A Concurrent Language for Argumentation*

Stefano Bistarelli[1][0000−0001−7411−9678] and Carlo Taticchi[2][0000−0003−1260−4672]

[1] University of Perugia, Italy - stefano.bistarelli@unipg.it
[2] Gran Sasso Science Institute, 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.

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 decreases 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 [15, 19], 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 [10], 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 [17, 5] and computational aspects (for example, a special track on dynamics [4] appeared in the Third International Competition on Computational Models of Argumentation).

Logical frameworks for argumentation, like the ones presented in [9, 11], 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* [18], 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 inserting (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

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 [10], Dung defines the building blocks of abstract argumentation: an Abstract Argumentation Framework is a pair $\langle Arg, R \rangle$ where $Arg \subseteq U$ is a set of arguments belonging to a "universe" U and R is a binary relation on Arg representing attacks[3]. 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 [10, 2] that reflect qualities which are likely to be desirable for "good" subsets of arguments. In the rest of this paper, we will denote the extension-based semantics (also referred to as Dung semantics), namely admissible, complete, stable, preferred, and grounded, with their respective abbreviation *adm*, *com*, *stb*, *prf* and *gde*, and generically with σ. Besides enumerating the extensions for a certain semantics σ, one of the most 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.

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* [6]. In that paper, Caminada also gives a definition for a reinstatement labelling, a total function that assigns a label to the arguments of an AF: 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. Moreover, there exists a connection between reinstatement labellings and the Dung-style semantics: 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.

[3] 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 insert arguments from U to Arg.

3 A Four-state Labelling Semantics

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 1, 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 1, 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.

Fig. 1. 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 [13], where arguments that are assigned the label + are accepted, those that are assigned the label − are rejected, those that are assigned both + and − are neither fully accepted nor fully rejected, and those that are not considered at all are assigned the empty set $\emptyset$. Each different label can be traced to a particular meaning. $\emptyset$ stands for "don't care" [13] 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). Even though the four-state labelling is more informative than the reinstatement labelling (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. To overcome this problem, in the following we establish a mapping between a modified four-state labelling and the classical semantics.

Definition 1 (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 1. *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 1 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.*

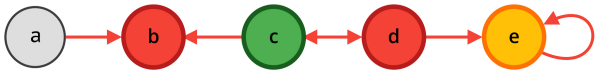

Fig. 2. Admissible labelling of an AF showed through colours. Argument c, highlighted in green, is the only *in*; red arguments b and d are *out*; the one in yellow, namely e, is *undec*; and the grey argument a are left with an empty label $\emptyset$.

4 The Concurrent Argumentation Language (CA)

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) [18], 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.

We replace the CC ask operation 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 CC tell operation (that we call *insert*) 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 1, while in Table 2 we give the definitions for the transition rules.

$$A ::= success \mid insert(Arg, R) \to A \mid rmv(Arg, R) \to A \mid E \mid A \| A \mid \exists_x A$$
$$E ::= test_c(a, l, \sigma) \to A \mid test_s(a, l, \sigma) \to A \mid check(Arg, R) \to A \mid E + E \mid E +_P E \mid E \|_G E$$

Table 1. CA syntax.

Suppose to have an agent A whose knowledge base is represented by an AF $F = \langle Arg, R \rangle$. An $insert(Arg', R')$ 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 an insertion, (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 insertion of only attack relations to the considered AF. It is as well possible to insert 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

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

Table 2. CA operational semantics.

$check(Arg', R')$ 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[4] 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

[4] The set of extensions $S_\sigma(F)$ is finite, thus both $test_c(a, l, \sigma)$ and $test_s(a, l, \sigma)$ are decidable.

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[5]. The parallel composition operator enables the specification of complex concurrent argumentation processes. For example, a debate involving many agents that asynchronously provide arguments can be modelled as a parallel composition of insert operations performed on the knowledge base.

Example 1. Consider the AF in Figure 3 (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 in parallel with agent B 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 insert 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 insert 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 3 (right).

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

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 insert/remove.

[5] We plan to use existential quantifiers to extend our work by allowing our agents to have local stores.

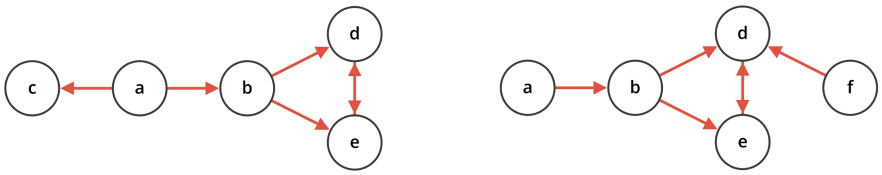

Fig. 3. The AF on the right is obtained starting from the one on the left trough the insertion of an argument f attacking d and the removal of c together with the attack (a, c).

4.1 Belief Revision and the AGM Framework

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 (as depicted in Figure 4) 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.

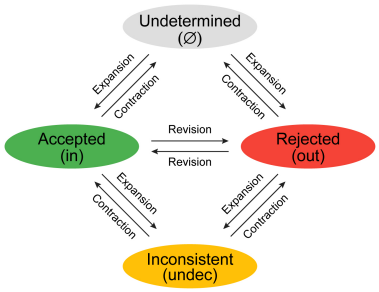

Fig. 4. Transitions between AGM beliefs states.

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, and provide building blocks for realizing complex interaction processes between agents. Negotiation, that aims to solve conflicts arising from the interaction between two or more parties with different individual goals, could be implemented through expansion operations, modelling the behaviour of an agent presenting claims towards its counterparts, and contraction, representing the act of retracting a condition to successfully conclude the negotiation. Inconsistent beliefs in a debate can be made accepted through a contraction, while expansion can make beliefs which state is undetermined acceptable. Agents involved in persuasive dialogue games have to elaborate strategies for supporting their beliefs and defeating the adversaries. Again, revision operations on the knowledge base can change 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 have 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 2 (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 $\oplus_{a,L}^{\sigma,l}(F)$, $\oslash_{a,L}^{\sigma,l}(F)$ *and* $\circledast_{a,L}^{\sigma,l}(F)$ *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* $L'(a) = l$.

Definition 2 can be extended in such a way to consider not only one labelling, but all the possible ones. We omit the discussion on this regard due to space constraints. It is important to note that the formalism we present is not monotone: the *insert* 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.

AGM operators have already been studied from the point of view of their implementation in work as [3, 8]. 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 2 can be implemented. Notice 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 simplicity of implementation.

Proposition 1. *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* $\oplus_{a,L}^{gde,l}(F)$ *could act as:*

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

Proposition 2. *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* $\exists b \in Arg \mid L(b) = in$, *insert* (b, a) *to* R
  - *otherwise, insert* $\langle\{b\}, \{(b, a)\}\rangle$ *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

Proposition 3. *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* $\exists b \in Arg \mid L(b) = in$, *insert* (b, a) *to* R *and then* $\forall c \in Arg \mid (a, c) \in R$, *insert* (b, c) *to* R
- *otherwise, insert* $\langle \{b\}, \{(b, a)\} \rangle$ *to* F *and then* $\forall c \in Arg \mid (a, c) \in R$, *insert* (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

Note that the argument extension revision we propose for grounded semantics in Proposition 3 is more restrictive than necessary, since ensure that all the arguments different from a (that is the argument to be revised) maintain the exact same labels, while Definition 2 only forbids to change the label to *undec*. The three introduced operators can be implemented in our language.

Proposition 4. *The argument extension expansion, contraction and revision in Propositions 2, 2 and 3, respectively, can be implemented in our language.*

As an example, an expansion operator is shown in Table 3.

$$
\begin{aligned}
\oplus^{gde,in}_{\substack{a,L \\ (L(a)=\emptyset)}}(F) &: insert(\{a\}, \{\}) \rightarrow success \\
\oplus^{gde,out}_{\substack{a,L \\ (L(a)=\emptyset)}}(F) &: \sum_{b \in Arg} (test_c(b, in, gde) \rightarrow insert(\{a\}, \{(b, a)\})) \rightarrow success \\
&\quad +_P insert(\{a, u\}, \{(u, a)\}) \rightarrow success \\
\oplus^{gde,undec}_{\substack{a,L \\ (L(a)=in)}}(F) &: \sum_{b \in Arg} (test_c(b, undec, gde) \rightarrow insert(\{\}, \{(b, a)\})) \rightarrow success \\
&\quad +_P insert(\{\}, \{(a, a)\}) \rightarrow success \\
\oplus^{gde,undec}_{\substack{a,L \\ (L(a)=out)}}(F) &: \Big\|_G \; (test_c(b, in, gde) \wedge check(\{\}, \{(b, a)\}) \\
&\quad \rightarrow insert(\{\}, \{(a, b)\})) \rightarrow success
\end{aligned}
$$

Table 3. Argument extension expansion operator (Proposition 1) in CA syntax where $test_c(a, S, \sigma) \rightarrow A$ is syntactic sugar for $\sum_{l \in S}(test_c(a, l, \sigma))$.

In devising operations of Definitions 2, 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. 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 $\oplus^{\sigma}_{a}$ 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).

5 Related Work

A formalism for expressing dynamics in AFs is defined in [17] 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. The introduced approach allows for generalising AFs, 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 [7]. Different kinds of revision are introduced, in which a new argument interacts with an already existing one. 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), [3] show under which conditions a set of arguments can be enforced (to become accepted) for a specific semantics. The notion of expansion we use in the presented work is very different from that in [3]. 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.

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. 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 operations 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 [16], 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 [12], 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.

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 [14]. 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. Alchourrón, C.E., Gärdenfors, P., Makinson, D.: On the logic of theory change: Partial meet contraction and revision functions. The Journal of Symbolic Logic **50**(02), 510–530 (Jun 1985)
2. Baroni, P., Caminada, M., Giacomin, M.: An introduction to argumentation semantics. Knowledge Eng. Review **26**(4), 365–410 (2011)
3. Baumann, R., Brewka, G.: Expanding argumentation frameworks: Enforcing and monotonicity results. In: Baroni, P., Cerutti, F., Giacomin, M., Simari, G.R. (eds.) Computational Models of Argument: Proceedings of COMMA 2010, Desenzano del Garda, Italy, September 8-10, 2010. Frontiers in Artificial Intelligence and Applications, vol. 216, pp. 75–86. IOS Press (2010)
4. Bistarelli, S., Kotthoff, L., Santini, F., Taticchi, C.: 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. CEUR Workshop Proceedings, vol. 2171, pp. 4–9. CEUR-WS.org (2018)
5. Boella, G., Kaci, S., van der Torre, L.W.N.: 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. Lecture Notes in Computer Science, vol. 6057, pp. 150–159. Springer (2009)

6. Caminada, M.: On the Issue of Reinstatement in Argumentation. In: Logics in Artificial Intelligence, 10th European Conference, JELIA 2006, Liverpool, UK, September 13-15, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4160, pp. 111–123. Springer (2006)
7. Cayrol, C., de Saint-Cyr, F.D., Lagasquie-Schiex, M.C.: 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. pp. 124–134. AAAI Press (2008)
8. Coste-Marquis, S., Konieczny, S., Mailly, J., Marquis, P.: Extension enforcement in abstract argumentation as an optimization problem. In: Yang, Q., Wooldridge, M.J. (eds.) Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015. pp. 2876–2882. AAAI Press (2015)
9. Doutre, S., Herzig, A., Perrussel, L.: 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)
10. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artificial Intelligence **77**(2), 321–357 (Sep 1995)
11. Dupin de Saint-Cyr, F., Bisquert, P., Cayrol, C., Lagasquie-Schiex, M.C.: Argumentation update in YALLA (Yet Another Logic Language for Argumentation). International Journal of Approximate Reasoning **75**, 57–92 (Aug 2016)
12. Dvorák, W., Dunne, P.E.: Computational problems in formal argumentation and their complexity. FLAP **4**(8) (2017)
13. Jakobovits, H., Vermeir, D.: Robust semantics for argumentation frameworks. J. Log. Comput. **9**(2), 215–261 (1999)
14. Maudet, N., Parsons, S., Rahwan, I.: 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. pp. 1–16 (2006)
15. Prakken, H.: An abstract framework for argumentation with structured arguments. Argument & Computation **1**(2), 93–124 (2010)
16. Prakken, H., Winter, M.D.: Abstraction in argumentation: Necessary but dangerous. In: Modgil, S., Budzynska, K., Lawrence, J. (eds.) Computational Models of Argument - Proceedings of COMMA 2018, Warsaw, Poland, 12-14 September 2018. Frontiers in Artificial Intelligence and Applications, vol. 305, pp. 85–96. IOS Press (2018)
17. Rotstein, N.D., Moguillansky, M.O., Garcia, A.J., Simari, G.R.: An abstract argumentation framework for handling dynamics. In: Proceedings of the Argument, Dialogue and Decision Workshop in NMR 2008, Sydney, Australia. pp. 131–139 (2008)
18. Saraswat, V.A., Rinard, M.: Concurrent constraint programming. In: Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL '90. pp. 232–245. ACM Press, San Francisco, California, United States (1990)
19. Toni, F.: A tutorial on assumption-based argumentation. Argument & Computation **5**(1), 89–117 (2014)