DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

P.O. Box: 22012 E-46071 Valencia (SPAIN)

Technical Report

Ref. No.:
Pages: 28
Title: Case-based Argumentation Framework. Dialogue Protocol
Author(s): Stella Heras and Vicente Botti and Vicente Julián
Date: June 24, 2011
Keywords: Argumentation, Case-Based Reasoning, Multi-Agent Systems

Vº Bº
Vicente Botti Stella Heras

1 Introduction

Large scale computer systems are now viewed in terms of the entities that participate in them, offering and consuming services [16]. Commonly, these systems are implemented as open Multi-Agent Systems (MAS) which software agents are able to interact to solve complex tasks. Agents are autonomous entities that can have their own knowledge resources, can play different roles and can have different objectives and preferences over values that they want to promote with their actions (e.g. an interested agent could want to promote its own *wealth* in spite of the *fairness* in a negotiation to allocate any scarce resource). In addition, they can be linked by different types of dependency relations in what we call an *agent society*[11]. On top of the simpler ability to interact, open MAS must include mechanisms for their agents to reach *agreements* [34] by also taking into account their social context. This context determines the way in which agents can reach these agreements. For instance, an agent representing a worker in a company could be forced to violate its preferences about the outcome of an agreement process due to a power dependency relation that commits it to accept the decisions of a superior.

Argumentation provides MAS with a framework that assures a rational communication, which allows agents to reach agreements when conflicts of opinion arise. The dialogue typology of Walton and Krabbe [38] has been adopted in MAS to classify the different types of dialogues between agents depending on the objective of the interaction. Other concept of the argumentation theory, the theory of *dialogue games* [10, 17] has also been applied to structure the dialogue between agents with different points of view. Formal dialogue games are interactions between several players (agents in our case) where each player moves by making utterances in accordance to a defined set of rules [23]. Recently, a wide range of approaches that formalise interaction protocols by using different dialogue games have been published [23]. Some examples of dialogue game protocols about specific types of dialogues are: inquiry [13], persuasion [2], negotiation [32] and deliberation [21].

In [11], we have proposed a case-based argumentation framework that allows agents of an agent society to reach agreements by taking into account their social context. In this report we present the communication protocol that agents of our argumentation framework use to interact when they engage in argumentation dialogues. Considerable research has been performed on the design of artificial agent communication languages, such as DARPA's *Knowledge Query and Manipulation Language (KQML)*[a] and the IEEE Foundation for Intelligent Physical Agents *Agent Communications Language (FIPA ACL)*[b]. These languages provide agents with a high flexibility of expression. However, in a dialogue agents can have too many choices of what to utter in each step of the conversation. Therefore, this flexibility can also be an important downside if it gives rise to a state-space explosion and leads agents to engage in unending dialogues [26, Chapter 13]. A possible solution for this problem consists in limiting the allowed set of utterances for each step of the dialogue by defining the agents communication protocol with a dialogue game.

The structure of this report is as follows: Section 2 briefly introduces our case-based argumentation framework for agent societies; Section 3 introduces the notation used along the report; Section 4 presents the syntax of the protocol, as the set of defined locutions, their combinatorial properties and the rules that govern the dialogue; after that, Section 5 provides the *axiomatic* semantics and the *operational* semantics of the locutions. The former defines the pre-conditions that should be met to put forward each locution (or set of locutions) and the post-conditions that apply before their utterance. The latter views each locution as a transition in an abstract state-machine that represents the possible stages that can be reached during the dialogue; finally, section 7 summarises the contents of this report.

2 Case-based Argumentation Framework

In [11] a case-based argumentation framework for agent societies has been proposed. Many argumentation frameworks require to create an explicit rule-based model of the domain [28]. In open MAS, the domain is highly dynamic and the set of rules that model it is difficult to specify in advance. However, tracking

[a] `www.cs.umbc.edu/research/kqml/`

[b] `www.fipa.org/repository/aclspecs.html`

the arguments that agents put forward in argumentation processes could be relatively simple. Therefore, these arguments can be stored as cases codified in a common language (e.g. an ontological language, as will be proposed in next section) that different agents are able to understand. Other important problem with rule-based systems arises when the knowledge-base must be updated (e.g. adding new knowledge that can invalidate the validity of a rule). Updates imply to check the knowledge-base for conflicting or redundant rules. Case-based systems are easier to maintain than rule-based systems since, in the worst case, the addition of new cases can give rise to updates in some previous cases, but does affect the correct operation of the system, although it can have an impact in its performance.

In this section, we briefly introduce the elements of our framework. Concretely, we propose three types of knowledge resources that the agents can use to generate, select and evaluate arguments:

A case-base with domain-cases that represent previous problems and their solutions. Agents can use this knowledge resource to generate their positions in a dialogue and arguments to support them. Also, the acquisition of new domain-cases increases the knowledge of agents about the domain under discussion.

A case-base with argument-cases that store previous argumentation experiences and their final outcome. Argument-cases have three main objectives: they can be used by agents 1) to generate new arguments; 2) to strategically select the best position to put forward in view of past argumentation experiences; and 3) to store the new argumentation knowledge gained in each agreement process, improving the agents' argumentation skills.

A database of Argumentation schemes with a set of argumentation schemes [39], which represent stereotyped patterns of common reasoning in the application domain where the framework is implemented. An argumentation scheme consists of a set of premises and a conclusion that is presumed to follow from them. Also, each argumentation scheme has associated a set of *critical questions* that represent potential attacks to the conclusion supported by the scheme. The concrete argumentation schemes to be used depend on the application domain.

The structure of domain-cases and the concrete set of argumentation schemes that an argumentation system that implements our framework has depends on the application domain. Argument-cases are the main structure that we use to computationally represent arguments in agent societies. In addition, their structure is generic and domain-independent. Argument-cases have the three possible types of components that usual cases of CBR systems have: the description of the state of the world when the case was stored (*Problem*); the solution of the case (*Conclusion*); and the explanation of the process that gave rise to this conclusion (*Justification*).

The problem description has a *domain context* that consists of the *premises* that characterise the argument. In addition, if we want to store an argument and use it to generate a persuasive argument in the future, the features that characterise its *social context* must also be kept. The social context of the argument-case includes information about the *proponent* and the *opponent* of the argument and about their *group*. Moreover, we also store the preferences (*ValPref*) of each agent or group over the set of *values* pre-defined in the system. Finally, the *dependency relation* between the proponent's and the opponent's roles is also stored. In our framework, we consider three types of dependency relations as defined in [9]: *Power*, when an agent has to accept a request from other agent because of some pre-defined domination relationship between them; *Authorisation*, when an agent has signed a contract with other agent to provide it with a service and hence, the contractor agent is able to impose its authority over the contracted agent and *Charity*, when an agent is willing to answer a request from other agent without being obliged to do so.

In the solution part, the *conclusion* of the case, the *value* promoted, and the *acceptability status* of the argument at the end of the dialogue are stored. The last feature shows if the argument was deemed *acceptable*, *unacceptable* or *undecided* in view of the other arguments that were put forward in the agreement process. In addition, the conclusion part includes information about the possible *attacks* that the argument received during the process. These attacks could represent the justification for an argument to be deemed unacceptable or else reinforce the persuasive power of an argument that, despite

being attacked, was finally accepted. Concretely, arguments in our framework can be attacked by putting forward *distinguishing premises* or *counter-examples* to them, as proposed in [5], and also by questioning the validity of the conclusion drawn from an argumentation scheme by instantiating a *critical question*.

A distinguishing premise is a premise that does not appear in the description of the problem to solve and has different values for two cases or a premise that appears in the problem description and does not appear in one of the cases. A counter-example for a case is a previous domain-case or an argument-case that was deemed acceptable, where the problem description of the counter-example matches the current problem to solve and also subsumes the problem description of the case, but proposing a different solution. A critical question is a question associated to an argumentation scheme that represents a potential way in which the conclusion drawn from the scheme can be attacked. Critical questions can be classified as *presumptions* that the proponent of the argumentation scheme has made or *exceptions* to the general inference rule that the scheme represents [29]. In the former case, the proponent has the burden of proof if the critical question is asked, whereas in the later the burden of proof falls on the opponent that has questioned the conclusion of the scheme. Therefore, if the opponent asks a critical question, the argument that supports this argumentation scheme remains temporally rebutted until the question is conveniently answered. This characteristic of argumentation schemes makes them very suitable to devise ways of attack the conclusions drawn from other agents.

Finally, the justification part of an argument-case stores the information about the knowledge resources that were used to generate the argument represented by the argument-case (the set of domain-cases and argument-cases). In addition, the justification of each argument-case has a *dialogue-graph* (or several) associated, which represents the dialogue where the argument was proposed. In this way, the sequence of arguments that were put forward in a dialogue is represented, storing the complete conversation as a directed graph that links argument-cases. This graph can be used later to improve the efficiency in an argumentation dialogue in view of a similar dialogue that was held in the past.

As pointed out before, in our framework agents can generate arguments from previous cases (domain-cases and argument-cases) and from argumentation schemes. However, note that the fact that a proponent agent uses one or several knowledge resources to generate an argument does not imply that it has to show all this information to its opponent. The argument-cases of the agents' argumentation systems and the structure of the actual arguments that are interchanged between agents is not the same. Thus, arguments that agents interchange are defined as tuples of the form:

Definition 2.1 (Argument)**.** *$Arg = \{\phi, v, < S >\}$, where ϕ is the conclusion of the argument, v is the value that the agent wants to promote with it and $< S >$ is a set of elements that support the argument (support set).*

This support set can consist of different elements, depending on the argument purpose. On one hand, if the argument provides a potential solution for a problem, the support set is the set of features (*premises*) that represent the context of the domain where the argument has been proposed (those premises that match the problem to solve and other extra premises that do not appear in the description of this problem but that have been also considered to draw the conclusion of the argument) and optionally, any knowledge resource used by the proponent to generate the argument (*domain-cases*, *argument-cases* or *argumentation schemes*). On the other hand, if the argument attacks the argument of an opponent, the support set can also include any of the allowed attacks in our framework (*critical questions*, *distinguishing premises* or *counter-examples*).

3 Preliminaries

Along this report we follow the standard that views utterances as composed by two layers: an internal layer that represents the *topics* of the dialogue and an external layer that consists of the *locutions* or performatives that define the allowed speech acts. On one hand, we assume that the topics of the inner layer can be represented with well-formed formulae of the Description Logic (DL) $\mathcal{SHOIN}(D)$ [12], which forms the basis of the Web Ontology Language OWL-DL. In our argumentation framework, we have designed an ontology called *ArgCBROnto* to define the representation language of arguments

and argumentation concepts[c]. Ontologies provide a common vocabulary to understand the structure of information among different software agents. In addition, ontologies allow to make assumptions about the domain explicit, which facilitates to change these assumptions as new knowledge about the domain is acquired. The high dynamism of the domains where open MAS operate gives rise to many changes in the domain knowledge that agents have available. Therefore, they must be able to efficiently handle the consequences of these changes. On the other hand, we use the standard operators and axioms of *modal logics* of knowledge and believe [33, Chapter 13] to define the semantics of locutions.

In DLs, the important notions of the domain are described by *concept descriptions*, which are expressions that are built from atomic *concepts* (unary predicates) and atomic *roles* (binary predicates relating concepts) using the concept and role constructors provided by the particular DL. The semantics of DLs is given in terms of *interpretations* [4]:

Definition 3.1 (Interpretation:)**.** *An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty set $\Delta^{\mathcal{I}}$, called the domain of $\mathcal{I}$, and a function $\cdot^{\mathcal{I}}$ that maps every concept to a subset of $\Delta^{\mathcal{I}}$, and every role name to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ such that, for all concepts C, D and all role names R,*
$\top^{\mathcal{I}} = \{\Delta^{\mathcal{I}} |$ for all $C^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, then $\top^{\mathcal{I}} = C^{\mathcal{I}} \cup \neg C^{\mathcal{I}}\}$,
$\bot^{\mathcal{I}} = \{\emptyset |$ for all $C \in \Delta^{\mathcal{I}}$, then $\bot^{\mathcal{I}} = C^{\mathcal{I}} \cap \neg C^{\mathcal{I}}\}$,
$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$, $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$, $\neg C^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$,
$(\exists R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} |$ there is some $y \in \Delta^{\mathcal{I}}$ with $\langle x, y\rangle \in R^{\mathcal{I}}$ and $y \in C^{\mathcal{I}}\}$,
$(\forall R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} |$ for all $y \in \Delta^{\mathcal{I}}$, if $\langle x, y\rangle \in R^{\mathcal{I}}$, then $y \in C^{\mathcal{I}}\}$. We say that $C^{\mathcal{I}}$ ($R^{\mathcal{I}}$) is the extension of the concept C (role name R) in the interpretation $\mathcal{I}$. If $x \in C^{\mathcal{I}}$, then we say that x is an instance of C in $\mathcal{I}$.

Table 1 shows the syntax and semantics of the constructors of $\mathcal{SHOIN}(D)$, using Roman upper-case letters to represent *concepts*, *datatypes* and *roles* and Roman lower-case letters to represent *individuals* and *data values*.

As any description logic, $\mathcal{SHOIN}(D)$ uses concept descriptions to build statements in a DL knowledge base $\mathcal{K}$ (the analogue of an ontology in OWL-DL), which typically comes in two parts: a *terminological (TBox)* and an *assertional (ABox)*. In the TBox, we can describe the relevant notions of an application domain by stating properties of concepts and roles and relationships between them. For instance, the notions of agents and arguments are defined in our argumentation framework with the concepts of *Agent* and *Argument* of the ArgCBROnto and the following axioms:

$SocialEntity \sqsubseteq Thing$

$Agent \sqsubseteq SocialEntity$

$Argument \sqsubseteq Thing$

The *properties* of an argument are defined with the roles $hasConclusion$, $promotesValue$ and $hasSupportSet$ and the following axioms and value restrictions:

$Argument \sqsubseteq \forall hasConclusion.Conclusion$

$Argument \sqsubseteq \forall promotesValue.Value$

$Argument \sqsubseteq \forall hasSupportSet.SupportSet$

which say that arguments can have three properties that relate them with objects of the class $Conclusion$, $Value$ and $SupportSet$. Correspondingly, the ABox represents the concrete data of the database $\mathcal{K}$, with the individuals of concepts (instances) and their properties. For instance, the ABox of the ArgCBROnto ontology can include an argument arg that promotes a value *solidarity*:

$Argument(arg)$

[c]The complete specification of the ontology is available at `users.dsic.upv.es/~vinglada/docs`.

Constructor Name	**Syntax**	**Semantics**
atomic concept **A**	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
datatypes **D**	D	$D^D \subseteq \Delta_D^{\mathcal{I}}$
abstrac role $\mathbf{R}_A$	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
datatype role $\mathbf{R}_D$	U	$U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta_D^{\mathcal{I}}$
individuals **I**	o	$o^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
data values	v	$v^{\mathcal{I}} = v^D$
inverse role	R^-	$(R^-)^{\mathcal{I}} = (R^{\mathcal{I}})^-$
conjunction	$C_1 \sqcap C_2$	$(C_1 \sqcap C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
disjunction	$C_1 \sqcup C_2$	$(C_1 \sqcup C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
negation	$\neg C_1$	$(\neg C_1)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C_1^{\mathcal{I}}$
oneOf	$\{o_1, ...\}$	$\{o_1, ...\}^{\mathcal{I}} = \{o_1^{\mathcal{I}}, ...\}$
exists restriction	$\exists R.C$	$(\exists R.C)^{\mathcal{I}} = \{x \mid \exists y.\, \langle x, y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$
value restriction	$\forall R.C$	$(\forall R.C)^{\mathcal{I}} = \{x \mid \forall y.\, \langle x, y\rangle \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
atleast restriction	$\geq nR$	$(\geq nR)^{\mathcal{I}} = \{x \mid \#(\{y.\, \langle x, y\rangle \in R^{\mathcal{I}}\}) \geq n\}$
atmost restriction	$\leq nR$	$(\leq nR)^{\mathcal{I}} = \{x \mid \#(\{y.\, \langle x, y\rangle \in R^{\mathcal{I}}\}) \leq n\}$
datatype exists	$\exists U.D$	$(\exists U.D)^{\mathcal{I}} = \{x \mid \exists y.\, \langle x, y\rangle \in U^{\mathcal{I}} \text{ and } y \in D^D\}$
datatype value	$\forall U.D$	$(\forall U.D)^{\mathcal{I}} = \{x \mid \forall y.\, \langle x, y\rangle \in U^{\mathcal{I}} \rightarrow y \in D^D\}$
datatype atleast	$\geq nU$	$(\geq nU)^{\mathcal{I}} = \{x \mid \#(\{y.\, \langle x, y\rangle \in U^{\mathcal{I}}\}) \geq n\}$
datatype atmost	$\leq nU$	$(\leq nU)^{\mathcal{I}} = \{x \mid \#(\{y.\, \langle x, y\rangle \in U^{\mathcal{I}}\}) \leq n\}$
datatype oneOf	$\{v_1, ...\}$	$\{v_1, ...\}^{\mathcal{I}} = \{v_1^{\mathcal{I}}, ...\}$
Axiom Name	**Syntax**	**Semantics**
concept inclusion	$C_1 \sqsubseteq C_2$	$C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
object role inclusion	$R_1 \sqsubseteq R_2$	$R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$
object role transitivity	$Trans(R)$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^+$
datatype role inclusion	$U_1 \sqsubseteq U_2$	$U_1^{\mathcal{I}} \subseteq U_2^{\mathcal{I}}$
individual inclusion[e]	$a : C$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
individual equality	$a = b$	$a^{\mathcal{I}} = b^{\mathcal{I}}$
individual inequality	$a \neq b$	$a^{\mathcal{I}} \neq b^{\mathcal{I}}$
concept existence	$\exists C$	$\#(C^{\mathcal{I}}) \geq 1$

Table 1: Syntax and Semantics of $\mathcal{SHOIN}(D)$ [12].

$promotesValue(arg, solidarity)$

On the other hand, the syntax of the external layer of utterances (locutions) is as proposed in [25]:

$locution(a_s, \phi)$ or $locution(a_s, a_r, \phi)$

where $Agent(a_s)$ (the sender) and $Agent(a_r)$ (the receiver) are individuals of the *Agent* concept and ϕ is the content of the utterance. The former locution is addressed to all participants in the dialogue, whereas the latter is specifically sent to $Agent(a_r)$. We denote the set of well-formed formulae in $\mathcal{SHOIN}(D)$ as $\mathcal{D}$. Then, $\phi \in \mathcal{D}$ can represent statements about problems to solve, evidences about the world or different types of arguments. Also, we denote the set of individuals members of the concept *Argument* as $\mathcal{A}$ such that $\forall arg \in \mathcal{A}, Argument(arg)$. Therefore, Φ is said to be an argument in support of ϕ if $\Phi \in \mathcal{A}/\Phi \vdash^+ \phi$. Correspondingly, Φ is said to be an argument against ϕ if $\Phi \in \mathcal{A}/\Phi \vdash^- \phi$.

Also, agents make *propositional commitments* (also known as dialogical commitments) with each locution that they put forward. Therefore, if an agent asserts some locution and other agent challenges it, the first has the commitment to provide reasons (or arguments) to justify the validity of such assertion or else, has to retract it. All commitments made by an agent during the dialogue are commonly stored in an individual database called *commitment store (CS)* [10] (there is one commitment store per agent), which is accessible by other agents that are engaged in a dialogue with the agent.

As pointed out before, we follow the standard notation of *modal logics* of knowledge and believe described in [33, report 13]. Thus, we use the modal operators

$K_i\phi$: "Agent a_i knows ϕ"

$B_i\phi$: "Agent a_i believes that ϕ is true"

$C_g\phi$: "ϕ is common knowledge for any agent in the group g if any agent of the group knows it and knows that it is common knowledge"

and the modal connective

$\Diamond\phi$ is satisfied now if ϕ is satisfied either now or at some future moment.

Note that here we make a distinction between what agents *know* (which is considered to be true) and what agents *believe* (which forms part of the mental state of an agent and can be true or not).For instance, all agents that belong to a society that represents the workers of a car rental company can *know* that the business manager *believe* that Volvo is the safest brand that they can offer to its customers. The workers know what the manager believe, and the fact that everybody knows the opinion of the manager is true. However, this doesn't mean that such opinion has to be true and in fact, any worker can believe that BMW is safer than Volvo. Therefore, the belief of the manager is subjective and depends on its individual knowledge.

In addition, as proposed in [25], we use the following simplified elements of *FIPA*'s communicative act library specification[f]:

Done[$locution(a_s, \phi)$*, preconditions]*

which indicates that $locution(a_s, \phi)$ (or correspondingly $locution(a_s, a_r, \phi)$) has been put forward by agent a_s (addressed to agent(s) a_r) with content ϕ and the specified *preconditions* hold before this utterance and

Feasible[condition, $locution(a_s, \phi)$*]*

which means that if *condition* can take place, $locution(a_s, \phi)$ (or correspondingly $locution(a_s, a_r, \phi)$) will be put forward by agent a_s (addressed to agent(s) a_r) with content ϕ.

Further notation that we use throughout this report is the next:

a_s: the $Agent(a_s)$ sender of the locution.

a_r: the $Agent(a_r)$ receiver of the locution.

arg_i: an $Argument(arg_i)$ of an $Agent(a_i)$.

SS_i: the $SupportSet(SS_i)$ of the $Argument(arg_i)$ that has put forward an $Agent(a_i)$.

CS_i: the commitment store of an $Agent(a_i)$.

q: the $Problem(q)$ under discussion.

p_i: the $Solution(p_i)$ (or position) proposed by an $Agent(a_i)$ to solve the $Problem(q)$.

[f] http://www.fipa.org/specs/fipa00037/SC00037J.html

4 Syntax

In this section we provide the syntax of the communication protocol that follow the agents of our argumentation framework. To formalise it, we follow the *dialogue game* approach proposed in [24] and extended in [26]. This approach is prospective (intended to modeling systems to represent the reality and that do not exist yet), which fits the objective of most open MAS. Other approaches to formalise dialogue systems have been reviewed in [27] (concretely, formal systems for persuasion dialogue). However, most of these proposals are retrospective (intended to reconstruct/explain what happened in a dialogue, using a legal dispute as typical example). Furthermore, they assume a consistent and presupposed *context* that represents fixed and indisputable knowledge that cannot be changed during the dialogue. This assumption cannot be made in open MAS where heterogeneous agents with potentially partial knowledge about the context of the dispute can enter or leave the system (and hence the dialogue) at any time.

Along this report, we assume that a set of agents with different positions (points of view) are arguing to reach an agreement to solve a complex problem. Thus, our basic notion of agreement consists of a solution for a generic problem that several agents must solve. At this level of abstraction, we assume that this is a generic problem of any type (e.g. resource allocation, classification, prediction, etc.) that could be described with a set of features. However, different notions of agreement can be found in the literature of agreement technologies [7, 8].

Next, we present the elements of the dialogue: the set of allowed *locutions*, the *commencement rules*, the *combination rules* that govern the course of the dialogue, the *commitment rules* that define the commitments that each agent makes when it utters each locution and how these commitments can be combined, the *rules for speaker order* and the *termination rules*. The dialogue game presented in this section is aimed at providing a communication protocol for agents that engage in an agreement process. This process can be seen as a *collaborative deliberation*, where all agents follow to select the best solution for a problem at hand and do not perceive any reinforcement or reward if their position is selected as the final solution to apply, as a *negotiation*, where agents try to convince other agents to apply its solution as the best for solving the problem and have individual *utility functions* that increase its perceived utility in that case, or also as a *persuasion*, where each agent tries to persuade the rest of agents to change their opinions and support its solution as the best option to solve the problem.

Locutions

The set of allowed locutions of our dialogue game are the following:

L1: $open\_dialogue(a_s, \phi)$, where ϕ is a problem q to solve in the system application domain. With this locution, an agent a_s opens the argumentation dialogue, asking other agents to collaborate or negotiate to solve a problem that it has been presented with.

L2: $enter\_dialogue(a_s, \phi)$, where ϕ is a problem q to solve in the system application domain. With this locution, an agent a_s engages in the argumentation dialogue to solve the problem.

L3: $withdraw\_dialogue(a_s, \phi)$, where ϕ is a problem q to solve in the system application domain. With this locution, an agent a_s leaves the argumentation dialogue to solve the problem.

L4: $propose(a_s, \phi)$, where ϕ is a position p. With this locution, an agent a_s puts forward the position p as its proposed solution to solve the problem under discussion in the argumentation dialogue.

L5: $why(a_s, a_r, \phi)$, where ϕ can be a position p or an argument $arg \in \mathcal{A}$. With this locution, an agent a_s challenges the position p or the argument arg of an agent a_r, asking it for a support argument.

L6: $noCommit(a_s, \phi)$, where ϕ is a position p. With this locution, an agent a_s withdraws its position p as a solution for the problem under discussion in the argumentation dialogue.

L7: $assert(a_s, a_r, \phi)$, where ϕ can be an argument $arg \in \mathcal{A}$ that supports a position, other argument or an objectively verifiable evidence about the system application domain. With this locution, an

agent a_s sends to an agent a_r an argument or an evidence that supports its position or a previous argument that a_r has put forward.

L8: $accept(a_s, a_r, \phi)$, where ϕ can be an argument $arg \in \mathcal{A}$ or a position p to solve a problem. With this locution, an agent a_s accepts the argument arg or the position p of an agent a_r. Also, this locution can be used at the end of the dialogue to inform all agents about the final position agreed as the best position to solve the problem. In that case, a_r denotes *all* individuals that belong to the concept *Agent*, except for the sender a_s ($all: \forall a_i, a_i \neq a_s / Agent(a_i)$).

L9: $attack(a_s, a_r, \phi)$, where ϕ is an argument $arg \in \mathcal{A}$ of an agent a_s. With this locution, an agent a_s challenges an argument of an agent a_r with its argument arg.

L10: $retract(a_s, a_r, \phi)$, where ϕ is an argument $arg \in \mathcal{A}$. With this locution, an agent a_s informs an agent a_r that it withdraws the argument arg that it put forward in a previous step of the argumentation dialogue.

Commencement Rules

The dialogue starts when an agent a_s is presented with a new problem q to solve. First, the agent tries to solve it by using its own knowledge resources. Then, it opens a dialogue with other agents by sending them the locution $open\_dialogue(a_s, a_r, q)$, where a_r can be any agent a_i that a_s knows. After that, a_i enters in the dialogue by posing the locution $enter\_dialogue(a_s, q)$ (where $a_s = a_i$). After that, if a_i has been able to found a solution for q, it proposes this initial position p to solve the problem q with the locution $propose(a_s, p)$ (where $a_s = a_i$) and waits for the challenges of other agents or for other position proposals. Otherwise, it can challenge the positions of other agents engaged in the dialogue with the locution $why(a_s, a_r, p)$ (where $a_s = a_i$).

Rules for the Combination of Locutions

The rules for the combination of locutions define which locution can be put forward at each step of the dialogue game. Figure 1 represents a state machine with the possible stages of our dialogue game protocol. As shown in the figure, the protocol has three main stages: the *opening* stage, where the agent that initiates the dialogue opens the argumentation process to solve a problem; the *argumentation* stage, where agents argue to reach an agreement about the best solution to apply to solve the problem; and the *closing* stage, where the final decision about the position selected to solve the problem is reported to all agents that have participated in the dialogue. Next, the stages of our dialogue game and the rules for the combination of locutions in each stage are presented.

Opening Stage:

The opening stage commences when an agent a_s wants to establish an agreement process with other agents to solve a problem q that it has been faced with. Then, it uses the locution $open\_dialogue(a_s, q)$ to start the dialogue.

Argumentation Stage:

The argumentation stage follows the opening stage. Here, agents argue to reach an agreement about the solution to apply to the problem q. As shown in Figure 2, this stage is divided into a set of substages which activation is defined by the following rules (for clarity reasons, substages are labelled with the name of the rule that applies in each case):

R1: Once the dialogue has been opened, any agent that has been informed about it can enter in by using the locution $enter\_dialogue(a_s, q)$.

R2: After entering the dialogue, an agent can propose its position p to solve the problem q by putting forward the locution $propose(a_s, p)$. Alternatively, the agent can challenge the positions of other agents engaged in the dialogue (without being proposed its own position) with the locution $why(a_s, a_r, p)$. Also, in this substage the agent can withdraw from the dialogue by using the locution $withdraw\_dialogue(a_s, q)$.

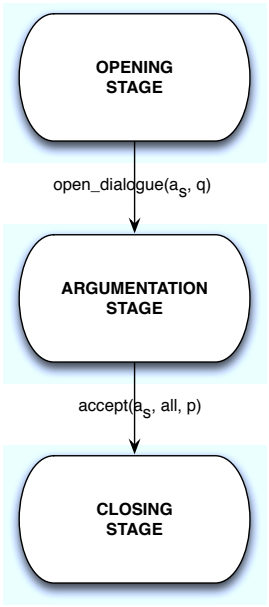

Figure 1: State Machine of the Dialogue Game

R3: In this substage, an agent that has proposed its position p to solve the problem q can be asked by other agent for an argument to support this position with the locution $why(a_s, a_r, p)$. Also, p can be accepted by an agent engaged in the dialogue, which reports the proponent agent with the locution $accept(a_s, a_r, p)$. Furthermore, the proponent agent can withdraw its position p with the locution $noCommit(a_s, p)$. Alternatively, it can leave the dialogue with the locution $withdraw\_dialogue(a_s, q)$.

R4: After being asked for an argument to support its position p, an agent can use its knowledge resources to provide the requester agent with this argument arg by means of the locution $assert(a_s, a_r, arg)$. Alternatively, it can withdraw its position p by using the locution $noCommit(a_s, p)$.

R5: An agent that has received a support or an attack argument from other agent can use its knowledge resources to create an attack argument arg and send it to the other agent with the locution $attack(a_s, a_r, arg)$. Also, the agent can accept the support argument and report to the other agent with the locution $accept(a_s, a_r, arg)$, where arg is the support argument received. In its turn, an agent that has asserted the argument arg can withdraw it with the locution $retract(a_s, a_r, arg)$.

R6: When an agent receives an attack argument from other agent, it analyses the type of the attack and can use its knowledge resources to try to rebut the attack. Therefore, if the attacking argument arg was a distinguishing premise or a counter-example ($arg = (DP \vee CE)$), the agent can distinguish the argument of the other agent with other distinguishing premise, or else counter-attack with other counter-example by using the locution $attack(a_s, a_r, arg)$. If the attacking argument was a critical question of the type presumption ($arg = CQ \wedge CQ.type = presumption$), the agent can use its knowledge resources to create and show to the other agent an argument arg with an evidence that supports that presumption by using the locution $assert(a_s, a_r, arg)$. Finally, if the attacking argument was a critical question of the type exception ($arg = CQ \wedge CQ.type = exception$), the agent can ask the other agent for an argument to support such critical question by stating the locution $why(a_s, a_r, arg)$. Alternatively, if the agent cannot rebut the attack, it can retract its argument with the locution $retract(a_s, a_r, arg)$. In its turn, any agent that has asserted the argument arg can withdraw it with the locution $retract(a_s, a_r, arg)$.

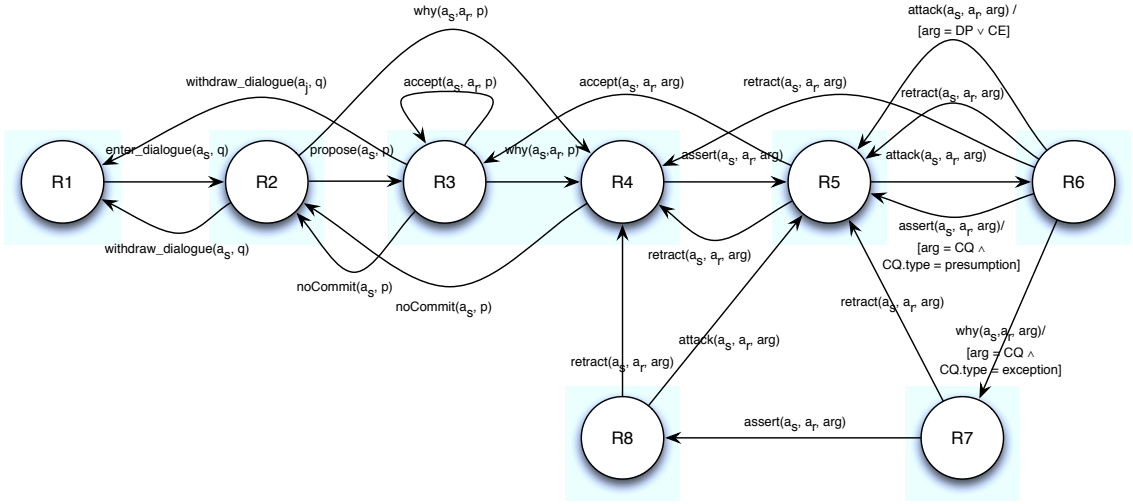

Figure 2: State Machine of the Argumentation Stage

R7: If an agent is asked by other agent for providing a support argument for its critical question of the type exception, this agent must use the locution $assert(a_s, a_r, arg)$ to assert an argument arg with an evidence to support such critical question attack or else, retract the attack by putting forward the locution $retract(a_s, a_r, arg)$.

R8: Once an agent has been provided by other agent with an evidence that supports the other agent's critical question of the type exception, this agent can retract its argument arg and report to the other with the locution $retract(a_s, a_r, arg)$ or else can try to generate an attack argument arg for the other agent's argument and send it the locution $attack(a_s, a_r, arg)$.

Also, note that any agent can withdraw its position at any stage of the dialogue. It implies that there are a transaction labelled with the locution $noCommit(a_s, p)$ from substages $R5...R8$ to substage $R2$, although they do not appear in Figure 2 for clarity purposes.

Closing Stage:

The closing stage can be activated at any time of the dialogue by the agent a_i that opened it. This stage is reached by putting forward the locution $accept(a_s, all, p)$ (where $a_s = a_i$) that informs all participating agents about the final position p agreed as the solution for the problem q. Here, the commitment store of all agents is deleted.

Commitment Rules

As pointed out before, agents make *dialogical commitments* with each locution that they put forward. These commitments are stored in an individual commitments database called *commitment store (CS)*. Also, the inclusion of a new commitment in the commitment store can make previous commitments to be inconsistent or invalid. Next, the commitment rules that define the commitments associated with each locution and how they inclusion in the commitment store affect to previous commitments are presented.

CR1: The locution $enter\_dialogue(a_s, q)$ gives rise to the creation of the commitment store CS_s of the sender agent.

CR2: The locution $propose(a_s, p)$ inserts the position p into the commitment store CS_s of the sender agent. If there is a previous position in CS_s, this position is replaced with the new position p. Thus, only one position can prevail in any commitment store.

CR3: The locution $withdraw\_dialogue(a_s, q)$ deletes the commitment store CS_s of the sender agent. This implies that the final agreement is only taken among the agents that remain listening in the substages $R2$ or $R3$. Also, agents cannot withdraw the dialogue before withdrawing any position that they have proposed with the locution $noCommit(a_s, p)$.

CR4: The locution $accept(a_s, a_r, p)$ inserts the position p into the commitment store CS_s of the sender. If there is a previous position in CS_s, this position is replaced with the new position p.

CR5: The locution $noCommit(a_s, p)$ deletes p from the commitment store CS_s of the sender.

CR6: The locution $why(a_s, a_r, p)$ commits the receiver to provide the sender with a supporting argument arg for p or else, to withdraw p with the locution *noCommit(a_s, p)*.

CR7: The locution $assert(a_s, a_r, arg)$ inserts the argument arg in the commitment store CS_s of the sender. Also, commitment stores cannot have inconsistent arguments. Then, if the conclusion of arg contradicts the conclusion of a previous argument stored in CS_s, the sender cannot put forward the locution $assert(a_s, a_r, arg)$ before deleting the inconsistent argument from CS_s with the locution *retract(a_s, a_r, arg)* addressed to any agent that is maintaining a dialogue with the sender.

CR8: The locution $accept(a_s, a_r, arg)$ inserts the argument arg into the commitment store CS_s of the sender. Again, commitment stores cannot have inconsistent arguments. Then, if the conclusion of arg contradicts the conclusion of a previous argument stored in CS_s, the sender cannot put forward the locution *assert(a_s, a_r, arg)* before deleting the inconsistent argument from CS_s with the locution *retract(a_s, a_r, arg)* addressed to any agent that is maintaining a dialogue with the sender.

CR9: The locution $retract(a_j, a_k, arg)$ deletes the argument arg from the commitment store CS_j of a_j.

CR10: The locution $attack(a_s, a_r, arg)$ inserts the argument arg in the commitment store CS_s of the sender. As pointed out before, commitment stores cannot have inconsistent arguments. Then, if the conclusion of arg contradicts the conclusion of a previous argument stored in CS_s, the sender cannot put forward the locution $attack(a_s, a_r, arg)$ before deleting the inconsistent argument from CS_s with the locution $retract(a_s, a_r, arg)$ addressed to any agent that is maintaining a dialogue with the sender. Also, if arg is a critical question of the type presumption, the locution $attack(a_s, a_r, arg)$ commits the receiver to providing an argument as evidence to support its last argument or else to retracting it.

CR11 : The locution $why(a_s, a_r, arg)$ where arg is an attack argument of the type *exception* put forward by the receiver to the last argument of the sender commits the receiver to having an argument to support its challenge or else to retracting it.

CR12: The locution $accept(a_s, all, p)$ $(all : \forall a_i, a_i \neq a_s / Agent(a_i))$ deletes the commitment stores of all agents that are still participating in the dialogue (including the initiator).

Rules for Speaker Order

During the dialogue, agents take turns to put forward locutions. Each time an agent a_s sends a locution to other agent a_r, it waits for an answer from a_r. However, any agent can held parallel argumentation dialogues with several agents. Then, in each of these dialogues, the argumentation succeeds as a two party dialogue between two agents, the one sending a locution to the other and waiting for a response. Nevertheless, the locution $open\_dialogue(a_s, q)$ is received by all agents of the society S_t. The locutions $accept(a_s, all, p)$, $propose(a_s, p)$, $noCommit(a_s, p)$ and $withdraw\_dialogue(a_s, p)$ are received by all agents that are engaged in the dialogue. Also, with these locutions, the sender agent does not wait for any response.

In this dialogue game protocol, we assume that all participating agents are able to see at each time the positions of the other agents by looking at their commitment stores. Also, when two agents are engaged in a dialogue, each agent has full access to the commitment store of the other. In this way, these agents can see the commitments associated to the arguments of their partners, but other agents can only have access to the positions proposed by each agent in the dialogue (also stored in the commitment stores). This preserves the privacy of the arguments that an agent puts forward in its argumentation dialogue with other agent. Note that if an agent wants to ask other agents for an opinion about an argument that it has received, it only has to send to these agents the argument, as it was its own argument. This simple rule allow us to use the same dialogue game to govern both collaborative deliberations and negotiations. In the former, all agents follow the common objective of proposing the best solution for a problem at hand. Therefore, there are not interested agents trying to take profit of the information interchanged between other agents to get a greater benefit with the final agreement reached. However, it could be the case in a negotiation, where each agent tries to increase its utility value perceived with the final agreement and use any extra information about other agents' knowledge and preferences to achieve that.

Termination Rules

The normal termination of the dialogue occurs when the argumentation process ends with all participating agents having proposed a prevailing position or having accepted the position of other agent. Then, agents may reach a decision about the final solution for the problem under discussion. In the ideal case, only the position of one participating agent prevails, while the other agents have withdrawn theirs and accepted this position by using the locution $accept(a_s, a_r, p)$. However, if at the end of the dialogue more than one position are still undefeated, agents can use a voting mechanism (selecting the position most accepted) or a random selection to decide the final outcome of the agreement process.

In any case, the agent a_i that opened the dialogue is responsible for reporting all participating agents the final position p selected as solution for the problem q at hand, by using the locution $accept(a_s, all, p)$ (where $a_s = a_i$). To avoid infinite dialogues, agents cannot put forward the same argument twice during a dialogue with other agent. Furthermore, a maximum time to reach an agreement can be established and agents must accept a position among those available at that moment to solve the problem.

Note that agents can maintain several parallel dialogues with other agents. Thus, once an agent has entered in the argumentation process with the locution *enter_dialogue(a_s, q)* it remains waiting to propose a position in the substage $R2$ or listening to incoming locutions of other agents in the substage $R3$. Then, the specific dialogue with an agent that has asked other agent for a support argument for its position p follows to the subsequent substages, but the agent still remains listening in $R3$ to other requests. Finally, the locution $noCommit(a_s, p)$ commits the sender to terminate any dialogue that its has started to defend p.

5 Semantics

In this section, we provide the formal semantics for the locutions of our dialogue game protocol. This semantics provides a common understanding about the properties of the communication language between agents. There are different methods to provide a communication language with a semantics [36], for instance, the *axiomatic* approach and the *operational* approach. The semantics of the locutions that define the communication language of the dialogue game presented in this report are provided in the following sections.

5.1 Axiomatic Semantics

The basic approach of semantics for communication languages is the axiomatic approach. With this approach, the meaning of the language is not explicitly defined but given in terms of properties that the language concepts satisfy [37]. Thus, in axiomatic semantics the semantics of a locution L is defined as a triple:

$\{pre\}\ L\ \{post\}$

where *pre* represents the preconditions that must hold before the locution is uttered and *post* represents the postconditions that apply after this utterance. Also, we can distinguish between *private* axiomatic semantics, where some preconditions or postconditions describe conditions of the dialogue that can only be observed by some agents and *public* axiomatic semantics, where all conditions are accessible to all agents. In our dialogue game protocol, the preconditions and postconditions of some locutions can only be observed by the sender and receiver agents. Thus, we present in this section the private axiomatic semantics for the locutions of our dialogue game. For clarity purposes, the preconditions that hold before the utterance of each locution in the communicative act *Done* are assumed to be the preconditions specified in the axiomatic semantics definition of the locution and thus, are omitted in the text.

Locutions Axiomatic Semantics

- **{*pre*}** $open\_dialogue(a_s, \phi)$ **{*post*}**

  pre : $Agent(a_s)$ wants to inform other agents of the society S_t about the proposition ϕ, which is a $Problem(q)$ to solve. Note that until agents do not enter the dialogue, their commitment stores CS are not created.
  $(K_s q) \wedge (\nexists CS_s)$

  post : All agents in the society S_t know that $Agent(a_s)$ wants to solve $Problem(q)$.
  $(\Diamond C_{S_t} K_s q)$

- **{*pre*}** $enter\_dialogue(a_s, \phi)$ **{*post*}**

  pre : $Agent(a_s)$ knows ϕ (the $Problem(q)$ reported by $Agent(a_i)$) and informs other participants of the $Group(g)$[g] that are engaged in the dialogue that it is willing to enter the dialogue to solve $Problem(q)$.
  $(Done[open\_dialogue(a_i, q), ...]) \wedge (K_s q)$

  post : Other participants of the $Group(g)$ are informed that $Agent(a_s)$ is willing to engage in a dialogue to solve $Problem(q)$. Also, the commitment store CS_s is created and $Agent(a_s)$ starts to belong to the $Group(g)$ of agents engaged in the dialogue.
  $(\Diamond C_g(a_s \in g)) \wedge (\exists CS_s)$

- **{*pre*}** $withdraw\_dialogue(a_s, \phi)$ **{*post*}**

  pre : $Agent(a_s)$ that has engaged in the argumentation dialogue to solve ϕ (the $Problem(q)$) wants to leave from the dialogue and report it to the other agents of the $Group(g)$ that are engaged in the dialogue. Note that agents cannot withdraw the dialogue before withdrawing any position $Solution(p)$ that they have proposed.
  $(Done[enter\_dialogue(a_s, q), K_s q]) \wedge (\nexists p \in CS_s)$

  post : Other participating agents of the $Group(g)$ know that $Agent(a_s)$ no longer participates in the dialogue to solve $Problem(q)$. Also, the commitment store CS_s of $Agent(a_s)$ is deleted.
  $(\Diamond C_g(a_s \notin g)) \wedge (\nexists CS_s)$

- **{*pre*}** $propose(a_s, \phi)$ **{*post*}**

  pre : An $Agent(a_s)$ that has engaged in a dialogue to solve ϕ (the $Problem(q)$) wants to propose its position $Solution(p)$ as a solution for the problem and reports it to the other agents of the $Group(g)$. An agent cannot propose a new position without withdrawing a previous $Solution(r)$ from its commitment store, if any.
  $(Done[enter\_dialogue(a_s, q), ...]) \wedge (B_s p) \wedge (\forall r \neq p)(\nexists r \in CS_s)$

[g] Agents know which other agents are participating in the dialogue by looking at their commitment stores.

post : Other participating agents of the $Group(g)$ know that $Agent(a_s)$ has proposed position $Solution(p)$ as solution for $Problem(q)$ and it is inserted in the commitment store CS_s of $Agent(a_s)$.
$(\Diamond C_g B_s p) \wedge (p \in CS_s)$

- **{*pre*}** $why(a_s, a_r, \phi)$ **{*post*}**
This locution has different semantics depending on its content ϕ. On one hand, if ϕ is a position $Solution(p)$ to solve the problem under discussion we have the following conditions:

pre : $Agent(a_s)$ wants to challenge $Agent(a_r)$ to provide a justification for the position $Solution(p)$.
$(Done[propose(a_r, p), B_r p]) \wedge (K_s B_r p) \wedge (p \notin CS_s)$

post : $Agent(a_r)$ knows that $Agent(a_s)$ does not believe $Solution(p)$ and has the dialogical commitment of justifying it with an $Argument(arg) \vdash^+ Solution(p)$ or else of withdrawing it.
$(\Diamond K_r \neg B_s p) \wedge ((Feasible[\exists arg / arg \vdash^+ p), assert(a_r, a_s, arg)]) \vee$
$(Feasible[\nexists arg / arg \vdash^+ p), noCommit(a_r, p)]))$

On the other hand, if ϕ is an attacking $Argument(arg_r)$ of $Agent(a_r)$ that poses a critical question $Premise(exc)$ of the type *exception* to a previous $Argument(arg_s)$ of $Agent(a_s)$ (such that $hasSupportSet(arg_r, SS_r) \wedge hasException(SS_r, exc) \wedge Argument(arg_r) \vdash^- Argument(arg_s)$) we have the following conditions:

pre : $Agent(a_s)$ wants to challenge $Agent(a_r)$ to provide a justification *Argument($arg_{r'}$)* for its attack $Argument(arg_r)$ to $Argument(arg_s)$.
$(Done[attack(a_r, a_s, arg_r), \neg B_r arg_s]) \wedge (K_s B_r arg_r) \wedge (arg_r \notin CS_s)$

post : $Agent(a_r)$ knows that $Agent(a_s)$ does not believe its $Argument(arg_r)$. Thus, it is committed to providing a justification $Argument(arg_{r'})$ for its attacking argument $Argument(arg_r)$ such that $Argument(arg_{r'}) \vdash^+$ *Argument(arg_r)* or else, to withdrawing the it.
$(\Diamond K_r \neg B_s arg_r) \wedge ((Feasible[\exists arg_{r'} / arg_{r'} \vdash^+ arg_r), assert(a_r, a_s, arg_{r'})]) \vee (Feasible[\nexists arg_{r'} / arg_{r'} \vdash^+ arg_r), retract(a_r, a_s, arg_r)]))$

- **{*pre*}** $noCommit(a_s, \phi)$ **{*post*}**

pre : $Agent(a_s)$ that has put forward ϕ (the position $Solution(p)$) wants to withdraw it and report this change to the other participating agents of the $Group(g)$ engaged in the dialogue.
$(p \in CS_s) \wedge (C_g B_s p)$

post : Other participating agents of the $Group(g)$ know that $Agent(a_s)$ no longer proposes $Solution(p)$ as its position to solve the problem at hand. Also, $Solution(p)$ is deleted from the commitment store CS_s of $Agent(a_s)$.
$(\Diamond C_g \neg B_s p) \wedge (p \notin CS_s)$

- **{*pre*}** $assert(a_s, a_r, \phi)$ **{*post*}**
This locution has different semantics depending on its content ϕ. However, in any case an argument cannot be inserted in the commitment store of an agent without deleting first any inconsistent argument. Then, on one hand, it could be the case that $Agent(a_r)$ has challenged the position $Solution(p)$ of $Agent(a_s)$. In this case, ϕ is an $Argument(arg)$ that supports this position, such that $Argument(arg) \vdash^+ Solution(p)$.

pre : An $Agent(a_s)$ wants to provide a justification for its position $Solution(p)$ and reports it to an agent $Agent(a_r)$ that has challenged it.
$(Done[why(a_r, a_s, p), ...]) \wedge (\exists arg)(arg \vdash^+ p) \wedge (\nexists arg_s \in CS_s)(arg \vdash^- arg_s)$

post : $Agent(a_r)$ knows that $Agent(a_s)$ has provided $Argument(arg)$ as a justification for its position and it is inserted in the commitment store CS_s of $Agent(a_s)$.
$(\Diamond K_r B_s arg) \wedge (arg \in CS_s)$

On the other hand, $Agent(a_r)$ could have attacked the argument $Argument(arg_s)$ with an $Argument(arg_r)$ that poses a critical question $Premise(pre)$ of the type *presumption* such that $hasSupportSet(arg_r, SS_r)$ $\wedge$ $hasPresumption(arg_r, pre)$ $\wedge$ $Argument(arg_r)$ $\vdash^-$ $Argument(arg_s)$. In this case, ϕ is an $Argument(arg_{s'})$ of $Agent(a_s)$ that supports its previous argument $Argument(arg_s)$, such that $Argument(arg_{s'}) \vdash^+ Argument(arg_s)$.

pre : $Agent(a_s)$ wants to rebut a critical question attack of the type *presumption* posed by $Agent(a_r)$, such that $Argument(arg_r) \vdash^- Argument(arg_s)$, but $Argument(arg_{s'}) \vdash^+ Argument(arg_s)$ and hence rebuts $Argument(arg_r)$.
$Done[attack(a_r, a_s, arg_r), ...] \wedge (\exists arg_{s'})(arg_{s'}$
$\vdash^+ arg_s) \wedge (\nexists arg_{s''} \in CS_s)(arg_{s'} \vdash^- arg_{s''})$

post : $Agent(a_r)$ knows that $Agent(a_s)$ has provided $Argument(arg_{s'})$ as a justification for its $Argument(arg_s)$ and it is inserted in the commitment store CS_s of $Agent(a_s)$.
$(\Diamond K_r B_s arg_{s'}) \wedge (arg_{s'} \in CS_s)$

Finally, $Agent(a_r)$ could have challenged the argument $Argument(arg_s)$ that poses a critical question $Premise(exc)$ of the type *exception* to its $Argument(arg_r)$ such that $hasSupportSet(arg_s, SS_s)$ $\wedge$ $hasException(SS_s, exc)$ $\wedge$ $Argument(arg_s)$
$\vdash^-$ $Argument(arg_r)$. In this case, ϕ is an $Argument(arg_{s'})$ of $Agent(a_s)$ that supports its critical question attack posed with $Argument(arg_s)$, such that
$Argument(arg_{s'}) \vdash^+ Argument(arg_s)$.

pre : $Agent(a_s)$ wants to support a critical question attack of the type *exception*, such that $Argument(arg_s) \vdash^- Argument(arg_r)$ and $Argument(arg_{s'}) \vdash^+ Argument(arg_s)$.
$Done[why(a_r, a_s, arg_s), ...] \wedge (\exists arg_{s'})(arg_{s'} \vdash^+ arg_s) \wedge$
$(\nexists arg_{s''} \in CS_s)(arg_{s'} \vdash^- arg_{s''})$

post : $Agent(a_r)$ knows that $Agent(a_s)$ has provided $Argument(arg_{s'})$ as a justification for its $Argument(arg_s)$ and it is inserted in the commitment store CS_s of $Agent(a_s)$.
$(\Diamond K_r B_s arg_{s'}) \wedge (arg_{s'} \in CS_s)$

- **{*pre*}** $attack(a_s, a_r, \phi)$ **{*post*}**
This locution has different semantics depending on its content ϕ, which represents different types of arguments. Again, in any case an argument cannot be inserted in the commitment store of an agent without deleting first any inconsistent argument. With the *attack* locution, the $Agent(a_s)$ puts forward an $Argument(arg_s)$ to attack the $Argument(arg_r)$ of an $Agent(a_r)$, such that $Argument(arg_s) \vdash^- Argument(arg_r)$. $Argument(arg_s)$ can be of different types. On one hand, if $Argument(arg_r)$ is a support argument with one or more premises in its support set, such that $hasSupportSet(arg_r, SS_r) \wedge Premise(pr_r) \wedge hasPremise(SS_r, pr_r)$, $Argument(arg_s)$ can be a *distinguishing-premise* attack.

pre : $Agent(a_s)$ wants to attack the support $Argument(arg_r)$ of an $Agent(a_r)$ with an $Argument(arg_s)$, such that $hasSupportSet(arg, SS_s) \wedge$
$Premise(DP) \wedge hasDistinguishingPremise(SS_s, DP)$.
$(Done[assert(a_r, a_s, arg_r), ...]) \wedge (\exists arg_s)(arg_s \vdash^- arg_r) \wedge (\nexists arg_{s'} \in CS_s)$
$(arg_s \vdash^- arg_{s'})$

post : $Agent(a_r)$ knows that $Agent(a_s)$ does not believe its support $Argument(arg_r)$ and $Argument(arg_s)$ is inserted into the commitment store CS_s of $Agent(a_s)$.
$(\Diamond K_r \neg B_s arg_r) \wedge (K_r B_s arg_s) \wedge (arg_s \in CS_s)$

On the other hand, if $Argument(arg_r)$ is a support argument with one or more *argument-cases* or *domain-cases* in its support set, such that
$hasSupportSet(arg_r,\ SS_r) \wedge Case(c_r) \wedge (hasDomainCase(SS_r, c_r) \vee$
$hasArgumentCase(SS_r, c_r))$, then $Argument(arg_s)$ can be a *counter-example* attack. In that case, the axiomatic semantics coincide with the previous case.

Alternatively, if $Argument(arg_r)$ is a support argument with one or more *argumentation-schemes* in its support set, such that $hasSupportSet(arg_r, SS_r) \wedge ArgumentationScheme(as_r) \wedge (hasArgumentationScheme(SS_r, as_r)$, then *Argument($arg_s$)* can be a *critical question* attack. In the case of critical questions of the type *presumption* the locution has the semantics specified next:

pre : $Agent(a_s)$ wants to attack the support $Argument(arg_r)$ of an $Agent(a_r)$ with an $Argument(arg_s)$, such that $hasSupportSet(arg, SS_s) \wedge Premise(pre_s) \wedge hasPresumption(SS_s, pre_s) \wedge hasPresumption(as_r, pr$
$(Done[assert(a_r, a_s, arg_r), ...]) \wedge (\exists arg_s)(arg_s \vdash^- arg_r) \wedge$
$(\nexists arg_{s'} \in CS_s)(arg_s \vdash^- arg_{s'})$

post : $Agent(a_r)$ knows that $Agent(a_s)$ does not believe its support *Argument(arg_r)* and it is committed to provide an $Argument(arg_{rr})$ to support it or else to withdrawing it. Also, $Argument(arg)$ is inserted into the commitment store CS_s of $Agent(a_s)$.
$(\Diamond K_r \neg B_s arg_r) \wedge (K_r B_s arg) \wedge ((Feasible[\exists arg_{rr} \vdash^+ arg_r),$
$assert(a_r, a_s, arg_{rr})]) \vee (Feasible[\nexists arg_{rr} \vdash^+ arg_r), retract(a_r, a_s, arg_r)])) \wedge (arg \in CS_s)$

In the case of critical questions of the type *exception* the locution has the following semantics:

pre : $Agent(a_s)$ wants to attack the support $Argument(arg_r)$ of an $Agent(a_r)$ with an $Argument(arg_s)$, such that $hasSupportSet(arg_s, SS_s) \wedge$
$Premise(pre_s) \wedge hasException(SS_s, pre_s) \wedge hasException(as_r, pre_s)$.
$(Done[assert(a_r, a_s, arg_r), ...]) \wedge (\exists arg_s)(arg_s \vdash^- arg_r) \wedge$
$(\nexists arg_{s'} \in CS_s)(arg_s \vdash^- arg_{s'})$

post : $Agent(a_r)$ knows that $Agent(a_s)$ does not believe its support *Argument(arg_r)*. Also, $Argument(arg_s)$ is inserted into the commitment store CS_s of $Agent(a_s)$.
$(\Diamond K_r \neg B_s arg_r) \wedge (K_r B_s arg_s) \wedge (arg_s \in CS_s)$

Finally, $Argument(arg_r)$ can be an attack argument to the $Argument(arg_{s'})$ of $Agent(a_s)$ with a *distinguishing-premise* or a *counter-example* in its support set such that $hasSupportSet(arg_r, SS_r)$ $\wedge ((Premise(pr_r) \wedge$
$hasDistinguishingPremise(SS_r, pr_r) \vee (Case(c_r) \wedge$
$hasCounterExample(SS_r, c_r)))$. Then, $Argument(arg_s)$ can be an attack argument that rebuts $Argument(arg_r)$ with other *counter-example* or *distinguishing-premise*.

pre : $Agent(a_s)$ wants to rebut the attack $Argument(arg_r)$ of an $Agent(a_r)$ with an $Argument(arg_s)$, such that $hasSupportSet(arg_s, SS_s) \wedge ((Case(c_s) \wedge hasCounterExample(SS_s, c_s)) \vee ((Premise(pr_s)$
$\wedge$
$hasDistinguishingPremise(SS_s, pr_s)))$.
$(Done[attack(a_r, a_s, arg_r), ...]) \wedge (\exists arg_s)(arg_s \vdash^- arg_r) \wedge$
$(\nexists arg_{s''} \in CS_s)(arg_s \vdash^- arg_{s''})$

post : $Agent(a_r)$ knows that $Agent(a_s)$ does not believe its attack $Argument(arg_r)$. Also, $Argument(arg_s)$ is inserted into the commitment store CS_s of $Agent(a_s)$.
$(\Diamond K_r \neg B_s arg_r) \wedge (K_r B_s arg_s) \wedge (arg_s \in CS_s)$

- **{*pre*}** $accept(a_s, a_r, \phi)$ **{*post*}**
This locution has different semantics depending on the content of ϕ. On one hand, ϕ can be a position $Solution(p)$ proposed by $Agent(a_r)$.

  pre : $Agent(a_s)$ wants to accept a position $Solution(p)$ proposed by $Agent(a_r)$.
  $(K_s B_r p) \wedge (B_s p)$

  post : $Agent(a_r)$ knows that $Agent(a_s)$ has accepted its position. Also, this position is inserted into the commitment store CS_s of $Agent(a_s)$ (and replaces a previous position if any).
  $(\Diamond K_r B_s p) \wedge (p \in CS_s) \wedge (\nexists p_s \in CS_s)(p_s \neq p)$

  On the other hand, ϕ can be an argument $Argument(arg_r)$ proposed by $Agent(a_r)$.

pre : $Agent(a_s)$ wants to accept the $Argument(arg_r)$ proposed by $Agent(a_r)$ and there is not any inconsistent argument in the commitment store CS_s of $Agent(a_s)$.
$(K_s B_r arg_r) \wedge (B_s arg_r) \wedge (\nexists arg_s \in CS_s)(arg_r \vdash^{-} arg_s)$

post : $Agent(a_r)$ knows that $Agent(a_s)$ has accepted its argument. Also, this argument is inserted into the commitment store CS_s of $Agent(a_s)$.
$(\Diamond K_r B_s arg_j) \wedge (arg_r \in CS_s)$

- **{*pre*}** $retract(a_s, a_r, \phi)$ **{*post*}**

  pre : $Agent(a_s)$ wants to withdraw ϕ, which is an $Argument(arg_s)$ from its commitment store CS_s and reports it to any agent of the $Group(g)$ that is engaged in a dialogue with it.
  $(\neg B_s arg_s) \wedge (C_g B_s arg_s)$

  post : Every agent of the $Group(g)$ knows that $Agent(a_s)$ no longer believes $Argument(arg_s)$ and it is deleted from its commitment store.
  $(\Diamond C_g \neg B_s arg_s) \wedge (arg_s \notin CS_s)$

- **{*pre*}** $accept(a_s, all, \phi)$ **{*post*}**

  pre : $Agent(a_s)$ wants to close the dialogue and report all agents of the $Group(g)$ engaged in it the final $Solution(p)$ agreed to apply to the $Problem(q)$ at hand.
  $(C_g q) \wedge (C_g p)$

  post : All agents $Agent(ag)$ of $Group(g)$ know that $Agent(a_s)$ believes $Solution(p)$. Also, their commitment stores are deleted and the dialogue ends.
  $(\Diamond C_g B_s p) \wedge (\forall ag \in g)(\nexists CS_{ag})$

The axiomatic semantics is typically used to provide preliminary specifications of communication languages, but the knowledge of only its properties is not sufficient to understand a language.Thus, next section complements this semantics with an additional form of semantics that provide meaning for the transitions between the stages of the dialogue.

5.2 Operational Semantics

The operational semantics views the dialogue game protocol as an abstract state machine and defines precisely the transitions between states. These transitions are triggered by the utterance of each locution. However, from some stages an agent can utter different locutions following different *agent decision mechanisms*, which are reasoning mechanisms that agents can use to choose the locution to utter in the next step of the dialogue among a set of candidates. These mechanisms depend on the knowledge that agents can infer from their knowledge resources or even on the specific design of agents. For instance, agents that are designed to be more competitive and, if possible, always put forward attack arguments or agents that are designed to remain listening and only engage in a dialogue if their positions or arguments are attacked. Figure 3 shows the decision mechanisms that agents can use in each substage of the argumentation stage of our protocol. For clarity purposes, arrows labelled with the decision mechanism *D8* (presented below) from substages *R5*, *R6*, *R7* and *R8* to substage *R2* are omitted in the figure.

To define the transition rules of our protocol we follow the notation of [25]:

$$\langle a_i, K, o \rangle$$

where a_i is an agent, K is a decision mechanism (or the terminal state T) and o is the output of the mechanism K (send a locution or remain listening to incoming locutions). Some transitions are labelled with the locutions that trigger them while others, which occur between the mechanisms of a single agent, remain unlabeled. Also, if no specific output is invoked we denote this by a period in the third parameter of the triple ($\langle a_i, K, . \rangle$).

Concretely, we have identified the following decision mechanisms:

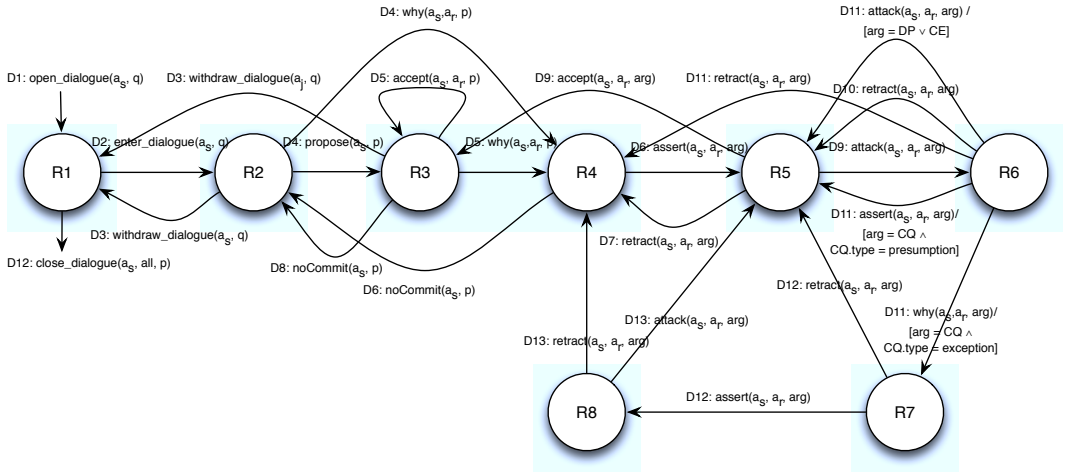

Figure 3: Decision Mechanisms of the Dialogue Game

- **D1 Open Dialogue:** A mechanism that allows an agent to open a dialogue with other agents of the society S_t that the agent belongs to, uttering the locution $open\_dialogue(a_s, q)$ or not. The output of this mechanism is: *send(open_dialogue(a_s, ϕ))*.

- **D2 Enter or Close Dialogue:** A mechanism that allows an agent to decide to engage in a dialogue and utter the locution $enter\_dialogue(a_s, q)$ or not. By this mechanism, the agent makes a query to its knowledge resources, trying to find a solution for the problem to solve. If the agent can provide a solution for the problem, the agent uses the mechanism to decide whether it enters in the dialogue or not. Alternatively, the agent that started the dialogue can also close it with the locution $accept(a_s, all, p)$. The outputs of this mechanism are: *send(enter_dialogue(a_s, ϕ))*, *listen()* or *send(close_dialogue(a_s, all, ϕ))*.

- **D3 Withdraw from Dialogue:** A mechanism that allows an agent to withdraw from the dialogue and put forward the locution $withdraw\_dialogue(a_s, q)$. The mechanism first checks that the agent has not any active position to solve the problem (agents cannot withdraw from the dialogue before withdrawing their positions). Possible outputs are: *send(withdraw_dialogue(a_s, ϕ))*.

- **D4 Propose or Challenge:** A mechanism that allows an agent to make a proposal to solve the problem under discussion and utter the locution $propose(a_s, p)$ or to challenge the positions of other agents uttering the locution $why(a_s, a_r, p)$. By this mechanism the agent uses its knowledge resources to generate and select the position to propose. If the agent has been able to generate a position to solve the problem, it uses the mechanism to decide whether to put forward that position. In any case, the agent can challenge other positions or remain listening to the utterances of other agents. The outcomes for this mechanism are: *send(propose(a_s, ϕ))*, *send(why(a_s, a_r, ϕ))* or *listen()*.

- **D5 Accept or Challenge:** A mechanism that allows an agent to query its knowledge resources and decide to accept or challenge the position of other agent. If the agent is able to generate the same position as its candidate to solve the problem, it can utter the locution $accept(a_s, a_r, p)$ to accept the other's position. Else, if the position cannot be generated or is generated but not ranked as the most suitable solution for the problem, the agent can use this mechanism and decide to accept the other agent's position or to challenge it with the locution $why(a_s, a_r, p)$. Thus, possible outcomes are: *send(accept(a_s, ϕ))* or *send(why(a_s, a_r, ϕ))*.

- **D6 Defend Position:** A mechanism that allows an agent to defend its position from a challenge or else, to withdraw it. By this mechanism the agent decides if it is able to use its knowledge resources

to provide the challenger with an argument that supports its position. In that case, it can utter the locution $assert(a_s, a_r, arg)$. Otherwise, the agent has to withdraw the position by using the locution $noCommit(a_s, p)$. Also, the agent that put forward the challenge can use this mechanism to listen for the answer to its challenge. The outcomes of this mechanism are: *send(assert(a_s, a_r, ϕ))*, *send(noCommit(a_s, ϕ))* or *listen()*.

- **D7 Withdraw Argument:** This mechanism allows an agent to decide whether to withdraw an argument that it has put forward, using the locution $retract(a_s, a_r, \phi)$. Possible outcomes are: *send(retract(a_s, a_r, ϕ))*.

- **D8 Withdraw Position:** A mechanism that allows an agent to decide whether to withdraw its proposed position with the locution $noCommit(a_s, p)$. The output of this mechanism is: *send(noCommit(a_s, ϕ))*.

- **D9 Accept or Attack:** A mechanism that allows an agent to query its knowledge resources and decide to accept or attack the argument of other agent. If the argument is consistent with the information inferred from the knowledge resources of the agent, it can utter the locution $accept(a_s, a_r, arg)$ to accept the other's argument. Else, if the argument is inconsistent and an attack argument can be generated from the knowledge resources, the agent can use this mechanism to decide to attack the argument by uttering the locution $attack(a_s, a_r, arg)$. Otherwise, if the argument cannot be decided (there is not enough information in the knowledge resources to support or rebut the argument) the agent also accepts it. Thus, possible outcomes are: *send(accept(a_s, ϕ))* or *send(attack(a_s, a_r, ϕ))*.

- **D10 Withdraw Attack:** This mechanism allows an agent to decide whether to withdraw an attack that it has put forward, using the locution $retract(a_s, a_r, \phi)$. Possible outcomes are: *send(retract(a_s, a_r, ϕ))* or *listen()*.

- **D11 Rebut Attack:** A mechanism that allows an agent to rebut an attack to its argument. By this mechanism the agent evaluates the attack argument received and queries its knowledge resources to search for information that supports or rebuts the attack. If the attack argument poses a critical question of the type *presumption*, the agent can rebut the attack by showing information that supports its argument with the locution $assert(a_s, a_r, \phi)$. If the attack argument poses a critical question of the type *exception*, the agent can rebut the attack by challenging it with the locution $why(a_s, a_r, \phi)$. Otherwise, if the attack argument poses a distinguishing-premise or a counter-example to the agent's argument, it can use the locution $attack(a_s, a_r, arg)$ to rebut the attack by counter-attacking with another distinguishing-premise or counter-example. In any case, if the agent is not able to rebut the attack with the information inferred from its knowledge resources, it can retract its argument by uttering the locution $retract(a_s, a_r, \phi)$. Therefore, the outcomes of this mechanism are: *send(assert(a_s, a_r, ϕ))*, *send(why(a_s, a_r, ϕ))*, *send(attack(a_s, a_r, ϕ))* or *send(retract(a_s, a_r, ϕ))*.

- **D12 Defend Argument:** This mechanism allows an agent to rebut a challenge to its argument, which poses a critical question of the type *exception*. With this mechanism, the agent queries its knowledge resources an tries to find information that supports its attack argument. In that case, the agent can rebut the attack by showing this information uttering the locution $assert(a_s, a_r, arg)$. Otherwise, the agent has to withdraw the attack by uttering $retract(a_s, a_r, arg)$. Also, the agent that put forward the challenge can use this mechanism to listen for the answer to its challenge. Possible outcomes are: *send(assert(a_s, a_r, ϕ))*, *send(retract(a_s, a_r, ϕ))* or *listen()*.

- **D13 Retract or Attack:** This mechanism allows an agent to counter-attack a critical question attack of the type *exception* posed to its argument. With this mechanism, the agent queries its knowledge resources to search for information that rebuts the attack. Then, if the agent finds such information, it can counter-attack uttering the locution $attack(a_s, a_r, \phi)$. Otherwise, the agent has to withdraw its argument uttering the locution $retract(a_s, a_r, \phi)$. Thus, the outcomes of the mechanism are: *send(attack(a_s, a_r, ϕ))* or *send(retract(a_s, a_r, ϕ))*.

Now, we define the transition rules of the operational semantics of our protocol.

- **TR1:** $\langle a_s, D1, send(open_dialogue(a_s, \phi))\rangle \xrightarrow{L1} \langle a_s, D2, .\rangle$
- **TR2:** $\langle a_s, D2, send(enter_dialogue(a_s, \phi))\rangle \xrightarrow{L2} \langle a_s, D3, .\rangle$
- **TR3:** $\langle a_s, D2, send(enter_dialogue(a_s, \phi))\rangle \xrightarrow{L2} \langle a_s, D4, .\rangle$
- **TR4:** $\langle a_s, D2, listen()\rangle \rightarrow \langle a_s, D2, .\rangle$
- **TR5:** $\langle a_s, D2, send(close_dialogue(a_s, all, \phi))\rangle \xrightarrow{L8} \langle all, T, .\rangle$
- **TR6:** $\langle a_s, D3, send(withdraw_dialogue(a_s, \phi))\rangle \xrightarrow{L3} \langle a_s, D2, listen()\rangle$
- **TR7:** $\langle a_s, D4, send(propose(a_s, p))\rangle \xrightarrow{L4} \langle a_s, D8, .\rangle$
- **TR8:** $\langle a_s, D4, send(propose(a_s, p))\rangle \xrightarrow{L4} \langle a_s, D5, .\rangle$
- **TR9:** $\langle a_s, D4, send(propose(a_s, p))\rangle \xrightarrow{L4} \langle a_r, D5, .\rangle$
- **TR10:** $\langle a_s, D4, send(why(a_s, a_r, \phi))\rangle \xrightarrow{L5} \langle a_s, D4, listen()\rangle$
- **TR11:** $\langle a_s, D4, send(why(a_s, a_r, \phi))\rangle \xrightarrow{L5} \langle a_r, D6, .\rangle$
- **TR12:** $\langle a_s, D4, listen()\rangle \rightarrow \langle a_s, D4, .\rangle$
- **TR13:** $\langle a_s, D8, send(noCommit(a_s, \phi))\rangle \xrightarrow{L6} \langle a_s, D4, listen()\rangle$
- **TR14:** $\langle a_s, D8, send(noCommit(a_s, \phi))\rangle \xrightarrow{L6} \langle a_s, D3, .\rangle$
- **TR15:** $\langle a_s, D5, send(accept(a_s, a_r, \phi))\rangle \xrightarrow{L8} \langle a_s, D5, .\rangle$
- **TR16:** $\langle a_s, D5, send(accept(a_s, a_r, \phi))\rangle \xrightarrow{L8} \langle a_r, D5, .\rangle$
- **TR17:** $\langle a_s, D5, send(why(a_s, a_r, \phi))\rangle \xrightarrow{L5} \langle a_s, D6, listen()\rangle$
- **TR18:** $\langle a_s, D5, send(why(a_s, a_r, \phi))\rangle \xrightarrow{L5} \langle a_r, D6, .\rangle$
- **TR19:** $\langle a_s, D6, listen()\rangle \rightarrow \langle a_s, D6, .\rangle$
- **TR20:** $\langle a_s, D6, send(assert(a_s, a_r, \phi))\rangle \xrightarrow{L7} \langle a_s, D7, .\rangle$
- **TR21:** $\langle a_s, D6, send(assert(a_s, a_r, \phi))\rangle \xrightarrow{L7} \langle a_s, D8, .\rangle$
- **TR22:** $\langle a_s, D6, send(assert(a_s, a_r, \phi))\rangle \xrightarrow{L7} \langle a_r, D9, .\rangle$
- **TR23:** $\langle a_s, D6, send(noCommit(a_s, \phi))\rangle \xrightarrow{L6} \langle a_s, D3, .\rangle$
- **TR24:** $\langle a_s, D6, send(noCommit(a_s, \phi))\rangle \xrightarrow{L6} \langle a_s, D4, listen()\rangle$
- **TR25:** $\langle a_s, D7, send(retract(a_s, a_r, \phi))\rangle \xrightarrow{L10} \langle a_s, D6, .\rangle$
- **TR26:** $\langle a_s, D9, send(accept(a_s, a_r, \phi))\rangle \xrightarrow{L8} \langle a_s, D3, .\rangle$
- **TR27:** $\langle a_s, D9, send(accept(a_s, a_r, \phi))\rangle \xrightarrow{L8} \langle a_s, D5, .\rangle$
- **TR28:** $\langle a_s, D9, send(accept(a_s, a_r, \phi))\rangle \xrightarrow{L8} \langle a_r, D8, .\rangle$

- **TR29:** $\langle a_s, D9, send(attack(a_s, a_r, \phi))\rangle \xrightarrow{L9} \langle a_s, D10, .\rangle$
- **TR30:** $\langle a_s, D9, send(attack(a_s, a_r, \phi))\rangle \xrightarrow{L9} \langle a_r, D8, .\rangle$
- **TR31:** $\langle a_s, D9, send(attack(a_s, a_r, \phi))\rangle \xrightarrow{L9} \langle a_r, D11, .\rangle$
- **TR32:** $\langle a_s, D10, listen()\rangle \rightarrow \langle a_s, D10, .\rangle$
- **TR33:** $\langle a_s, D10, send(retract(a_s, a_r, \phi))\rangle \xrightarrow{L10} \langle a_s, D9, .\rangle$
- **TR34:** $\langle a_s, D10, send(retract(a_s, a_r, \phi))\rangle \xrightarrow{L10} \langle a_r, D7, .\rangle$
- **TR35:** $\langle a_s, D10, send(retract(a_s, a_r, \phi))\rangle \xrightarrow{L10} \langle a_r, D8, .\rangle$
- **TR36:** $\langle a_s, D11, send(assert(a_s, a_r, \phi))\rangle \xrightarrow{L7} \langle a_s, D7, .\rangle$
- **TR37:** $\langle a_s, D11, send(assert(a_s, a_r, \phi))\rangle \xrightarrow{L7} \langle a_s, D8, .\rangle$
- **TR38:** $\langle a_s, D11, send(assert(a_s, a_r, \phi))\rangle \xrightarrow{L7} \langle a_r, D9, .\rangle$
- **TR39:** $\langle a_s, D11, send(why(a_s, a_r, \phi))\rangle \xrightarrow{L5} \langle a_s, D12, listen()\rangle$
- **TR40:** $\langle a_s, D11, send(why(a_s, a_r, \phi))\rangle \xrightarrow{L5} \langle a_r, D8, .\rangle$
- **TR41:** $\langle a_s, D11, send(why(a_s, a_r, \phi))\rangle \xrightarrow{L5} \langle a_r, D12, .\rangle$
- **TR42:** $\langle a_s, D11, send(attack(a_s, a_r, \phi))\rangle \xrightarrow{L9} \langle a_s, D7, .\rangle$
- **TR43:** $\langle a_s, D11, send(attack(a_s, a_r, \phi))\rangle \xrightarrow{L9} \langle a_s, D8, .\rangle$
- **TR44:** $\langle a_s, D11, send(attack(a_s, a_r, \phi))\rangle \xrightarrow{L9} \langle a_r, D9, .\rangle$
- **TR45:** $\langle a_s, D11, send(retract(a_s, a_r, \phi))\rangle \xrightarrow{L10} \langle a_s, D6, .\rangle$
- **TR46:** $\langle a_s, D11, send(retract(a_s, a_r, \phi))\rangle \xrightarrow{L10} \langle a_r, D6, listen()\rangle$
- **TR47:** $\langle a_s, D12, listen()\rangle \rightarrow \langle a_s, D12, .\rangle$
- **TR48:** $\langle a_s, D12, send(assert(a_s, a_r, \phi))\rangle \xrightarrow{L7} \langle a_s, D8, .\rangle$
- **TR49:** $\langle a_s, D12, send(assert(a_s, a_r, \phi))\rangle \xrightarrow{L7} \langle a_r, D13, .\rangle$
- **TR50:** $\langle a_s, D12, send(retract(a_s, a_r, \phi))\rangle \xrightarrow{L10} \langle a_s, D7, .\rangle$
- **TR51:** $\langle a_s, D12, send(retract(a_s, a_r, \phi))\rangle \xrightarrow{L10} \langle a_s, D8, .\rangle$
- **TR52:** $\langle a_s, D12, send(retract(a_s, a_r, \phi))\rangle \xrightarrow{L10} \langle a_r, D9, .\rangle$
- **TR53:** $\langle a_s, D13, send(attack(a_s, a_r, \phi))\rangle \xrightarrow{L9} \langle a_s, D7, .\rangle$
- **TR54:** $\langle a_s, D13, send(attack(a_s, a_r, \phi))\rangle \xrightarrow{L9} \langle a_s, D8, .\rangle$
- **TR55:** $\langle a_s, D13, send(attack(a_s, a_r, \phi))\rangle \xrightarrow{L9} \langle a_r, D9, .\rangle$
- **TR56:** $\langle a_s, D13, send(retract(a_s, a_r, \phi))\rangle \xrightarrow{L10} \langle a_s, D6, .\rangle$
- **TR57:** $\langle a_s, D13, send(retract(a_s, a_r, \phi))\rangle \xrightarrow{L10} \langle a_r, D6, listen()\rangle$

These transition rules provides the operational semantics of the dialogue, defining which is the range of potential decisions that agents can make in each stage of the dialogue.

6 Related Work

Dialogue games are interactions between two or more players, where each player 'moves' by making statements observing a pre-defined set of rules [23]. They are a specific type of games of the *game theory* that are different from the classical games studied in the economy research area in the sense that the profits or losses for the victory or defeat are not considered. Another important difference is that in dialogue games participants are not able to model the potential moves or other participants by using some uncertainty measure, for instance, some probabilistic measure. Dialogue games have been used with multiple purposes in computational linguistics, AI [6] and philosophy (concretely in argumentation theory [10, 17]). In CBR systems dialogue games have been applied to model human reasoning about legal precedents [30]. In MAS, their more recent and successful application consist in using them as a tool for the specification of communication protocols between agents. Thus, we can find abundant bibliography that formalises agent interaction protocols by using different dialogue games [1, 18]. Some other examples of dialogue game protocols about specific types of dialogues are: *information seeking* [13], *persuasion* [30, 3, 40], *negotiation* [20, 31, 14], *inquiry* [22] and *deliberation* [21]. To our knowledge, no research has been done to propose a dialogue game based on case-based resources that agents can use to manage agreement processes in agent societies. In the protocol presented this work, we do not focus on a specific type of dialogue, but we have proposed a generic dialogue game that can be used in deliberative, persuasive or negotiation dialogues where a group of agents must reach an agreement about the solution to apply to a generic problem of any type (e.g. resource allocation, classification, prediction, etc.) that could be described with a set of features.

A particular element of dialogue games, *commitment stores*, has been widely used in the area of MAS. The fact that an agent utters certain proposition during the dialogue means that this agent incurs certain level of commitment to this proposition and its implications or, at least, that the agent has certain support to justify this utterance. The concept of commitment stores comes from the study of *fallacies* (poor reasoning patterns that in some way imitate valid reasoning patterns) developed by Hamblin in [10]. According to this work, formal reasoning systems have public commitment stores for each participant, whose commitments can be withdrawn under certain circumstances. The inclusion of a new commitment gives rise to a previous verification that guarantees the coherence of the information of the store. Following Hamblin's approach, commitments have a purely dialogical processing (he calls them *propositional commitments*) and represent beliefs that do not necessary correspond with the actual beliefs of the participant. Furthermore, commitments may not hold out of the dialogue context. In this work, we use the concept of dialogue games to model the interaction between the agents that belong to a society. In doing so, we assume that the commitments that the agents make during the dialogue are stored in commitment stores accessible to the participants of the dialogue. Also, we endorse the view of Hamblin and define our notion of commitments as propositional commitments that agents incur during the dialogue, with no effect once the dialogue is terminated.

Other approach for the concept of commitment was provided by Walton and Krabbe in [38]. In this work commitments are understood as obligations of participants to incur, maintain or execute certain curse of action (they are *action commitments*). In this case, the commitments made during the dialogue can force the participants to perform certain actions out of the dialogue context. For these authors, commitments can also represent the fact of uttering statement in the dialogue. Therefore, propositional commitments are viewed as a specific type of action commitments.

Finally, a different approach for commitments was presented by Singh in [35], who proposes a social semantics for the agent communication languages. According to Singh, the participants of the dialogue have to express their *social commitments*. These commitments represent their beliefs about certain propositions and their intentions to execute actions in the future.

Despite the prolific applications of dialogue games in MAS, as discussed by Maudet in [19], a commonly accepted theory of dialogue games that is generic and suitable to any type of dialogue does not exist yet. However, there are a common set of requirements among the models based on dialogue games which define their *syntax*. Based on Maudet's requirements and the approaches found in the literature, in [23] a definition for the components that a dialogue game should have is proposed. However, a different view of the elements of dialogue games is presented in [30].

The components of the approach of McBurney and Parsons are the following:

- *Commencement rules*: rules that define the circumstances under which the dialogue commences.
- *Locutions*: rules which indicate what utterances are permitted. In legal contexts, for instance, locutions allow participants to express propositions, opponents to refute these propositions and again participants to refute this rebuttal justifying their propositions. Justifications imply to present proofs or arguments that defend these propositions.
- *Rules for combination of locutions*: rules that define the dialogical contexts under which particular locutions are permitted or not, or obligatory or not.
- *Commitment rules*: rules which define the circumstances under which participants incur dialogical commitment by their utterances. Also, these rules define how commitments are combined when utterances incurring conflicting commitments are made.
- *Rules for speaker order*: rules which define the order in which speakers can make utterances.
- *Termination rules*: rules that define the circumstances under which the dialogue ends.

Following Prakken's approach the common elements of dialogue systems are:

- A *topic language* $\mathcal{L}_t$, closed under classical negation.
- A *communication language* $\mathcal{L}_c$, where the set of *dialogues*, denoted by $\mathcal{M}^{\leq\infty}$, is the set of sequences from $\mathcal{L}_c$, and the set of *finite dialogues*, denoted by $\mathcal{M}^{<\infty}$, is the set of all finite sequences from $\mathcal{L}_c$.
- A *dialogue purpose*.
- A set $\mathcal{A}$ of *participants*, and a set $\mathcal{R}$ of *roles*, defined as disjoint subsets of $\mathcal{A}$. A participant a may or may not have a, possibly inconsistent, *belief base* $\Sigma_a \subseteq Pow(\mathcal{L}_t)$, which may or may not change during the dialogue. Furthermore, each participant has a, possibly empty set of commitments $\mathcal{C}_a \subseteq \mathcal{L}_t$, which usually changes during the dialogue.
- A *context* $\mathcal{K} \subseteq \mathcal{L}_\sqcup$, containing the knowledge that is presupposed and must be respected during the dialogue. The context is assumed consistent and remains the same throughout a dialogue.
- A *logic* L for $\mathcal{L}_t$, which may or may not be monotonic and which may or may not be argument-based.
- A set of *effect rules* $\mathcal{E}$ for $\mathcal{L}_c$, specifying for each utterance $\varphi \in \mathcal{L}_c$ its effects on the commitments of the participants.
- A *protocol* P for $\mathcal{L}_c$, specifying the legal moves at each stage of a dialogue. It is useful (although not strictly necessary) to explicitly distinguish elements of a protocol that regulate turntaking and termination.
- *Outcome rules* O, defining the outcome of the dialogue. For instance, in a negotiation the outcome is an allocation of resources, in a deliberation it is a decision on a course of action, and in persuasion dialogue it is a winner and a loser of the persuasion dialogue.

The approach of McBurney and Parsons is prospective (looking forward to model systems that do not exist yet). Opposite to this proposal, Prakken's approach is retrospective (looking back to reconstruct or explain what happened in a dialogue). Therefore, McBurney and Parson's approach can be considered as more suitable for modelling the dialogue between a set of heterogeneous agents whose interactions will determine the dynamics and operation of the system. In this work, hence, we have followed this approach. By contrast, Prakken's approach assumes a presupposed knowledge about the domain, which remains inalterable throughout the dialogue. However, in open MAS the context can also be changed as new agents enter in the system and new common knowledge is available.

Together with the definition of the syntax, a definition of semantics must be specified to provide a formal definition of the dialogue game. This semantics is concerned with the truth of falsity of utterances. The semantics of a dialogue game have the following functions [26, Chapter 13]:

- To provide a shared understanding to participants of the meaning of utterances, sequences of utterances and dialogues.
- To provide a shared understanding to designers of agent protocols of the meaning of utterances, sequences of utterances and dialogues.
- To provide a means of studying the properties of the protocol formally and with rigour.
- To provide a means of comparing protocols formally and with rigour.
- To provide a means of readily implementing protocols in production systems.
- To help ensure that implementation of agent communications in open MAS is undertaken uniformly.

There are different types of semantics for agent communication protocols and dialogue games [37]. One type of semantics, the *axiomatic* semantics, defines each locution of the protocol in terms of the pre-conditions that must exists before the locution can be uttered and the post-conditions which apply after its utterance. Axiomatic semantics can be *public* or *private*. In the former, the pre-conditions and post-conditions describe states or conditions of the dialogue that are publicly observable by all its participants whereas in the later some pre-conditions or post-conditions describe states or conditions of the dialogue that are only observable by some participants. Other type of semantics is called *operational* semantics. This semantics views the dialogue game protocol as an abstract state machine and defines precisely the transitions between states. The transitions are triggered by the utterance of each locution. The dialogue game proposed in this report has been formalised by specifying its axiomatic and operational semantics.

In a third type of semantics, the *denotational* semantics, each element of the language syntax is assigned a relationship to an abstract mathematical entity (its denotation). The *possible worlds* of Kripke [15] is an example of such semantics. Finally, there are a specific type of denotational semantics, the *game-theoretic* semantics, where each well-formed statement of the language is associated with a conceptual game between two players, a protagonist and an antagonist. A statement is considered to be true if there is a winning strategy for the protagonist in the associated game (a rule giving that player moves such that executing them guarantees the player can win the game, no matter what moves are made by the antagonist).

Game theoretic semantics are usually applied to abstract argumentation frameworks where the strategies of agents determine which argument(s) they will reveal in each argumentation step. However, they assume the existence of a pre-defined utility function about the payoff that an agent gets for the fact of winning the dialogue or having accepted more or less arguments. Finally, game theory assumes complete knowledge of the space of arguments proposed in the argumentation framework. This assumption is unrealistic in an argumentation dialogue between heterogeneous agents which have individual and private knowledge resources to generate arguments.

7 Conclusions

This report has presented the dialogue game protocol that agents of a case-based argumentation framework can use to interact and engage in argumentation dialogues. First, the syntax of the protocol has been detailed by defining its locutions, commencement rules, rules for the combination of locutions, commitment rules, rules for the speaker order and termination rules.

Finally, the *axiomatic* semantics and the *operational* semantics of the locutions are defined. The former specifies the pre-conditions that should be met to put forward each locution (or set of locutions) and the post-conditions that apply before their utterance. The latter views each locution as a transition in an abstract state-machine that represents the possible stages that can be reached during the dialogue.

Funding

This work is supported by the Spanish government grants [CONSOLIDER-INGENIO 2010 CSD2007-00022, TIN2008-04446, and TIN2009-13839-C03-01] and by the GVA project [PROMETEO 2008/051].

References

[1] L. Amgoud, N. Maudet, and S. Parsons. Modelling dialogues using argumentation. In *4th International Conference on MultiAgent Systems, ICMAS-00*. IEEE Press, 2000.

[2] K. Atkinson. A dialogue game protocol for multi-agent argument over proposals for action. *Autonomous Agents and Multi-Agent Systems. Special issue on Argumentation in Multi-Agent Systems*, 11(2):153–171, 2005.

[3] K. Atkinson. *What Should We Do?: Computational Representation of Persuasive Argument in Practical Reasoning*. PhD thesis, Liverpool University, 2005.

[4] F. Baader, I. Horrocks, and U. Sattler. *Handbook of Knowledge Representation*, chapter Description Logics, pages 135–179. Elsevier, 2007.

[5] T. Bench-Capon and G. Sartor. A model of legal reasoning with cases incorporating theories and values. *Artificial Intelligence*, 150(1-2):97–143, 2003.

[6] T. J. Bench-Capon. Specification and implementation of toulmin dialogue game. In *International Conferences on Legal Knowledge and Information Systems, JURIX-98*, Frontiers of Artificial Intelligence and Applications, pages 5–20. IOS Press, 1998.

[7] C. Carrascosa and M. Rebollo. Modelling agreement spaces. In *IBERAMIA 2008 Workshop on Agreement Technologies, WAT-08*, pages 79–88, 2008.

[8] C. Carrascosa and M. Rebollo. Agreement spaces for counselor agents. In *8th International Conference on Autonomous Agents and Multiagent Systems, AAMAS-09*, pages 1205–1206. ACM Press, 2009.

[9] F. Dignum and H. Weigand. Communication and Deontic Logic. In R. Wieringa and R. Feenstra, editors, *Information Systems - Correctness and Reusability. Selected papers from the IS-CORE Workshop*, pages 242–260. World Scientific Publishing Co., 1995.

[10] C. L. Hamblin. *Fallacies*. Methuen Co. Ltd., 1970.

[11] S. Heras, V. Botti, and V. Julián. A computational argumentation framework for agent societies. Technical Report http://hdl.handle.net/10251/11034, Universitat Politècnica de València, 2011.

[12] I. Horrocks and P. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. *Journal of Web Semantics*, 1(4):345–357, 2004.

[13] J. Hulstijn. *Dialogue Models for Inquiry and Transaction*. PhD thesis, University of Twente, 2000.

[14] N. C. Karunatillake, N. R. Jennings, I. Rahwan, and P. McBurney. Dialogue Games that Agents Play within a Society. *Artificial Intelligence*, 173(9-10):935–981, 2009.

[15] S. Kripke. A completeness proof in modal logic. *Journal of Symbolic Logic*, 24:1–14, 1959.

[16] M. Luck and P. McBurney. Computing as interaction: agent and agreement technologies. In *IEEE International Conference on Distributed Human-Machine Systems*. IEEE Press, 2008.

[17] J. D. MacKenzie. Question-begging in non-cumulative systems. *Philosophical Logic*, 8(1):117–133, 1978.

[18] N. Maudet and B. Chaib-draa. Commitment-based and dialogue-game based protocols-news trends in agent communication language. *Knowledge Engineering Review*, 17(2):157–179, 2002.

[19] N. Maudet and F. Evrard. A generic framework for dialogue game implementation. In *2nd Workshop on Formal Semantics and Pragmatics of Dialogue*, pages 185–198. University of Twente, 1998.

[20] P. McBurney, R. M. V. Eijk, S. Parsons, and L. Amgoud. A dialogue game protocol for agent purchase negotiations. *Autonomous Agents and Multi-Agent Systems*, 7(3):235–273, 2003.

[21] P. McBurney, D. Hitchcock, and S. Parsons. The eightfold way of deliberation dialogue. *International Journal of Intelligent Systems*, 22(1):95–132, 2007.

[22] P. McBurney and S. Parsons. Representing epistemic uncertainty by means of dialectical argumentation. *Annals of Mathematics and Artificial Intelligence, Special Issue on Representations of Uncertainty*, 32(1-4):125–169, 2001.

[23] P. McBurney and S. Parsons. Dialogue games in multi-agent systems. *Informal Logic. Special Issue on Applications of Argumentation in Computer Science*, 22(3):257–274, 2002.

[24] P. McBurney and S. Parsons. Games that agents play: A formal framework for dialogues between autonomous agents. *Journal of Logic, Language and Information*, 11(3):315–334, 2002.

[25] P. McBurney and S. Parsons. Locutions for argumentation in agent interaction protocols. In *Revised Proceedings of the International Workshop on Agent Communication, AC-04*, volume 3396 of *LNAI*, pages 209–225. Springer, 2004.

[26] P. McBurney and S. Parsons. *Argumentation in Artificial Intelligence*, chapter Dialogue games for agent argumentation, pages 261–280. Springer, 2009.

[27] H. Prakken. Formal systems for persuasion dialogue. *The Knowledge Engineering Review*, 21:163–188, 2006.

[28] H. Prakken. An abstract framework for argumentation with structured arguments. *Argument and Computation*, (1):93–124, 2010.

[29] H. Prakken, C. Reed, and D. Walton. Dialogues about the burden of proof. In *Proceedings of the 10th International Conference on Artificial Intelligence and Law, ICAIL-05*, pages 115–124. ACM Press, 2005.

[30] H. Prakken and G. Sartor. Modelling reasoning with precedents in a formal dialogue game. *Artificial Intelligence and Law*, 6:231–287, 1998.

[31] F. Sadri, F. Toni, and P. Torroni. Dialogues for negotiation: Agent varieties and dialogue sequences. In *Revised Papers from the 8th International Workshop on Intelligent Agents VIII, ATAL-01*, volume 2333, pages 405–421. Springer, 2001.

[32] F. Sadri, F. Toni, and P. Torroni. Logic agents, dialogue, negotiation - an abductive approach. In *Convention of The Society for the Study of Artificial Intelligence and the Simulation of Behaviour, AISB-01*. AISB, 2001.

[33] Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game Theoretic and Logical Foundations*. Cambridge University Press, 2009.

[34] C. Sierra, V. Botti, and S. Ossowski. Agreement Computing. *KI - Künstliche Intelligenz*, DOI: 10.1007/s13218-010-0070-y, 2011.

[35] M. Singh. A social semantics for agent communication languages. volume 1916 of *LNCS*, pages 31–45. Springer, 2000.

[36] R. D. Tennent. *Semantics of Programming Languages*. Prentice Hall, 1991.

[37] R. M. van Eijk. Semantics of Agent Communication: An Introduction. In *Foundations and Applications of Multi-Agent Systems, UKMAS 1996-2000, Selected Papers*, volume 2403 of *LNAI*, pages 152–168. Springer-Verlag, 2002.

[38] D. Walton and E. C. W. Krabbe. *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. State University of New York Press, 1995.

[39] D. Walton, C. Reed, and F. Macagno. *Argumentation Schemes*. Cambridge University Press, 2008.

[40] M. Wardeh, T. Bench-Capon, and F. P. Coenen. PISA - Pooling Information from Several Agents: Multiplayer Argumentation From Experience. In *Proceedings of the 28th SGAI International Conference on Artificial Intelligence, AI-2008*, pages 133–146. Springer, 2008.