A Concurrent Language for Argumentation: Preliminary Notes

Stefano Bistarelli
University of Perugia, Italy
http://www.dmi.unipg.it/bista/
stefano.bistarelli@unipg.it

Carlo Taticchi
Gran Sasso Science Institute, L'Aquila, Italy
carlo.taticchi@gssi.it

Abstract

While agent-based modelling languages naturally implement concurrency, the currently available languages for argumentation do not allow to explicitly model this type of interaction. In this paper we introduce a concurrent language for handling process arguing and communicating using a shared argumentation framework (reminding shared constraint store as in concurrent constraint). We introduce also basic expansions, contraction and revision procedures as main bricks for enforcement, debate, negotiation and persuasion.

2012 ACM Subject Classification Computing methodologies → Knowledge representation and reasoning; Theory of computation → Concurrency; Computing methodologies → Concurrent programming languages

Keywords and phrases Argumentation, Concurrent Language, Debating, Negotiation, Belief Revision

Digital Object Identifier 10.4230/OASIcs.Gabbrielli.2020.9

Funding This work was partially supported by "Argumentation 360" (Ricerca di Base 2017–2019), "RACRA" (Ricerca di Base 2018–2020) and "ASIA" (Social Interaction with Argumentation – GNCS-INDAM).

1 Introduction

Many applications in the field of artificial intelligence aim to reproduce the human behaviour and reasoning in order to allow machines to think and act accordingly. One of the main challenges in this sense is to provide tools for expressing a certain kind of knowledge in a formal way so that the machines can use it for reasoning and infer new information. Argumentation Theory provides formal models for representing and evaluating arguments that interact with each other. Consider, for example, two people arguing about whether lowering taxes is good or not. The first person says that a) lowering taxes would increase productivity; the second person replies with b) a study showed that productivity decrease when taxes are lowered; then, the first person adds c) the study is not reliable since it uses data from unverified sources. The dialogue between the two people is conducted through three main arguments (a,b and c) whose internal structure can be represented through different formalisms [26, 30], and for which we can identify the relations b attacks a and c attacks b. In this paper, we use the representation for Argumentation Frameworks introduced by Dung [18], in which arguments are abstract, that is their internal structure, as well as their origin, is left unspecified. Abstract Argumentation Frameworks (AFs), have been widely studied from the point of view of the acceptability of arguments and, recently, several authors

have investigated the dynamics of AFs, taking into account both theoretical [28, 4, 10] and computational aspects (for example, a special track on dynamics [7] appeared in the Third International Competition on Computational Models of Argumentation[1]).

Logical frameworks for argumentation, like the ones presented in [17, 19], have been introduced to fulfil the operational tasks related to the study of dynamics in AFs, such as the description of AFs, the specification of modifications, and the search for sets of "good" arguments. Although some of these languages could be exploited to implement applications based on argumentation, for instance to model debates among political opponents, none of them consider the possibility of having concurrent interactions or agents arguing with each other. This lack represents a significant gap between the reasoning capacities of AFs and their possible use in real-life tools. As an example, consider the situation in which two debating agents share a knowledge base, represented by an AF, and both of them want to update it with new information, in such a way that the new beliefs are consistent with the previous ones. The agents can act independently and simultaneously. Similarly to what happens in concurrent programming, if no synchronization mechanism is taken into account, the result of update or revision can be unpredictable and can also lead to the introduction of inconsistencies.

Motivated by the above considerations, we introduce a concurrent language for argumentation (CA) that aims to be used also for modelling different types of interaction between agents (as negotiations, persuasion, deliberation and dialogues). In particular, our language allows for modelling concurrent processes, inspired by notions such as the *Ask-and-Tell constraint system* [29], and using AFs as centralised store. The language is thus endowed with primitives for the specification of interaction between agents through the fundamental operations of adding (or removing) and checking arguments and attacks. Besides specifying a logic for argument interaction, our language can model debating agents (e.g., chatbots) that take part in a conversation and provide arguments.

Alchourrón, Gärdenfors, and Makinson (AGM) theory [1] gives operations (like expansion, contraction, revision) for updating and revising beliefs on a knowledge base. We propose a set of AGM-style operations that allow for modifying an AF (which constitutes the shared memory our agents access to communicate) and changing the status of its arguments so as to allow the implementation of more complex operations, like negotiation and the other forms of dialogues.

The rest of this paper is structured as follows: in Section 2 we recall some notions from Argumentation Theory; in Section 3 we define a labelling semantics for AFs upon which the agents build their beliefs; in Section 4 we present the syntax and the operational semantics of our concurrent language, together with some high level operations that realize the interaction between agents; in Section 5 we discuss existing formalisms from the literature that bring together argumentation and multiagent systems, highlighting the contact points and the differences with our work; Section 6 concludes the paper with final remarks and perspectives on future work.

2 Abstract Argumentation Frameworks

In this section, we briefly recall the basic concepts we refer to in our proposal. In particular, we give the fundamental definition for Abstract Argumentation Frameworks, together with the notions of acceptable argument and argumentation semantics.

[1] ICCMA2019 website: `http://iccma2019.dmi.unipg.it`.

Argumentation is an interdisciplinary field that aims to understand and model the human natural fashion of reasoning. In Artificial Intelligence, argumentation theory allows one to deal with uncertainty in non-monotonic (defeasible) reasoning, and it is used to give a qualitative, logical evaluation to sets of interacting arguments, called extensions. In his seminal paper [18], Dung defines the building blocks of abstract argumentation.

▶ **Definition 1** (AFs). *Let U be the set of all possible arguments, which we refer to as the "universe". An Abstract Argumentation Framework is a pair $\langle Arg, R\rangle$ where $Arg \subseteq U$ is a set of arguments and R is a binary relation on Arg representing attacks*[2]*.*

AFs can be represented through directed graphs, that we depict using the standard conventions. For two arguments $a, b \in Arg$, $(a, b) \in R$ represents an attack directed from a against b. Moreover, we say that an argument b is *defended* by a set $B \subseteq Arg$ if and only if, for every argument $a \in Arg$, if $R(a, b)$ then there is some $c \in B$ such that $R(c, a)$.

The goal is to establish which are the acceptable arguments according to a certain semantics, namely a selection criterion. Non-accepted arguments are rejected. Different kinds of semantics have been introduced [18, 2] that reflect qualities which are likely to be desirable for "good" subsets of arguments. We first give the definition for the extension-based semantics (also referred to as Dung semantics), namely admissible, complete, stable, preferred, and grounded semantics (denoted with *adm*, *com*, *stb*, *prf* and *gde*, respectively, and generically with σ).

▶ **Definition 2** (Extension-based semantics). *Let $F = \langle Arg, R\rangle$ be an AF. A set $E \subseteq Arg$ is conflict-free in F, denoted $E \in S_{cf}(F)$, if and only if there are no $a, b \in E$ such that $(a, b) \in R$. For $E \in S_{cf}(F)$ we have that:*

- *$E \in S_{adm}(F)$ if each $a \in E$ is defended by E;*
- *$E \in S_{com}(F)$ if $E \in S_{adm}(F)$ and $\forall a \in Arg$ defended by E, $a \in E$;*
- *$E \in S_{stb}(F)$ if $\forall a \in Arg \setminus E$, $\exists b \in E$ such that $(b, a) \in R$;*
- *$E \in S_{prf}(F)$ if $E \in S_{adm}(F)$ and $\nexists E' \in S_{adm}(F)$ such that $E \subset E'$;*
- *$E \in S_{gde}(F)$ if $E \in S_{com}(F)$ and $\nexists E' \in S_{com}(F)$ such that $E' \subset E$.*

Moreover, if E satisfies one of the above semantics, we say that E is an extension for that semantics (for example, if $E \in S_{adm}(F)$ we say that E is an admissible extension).

The different semantics described in Definition 2 corresponds to different styles of reasoning, each of which may be more appropriate for being applied to a particular application domain. The characterisation of the reasoning requirements for the various domains is still a largely open research problem [3] and can only be based on general criteria rather than on specific cases. The stable semantics can be considered the strongest one: the accepted arguments attack all the others in the framework. Since a stable extension may not exist, the preferred semantics can be used as a valid alternative. The preferred semantics, in turn, does not have a unique extension, making the grounded semantics (that always exists and admits exactly one solution) an overall good option for establishing which arguments have to be accepted.

A partial order can be defined among the set of extensions for the different semantics. In detail, we know that $S_{stb}(F) \subseteq S_{prf}(F) \subseteq S_{com}(F) \subseteq S_{adm}(F) \subseteq S_{cf}(F)$ and $S_{gde}(F) \subseteq S_{com}(F)$. Besides enumerating the extensions for a certain semantics σ, one of the most

[2] We introduce both U and $Arg \subseteq U$ (not present in the original definition by Dung) for our convenience, since in the concurrent language that we will define in Section 4 we use an operator to dynamically add arguments from U to Arg.

common tasks performed on AFs is to decide whether an argument a is accepted in some extension of $S_\sigma(F)$ or in all extensions of $S_\sigma(F)$. In the former case, we say that a is *credulously* accepted with respect to σ; in the latter, a is instead *sceptically* accepted with respect to σ. The grounded semantics, in particular, coincides with the set of arguments sceptically accepted by the complete ones.

▶ **Example 3.** In Figure 1 we provide an example of AF where sets of extensions are given for all the mentioned semantics[3]. We discuss some details: the singleton $\{e\}$ is not conflict-free because e attacks itself. The argument b is not contained in any admissible extension because no other argument (included itself) defends b from the attack of a. The empty set $\{\}$, and the singletons $\{c\}$ and $\{d\}$ are not complete extensions because a, which is not attacked by any other argument, has to be contained in all complete extensions. Only the maximal (with respect to set inclusion) admissible extensions $\{a, c\}$ and $\{a, d\}$ are preferred, while the minimal complete $\{a\}$ is the (unique) grounded extension. Then, the arguments in the subset $\{a, d\}$, that conduct attacks against all the other arguments (namely b, d and e), represent a stable extension. To conclude the example, we want to point out that argument a is sceptically accepted with respect to the complete semantics, since it appears in all three subsets of $S_{com}(F)$. On the other hand, argument c, that is in just one complete extension, is credulously accepted with respect to the complete semantics.

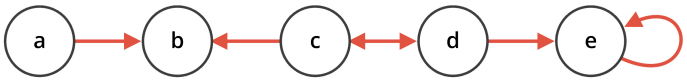

Figure 1 An argumentation framework F for which we compute the following sets of extensions: $S_{cf}(F) = \{\{\},\{a\},\{b\},\{c\},\{d\},\{a,c\},\{a,d\},\{b,d\}\}$, $S_{adm}(F) = \{\{\},\{a\},\{c\},\{d\},\{a,c\},\{a,d\}\}$, $S_{com}(F) = \{\{a\},\{a,c\},\{a,d\}\}$, $S_{prf}(F) = \{\{a,c\},\{a,d\}\}$, $S_{stb}(F) = \{\{a,d\}\}$, and $S_{gde}(F) = \{\{a\}\}$.

Many of the above-mentioned semantics (such as the admissible and the complete ones) exploit the notion of defence in order to decide whether an argument is part of an extension or not. The phenomenon for which an argument is accepted in some extension because it is defended by another argument belonging to that extension is known as *reinstatement* [11]. In that paper, Caminada also gives a definition for a reinstatement labelling.

▶ **Definition 4** (Reinstatement labelling). *Let $F = \langle Arg, R\rangle$ be an AF and $\mathbb{L} = \{in, out, undec\}$. A labelling of F is a total function $L : Arg \to \mathbb{L}$. We define $in(L) = \{a \in Arg \mid L(a) = in\}$, $out(L) = \{a \in Arg \mid L(a) = out\}$ and $undec(L) = \{a \in Arg \mid L(a) = undec\}$. We say that L is a reinstatement labelling if and only if it satisfies the following:*

- *$\forall a, b \in Arg$, if $a \in in(L)$ and $(b, a) \in R$ then $b \in out(L)$;*
- *$\forall a \in Arg$, if $a \in out(L)$ then $\exists b \in Arg$ such that $b \in in(L)$ and $(b, a) \in R$.*

In other words, an argument is labelled *in* if all its attackers are labelled *out*, and it is labelled *out* if at least an *in* node attacks it. In all other cases, the argument is labelled *undec*. A labelling-based semantics [2] associates with an AF a subset of all the possible labellings. In Figure 2 we show an example of reinstatement labelling on an AF. Moreover, there exists a connection between reinstatement labellings and the Dung-style semantics.

[3] The examples are made using the ConArg suite [8]. Web interface: `http://www.dmi.unipg.it/conarg`.

This connection is summarised in Table 1: the set of *in* arguments in any reinstatement labelling constitutes a complete extension; then, if no argument is *undec*, the reinstatement labelling provides a stable extension; if the set of *in* arguments (or the set of *out* arguments) is maximal with respect to all the possible labellings, we obtain a preferred extension; finally the grounded extension is identified by labellings where either the set of *undec* arguments is maximal, or the set of *in* (respectively *out*) arguments is maximal.

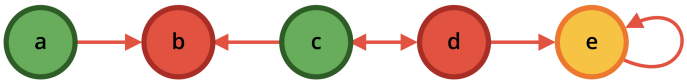

Figure 2 an example of AF in which reinstatement labelling is showed by using colours. Arguments a and c highlighted in green are *in*, red ones (b and d) are *out*, and the the yellow argument e (that attacks itself) is *undec*.

Table 1 Reinstatement labelling vs semantics.

Labelling restrictions		Semantics
no restrictions		complete
empty	*undec*	stable
maximal	*in*	preferred
maximal	*out*	preferred
maximal	*undec*	grounded
minimal	*in*	grounded
minimal	*out*	grounded

Reinstatement labelling allows to inspect AFs on a finer grain than Dung's extensions, since the *undec* label identifies arguments that are not acceptable, but still not directly defeated by accepted arguments. However, the information brought by the *undec* label can be misleading. Consider for example an AF in which two arguments a and b are attacking each other (Figure 3, left). A possible labelling for such a framework would label both arguments as *undec*. Indeed, we cannot decide whether, in general, it is worth accepting a (or b). Consider now a second AF composed of two arguments c and d where only c attacks d and both arguments are labelled as *undec* (Figure 3, right). At this point, one could conclude that it is not possible to univocally establish whether c is a good argument or not, similarly to what happens in the previous example. However, in this case the fact of c being *undec* does not depend on the structure of the framework, but rather on the choice of just ignoring it.

Figure 3 Two AFs where all arguments are labelled *undec*. The one on the left has two undistinguishable arguments a and b, while argument c of the AF on the right is arguably better than d, from the point of view of acceptability.

Ambiguity of the *undec* label is solved in the four-state labelling introduced by [22], where arguments that are assigned the label *in* are accepted, those that are assigned the label *out* are rejected, those that are assigned both *in* and *out* (which we denote as *undec*) are neither fully accepted nor fully rejected, and those that are not considered at all are assigned the empty set $\emptyset$. The labelling of [22] is defined as follow.

▶ **Definition 5** (Four-state labelling). *A four-state labelling consists of a total mapping* $L : Arg \to 2^{\{in,out\}}$ *that satisfies the following conditions:*

- $\forall a \in Arg$, *if* $out \in L(a)$, *then* $\exists b \in Arg$ *such that* $(b, a) \in R$ *and* $in \in L(b)$;
- $\forall a \in Arg$, *if* $in \in L(a)$, *then* $\forall b \in Arg$ *such that* $(b, a) \in R$, $out \in L(b)$;
- $\forall a \in Arg$, *if* $in \in L(a)$, *then* $\forall c$ *such that* $(a, c) \in R$, $out \in L(c)$.

A four-state labelling is said to be total[4] *if and only if* $\forall a \in Arg$, $L(a) \neq \emptyset$. *A labelling which is not total is called partial. Moreover, the four labels form the lattice of Figure 4, in which undec (that is the set* $\{in, out\}$*) is the top element and* $\emptyset$ *is the bottom.*

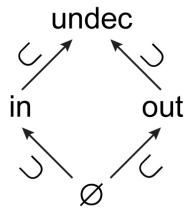

Figure 4 Lattice of labels in the four-state labelling.

We show an example of labelling in Figure 5, where all four labels are used. Note that the arguments labelled *in* and *out* in the figure do not satisfy the condition of the reinstatement labelling. Even though the labelling of Definition 5 is more informative than

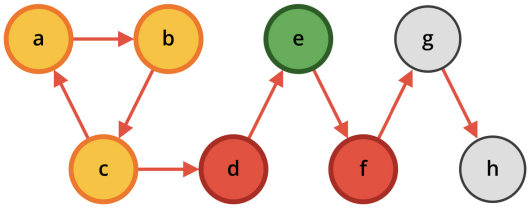

Figure 5 Labelling of an AF showed through colours. Argument e, highlighted in green, is the only *in*; red arguments d and f are *out*; those in yellow, i.e., a, b and c, are *undec*; and the grey arguments g and h are left with an empty label $\emptyset$.

the reinstatement labelling of Definition 4 (that does not comprehend an empty label), there is no direct connection between labellings and extensions of a certain semantics, as it happens for the reinstatement labelling.

3 A Four-state Labelling Semantics

We showed in the previous section that both reinstatement and four-state labellings have both pros and cons. The labelling by Caminada does not allow to leave unlabelled arguments that we do not want to consider in computing acceptability and forces all arguments that are

[4] The total labelling is called "complete" in the original definition [22]. We changed it to avoid ambiguity with the complete semantics.

neither *in* nor *out* to be labelled *undec*. On the other hand, the set of arguments labelled *in* by the reinstatement labelling showed in Definition 4 always correspond to a complete extension and some other semantics can be obtained by applying restrictions on the labelling itself (see Table 1), while four-state labelling does not necessarily correspond to any particular extension. To overcome this problem, in the following we establish a mapping between a modified four-state labelling and the classical semantics of Definition 2.

The labelling of an AF gives information about the acceptability of the arguments in the framework (according to the various Dung's semantics) and can be used by intelligent agents to represent the state of their beliefs. Each different label can be traced to a particular meaning. $\emptyset$ stands for "don't care" [22] and identifies arguments that are not considered by the agents. For instance, arguments in $U \setminus Arg$, that are only part of the universe, but not of the shared AF, are labelled with $\emptyset$ since they are outside the interest of the agents. Accepted and rejected arguments (labelled as *in* and *out*, respectively), allow agents to discern true beliefs from the false ones. At last, *undec* arguments possess both *in* and *out* labels, meaning that agents cannot decide about the acceptability of a belief ("don't know", indeed).

▶ **Definition 6** (Four-state labelling semantics). *Let U be a universe of arguments, $F = \langle Arg, R\rangle$ an AF with $Arg \subseteq U$ and $R \subseteq Arg \times Arg$ the arguments and attacks. L is a four-state labelling on F if and only if*

- $\forall a \in U \setminus Arg. L(a) = \emptyset$;
- $\forall a \in Arg$, *if* $out \in L(a)$, *then* $\exists b \in Arg$ *such that* $(b, a) \in R$ *and* $in \in L(b)$;
- $\forall a \in Arg$, *if* $in \in L(a)$, *then* $\forall b \in Arg$ *such that* $(b, a) \in R$, $out \in L(b)$;
- $\forall a \in Arg$, *if* $in \in L(a)$, *then* $\forall c$ *such that* $(a, c) \in R$, $out \in L(c)$.

Moreover,

- *L is a conflict-free labelling if and only if:*
  - $L(a) = \{in\} \implies \forall b \in Arg \mid (b, a) \in R. L(b) \neq \{in\}$ *and*
  - $L(a) = \{out\} \implies \exists b \in Arg \mid (b, a) \in R \wedge L(b) = \{in\}$
- *L is an admissible labelling if and only if:*
  - $L(a) = \{in\} \implies \forall b \in Arg \mid (b, a) \in R. L(b) = \{out\}$ *and*
  - $L(a) = \{out\} \implies \exists b \in Arg \mid (b, a) \in R \wedge L(b) = \{in\}$
- *L is a complete labelling if and only if:*
  - $L(a) = \{in\} \iff \forall b \in Arg \mid (b, a) \in R. L(b) = \{out\}$ *and*
  - $L(a) = \{out\} \iff \exists b \in Arg \mid (b, a) \in R \wedge L(b) = \{in\}$
- *L is a stable labelling if and only if:*
  - *L is a complete labelling and*
  - $\nexists a \in Arg \mid L(a) = \{in, out\}$
- *L is a preferred labelling if and only if:*
  - *L is an admissible labelling and*
  - $\{a \mid L(a) = \{in\}\}$ *is maximal among all the admissible labellings*
- *L is a grounded labelling if and only if:*
  - *L is a complete labelling and*
  - $\{a \mid L(a) = \{in\}\}$ *is minimal among all the complete labellings*

We can show there is a correspondence between labellings satisfying the restrictions given in the definition above and the extensions of a certain semantics. We use the notation $L \in S_\sigma(F)$ to identify a labelling L corresponding to an extension of the semantics σ with respect to the AF F.

▶ **Theorem 7.** *A four-state labelling L of an AF $F = \langle Arg, R\rangle$ is a conflict-free (respectively admissible, complete, stable, preferred, grounded) labelling as in Definition 6 if and only if the set I of arguments labelled in by L is a conflict-free (respectively admissible, complete, stable, preferred, grounded) extension of F.*

Proof. We sketch the proof for the admissible labelling. The conflict-free case is obtained through a similar reasoning and the remaining can be constructed as in [12].

$\Rightarrow$) Consider an admissible labelling L on $F = \langle Arg, R\rangle$. We have to show that there are no $a, b \in I$ such that $(a, b) \in R$ and that each $a \in I$ is defended by I. First of all, arguments labelled *in* by L can only be attacked by *out* arguments, so for all $a, b \in I$ we have $(a, b) \notin R$. Then, if a is attacked by an argument b (which we know must be *out*) that argument is necessarily in turn attacked by at least one *in*. We conclude that I defends all its elements and therefore it is an admissible extension.

$\Leftarrow$) We have an admissible extension E composed of arguments labelled *in* by L, and we know that all arguments in E does not attack each other and are defended by E. Hence, *in* arguments of L cannot be attacked by other arguments with the label *in*. Finally, arguments that are attacked from E are *out*. ◀

In the next session, where we present our concurrent language for argumentation, the labelling of Definition 6 is used to implement both primitives and high level operations that rely on the acceptability state of agent's belief and are able to change the underlying knowledge base accordingly.

4 The Language

Agents/processes in a distributed/concurrent system can perform operations that affect the behaviour of other components. The indeterminacy in the execution order of the processes may lead to inconsistent results for the computation or even cause errors that prevent particular tasks from being completed. We refer to this kind of situation as a *race condition*. If not properly handled, race conditions can cause loss of information, resource starvation and deadlock. In order to understand the behaviour of agents and devise solutions that guarantee correct executions, many formalisms have been proposed for modelling concurrent systems. Concurrent Constraint Programming (CC) [29], in particular, relies on a constraint store of shared variables in which agents can read and write in accordance with some properties posed on the variables. The basic operations that can be executed by agents in the CC framework are a blocking *Ask* and an atomic *Tell*. These operations realise the interaction with the store and also allow one to deal with partial information.

Starting from the CC syntax, we enrich the ask and tell operators in order to handle the interaction with an AF used as knowledge base for the agents. We replace the ask with three decisional operations: a syntactic *check* that verifies if a given set of arguments and attacks is contained in the knowledge base, and two semantic *test* operations that we use to retrieve information about the acceptability of arguments in an AF. The tell operation (that we call *add*) augments the store with additional arguments and attack relations. We can also remove parts of the knowledge base through a specifically designed removal operation. Finally, a guarded parallel composition $\|_G$ allows for executing all the operations that satisfy some given conditions, and a prioritised operator $+_P$ is used to implement if-then-else constructs. The syntax of our concurrent language for argumentation is presented in Table 2, while in Table 3 we give the definitions for the transition rules.

Suppose to have an agent A whose knowledge base is represented by an AF $F = \langle Arg, R\rangle$. An $add(Arg', R')$ action performed by the agent results in the addition of a set of arguments $Arg' \subseteq U$ (where U is the universe) and a set of relations R' to the AF F. When performing

Table 2 CA syntax.

$$
\begin{aligned}
P &::= C.A \\
C &::= p(a,l,\sigma) :: A \mid C.C \\
A &::= success \mid add(Arg,R) \to A \mid rmv(Arg,R) \to A \mid E \mid A \| A \mid \exists_x A \mid p(a,l,\sigma) \\
E &::= test_c(a,l,\sigma) \to A \mid test_s(a,l,\sigma) \to A \mid check(Arg,R) \to A \\
&\quad \mid E + E \mid E +_P E \mid E \|_G E
\end{aligned}
$$

an Addition, (possibly) new arguments are taken from $U \setminus Arg$. We want to make clear that the tuple (Arg', R') is not an AF, indeed it is possible to have $Arg' = \emptyset$ and $R' \neq \emptyset$, which allows to perform an addition of only attack relations to the considered AF. It is as well possible to add only arguments to F, or both arguments and attacks. Intuitively, $rmv(Arg, R)$ allows to specify arguments and/or attacks to remove from the knowledge base. Removing an argument from an AF requires to also remove the attack relations involving that argument and trying to remove an argument (or an attack) which does not exist in F will have no consequences. The operation $check(Arg', R')$ is used to verify whether the specified arguments and attack relations are contained in the set of arguments and attacks of the knowledge base, without introducing any further change. If the check is positive, the operation succeeds, otherwise it suspends. We have two distinct test operations, both requiring the specification of an argument $a \in A$, a label $l \in \{in, out, undec, \emptyset\}$ and a semantics $\sigma \in \{adm, com, stb, prf, gde\}$. The credulous $test_c(a, l, \sigma)$ succeeds if there exists at least an extension of $S_\sigma(F)$ whose corresponding labelling L is such that $L(a) = l$; otherwise (in the case $L(a) \neq l$ in all labellings) it suspends. The sceptical $test_s(a, l, \sigma)$ succeeds[5] if a is labelled l in all possible labellings $L \in S_\sigma(F)$; otherwise (in the case $L(a) \neq L$ in some labellings) it suspends. The guarded parallelism $\|_G$ is designed to execute all the operations for which the guard in the inner expression is satisfied. More in detail, $E_1 \|_G E_2$ is successful when either E_1, E_2 or both are successful and all the operations that can be executed are executed. This behaviour is different both from classical parallelism (for which all the agents have to terminate in order for the procedure to succeed) and from nondeterminism (that only selects one branch). The operator $+_P$ is left-associative and realises an if-then-else construct: if we have $E_1 +_P E_2$ and E_1 is successful, than E_1 will be always chosen over E_2, even if also E_2 is successful, so in order for E_2 to be selected, it has to be the only one that succeeds. Differently from nondeterminism, $+_P$ prioritises the execution of a branch when both E_1 and E_2 can be executed. Moreover, an if-then-else construct cannot be obtained starting from nondeterminism since of our language is not expressive enough to capture success or failure conditions of each branch.

The remaining operators are classical concurrency compositions: an agent in a parallel composition obtained through $\|$ succeeds if all the agents succeed; any agent composed through $+$ is chosen if its guards succeeds; the existential quantifier $\exists_x A$ behaves like agent A where variables in x are local to A[6]. The parallel composition operator enables the specification of complex concurrent argumentation processes. For example, a debate

[5] The set of extensions $S_\sigma(F)$ is finite, thus both $test_c(a, l, \sigma)$ and $test_s(a, l, \sigma)$ are decidable.
[6] We plan to use existential quantifiers to extend our work by allowing our agents to have local stores.

Table 3 CA operational semantics.

$\langle add(Arg', R') \to A, \langle Arg, R\rangle\rangle \longrightarrow \langle A, \langle Arg \cup Arg', R \cup R'\rangle\rangle$	Addition
$\langle rmv(Arg', R') \to A, \langle Arg, R\rangle\rangle \longrightarrow \langle A, \langle Arg \setminus Arg', R \setminus \{R' \cup R''\}\rangle\rangle$ where $R'' = \{(a,b) \in R \mid a \in Arg' \vee b \in Arg'\}$	Removal
$\dfrac{Arg' \subseteq Arg \wedge R' \subseteq R}{\langle check(Arg', R') \to A, \langle Arg, R\rangle\rangle \longrightarrow \langle A, \langle Arg, R\rangle\rangle}$	Check
$\dfrac{\exists L \in S_\sigma(F) \mid l \in L(a)}{\langle test_c(a, l, \sigma) \to A, F\rangle \longrightarrow \langle A, F\rangle}$	Credulous Test
$\dfrac{\forall L \in S_\sigma(F).l \in L(a)}{\langle test_s(a, l, \sigma) \to A, F\rangle \longrightarrow \langle A, F\rangle}$	Sceptical Test
$\dfrac{\langle A_1, F\rangle \longrightarrow \langle A_1', F'\rangle}{\langle A_1 \| A_2, F\rangle \longrightarrow \langle A_1' \| A_2, F'\rangle}$ $\langle A_2 \| A_1, F\rangle \longrightarrow \langle A_2 \| A_1', F'\rangle$ $\quad$ $\dfrac{\langle A_1, F\rangle \longrightarrow \langle success, F'\rangle}{\langle A_1 \| A_2, F\rangle \longrightarrow \langle A_2, F'\rangle}$ $\langle A_2 \| A_1, F\rangle \longrightarrow \langle A_2, F'\rangle$	Parallelism
$\dfrac{\langle E_1, F\rangle \longrightarrow \langle A_1, F\rangle, \langle E_2, F\rangle \not\longrightarrow}{\langle E_1 \|_G E_2, F\rangle \longrightarrow \langle A_1, F\rangle}$ $\langle E_2 \|_G E_1, F\rangle \longrightarrow \langle A_1, F\rangle$	Guarded Parallelism (1)
$\dfrac{\langle E_1, F\rangle \longrightarrow \langle A_1, F\rangle, \langle E_2, F\rangle \longrightarrow \langle A_2, F\rangle}{\langle E_1 \|_G E_2, F\rangle \longrightarrow \langle A_1 \| A_2, F\rangle}$	Guarded Parallelism (2)
$\dfrac{\langle E_1, F\rangle \longrightarrow \langle A_1, F\rangle}{\langle E_1 + E_2, F\rangle \longrightarrow \langle A_1, F\rangle}$ $\langle E_2 + E_1, F\rangle \longrightarrow \langle A_1, F\rangle$	Nondeterminism
$\dfrac{\langle E_1, F\rangle \longrightarrow \langle A_1, F\rangle}{\langle E_1 +_P E_2, F\rangle \longrightarrow \langle E_1, F\rangle}$	If Then Else (1)
$\dfrac{\langle E_1, F\rangle \not\longrightarrow, \langle E_2, F\rangle \longrightarrow \langle A_2, F\rangle}{\langle E_1 +_P E_2, F\rangle \longrightarrow \langle E_2, F\rangle}$	If Then Else (2)
$\dfrac{\langle A[y/x], F\rangle \longrightarrow \langle A', F'\rangle}{\langle \exists_x A, F\rangle \longrightarrow \langle A', F'\rangle}$ with y fresh	Hidden Variables
$\langle p(b, m, \gamma), F\rangle \longrightarrow \langle A[b/a, m/l, \gamma/\sigma], F\rangle$ when $p(a, l, \sigma) :: A$	Procedure Call

involving many agents that asynchronously provide arguments can be modelled as a parallel composition of add operations performed on the knowledge base. Concluding, P is the class of programs, and the procedure call C has three parameters that allow the implementation of operators which takes into account an argument, a label and a semantics. Below, we give an example of a CA program.

▶ **Example 8.** Consider the AF in Figure 6 (left), where the complete semantics is the set $\{\{a\},\{a,e\},\{a,d\}\}$ and the preferred coincides with $\{\{a,d\},\{a,e\}\}$. An agent A wants to perform the following operation: if argument d is labelled *out* in all complete extensions, then remove the argument c from the knowledge base. At the same time, an agent B wants to add an argument f attacking d only if e is labelled *in* in some preferred extension. If A is the first agent to be executed, the sceptical test on argument d will suspend, since d belongs to the complete extension $\{a,d\}$. The credulous test performed by agent B, instead, is successful and so it can proceed to add an argument f that defeats d. Now d is sceptically rejected by the complete semantics and agent A can finally remove the argument c. After the execution of the program below, we obtain the AF of Figure 6 (right).

$$A : test_s(d, out, com) \rightarrow rmv(\{c\},\{(a,c)\}) \rightarrow success$$
$$B : test_c(e, in, prf) \rightarrow add(\{f\},\{(f,d)\}) \rightarrow success$$
$$P : A \| B$$

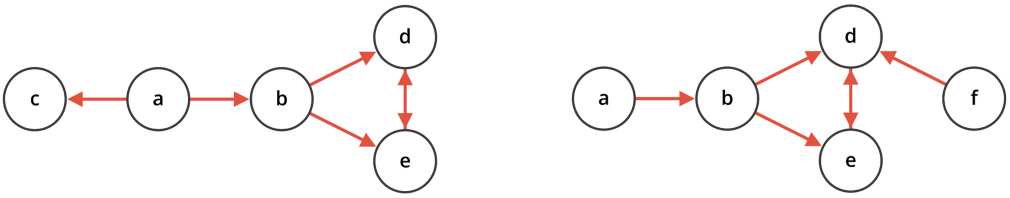

Figure 6 The AF on the right is obtained starting from the one on the left trough the addition of an argument f attacking d and the removal of c together with the attack (a,c).

As we will see in the next session, we aim to use the operators of our language to model the behaviour of agents involved in particular argumentative processes (such as persuasion and negotiation). Note that the language is very permissive: there are no constraints on which arguments or attacks an agent can add/remove. Future work include the partition of arguments and attack with respect to the owner's capabilities and restrict permissions on legal moves.

4.1 Belief Revision and the AGM Framework

Interaction between agents can be modelled in different ways, according to the purposes of the communication. Negotiating agents need to find a common agreement that is beneficial to all, while, for instance, an agent with the goal of persuading its opponents has to both defend its position from the attacks of the other agents and defeat all the arguments against its proposal. The operations needed for the implementation of such kind of interactions must be able to modify the knowledge base shared between the communicating parts so as to model the behaviour of the agents. In particular, usually agents interact modifying part of the shared AF, trying to change the state of acceptance of an argument, often alternating with other agents or concurrently performing syntactic changes to the AF.

The AGM framework [1] provides an approach to the problem of revising knowledge basis by using theories (deductively closed sets of formulae) to represent the beliefs of the agents. A formula α in a given theory can have different statuses for an agent, according to its knowledge base K. If the agent can deduce α from its beliefs, then we say that α is *accepted* ($K \vdash \alpha$). Such a deduction corresponds with the entailment of α by the knowledge base. If the agent can deduce the negation of α, then we say that α is *rejected* ($K \vdash \neg\alpha$). Otherwise, the agent cannot deduce anything and α is *undetermined*. The correspondence between accepted/rejected beliefs and *in*/*out* arguments in a labelling is straightforward. Since the undetermined status represents the absence of a piece of information (nothing can be deduced in favour of either accepting or rejecting a belief) it can be mapped into the empty label $\emptyset$. Finally, the *undec* label is assigned to arguments that are both *in* and *out*, boiling down to the notion of inconsistency in AGM. The empty label, in particular, plays a fundamental role in identifying new arguments that agents can bring to the debate to defend (or strengthen) their position. The status of a belief can be changed through some operations (namely expansion $\oplus$, contraction $\oslash$ and revision $\circledast$) on the knowledge base, as depicted in Figure 7 (notice the similarity with the lattice in Figure 4).

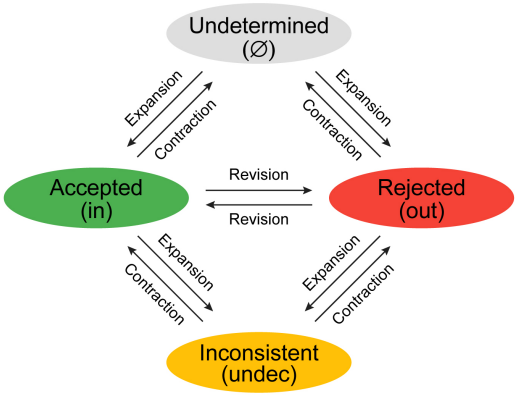

Figure 7 Transitions between AGM beliefs states.

An expansion basically brings new pieces of information to the base, allowing for undetermined belief to become either accepted or refused. A contraction, on the contrary, reduces the information an agent can rely on in making its deduction, and an accepted (or refused) belief can become undetermined. A revision introduces conflicting information, making acceptable belief refused and vice-versa. The AGM framework also defines three sets of rationality postulates (one for each operation) that any good operator should satisfy. To give an example, if we want to add a new belief on a knowledge base, then we expect that no other information in the base is removed. AGM operators provide building blocks for realizing complex interaction processes between agents. Below, we provide some examples:

- **Negotiation** is a process that aims to solve conflicts arising from the interaction between two or more parties that have different individual goals (for instance, a request of computational resources in a distributed network), and its outcome is an agreement that translates in common benefits for all participants. Expansion, here, can be used to model the behaviour of an agent presenting claims towards its counterparts, while contraction represents the act of retracting a condition to successfully conclude the negotiation.

- Contrary to negotiation, a **debate** takes place when the goal of the agents in the system is to promote their own point of view and thus "convince" the others about a conclusion or a statement. A debate [21] can be considered as a mechanism through which a decision maker extracts information from two (or more) counterparts, each of them holding different positions with respect to the right choice. In a multi-agent system, a debate is a process carried out as the interaction between more parties, each of them trying to provide arguments strong enough to support their own conclusion. In this case, agents can make their beliefs accepted in different ways, exploiting AGM operators: inconsistent beliefs can be made accepted through a contraction, while expansion can make beliefs which state is undetermined acceptable.
- The notion of **persuasion** in dialogue games [25] aims to solve conflicts of points of view between two counterparts. In order to persuade the opponent, an agent has to defend its position by replying to every attack against its initial claim. If it fails, the opponent wins the game. Agents involved in this kind of persuasive dialogue games have to elaborate strategies [23], for supporting their beliefs and defeating the adversaries, that consist in a sequence of actions to perform in the system. Again, revision operations on the knowledge base are responsible for changing the status of the beliefs of a persuaded agent.

As for knowledge basis in belief revision, AFs can undergo changes that modify the structure of the framework itself, either integrating new information (and so increasing the arguments and the attacks in the AF) or discarding previously available knowledge. Agents using AFs as the mean for exchanging and inferring information has to rely on operations able to modify such AFs. Besides considering the mere structural changes, also modifications on the semantics level need to be addressed by the operations performed by the agents. In the following, we define three operators for AFs, namely *argument expansion*, *contraction* and *revision*, that comply with classical operators of AGM and that can be built as procedures in our language.

The argumentation frameworks $\langle Arg, R\rangle$ we use as the knowledge base for our concurrent agents are endowed with a universe of arguments U that are used to bring new information. Since arguments in $U \setminus Arg$ do not constitute an actual part of the knowledge base, they are always labelled $\emptyset$, until they are added into the framework and acquire an *in* and/or an *out* label. Notice also that changes to the knowledge base we are interested in modelling are restricted to a single argument at a time, miming the typical argument interaction in dynamic AF.

▶ **Definition 9** (Argument extension expansion, contraction, revision). *Let $F = \langle Arg, R\rangle$ be an AF on the universe U, $Arg \subseteq U$, $R \subseteq Arg \times Arg$, σ a semantics, $L \in S_\sigma(F)$ a given labelling, and $a \in U$ an argument.*

- *An argument extension expansion $\oplus^\sigma_{a,L} : AF \to AF$ computes a new AF $F' = \oplus^\sigma_{a,L}(F)$ with semantics $S_\sigma(F')$ for which $\exists L' \in S_\sigma(F')$ such that $L'(a) \supseteq L(a)$ (if $L'(a) \supset L(a)$ the expansion is strict).*
- *An argument extension contraction $\oslash^\sigma_{a,L} : AF \to AF$ computes a new AF $F' = \oslash^\sigma_{a,L}(F)$ with semantics $S_\sigma(F')$ for which $\exists L' \in S_\sigma(F')$ such that $L(a) \supseteq L'(a)$ (if $L(a) \supset L'(a)$ the expansion is strict).*
- *An argument extension revision $\circledast^\sigma_{a,L} : AF \to AF$ computes a new AF $F' = \circledast^\sigma_{a,L}(F)$ with semantics $S_\sigma(F')$ for which $\exists L' \in S_\sigma(F')$ such that if $L(a) = in/out$, then $L'(a) = out/in$ and $\forall b \in Arg$ with $b \neq a$, $L'(b) = L(b) \vee L'(b) \neq undec$ (that is no inconsistencies are introduced).*

Moreover, we denote with $\oplus^{\sigma,l}_{a,L}(F)$, $\oslash^{\sigma,l}_{a,L}(F)$ and $\circledast^{\sigma,l}_{a,L}(F)$ an argument extension expansion, contraction and revision, respectively, that computes an AF F' with semantics $S_\sigma(F')$ for which $\exists L' \in S_\sigma(F')$ such that $L'(a) = l$.

When performing an argument extension expansion (or contraction, or revision) for a certain argument a of an AF F, the operators of Definition 9 take into account a single labelling of the semantics σ and there is no control over the other labellings, for which a can have its label arbitrarily changed. For example, an argument extension expansion that increases the number of labels of a with respect to a chosen labelling L, may reduce that number in a different labelling. Therefore, we introduce a further definition that considers all the possible labellings $\mathcal{L}^F_\sigma$ of $S_\sigma(F)$. To compare the various labels an argument can have in different labellings, we refer to the order in Figure 4 and, calling $\mathcal{L}^F_{\sigma\downarrow_a}$ the multi-set of the labels a has in the various $L \in \mathcal{L}^F_\sigma$, we say that $\mathcal{L}^{F'}_{\sigma\downarrow_a} \supseteq \mathcal{L}^F_{\sigma\downarrow_a}$ if there exists an injective function $f : \mathcal{L}^F_\sigma \rightarrowtail \mathcal{L}^{F'}_\sigma$ such that $\forall l \in \mathcal{L}^F_{\sigma\downarrow_a}.l \leq f(l)$. Moreover, we use the notation $\mathcal{L}^F_{\sigma\downarrow_a}|_l$ to restrict to l labels in the multi-set $\mathcal{L}^F_{\sigma\downarrow_a}$, where $l = \{\emptyset, in, out, undec\}$.

▶ **Definition 10** (Argument semantics expansion, contraction, revision). *Let $F = \langle Arg, R\rangle$ be an AF on the universe U, $Arg \subseteq U$, $R \subseteq Arg \times Arg$, σ a semantics, and $a \in U$ an argument.*

- *An argument semantics expansion $\oplus^\sigma_a : AF \to AF$ computes a new AF $F' = \oplus^\sigma_a(F)$ with semantics $S_\sigma(F')$ such that $\mathcal{L}^{F'}_{\sigma\downarrow_a} \supseteq \mathcal{L}^F_{\sigma\downarrow_a}$.*
- *An argument semantics contraction $\oslash^\sigma_a : AF \to AF$ computes a new AF $F' = \oslash^\sigma_a(F)$ with semantics $S_\sigma(F')$ such that $\mathcal{L}^F_{\sigma\downarrow_a} \supseteq \mathcal{L}^{F'}_{\sigma\downarrow_a}$.*
- *An argument semantics revision $\circledast^\sigma_a : AF \to AF$ computes a new AF $F' = \circledast^\sigma_a(F)$ with semantics $S_\sigma(F')$ such that $\forall b \in Arg. \left|\mathcal{L}^F_{\sigma\downarrow_b}|_{undec}\right| \geq \left|\mathcal{L}^{F'}_{\sigma\downarrow_b}|_{undec}\right|$ (that is no inconsistencies are introduced), and:*
  - *in-to-out revision: $\left|\mathcal{L}^F_{\sigma\downarrow_a}|_{out}\right| < \left|\mathcal{L}^{F'}_{\sigma\downarrow_a}|_{out}\right| \wedge \left|\mathcal{L}^F_{\sigma\downarrow_a}|_{in}\right| > \left|\mathcal{L}^{F'}_{\sigma\downarrow_a}|_{in}\right|$;*
  - *out-to-in revision: $\left|\mathcal{L}^F_{\sigma\downarrow_a}|_{in}\right| < \left|\mathcal{L}^{F'}_{\sigma\downarrow_a}|_{in}\right| \wedge \left|\mathcal{L}^F_{\sigma\downarrow_a}|_{out}\right| > \left|\mathcal{L}^{F'}_{\sigma\downarrow_a}|_{out}\right|$;*

It is important to note that the formalism we present is not monotone: the *add* operation may lead to a contraction, reducing the number of arguments with the labels *in* and/or *out*. Similarly, the removal of an argument may lead to an expansion (this is the case of Figure 8).

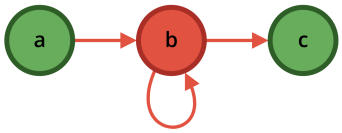

Figure 8 Example of argument extension expansion. Removing the *in* argument a makes both b and c *undec*.

AGM operators have already been studied from the point of view of their implementation in work as [5, 14], especially with regard to enforcement. However, in the previous literature, realisability of extensions and not of single arguments is considered. The implementation of an argument expansion/contraction/revision operator changes according to the semantics we take into account. In the following, we consider the grounded semantics and show how the operators of Definitions 9 can be implemented. For the grounded semantics, that only has one extension, Definitions 9 and 10 coincide. Notice also that there exist many ways to obtain expansion, contraction and revision. We chose one that leverage between minimality with respect to the changes required in the framework and linearity of implementation.

▶ **Proposition 11.** *Let $F = \langle Arg, R\rangle$ be an AF on the universe U, $Arg \subseteq U$, $R \subseteq Arg \times Arg$, $a \in U$ an argument, and L the unique grounded labelling. A possible argument extension expansion $\oplus^{gde,l}_{a,L}(F)$ could act as:*

- *if* $L(a) = \emptyset$ *and* $l = in$*, add* a *to* Arg
- *if* $L(a) = \emptyset$ *and* $l = out$*,*
  - *if* $\exists b \in Arg \mid L(b) = in$*, add* $\langle \{a\}, \{(b,a)\} \rangle$ *to* F
  - *otherwise, add* $\langle \{a,b\}, \{(b,a)\} \rangle$ *to* F
- *if* $L(a) = in$ *and* $l = undec$*,*
  - *if* $\exists b \in Arg \mid L(b) = undec$*, add* (b,a) *to* R
  - *otherwise, add* (a,a) *to* R
- *if* $L(a) = out$ *and* $l = undec$*,*
  - $\forall b \in Arg \mid L(b) = \{in\} \wedge (b,a) \in R$*, add* (a,b) *to* R

Proof. If a has an empty label, it means that $a \in U \setminus Arg$, since the grounded labelling assigns a label different from $\emptyset$ to all arguments in Arg. It is then sufficient to add a to the set of considered arguments Arg to make it *in*. If the freshly added argument is attacked by another *in* argument, it becomes *out*. Continuing, a is labelled *undec* in the grounded labelling only if it is attacked by an *undec* argument (included a itself), thus, to make an *in* argument a become *undec* we can look for an argument b in Arg that is already labelled as *undec*. If we find such a b then it is sufficient to add the attack relation from b to a to the store. Otherwise, we make a attack itself. Finally, if we want an *out* argument a to become *undec*, we make it attack back all its *in* attackers. Doing so, we obtain three distinct complete labellings: one in which a is accepted and its attackers are not, another one in which the opposite situation occurs, and the third labelling in which neither a nor its attackers are fully accepted or rejected (that is they are *undec*). Hence, a will be *undec* in the minimal complete labelling (that, by Definition 6, is also grounded). ◀

▶ **Proposition 12.** *Let* $F = \langle Arg, R \rangle$ *be an AF on the universe* U*,* $Arg \subseteq U$*,* $R \subseteq Arg \times Arg$*,* $a \in U$ *an argument, and* L *the unique grounded labelling. A possible argument extension contraction* $\oslash_{a,L}^{gde,l}(F)$ *could act as:*

- *if* $L(a) = undec$ *and* $l = in$*,* $\forall b \in Arg \mid L(b) = undec$*, remove* (b,a) *from* R
- *if* $L(a) = undec$ *and* $l = out$*,*
  - *if* $\exists b \in Arg \mid L(b) = in$*, add* (b,a) *to* R
  - *otherwise, add* $\langle \{b\}, \{(b,a)\} \rangle$ *to* F
- *if* $L(a) = in$ *and* $l = \emptyset$*, remove* a *(and all attacks involving* a*) from* F
- *if* $L(a) = out$ *and* $l = \emptyset$*, remove* a *(and all attacks involving* a*) from* F

Proof. Consider a grounded labelling. An *undec* argument a can become *in* by removing all attacks coming from *undec* arguments (included a itself). Indeed an argument is *undec* only if it is attacked by another *undec*. Note that a cannot be attacked by *in* arguments, otherwise it would have been *out*. Therefore, after the changes a is only attacked by *out* arguments, and thus is *in*. Alternatively, a can become *out* when it is attacked by another *in* argument b (when the store does not contain *in* arguments, we add one from the universe). If a is either *in* or *out*, instead, we can contract its label to *undec* through the removal of a itself form the store. ◀

▶ **Proposition 13.** *Let* $F = \langle Arg, R \rangle$ *be an AF on the universe* U*,* $Arg \subseteq U$*,* $R \subseteq Arg \times Arg$*,* $a \in U$ *an argument, and* L *the unique grounded labelling. A possible argument extension revision* $\circledast_{a,L}^{gde,l}(F)$ *could act as:*

- *if* $L(a) = in$*,*
  - *if* $\exists b \in Arg \mid L(b) = in$*, add* (b,a) *to* R *and then* $\forall c \in Arg \mid (a,c) \in R$*, add* (b,c) *to* R
  - *otherwise, add* $\langle \{b\}, \{(b,a)\} \rangle$ *to* F *and then* $\forall c \in Arg \mid (a,c) \in R$*, add* (b,c) *to* R
- *if* $L(a) = out$*,* $\forall b \in Arg \mid L(b) \in \{in, undec\}$*, remove* (b,a) *from* R *and then* $\forall c \in Arg \mid (a,c) \in R \wedge L(c) \in \{in, undec\}$*, remove* (a,c) *from* R

Proof. Given a grounded labelling we want to change the label of a from *in* to *out* (or vice versa), while preserving the labels of all other arguments. If a is *in*, we can look for another argument b labelled *in* and make b attack a, together with all other arguments attacked by a. If the store does not contain any *in* argument, we take one from the universe. If a is *out*, we remove all the attacks coming from *in* and *undec* arguments, so that the only attacks left come from *out* arguments and a becomes *in*. To preserve the labels of the other arguments, all attacks from a towards *in* and *undec* are removed, since they would have become *out* after the revision of a. *out* arguments attacked by a does not need further adjustments. ◀

Note that the argument extension revision we propose for grounded semantics in Proposition 13 is more restrictive than necessary, since ensure all the arguments different from a (that is the argument to be revised) to maintain the exact same labels, while Definition 9 only forbids to change the label to *undec*. For each operator, we also show how to implement it in our language.

▶ **Proposition 14.** *The argument extension expansion, contraction and revision in Propositions 12, 12 and 13, respectively, can be implemented in our language.*

Proof. We show an example of possible implementations in Tables 4, 5 and 6. We make use of some syntactic sugar to simplify the presentation of the results. Let be $|Arg| = n$:

- $E_1 \wedge E_2 \to A$ represents $E_1 \to E_2 \to A$;
- $E_1 \vee E_2 \to A$ represents $E_1 \to A + E_2 \to A$;
- *true* represents a dummy $check(\{\}, \{\})$;
- $\sum_{a \in Arg} (E(a))$ represents $E(a_1) + E(a_2) + \cdots + E(a_n)$, $\forall a_i \in Arg$;
- $\Big\|_{G}^{a \in Arg} (E(a))$ represents $E(a_1) \|_G E(a_2) \|_G \ldots \|_G E(a_n)$, $\forall a_i \in Arg$;
- $test_c(a, S, \sigma) \to A$ represents $\sum_{l \in S} (test_c(a, l, \sigma))$.

We also use the letter u to identify fresh arguments taken from $U \setminus Arg$. ◀

We want to emphasize that guarded parallelism $\|_G$ and if then else constructs realised through $+_P$ are crucial for the implementation of the aforementioned operators. For instance, we use $\|_G$ in the argument extension contraction (Table 5) to remove all and only the attacks towards a coming from *undec* arguments. This behaviour cannot be achieved through classical parallelism (which only succeeds when all the branches terminates). The operator $+_P$, instead, is used in Table 4 to realise the expansion from $\emptyset$ to *out*: if an *in* argument b can be found in the framework, then we add an attack from b to a; otherwise we have to introduce, beforehand, an *in* argument. Without an if then else construct it is not possible to prioritise the choice of looking for an existing *in* argument and an agent could arbitrarily add a new argument even if it is not needed.

In devising operations of Definitions 9 and 10, that allow agents for changing the labels of arguments in a shared knowledge base with respect to a given semantics, we reinterpret AGM operators for expansion, contraction and revision. In particular, our operations are restricted to a single argument, rather than considering a set of beliefs as in other approaches like [14] and [5]. Nonetheless, we maintain similarities with the AGM theory, to the point that we can highlight some similarities with the original postulates of [1] that characterise rational operators performing expansion, contraction and revision of beliefs in a knowledge base. Consider for instance an argument a of an AF F and a semantics σ. An argument semantics expansion $\oplus_a^\sigma$ produces as output an AF F' for which no labelling $L' \in S_\sigma(F')$ is such that a has less labels in L' than in any labelling L of F (i.e., the number of labels assigned to a either remains the same or increases after the expansion).

Table 4 Argument extension expansion operator (Proposition 11) in CA syntax.

$$\oplus^{gde,in}_{\substack{a,L\\(L(a)=\emptyset)}}(F) : add(\{a\},\{\}) \rightarrow success$$

$$\oplus^{gde,out}_{\substack{a,L\\(L(a)=\emptyset)}}(F) : \sum_{b\in Arg} (test_c(b,in,gde) \rightarrow add(\{a\},\{(b,a)\})) \rightarrow success$$
$$+_P$$
$$add(\{a,u\},\{(u,a)\}) \rightarrow success$$

$$\oplus^{gde,undec}_{\substack{a,L\\(L(a)=in)}}(F) : \sum_{b\in Arg} (test_c(b,undec,gde) \rightarrow add(\{\},\{(b,a)\})) \rightarrow success$$
$$+_P$$
$$add(\{\},\{(a,a)\}) \rightarrow success$$

$$\oplus^{gde,undec}_{\substack{a,L\\(L(a)=out)}}(F) : \|_{G\ b\in Arg} (test_c(b,in,gde) \wedge check(\{\},\{(b,a)\}) \rightarrow add(\{\},\{(a,b)\})) \rightarrow success$$

Table 5 Argument extension contraction operator (Proposition 12) in CA syntax.

$$\oslash^{gde,in}_{\substack{a,L\\(L(a)=undec)}}(F) : \|_{G\ b\in Arg} (test_c(b,undec,gde) \rightarrow rmv(\{\},\{(b,a)\})) \rightarrow success$$

$$\oslash^{gde,out}_{\substack{a,L\\(L(a)=undec)}}(F) : \sum_{b\in Arg} (test_c(b,in,gde) \rightarrow add(\{\},\{b,a\})) \rightarrow success$$
$$+_P$$
$$add(\{u\},\{u,a\}) \rightarrow success$$

$$\oslash^{gde,\emptyset}_{\substack{a,L\\(L(a)=in)}}(F) : rmv(\{a\},\{\}) \rightarrow success$$

$$\oslash^{gde,\emptyset}_{\substack{a,L\\(L(a)=out)}}(F) : rmv(\{a\},\{\}) \rightarrow success$$

5 Related Work

A formalism for expressing dynamics in AFs is defined in [28] as a *Dynamic Argumentation Framework* (DAF). The aim of that paper is to provide a method for instantiating Dung-style AFs by considering a universal set of arguments U. A DAF consists of an AF $\langle U, R\rangle$ and a set of evidence, which has the role of restricting $\langle U, R\rangle$ to possible arguments and relations, so to obtain a static instance of the framework. DAFs are built starting from argumental structures, in which a tree of arguments supports a claim (corresponding to the root of the tree), and then adding attacks between argumental structures. The dynamic component of a DAF is thus the set of evidence. The introduced approach allows for generalising AFs,

Table 6 Argument extension revision operator (Proposition 13) in CA syntax.

$$
\begin{aligned}
\circledast^{gde,out}_{\substack{a,L\\(L(a)=in)}}(F) : & \sum_{b\in Arg} (test_c(b,in,gde) \rightarrow add(\{\},\{(b,a)\}) \\
& \quad \rightarrow \|_{\substack{G\\c\in Arg}} (check(\{c\},\{a,c\}) \rightarrow add(\{\},\{(b,c)\})) \parallel_G true \rightarrow success \\
& +_P \\
& add(\{b\},\{(b,a)\}) \\
& \quad \rightarrow \|_{\substack{G\\c\in Arg}} (check(\{c\},\{a,c\}) \rightarrow add(\{\},\{(b,c)\})) \parallel_G true) \rightarrow success \\
\circledast^{gde,in}_{\substack{a,L\\(L(a)=out)}}(F) : & \|_{\substack{G\\b\in Arg}} (test_c(b,\{in,undec\},gde) \rightarrow rmv(\{\},\{(b,a)\})) \\
& \quad \rightarrow \|_{\substack{G\\c\in Arg}} (\\
& \qquad test_c(c,\{in,undec\},gde) \wedge check(\{c\},\{a,c\}) \\
& \qquad\quad \rightarrow rmv(\{\},\{(a,c)\}) \parallel_G true \\
& \qquad) \rightarrow success
\end{aligned}
$$

adding the possibility of modelling changes, but, contrary to our study, it does not consider how such modifications affect the semantics and does not allow to model the behaviour of concurrent agents.

The impact of modifications on an AF in terms of sets of extensions is studied in [13]. Different kinds of revision are introduced, in which a new argument interacts with an already existing one. The authors describe different kinds of revision differing in the number of extensions that appear in the outcome, with respect to a semantics: a *decisive* revision allows to obtain a unique non-empty extension, a *selective* revision reduces the number of extensions (to a minimum of two), while a *questioning* one increases that number; a destructive *revision* eliminates all extensions, an *expansive* revision maintain the number of extension and increases the number of accepted arguments; a *conservative* revision does not introduce changes on the semantics level (and is strictly connected to the notion of robustness [9]), and an *altering* revision insert and delete arguments in the extensions. All these revisions are obtained through the addition of a single argument, together with a single attack relation either towards or from the original AF, and can be implemented as procedures of our language. The review operator we define in the syntax of our language (as the other two operator for expansion and contraction), instead, does not consider whole extensions, but just an argument at a time, allowing communicating agents to modify their beliefs in a finer grain.

Focusing on syntactic expansion of an AF (the mere addition of arguments and attacks), [5] show under which conditions a set of arguments can be enforced (to become accepted) for a specific semantics. Moreover, since adding new arguments and attacks may lead to a decrease in term of extensions and accepted arguments, the authors also investigate whether an expansion behave in a monotonic fashion, thus preserving the status of all originally accepted arguments. The study is only conducted on the case of weak expansion (that adds further arguments which do not attack previous arguments). The notion of expansion we use in the presented work is very different from that in [5]. First of all, we take into

account semantics when defining the expansion, making it more similar to an enforcement itself: we can increment the labels of an argument so to match a desired acceptance status. Then, our expansion results to be more general, being able to change the status of a certain argument not only to accepted, but also rejected, undecided or undetermined. This is useful, for instance, when we want to diminish the beliefs of an opponent agent.

Enforcing is also studied in [14], where the authors consider an expansion of the AF that only allows the addition of new attack relations, while the set of arguments remains the same (differently from [5]). It is shown, indeed, that if no new argument is introduced, it is always possible to guarantee the success of enforcement for any classical semantics. Also in this case, we want to highlight the differences with our work. Starting from the modifications allowed into the framework, we are not limited to only change the set of relations, since we implement procedures that also add and remove arguments. Moreover, the operators we define are not just enforcement operators, since they allow to modify the acceptability status of a single argument of an AF.

In our model, AFs are equipped with a universe of arguments that agents use to insert new information in the knowledge base. The problem of combining AFs is addressed in [6], that study the computational complexity of verifying if a subset of argument is an extension for a certain semantics in incomplete argumentation frameworks obtained by merging different beliefs. The incompleteness is considered both for arguments and attack relations. Similarly to our approach, arguments (and attacks) can be brought forward by agents and used to build new acceptable extensions. On the other hand, the scope of [6] is focused on a complexity analysis and does not provide implementations for the merging.

6 Conclusion and Future Work

We introduced a concurrent language for argumentation, that can be used by (intelligent) agents to implement different forms of communications. The agents involved in the process share an abstract argumentation framework that serves as a knowledge base and where arguments represent the agreed beliefs. The framework can be changed via a set of primitives that allow the addition and the removal of arguments and attacks. All agents have at their disposal a universe of "unused" arguments to chose from when they need to introduce new information. In order to take into account the justification status of such beliefs (which can be accepted, rejected, undetermined and inconsistent) we considered a four-state labelling semantics. Besides operations at a syntactic level, thus, we also defined semantic operation that verify the acceptability of the arguments in the store. Finally, to allow agents for realising more complex forms of communication (like negotiation and persuasion), we presented three AGM-style operators, namely of expansion, contraction and revision, that change the status of a belief to a desired one; we also showed how to implement them in our language.

For the future, we plan to extend this work in many directions. First of all, given the known issues of abstract argumentation [27], we want to consider structured AFs and provide an implementation for our expansion, contraction and revision operators, for which a different store (structured and not abstract, indeed) need to be considered. The concurrent primitives are already general enough and do not require substantial changes. To obtain a spendable implementation, we will consider operations that can be done in polynomial time [20], for instance by using the grounded semantics, for which finding and checking extension is a easy task from the point of view of computational complexity. We also plan to provide a real implementation of our language that can be used for both research purposes and practical applications.

To further improve the capabilities of our agents and make it more appealing for real-life applications, we want to extend our language with the ability to handle processes involving time-critical aspects, in a similar way as CC is extended with temporal logic in [16, 15]. In this way, we could implement operations that also take into account time constraints. The shared store could also be shaped as a subsumptive hierarchy, able to handle various relations among the arguments.

On the operations level, we are currently only able to modify the acceptance status of the arguments, without further considerations on the obtained semantics. To gain control also over changes on the set of extensions, we want to introduce operators able to obtain a specified semantics (when possible) or to leave it unchanged (this can be done relying on the notion of robustness [9]). Another study we could conduct over the operators concerns their (non-)monotonicity. Since, in the current state of the work, operations like the removal of an argument can lead to an expansion into the considered AF, we would like to investigate the conditions under which, for instance, a contraction can be the only consequence of a removal. To this extent, also other operations on beliefs (like extraction, consolidation and merging) could be taken into account.

As a final consideration, whereas in real-life cases it is always clear which part involved in a debate is stating a particular argument, AFs do not hold any notion of "ownership" for arguments or attacks, that is, any bond with the one making the assertion is lost. To overcome this problem, we want to implement the possibility of attaching labels on (groups of) arguments and attacks of AFs, in order to preserve the information related to whom added a certain argument or attack, extending and taking into account the work in [24]. Consequently, we can also obtain a notion of locality (or scope) of the belief in the knowledge base: arguments owned by a given agents can be placed into a local store and used in the implementation of specific operators through hidden variables.

References

1 Carlos E. Alchourrón, Peter Gärdenfors, and David Makinson. On the logic of theory change: Partial meet contraction and revision functions. *The Journal of Symbolic Logic*, 50(02):510–530, June 1985. doi:10.2307/2274239.

2 Pietro Baroni, Martin Caminada, and Massimiliano Giacomin. An introduction to argumentation semantics. *Knowledge Eng. Review*, 26(4):365–410, 2011. doi:10.1017/S0269888911000166.

3 Pietro Baroni and Massimiliano Giacomin. On principle-based evaluation of extension-based argumentation semantics. *Artif. Intell.*, 171(10-15):675–700, 2007. doi:10.1016/j.artint.2007.04.004.

4 Ringo Baumann. What Does it Take to Enforce an Argument? Minimal Change in abstract Argumentation. *Frontiers in Artificial Intelligence and Applications*, pages 127–132, 2012. doi:10.3233/978-1-61499-098-7-127.

5 Ringo Baumann and Gerhard Brewka. Expanding argumentation frameworks: Enforcing and monotonicity results. In Pietro Baroni, Federico Cerutti, Massimiliano Giacomin, and Guillermo Ricardo Simari, editors, *Computational Models of Argument: Proceedings of COMMA 2010, Desenzano del Garda, Italy, September 8-10, 2010*, volume 216 of *Frontiers in Artificial Intelligence and Applications*, pages 75–86. IOS Press, 2010. doi:10.3233/978-1-60750-619-5-75.

6 Dorothea Baumeister, Daniel Neugebauer, Jörg Rothe, and Hilmar Schadrack. Verification in incomplete argumentation frameworks. *Artif. Intell.*, 264:1–26, 2018. doi:10.1016/j.artint.2018.08.001.

7 Stefano Bistarelli, Lars Kotthoff, Francesco Santini, and Carlo Taticchi. Containerisation and Dynamic Frameworks in ICCMA'19. In *Proceedings of the Second International Workshop on Systems and Algorithms for Formal Argumentation (SAFA 2018) Co-Located with the 7th International Conference on Computational Models of Argument (COMMA 2018), Warsaw, Poland, September 11, 2018*, volume 2171 of *CEUR Workshop Proceedings*, pages 4–9. CEUR-WS.org, 2018.

8 Stefano Bistarelli and Francesco Santini. Conarg: A constraint-based computational framework for argumentation systems. In *IEEE 23rd International Conference on Tools with Artificial Intelligence, ICTAI 2011, Boca Raton, FL, USA, November 7-9, 2011*, pages 605–612. IEEE Computer Society, 2011. doi:10.1109/ICTAI.2011.96.

9 Stefano Bistarelli, Francesco Santini, and Carlo Taticchi. On Looking for Invariant Operators in Argumentation Semantics. In *Proceedings of the Thirty-First International Florida Artificial Intelligence Research Society Conference, FLAIRS 2018, Melbourne, Florida, USA. May 21-23 2018.*, pages 537–540, 2018.

10 Guido Boella, Souhila Kaci, and Leendert W. N. van der Torre. Dynamics in Argumentation with Single Extensions: Attack Refinement and the Grounded Extension (Extended Version). In *Argumentation in Multi-Agent Systems, 6th International Workshop, ArgMAS 2009. Revised Selected and Invited Papers*, volume 6057 of *Lecture Notes in Computer Science*, pages 150–159. Springer, 2009. doi:10.1007/978-3-642-12805-9_9.

11 Martin Caminada. On the Issue of Reinstatement in Argumentation. In *Logics in Artificial Intelligence, 10th European Conference, JELIA 2006, Liverpool, UK, September 13-15, 2006, Proceedings*, volume 4160 of *Lecture Notes in Computer Science*, pages 111–123. Springer, 2006.

12 Martin Caminada. On the Issue of Reinstatement in Argumentation. In Michael Fisher, Wiebe van der Hoek, Boris Konev, and Alexei Lisitsa, editors, *Logics in Artificial Intelligence, 10th European Conference, JELIA 2006, Liverpool, UK, September 13-15, 2006, Proceedings*, volume 4160 of *Lecture Notes in Computer Science*, pages 111–123. Springer, 2006.

13 Claudette Cayrol, Florence Dupin de Saint-Cyr, and Marie-Christine Lagasquie-Schiex. Revision of an Argumentation System. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference, KR 2008, Sydney, Australia, September 16-19, 2008*, pages 124–134. AAAI Press, 2008.

14 Sylvie Coste-Marquis, Sébastien Konieczny, Jean-Guy Mailly, and Pierre Marquis. Extension enforcement in abstract argumentation as an optimization problem. In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 2876–2882. AAAI Press, 2015. URL: http://ijcai.org/Abstract/15/407.

15 Frank S. de Boer, Maurizio Gabbrielli, and Maria Chiara Meo. Semantics and expressive power of a timed concurrent constraint language. In Gert Smolka, editor, *Principles and Practice of Constraint Programming – CP97, Third International Conference, Linz, Austria, October 29 – November 1, 1997, Proceedings*, volume 1330 of *Lecture Notes in Computer Science*, pages 47–61. Springer, 1997. doi:10.1007/BFb0017429.

16 Frank S. de Boer, Maurizio Gabbrielli, and Maria Chiara Meo. A timed concurrent constraint language. *Inf. Comput.*, 161(1):45–83, 2000. doi:10.1006/inco.1999.2879.

17 Sylvie Doutre, Andreas Herzig, and Laurent Perrussel. A Dynamic Logic Framework for Abstract Argumentation. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*, 2014.

18 Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–357, September 1995. doi:10.1016/0004-3702(94)00041-X.

19 Florence Dupin de Saint-Cyr, Pierre Bisquert, Claudette Cayrol, and Marie-Christine Lagasquie-Schiex. Argumentation update in YALLA (Yet Another Logic Language for Argumentation). *International Journal of Approximate Reasoning*, 75:57–92, August 2016. doi:10.1016/j.ijar.2016.04.003.

20 Wolfgang Dvorák and Paul E. Dunne. Computational problems in formal argumentation and their complexity. *FLAP*, 4(8), 2017. URL: http://www.collegepublications.co.uk/downloads/ifcolog00017.pdf.

21 Jacob Glazer and Ariel Rubinstein. Debates and Decisions: On a Rationale of Argumentation Rules. *Games and Economic Behavior*, 36(2):158–173, 2001. doi:10.1006/game.2000.0824.

22 Hadassa Jakobovits and Dirk Vermeir. Robust semantics for argumentation frameworks. *J. Log. Comput.*, 9(2):215–261, 1999. doi:10.1093/logcom/9.2.215.

23 Magdalena Kacprzak, Katarzyna Budzynska, and Olena Yaskorska. A logic for strategies in persuasion dialogue games. In *Advances in Knowledge-Based and Intelligent Information and Engineering Systems – 16th Annual KES Conference, San Sebastian, Spain, 10-12 September 2012*, volume 243 of *Frontiers in Artificial Intelligence and Applications*, pages 98–107. IOS Press, 2012. doi:10.3233/978-1-61499-105-2-98.

24 Nicolas Maudet, Simon Parsons, and Iyad Rahwan. Argumentation in Multi-Agent Systems: Context and Recent Developments. In *Argumentation in Multi-Agent Systems, Third International Workshop, ArgMAS 2006, Hakodate, Japan, May 8, 2006, Revised Selected and Invited Papers*, pages 1–16, 2006. doi:10.1007/978-3-540-75526-5_1.

25 Henry Prakken. Models of Persuasion Dialogue. In *Argumentation in Artificial Intelligence*, pages 281–300. Springer, 2009. doi:10.1007/978-0-387-98197-0_14.

26 Henry Prakken. An abstract framework for argumentation with structured arguments. *Argument & Computation*, 1(2):93–124, 2010. doi:10.1080/19462160903564592.

27 Henry Prakken and Michiel De Winter. Abstraction in argumentation: Necessary but dangerous. In Sanjay Modgil, Katarzyna Budzynska, and John Lawrence, editors, *Computational Models of Argument – Proceedings of COMMA 2018, Warsaw, Poland, 12-14 September 2018*, volume 305 of *Frontiers in Artificial Intelligence and Applications*, pages 85–96. IOS Press, 2018. doi:10.3233/978-1-61499-906-5-85.

28 Nicolas D. Rotstein, Martın O. Moguillansky, Alejandro J. Garcia, and Guillermo R. Simari. An abstract argumentation framework for handling dynamics. In *Proceedings of the Argument, Dialogue and Decision Workshop in NMR 2008, Sydney, Australia*, pages 131–139, 2008.

29 Vijay A. Saraswat and Martin Rinard. Concurrent constraint programming. In *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages – POPL '90*, pages 232–245, San Francisco, California, United States, 1990. ACM Press. doi:10.1145/96709.96733.

30 Francesca Toni. A tutorial on assumption-based argumentation. *Argument & Computation*, 5(1):89–117, 2014. doi:10.1080/19462166.2013.869878.