r.t. K then G' is preferred to G or $\langle G', S', R'\rangle = \langle \top, \emptyset, \emptyset\rangle$.
- a changes his n-KB to $K' = \langle P \cup^r (R \cap H), L, B, H, N^{\prec}\rangle$ after his proposal response.

Negotiations among adaptive agents are guaranteed to terminate.

3 ASP-Prolog

The ASP-Prolog system [8] represents an extension of a modular Prolog system which enables the integration of Prolog-style reasoning with Answer Set Programming (ASP). The first implementation of ASP-Prolog dates back to 2004 [2], and we have recently embarked in a redesign and re-implementation of the system using more modern Prolog and ASP technology, as discussed in [8].

An ASP-Prolog program is composed of a hierarchy of modules, where each module can be declared to contain either Prolog code or ASP code. Each module provides an interface which enables to export predicate declarations and import declarations from other modules. In the current implementation, the root of the module hierarchy is expected to be a Prolog module, that can be interacted with using the traditional Prolog-style query-answering mechanism.

Each module is provided with the ability to access the intended models of other modules; in particular, the intended models of each module (i.e., the least Herbrand model of a Prolog module and the answer sets of an ASP module) are themselves automatically realized as individual modules, that can be queried. The interactions among modules can be realized using the following built-in constructs:

- m : `model`(X) succeeds if X is the name of a module representing one of the intended models of the module m—i.e., a module representing the least Herbrand model (resp. an answer set) of a Prolog (resp. ASP) module m.
- $m : p$ succeeds if the atom p is entailed in all the intended models of the module m; in particular, if m represents an answer set of another module, then this test will simply verify whether p is entailed by that particular answer set.
- The content of modules can be retrieved (using the predicate m : `clause`(R)) and modified by other modules. The predicates `assert` and `retract` can be applied to add or remove clauses from a module.

Let us assume we have one Prolog module $\mathtt{p}_1$ and two ASP modules $\mathtt{q}_1$ and $\mathtt{q}_2$:

$\mathbf{p}_1$	$\mathbf{q}_1$	$\mathbf{q}_2$
	`:- asp_module.`	`:- asp_module.`
`:- use_module(q1).`		`:- use_module(q1).`
`:- use_module(q2).`		
`p :- q1:r.`	`r :- not s.`	`a :- not s.`
`s :- q2:a.`	`h.`	`s :- not a.`
`t :- q2:model(X), X:a.`		`q :- q1:h.`
`v :- q1:assert(s), q1:r`		

The queries $\mathtt{p}_1$`:p`, $\mathtt{p}_1$`:t`, and $\mathtt{q}_1$`:q` are successful, while $\mathtt{p}_1$`:s` fails. On the other hand, the execution of $\mathtt{p}_1$`:v` would fail, adding the fact `s` to the module $\mathtt{q}_1$.

4 A Platform for Negotiation Systems

In this section, we describe an ASP-Prolog based platform for the development of negotiation agents which employ ALD in their negotiation. The organization of the platform is illustrated in Figure 1. At the lowest level, we develop an ASP-Prolog layer that

provides an implementation of abductive logic programming. This is used, in turn, to enable the representation of n-KBs and to handle the basic handling of negotiation proposals. At the top level, we have Prolog programs that can interact and coordinate the execution of agents performing negotiation. The layers of this architecture are discussed in the following subsections.

4.1 ALP Modules

Since the semantics of ALP and ALP with preferences is defined by answer sets of extended logic programs, it is natural to view ALPs as another type of modules that can be accessed and used in the same way as other modules in ASP-Prolog. To achieve this goal, we extend ASP-Prolog with the following predicates:

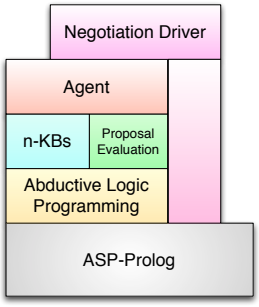

Fig. 1: Negotiation Architecture

• `use_alp(+Name, +`P^r`, +`P^a`, +[Options])`: this predicate has the same effect as the predicate `use_module(`P`)` for a Prolog or an ASP-module. It compiles the program (P^r, P^a) with the corresponding options specified in the list $Options$ into a new ASP module, $Name$. The compilation process is illustrated in Algorithm 1. Similarly to the case of ASP modules, the ALP belief sets of (P^r, P^a) are created as (sub)modules of the module produced by the algorithm. Accessing a belief set of the program or its content is done in the same way as for ASP modules. Some shorthands are also provided, e.g., `use_alp(+`P^r`, +`P^a`)` where the module name is specified as P^r.

• `most_preferred(+Name, ?M)`: this predicate allows users to query or check for a most preferred belief set of the ALP program referred to by $Name$.

• `more_preferred(+Name, ?`M_1`, ?`M_2`)`: this predicate allows users to compare two belief sets w.r.t. the $prefer$ relation.

Algorithm 1 : ALP Compile

Require: An abductive logic program $P = (P^r, P^a)$; a name $Name$
Ensure: An ASP module $Name$.
Compile P^a program into a regular answer set program P_1^a;
- introduce the atom $ok(n)$ in the body of rule n;
- generate choice rules of the form $0\{ok(n)\}1$. for each rule n in P^a.

Generate an ASP module containing the rules $P' = P^r \cup P_1^a$.

Example 9. Consider the ALP $P = (P^r, P^a)$ with preference:

$$P^r = \{\, prefer(n_1, n_2), \quad s \leftarrow, \quad \leftarrow \; not\, p,\; not\, q \,\},$$
$$P^a = \{\, n_1 : \; p \leftarrow \; not\, r, \quad n_2 : \; q \leftarrow \; not\, r \,\}.$$

It is easy to see that P has three belief sets $A_1 = \{p, s, prefer(n_1, n_2)\}$, $A_2 = \{q, s, prefer(n_1, n_2)\}$, and $A_3 = \{p, q, s, prefer(n_1, n_2)\}$ which are belief sets of $P^r \cup \{n_1\}$, $P^r \cup \{n_2\}$, $P^r \cup \{n_1, n_2\}$, respectively.[3] Suppose that P^r and P^a are

[3] Rule labels refer to the rules in P^a.

stored in the files `pr.lp` and `pa.lp` respectively. The command `?- use_alp(p, pr, pa)` compiles $P = (P^r, P^a)$ into the module `p`. It also creates three submodules `p1`, `p2`, and `p3` which correspond to A_1, A_2, and A_3, respectively. The next table displays some queries to p and the corresponding answers.

Query	Answer
`?- p:model(X).`	`X = p1; X = p2; X = p3`
`?- findall(E,(current_predicate(p:P/A), functor(E,P,A),p:E),L).`	`L = [s,prefer(n1,n2)]`
`?- p:model(X), X:p.`	`X = p1; X = p3`
`?- most_preferred(p, X).`	`X = p1`

The first query asks for belief sets of `p`. The second query asks for all atoms that are true in `p`, i.e., those belonging to all belief sets of P. The third query identifies models in which p (the atom) is true. The last query asks for a most preferred belief set of `p`. □

4.2 Negotiation Agent

The n-KB layer of Fig. 1 allows the high-level description of the n-KB of a negotiation agent and its mapping to the corresponding ALP module. A negotiation agent is described by a n-KB $K = \langle P, L, B, H, N^{\prec} \rangle$, where $P = (P^r, P^a)$, (L, B) is a disinformation w.r.t. P, H is set of assumptions, and $N^{\prec}$ is set of negotiated conditions. For convenience, we introduce a simple specification language for agents as shown in Table 1 (left). The complete description of a n-KB is provided in a file, with different sections corresponding to the different components of the n-KB.

Syntax	Example: Seller agent
`declare_pr:`	`declare_pr:`
The program P^r	The program P_s^r (Example 2)
`declare_pa:`	`declare_pa:`
The program P^a	The program P_s^a (Example 2)
`declare_lying:`	`declare_lying:`
Literals in L	$quality_B$.
`declare_bs:`	`declare_bs:`
Literals in B	$quality_A$.
`declare_hypotheses:`	`declare_hypotheses:`
Literals in H	$age \geq 65$. $student$.
	$cash$. $mail\_list$
`declare_goal:`	`declare_goal:`
Literals in N and	$high\_pr$.
the preference order $\prec$	low_pr. $lowest\_pr$.
	$prefer(low\_pr, lowest\_pr)$.
	$prefer(high\_pr, low\_pr)$.
	$prefer(high\_pr, lowest\_pr)$.

Table 1. Agent Specification

An agent is compiled into ALP (and, in turn, into ASP as discussed in the previous section) by the command `load_agent(+Agent, +File, +[Options])`, where

$Agent$ is the name of the agent, $File$ is file encoding the agent specification, and the list of options $[Options]$ for use with the program $P = (P^r, P^a)$. The command will compile the n-KB into a module, named $Agent$, whose submodules correspond to belief sets of the ALD-program $\delta(P, L, B)$. This allows the users to access belief sets, compute most preferred belief sets, as well as compare belief sets of $\delta(P, L, B)$. The overall structure of the compiler is sketched in Algorithm 2.

Example 10. Consider the n-KB K_s in Ex. 4. Suppose that the n-KB is stored in `seller.lp`. Its description is shown on the right in Table 1. The query `?- load_agent(s,seller)` will compile the n-KB into an ALP module named `s`, denoting the agent s. The module has submodules corresponding to the belief sets of $\delta(P_s, L_s, B_s)$ (Ex. 3) whose content can be accessed using the ASP-Prolog module interface discussed earlier. □

Algorithm 2 : n-KB Compilation

Require: An Agent name
Require: A n-KB $K = \langle P, L, B, H, N^{\prec} \rangle$ where $P = (P^r, P^a)$
Ensure: An ALP module representing $\delta(P, L, B)$.
Compute $I = \{\, r \mid r \in P^r \text{ and } head(r) \cap L^{\neg} \neq \emptyset \}$
$P_1^a = P^a \cup I \cup L \cup B$
Assign labels to the rules in P_1^a
Compute

$$\Phi = \{\, prefer(n_i, n_j) \mid n_i \in I \text{ and } n_j \in P^a \cup H \} \cup \{\, prefer(n_j, n_t) \mid n_j \in P^a \cup H \cup I \text{ and } n_t \in (L \cup B) \}$$

$P_1^r = P^r \setminus I \cup \Phi$
Compile the program $P_1 = (P_1^r, P_1^a)$ using the `use_alp` predicate.

4.3 Computing and Evaluating Proposals

The proposal evaluation layer (Fig. 1) provides a collection of predicates to generate general proposals and to evaluate proposals for acceptability, rejectability, and negotiability. Let us discuss the key predicates; all examples refer to s and b as the seller and buyer whose n-KBs are specified in Examples 4 and 5. We assume that the n-KBs are already compiled into modules s and b, respectively, using the `load_agent` command.

- `proposal(+Agent,[?G,?S,?R])`: this predicate succeeds if $\langle G, S, R \rangle$ is a proposal for the agent $Agent$. Note that this predicate can be used to generate as well as test a proposal. For example, the query `?- proposal(s,[G,S,R])` generates an arbitrary proposal that the seller agent `s` can create, given her n-KB (Ex. 10); one possible answer is `G = [`low_pr`], S = [], R = [`$product_A$`]`.
- `proposal(+Agent,?M,[?G,?S,?R])`: $\langle G, S, R \rangle$ is a proposal for the agent $Agent$ with supporting belief set M. For example, assume that the module s has a submodule $s1$ containing low_pr, $student$, and $product_B$; the query

  `?- proposal(s,s1,[G,S,R]).`

  asks for a possible proposal supported by the belief set described by module $s1$; it returns the answer `G = [`low_pr`], S = [`$student$`], R = [`$product_B$`]`. A most preferred proposal for s can be obtained by the query:

  `?- most_preferred(s, M), proposal(s,M,[G,S,R]).`

- `acceptable(+Agent,[+G,+S,+R])`: $\langle G, S, R\rangle$ is an acceptable proposal for the agent $Agent$. For example, the query
  `?- acceptable(s, [[`$high\_pr$`],[`$product_A$`], []]).`
  asks if $\langle high\_pr, \{product_A\}, \emptyset\rangle$ is acceptable for agent s; this query succeeds.
- `negotiable(+Agent,[+G,+S,+R])`: $\langle G, S, R\rangle$ is a negotiable proposal for the agent $Agent$. The query
  `?- negotiable(s, [[`$high\_pr$`],[`$product_A$`, `$quality_A$`], []]).`
  checks whether the proposal $\langle high\_pr, \{product_A, quality_A\}, \emptyset\rangle$ is negotiable for the agent s, and it succeeds.
- `rejectable(+Agent,[+G,+S,+R])`: $\langle G, S, R\rangle$ is a rejectable proposal for the agent $Agent$. The query
  `?- rejectable(b, [[`$high\_pr$`],[`$product_A$`, `$quality_A$`, `$maker_C$`], []]).`
  checks if $\langle high\_pr, \{product_A, quality_A, maker_C\}, \emptyset\rangle$ is rejectable for agent b; it succeeds, since this proposal is neither acceptable nor negotiable for b.

4.4 Agents: Responses, Strategies, and Negotiations

The design of a negotiation agent requires, in addition to its knowledge, the ability of applying strategies for the generation of proposals (e.g., act as an adaptive agent) and for the selection of responses (e.g., develop conscious responses). To this end, our negotiation architecture provides predicates for computing conscious responses and updates of agents. We next describe these predicates. Other strategies can be realized (as relatively simple Prolog modules) and their development is part of the future work.

- `response(+Agent,[?G2,?S2,?R2],[+G1,+S1,+R1])`: $\langle G2, S2, R2\rangle$ is a conscious response (by $Agent$) to the proposal $\langle G1, S1, R1\rangle$. Note that the predicate can be used to check responses (for consciousness) or to generate conscious responses. For example, the answer to the query
  `?- response(b,[G,S,R],[[low_pr],[cash],[product`$_A$`, quality`$_A$`, maker`$_C$`]])`
  which asks for a conscious response by b to the proposal
  $\langle\ low\_pr, \{cash\}, \{product_A, quality_A, maker_C\}\ \rangle$ is
  `G=[`low_pr`], S=[`$product_A$`, `$quality_A$`, `$maker_C$`], R = [`$cash$`]`
- `update_kb(+Agent, +Type, +Flag,+Value)`: where $Type$ is one of {`pr`, `pa`, `lying`, `bs`, `hypotheses`, `goal`} and $Flag$ is either `true` or `false`. $Type$ specifies the part of the agent's KB, that needs to be updated. $Flag$ is true (false) indicating whether the content specified in $Value$ has to be added or removed ($Value$ could be either a constant or a list of constants). This predicate updates the `declare_type` part of the agent. For instance, the query
  `?- update_kb(s, pr, true, `$student$`)`
  updates the program P^r of the seller n-KB with the fact $student$, i.e., adding the fact $student$ to the program P^r. Intuitively, this query should be executed after the seller learns that the buyer is a student. On the other hand, the query
  `?- update_kb(s, hypotheses, false, `$student$`)`
  updates the set of hypothesis H_s of the seller by removing $student$ from H_s, e.g., after she learns that the buyer is indeed a student.

4.5 A Program for Automated Negotiation

The top-level of the proposed framework is a *coordinator agent*; the role of the coordinator is to control the exchanges between the negotiating agents, enforcing the proper ordering in the exchanges and acting as a communication channel among the negotiating agents. The coordinator agent is a relatively simple Prolog module. The entry point of the coordinating agent is the predicate `main`, which receives in input the names of the two negotiating agents and the files containing their n-KBs:

```
main(Name1, Agent1, Name2, Agent2, [G,S,R]) :-
   load_agent(Name1, Agent1), load_agent(Name2, Agent2),
   negotiation(Name1, Name2, [G,S,R]).
```

The negotiation starts with an agent identifying a most preferred proposal and proposing it to the other agent. The proposal identified by $[G, S, R]$ is the final outcome of a negotiation. The actual negotiation process is an iterative process (implemented by the predicate `round`), which alternates generation of responses between the two agents. In each round, the receiving agent computes a counter proposal and updates her n-KB. The coordinator agent updates the history of the negotiation to ensure that agents do not repeat their answers. Multiple traces can be obtained by asking for different answers.

```
1: negotiation(A, B, [G1,S1,R1]):-
2:   most_preferred(A,M), proposal(A,M,[G,S,R]), print_prop(A,G,S,R),
3:   round(A,B,[G,S,R],[(A,[],[G,S,R])], [G1,S1,R1]).
4: round(A, B, [G,S,R],History,[G,S,R]):-  print_prop(A,G,S,R),
5:    response(B,[G1,S1,R1],[G,S,R]),
6:    check_repeated((B,[G,S,R],[G1,S1,R1]),History),
7:    ([G1,S1,R1]==[true,[],[]] -> write('Accepted') ;
8:    ([G1,S1,R1]==[false,[],[]]-> write('Rejected') ;
9:    append([(B,[G,S,R],[G1,S1,R1])],History,History1),
10:   agent(B, Hyp, _, _, _), intersection(Hyp, R, SH1),
11:   negated(Hyp, HypN), intersection(HypN, R, SH2),
12:   union(SH1, SH2, SH), update_KB(B, pr, true, SH),
13:   round(B, A, [G1,S1,R1],History1,[G1,S1,R1])        )).
```

where `agent(·)` provides the components of the agent and `negated`(S, S') is true for $S' = S^{\neg}$. The subgoals in lines 7–12 are used to update the agent (see definition of adaptive agent), while line 6 avoids repetitions of proposals.

Example 11. The following is an example of some negotiations when running the query

```
?- main(s, seller, b, buyer, [G,S,R]).
```

```
Agent s:  proposal {[high_pr],[],[]}
G=[high_pr], S=[], R=[]
Rejected yes ? ;
Agent b: proposal {[low_pr],[qualityB,makerC,productB],[]}
Agent s: proposal {[low_pr],[cash],[-(qualityB)]}
Agent b: proposal {[low_pr],[qualityA,makerC,productA],[]}
Agent s: proposal {[low_pr],[cash],[]}
Agent b: proposal {[low_pr],[qualityA,makerC,productA],[cash]}
Accepted yes ?
```

The seller starts with $\langle\{high\_pr\}, \emptyset, \emptyset\rangle$, which the buyer rejects. The buyer proposes $\langle\{low\_pr\}, \{quality_B, maker_C, product_B\}, \emptyset\rangle$. This does not work for the seller, as he does not want to lie about $quality_B$. The buyer proposes the alternative $\langle\{low\_pr\}$, $\{quality_A, maker_C, product_A\}, \emptyset\rangle$ for which the seller wants cash. The buyer agrees and the seller agrees to sell the product, with BS about the quality of product A. □

5 Conclusion

In this paper, we introduced the design of a logic programming platform to implement negotiating agents. The model of negotiation supported by the proposed platform enables the representation of agents with incomplete knowledge and capable of choosing dishonest answers in building their offers and counter-offers. The architecture has been entirely developed using the ASP-Prolog system, taking advantage of the ability of combining Prolog and ASP modules. We believe this architecture is quite unique in the level of flexibility provided and in its ability to support easy extensions to capture, e.g., different agent strategies and behaviors.

We are currently extending the architecture to provide several built-in agent strategies (e.g., to represent agents with different levels of dishonesty), allow actual concurrency in the agents' interactions (e.g., through a Linda-style blackboard), and implementing real-world scenarios.

References

1. W. Chen, M. Zhang, and N. Foo. Repeated negotiation of logic programs. In *Proc. 7th Workshop on Nonmonotonic Reasoning, Action and Change*. , 2006.
2. O. El-Khatib, E. Pontelli, and T. Son. ASP-Prolog: A System for Reasoning about Answer Set Programs in Prolog. In *PADL*, pages 148–162. Springer, 2004.
3. S. S. Fatima, M. Wooldridge, and N. R. Jennings. Bargaining with incomplete information. *Ann. Math. Artif. Intell.*, 44(3):207–232, 2005.
4. H. G. Frankfurt. *On Bullshit*. Princeton Univ. Press, 2005.
5. A. C. Kakas et al. Agent planning, negotiation and control of operation. *ECAI*, 2004.
6. S. Kraus. Negotiation and cooperation in multi-agent environments. *AIJ*, 94(1-2), 1997.
7. J. E. Mahon. Two definitions of lying. *J. Applied Philosophy*, 22(2):211–230, 2008.
8. E. Pontelli, T. C. Son, and N.-H. Nguyen. Combining Answer Set Programming and Prolog: The ASP-Prolog System, To appear in LNAI 6565, 2011.
9. F. Sadri, F. Toni, and P. Torroni. An abductive logic programming architecture for negotiating agents. In *JELIA*, pages 419–431. Springer-Verlag, 2002.
10. C. Sakama and K. Inoue. Prioritized Logic Programming and its Application to Commonsense Reasoning. *Artificial Intelligence*, 123(1-2):185–222, 2000.
11. C. Sakama and K. Inoue. Negotiation by abduction and relaxation. In *AAMAS*, pages 1018–1025. ACM Press, 2007.
12. C. Sakama, T. C. Son, and E. Pontelli. A logical formulation for negotiation among dishonest agents. `www.cs.nmsu.edu/~tson/papers/neg2010.pdf`, 2010.
13. T. C. Son and C. Sakama. Negotiation using logic programming with consistency restoring rules. In *IJCAI*, pages 930–935, 2009.
14. M. Wooldridge and S. Parsons. Languages for negotiation. In *ECAI*, 2000.
15. G. Zlotkin and J. S. Rosenschein. Incomplete information and deception in multi-agent negotiation. In *IJCAI*, pages 225–231, 1991.