CoQuiAAS: A Constraint-based Quick Abstract Argumentation Solver

Jean-Marie Lagniez Emmanuel Lonca Jean-Guy Mailly
CRIL – U. Artois, CNRS
Lens, France
{lagniez,lonca,mailly}@cril.univ-artois.fr

***Abstract*—Nowadays, argumentation is a salient keyword in artificial intelligence. The use of argumentation techniques is particularly convenient for thematics such that multiagent systems, where it allows to describe dialog protocols (using persuasion, negotiation, ...) or on-line discussion analysis; it also allows to handle queries where a single agent has to reason with conflicting information (inference in the presence of inconsistency, inconsistency measure). This very rich framework gives numerous reasoning tools, thanks to several acceptability semantics and inference policies.**

On the other hand, the progress of SAT solvers in the recent years, and more generally the progress on Constraint Programming paradigms, lead to some powerful approaches that permit tackling theoretically hard problems.

The needs of efficient applications to solve the usual reasoning tasks in argumentation, together with the capabilities of modern Constraint Programming solvers, lead us to study the encoding of usual acceptability semantics into logical settings. We propose diverse use of Constraint Programming techniques to develop a software library dedicated to argumentative reasoning. We present a library which offers the advantages to be generic and easily adaptable. We finally describe an experimental study of our approach for a set of semantics and inference tasks, and we describe the behaviour of our solver during the First International Competition on Computational Models of Argumentation.

I. Introduction

An abstract argumentation framework [1] is a directed graph where the nodes represent abstract entities called *arguments* and the edges represent *attacks* between these arguments. This simple and elegant setting is used as well in processes concerning a single agent as in multiagent scenarios. Concerning single agent cases, the agent may have to reason from conflicting pieces of information, which leads her to build an argumentation framework from an inconsistent knowledge base to infer non-trivial conclusions [2]. In multiagent settings, argumentation is used to model dialogs between several agents [3] or to analyze on-line discussion between social network users [4]. The meaning of such a graph is determined by an *acceptability semantics*, which indicates which properties a set of arguments must satisfy to be considered as a "solution" of the problem; such a set of arguments is then called an *extension*.

Currently, a strong tendency in the argumentation community is the development of software approaches to compute the different reasoning tasks on argumentation frameworks, with respect to the usual semantics[1]. Given a semantics σ and an argumentation framework F, the most usual requests consist in computing one (or every) σ-extension of F, and determining if an argument belongs to at least one (or every) σ-extension of F. It is well-known that most of the pairs composed of a semantics and a request lead to a high complexity problem [6], [7]; thus, these tasks require the development of practically efficient tools to be computed in a reasonable time. To reach this goal, we propose in this article the use of Constraint Programming techniques, since this domain already proposes some very efficient solutions to solve high complexity combinatorial problems. In this paper, we are in particular interested in propositional logic and some formalisms derived from it. More precisely, we propose some encodings in conjunctive normal form (CNF) to solve problems from the first level of the polynomial hierarchy, and some encodings in the *Partial Max-SAT* formalism for higher complexity problems; these encodings allow us to solve reasoning tasks concerning four usual semantics and four usual requests. We take advantage of these encodings to solve these reasoning tasks, using some state-of-the-art approaches and software, which have proven their practical efficiency.

We have encoded those approaches for argumentation-based reasoning in a software library called CoQuiAAS. The aim of CoQuiAAS is twofold. First, we provide some efficient algorithms to tackle the main requests for the usual semantics. Then, our framework is designed to be upgradable: one may easily add some new parameters (request, semantics), or realize new algorithm for the tasks which are already implemented.

In this paper, we first present the basic notions concerning argumentation and the problems we provide reductions to: the Satisfiability problem – also known as SAT problem, which consist in deciding whether a propositional formula admits a model – and the search of Maximal Satisfiable Subsets (MSS) of constraints in a *Partial Max-SAT* instance. After this presentation, we detail the encodings we employed to translate argumentation problems into SAT and MSS problems. We present in the section IV the design of the library we provide, CoQuiAAS. At last, we give some experimental results of our approaches in section V, and we compare the conception and the request handled by our library to some existing softwares which tackle argumentation-based reasoning problems. The strength of CoQuiAAS is confirmed by the award received at the ICCMA'15.

[1] See [5] for more details.

II. Background Notions

A. Abstract Argumentation

Several models have been used to formalize argument-based reasoning. In this paper, we consider Dung's framework, which is one of the most well-known settings for argumentation related problems [1].

Definition 1. *An* argumentation framework *is a directed graph* $F = \langle A, R \rangle$ *where* A *is a finite set of abstract entities called* arguments *and* $R \subseteq A \times A$ *a binary relation called* attack relation.

The intuitive meaning of the attack relation comes from the usual proceedings of a debate. If a first argument a_1 is put forward without any contradiction, there is nothing to prevent an agent from considering a_1 as true. But, if someone puts forwards an argument a_2 (which is *a priori* acceptable) which attacks a_1, then a_1 cannot be accepted anymore, unless it is then defended. This intuitive notion of defense can be formalized as follows.

Definition 2. *Let* $F = \langle A, R \rangle$ *be an argumentation framework and* $a_1, a_2, a_3 \in A$ *three arguments.*

- *The argument* a_1 *is* attacked *by the argument* a_2 *in* F *if and only if* $(a_2, a_1) \in R$.
- *Then we says that* a_3 defends a_1 *against* a_2 *in* F *if and only if* $(a_3, a_2) \in R$.

These notions are generalized to attack and defense by a set of arguments $E \subseteq A$.

- *The argument* a_1 *is* attacked *by the set of arguments* E *in* F *if and only if* $\exists a_i \in E$ *such that* $(a_i, a_1) \in R$.
- *The set of arguments* $E \subseteq A$ defends a_1 *against* a_2 *in* F *if and only if* E *attacks* a_2.

For instance, in the argumentation framework described at Figure 1, the argument a_1 attacks the argument a_2, and a_2 defends himself against this attack. Intuitively, one may want at most one of these two arguments to be accepted, as considering one as accepted makes the second one attacked.

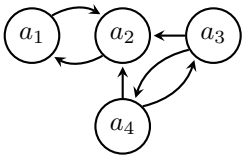

Fig. 1. An Example of Argumentation Framework. Each vertex is an argument, while an edge from a_i to a_j denotes the fact a_i attacks a_j.

When reasoning with an argumentation framework, an agent has to determine which arguments can jointly be accepted. Several properties may be defined for a set of arguments to be considered as a reasonable "solution" of the argumentation framework. Among these properties, two in particular are required by all the usual semantics of the literature:

- *conflict-freeness*: $E \subseteq A$ is conflict-free in F if and only if $\nexists a_i, a_j \in E$ such that $(a_i, a_j) \in R$;
- *admissibility*: a conflict-free set $E \subseteq A$ is admissible if and only if $\forall a_i \in E$, E defends a_i against all its attackers.

Conflict-freeness and admissibility are required by each acceptability semantics proposed by Dung.

Definition 3. *Let* $F = \langle A, R \rangle$ *be an argumentation framework.*

- *A conflict-free set* $E \subseteq A$ *is a* complete extension *of* F *if and only if* E *contains each argument that is defended by* E.
- *A set* $E \subseteq A$ *is a* preferred extension *of* F *if and only if* E *is a maximal element (with respect to* $\subseteq$*) among the complete extensions of* F.
- *A conflict-free set* $E \subseteq A$ *is a* stable extension *of* F *if and only if* E *attacks each argument which does not belong to* E.
- *A set* $E \subseteq A$ *is a* grounded extension *of* F *if and only if* E *is a minimal element (with respect to* $\subseteq$*) among the complete extensions of* F.

Given a semantics σ, $Ext_\sigma(F)$ *denotes the* σ*-extensions of* F. *The previous semantics are commonly denoted, respectively,* CO, PR, ST *and* GR.

These four acceptability semantics are illustrated in Section III for the example presented at Figure 1.

Dung proved that for each argumentation framework F,

- F has exactly one grounded extension;
- F has at least one preferred extension;
- each stable extension of F is a preferred extension of F;
- F may admit no stable extension.

Given a semantics σ, several decision problems can be considered. First, an interesting question is to know if a given argument a is skeptically or credulously accepted. The status of a is given by the following definitions:

- F accepts a skeptically with respect to the semantics σ if and only if $\forall \epsilon \in Ext_\sigma(F), a \in \epsilon$;
- F accepts a credulously with respect to the semantics σ if and only if $\exists \epsilon \in Ext_\sigma(F)$ such that $a \in \epsilon$.

Obviously, both these statuses collapse for the grounded semantics, since an argumentation framework possesses exactly one grounded extension. The skeptical acceptance (respectively credulous acceptance) decision problem is commonly denoted by DC (respectively DS).
Another interesting decision problem is $Exist$, which is the problem of determining whether a non-empty extension exists or not for the given semantics.

The complexity of these decision problems is summed up in Table I (which gathers results from other publications [6], [7], [8]).

TABLE I. COMPLEXITY OF INFERENCE PROBLEMS FOR THE USUAL SEMANTICS. C$-c$ MEANS THAT THE CONSIDERED DECISION PROBLEM IS COMPLETE FOR THE COMPLEXITY CLASS C.

Semantics	GR	ST	PR	CO
DC	P	NP$-c$	NP$-c$	NP$-c$
DS	P	coNP$-c$	$\Pi_2^P - c$	P$-c$
$Exist$	P	NP$-c$	NP$-c$	NP$-c$

Obviously, the complexity $Exist$ is a lower bound of the complexity of the computation of an extension, while the complexity of DS is a lower bound of the enumeration of the extensions.

B. Propositional Logic

The alphabet of propositional logic is the combination of a set of Boolean variables (generally denoted PS) and a set of three usual operators: the negation operator $\neg$ (unary), the conjunction operator $\wedge$ (binary) and the disjunction operator $\vee$ (binary), each of them being used to connect formulae (the variables themselves are atomic formulae). Insofar as a propositional formula is built in an inductive way (since the connectives apply on formulae), it can be seen as a rooted, directed, acyclic graph. Given an assignment of the set of propositional variables (called *interpretation*), a propositional formula is evaluated to *true* if and only if the root node of the formula is evaluated to *true*. In order to determine the value of this node given by an interpretation, one may compute the value of each node in reverse topological order; indeed, the value of the leaves is known – since the leaves are the nodes which correspond to the variables, which truth value is given by the interpretation – and the values of internal nodes is determined by the semantics of the corresponding connectives: a negation node ($\neg$) has the value *true* if and only if its child has the value *false*, a node $\wedge$ (respectively $\vee$) has the value *true* if and only if both its children (respectively at least one of its children) have (respectively has) the value *true*. When an interpretation leads a formula to be *true*, we call it a *model* of the formula. A formula is said to be *consistent* if and only if it admits at least one model. As a convention, we represent an interpretation by the set of variables which are *true* with respect to this interpretation. In addition to the previous connectives, we define the implication ($\Rightarrow$, defined by $a \Rightarrow b \equiv \neg a \vee b$) and equivalence ($\Leftrightarrow$, defined by $a \Leftrightarrow b \equiv (a \Rightarrow b) \wedge (b \Rightarrow a)$) connectives. When a propositional formula Φ is equivalent to a conjunction $\phi_1 \wedge \cdots \wedge \phi_n$, we can represent Φ as a set $\{\phi_1, \ldots, \phi_n\}$.

In our study, we deal with encodings in NNF formulae, meaning some propositional formulae where the negation operator is only applied on variables (that is, the leaves of the formula). However, CoQuiAAS uses SAT solvers, which are only able to tackle propositional formulae in conjunctive normal form, so a translation step from NNF to CNF is required between the encodings which exist in the literature and the ones that we actually use in our software library. It does not influence the generality of our approaches, since each propositional formula can be translated in polynomial time into an equivalent CNF formula.

The CNF formulae correspond to conjunctions of disjunctions of literals (a literal is a propositional variable or its negation), that is formulae written as $\wedge_{i=1}^{n}(\vee_{j=1}^{n_i} l_{i,j})$. These formulae are interesting since a disjunction of literals (called a *clause*) allows to represent a constraint in an simple manner; so, a CNF formula is in fact a set of constraints, and a model a such a formula is an assignment of a truth value to each variable such that no constraint of the problems is violated.

Although this formalism is well suited to represent a problem, searching a model of a CNF formula is theoretically complex (NP-hard [9]). However, the modern *SAT solvers* are able to solve these problems very efficiently, tackling gradually more and more imposing ones [10]. There exists problems which do not have any model, meaning that there is no interpretation such that each constraint is satisfied. In this case, an interesting question is to determine an interpretation which maximizes the number of satisfied constraints: this problem is called *Max-SAT* [10]. We can generalize this problem, giving a weight to each constraint – now the question is to maximize the sum of the weights of the satisfied constraints – this is the problem *Weighted Max-SAT*. If some constraints have an infinite weight (which means that they have to be satisfied), then the problems are said to be "partial": we thus consider the problems *Partial Max-SAT* and *Weighted Partial Max-SAT*.

Discovering an optimal solution of a Max-SAT instance allows to determine a set of constraints from the initial formula which is consistent, such that adding any other constraint from the initial problem makes this new problem inconsistent [11]; a set of constraints which has this property is called a *maximal satisfiable subset* (MSS) [12]. Given the set of constraints ϕ of a problem and a set of constraints ψ which is a MSS of ϕ, we say that $\overline{\psi} = \phi \setminus \psi$ is a *coMSS* (or *MCS*) of the formula [11]. We remark that the optimal solutions of the Max-SAT problem are only a subset of all the MSS of a formula.

It is interesting to notice that the algorithms developed in Constraint Programming to solve Max-SAT problems or to extract a MSS of a formula generally use a classic SAT solver based on the Minisat incremental interface as a black box [13], [14], [15] to perform the search using several consistency tests in a successive way. Concerning MSS extraction through such a solver, we can for instance mention the algorithms BLS [16] and CMP [17]. Let us conclude by noticing that there also exists some softwares dedicated to MSS extraction which use a Max-SAT solver as a black box [11].

III. LOGICAL ENCODINGS FOR ABSTRACT ARGUMENTATION

The literature already contains some examples of encodings which allow the translation of some usual requests of argumentation into propositional logic [18]. We take advantage of the encodings proposed by Besnard and Doutre to propose some approaches allowing to compute the extensions of an argumentation framework, and also to determine if an argument is skeptically or credulously accepted by an argumentation framework. Our encodings are based on the language of the NNF formulae, defined with the usual connectives on the set of Boolean variables $V_A = \{x_{a_i} \mid a_i \in A\}$. Concerning the set of propositional variables, x_{a_i} denotes the fact the argument a_i is accepted by the given argumentation framework. For a matter a readability, we use in the following a_i rather than x_{a_i}.

Let us first recall the encoding of stable semantics defined in [18].

Proposition 1. *Let $F = \langle A, R\rangle$ be an argumentation framework. $E \subseteq A$ is a stable extension of F if and only if E is a model of the formula below.*

$$\Phi^F_{st} = \bigwedge_{a_i \in A} [a_i \Leftrightarrow (\bigwedge_{a_j \in A | (a_j,a_i) \in R} \neg a_j)]$$

In addition to the computation of a single extension, this encoding also allows us to answer other well-known requests for the stable semantics, such that the enumeration of the whole set of extension or the acceptability states of the arguments.

Proposition 2. *Let $F = \langle A, R\rangle$ be an argumentation framework and $a_i \in A$ an argument.*

- *Computing a stable extension of F is equivalent to the computation of a model of Φ^F_{st}.*
- *Enumerating the stable extensions of F is equivalent to the enumeration of the models of Φ^F_{st}.*
- *Determining if a_i is credulously accepted by F with respect to the stable semantics is equivalent to determine the consistency of $\Phi^F_{st} \wedge a_i$.*
- *Determining if a_i is skeptically accepted by F with respect to stable semantics is equivalent to determine if $\Phi^F_{st} \wedge \neg a_i$ is inconsistent.*

Example 1. *When instantiating Φ^F_{st} with the argumentation framework given at Figure 1, we obtain the formula*

$$(a_1 \Leftrightarrow \neg a_2), (a_2 \Leftrightarrow \neg a_1 \wedge \neg a_3 \wedge \neg a_4),$$
$$(a_3 \Leftrightarrow \neg a_4), (a_4 \Leftrightarrow \neg a_3)$$

whose the models are $\{a_1, a_3\}$ and $\{a_1, a_4\}$, which correspond to the stable extensions of F.

As we noticed previously, SAT solvers only deal with CNF formulae. To address this issue, we translate the NNF formula given above into the CNF formula Φ^F_{st} given below.

$$\Psi^F_{st} = \bigwedge_{a_i \in A} (a_i \vee \bigvee_{a_j \in A | (a_j,a_i) \in R} a_j),$$
$$\bigwedge_{a_i \in A} [\bigwedge_{a_j \in A | (a_j,a_i) \in R} (\neg a_i \vee \neg a_j)]$$

Similarly to the stable semantics, Besnard and Doutre proposed a NNF encoding for the complete semantics.

Proposition 3. *Let $F = \langle A, R\rangle$ be an argumentation framework. $E \subseteq A$ is a complete extension of F if and only if E is a model of the formula below.*

$$\Phi^F_{co} = \bigwedge_{a_i \in A} [a_i \Rightarrow (\bigwedge_{a_j \in A | (a_j,a_i) \in R} \neg a_j)$$
$$\wedge (a_i \Leftrightarrow (\bigwedge_{a_j \in A | (a_j,a_i) \in R} \bigvee_{a_k \in A | (a_k,a_j) \in R} a_k))]$$

Then, as well as the stable semantics, we translate this NNF formula into a CNF one to be able to use SAT solvers in order to handle our requests. This time, we add additional variables P_a defined as equivalent to the disjunction of the attackers of the argument a. These auxiliary variables allow us to write a CNF formula Ψ^F_{co}, such that there is a bijection between the models of Φ^F_{co} and the models of Ψ^F_{co}, in a more elegant way than a naive translation from NNF into CNF.

$$\Psi^F_{co} = \bigwedge_{a_i \in A} (\neg a_i \vee \neg P_{a_i}),$$
$$\bigwedge_{a_i \in A} (a_i \vee \bigvee_{a_j \in A | (a_j,a_i) \in R} \neg P_{a_j}),$$
$$\bigwedge_{a_i \in A} (\bigwedge_{a_j \in A | (a_j,a_i) \in R} (\neg a_i \vee P_{a_j})),$$
$$\bigwedge_{a_i \in A} (\neg P_{a_i} \vee \bigvee_{a_j \in A | (a_j,a_i) \in R} a_j),$$
$$\bigwedge_{a_i \in A} [\bigwedge_{a_j \in A | (a_j,a_i) \in R} (P_{a_i} \vee \neg a_j)]$$

Proposition 4. *Let $F = \langle A, R\rangle$ be an argumentation framework and $a_i \in A$ an argument.*

- *Computing a complete extension of F is equivalent to the computation of a model of Φ^F_{co}.*
- *Enumerating the complete extensions of F is equivalent to the enumeration of the models of Φ^F_{co}.*
- *Determining if a_i is credulously accepted by F with respect to complete semantics is equivalent to determine the consistency of $\Phi^F_{co} \wedge a_i$.*
- *Determining if a_i is skeptically accepted by F with respect to complete semantics is equivalent to determine if $\Phi^F_{co} \wedge \neg a_i$ is inconsistent.*

Example 2. *When instantiating Φ^F_{co} with the argumentation framework given at Figure 1, we obtain the formula*

$$(a_1 \Rightarrow \neg a_2) \wedge (a_1 \Leftrightarrow a_1 \vee a_3 \vee a_4),$$
$$(a_2 \Rightarrow \neg a_1 \wedge \neg a_3 \wedge \neg a_4) \wedge (a_2 \Leftrightarrow a_2 \wedge a_4 \wedge a_3),$$
$$(a_3 \Rightarrow \neg a_4) \wedge (a_3 \Leftrightarrow a_3),$$
$$(a_4 \Rightarrow \neg a_3) \wedge (a_4 \Leftrightarrow a_4)$$

whose the models are the models of Φ^F_{st}, together with $\{a_1\}$ and $\emptyset$. Thus, the complete extensions of F are $Ext_{co}(F) = \{\{a_1, a_3\}, \{a_1, a_4\}, \{a_1\}, \emptyset\}$.

The notions of minimality and maximality with respect to $\subseteq$ are not easy to express in propositional logic. So, we simply define a grounded extension (respectively preferred) as a minimal (respectively maximal) model with respect to $\subseteq$ of Φ^F_{co}. We notice that we do not have to compute minimal or maximal models to tackle the grounded. Indeed, applying the unit propagation on Φ^F_{co} – at decision level 0, that is the literals propagated without any assumption – proves enough to compute the grounded extension.

Proposition 5. *Let $F = \langle A, R\rangle$ be an argumentation framework and $a_i \in A$ an argument.*

- *Computing the (only) grounded extension of F is equivalent to compute the literals propagated at decision level 0 in Φ^F_{co}.*
- *Determining if a_i is accepted (both credulously and skeptically) by F with respect to grounded semantics is equivalent to determine if a_i is propagated at decision level 0 in Φ^F_{co}.*

Example 3. *Applying the unit propagation in Φ_{co}^F defined at Example 2 leads to the empty set: grounded semantics applied to F gives $Ext_{gr}(F) = \{\emptyset\}$.*

Computing the preferred extensions of F require a slightly different encoding. We remark that a maximal model of Φ_{co}^F is a MSS of the weighted formula Φ_{pr}^F defined below.

Proposition 6. *Let $F = \langle A, R\rangle$ be an argumentation framework. $E \subseteq A$ is a preferred extension of F if and only if E is a MSS of the weighted formula*

$$\Phi_{pr}^F = \{(\Phi_{co}^F, +\infty), (a_1, 1), \cdots, (a_n, 1)\}$$

It is not necessary to extract a MSS of a Partial Max-SAT instance for each request related to the preferred semantics. Indeed, determining if an argument is credulously accepted is known to be NP-complete for the preferred semantics. It is in fact exactly equivalent to determine if it is credulously accepted for the complete semantics.

Proposition 7. *Let $F = \langle A, R\rangle$ be an argumentation framework and $a_i \in A$ an argument.*

- *Computing a preferred extension of F is equivalent to the computation of a MSS of Φ_{pr}^F.*
- *Enumerating the preferred extensions of F is equivalent to enumerate the whole set of MSSes of Φ_{pr}^F.*
- *Determining if a_i is credulously accepted by F with respect to the preferred semantics is equivalent to determine the consistency of $\Phi_{co}^F \wedge a_i$.*
- *Determining if a_i is skeptically accepted by F with respect to the preferred semantics is equivalent to determine if a_i belongs to each MSS of Φ_{pr}^F.*

Example 4. *Coming back to the argumentation framework F given at Figure 1, Φ_{pr}^F is the weighted formula*

$$\{(\Phi_{co}^F, +\infty), (a_1, 1), (a_2, 1), (a_3, 1), (a_4, 1)\}$$

whose the MSS are $\{a_1, a_3\}$ and $\{a_1, a_4\}$, which are the preferred extensions of F.

IV. CoQuiAAS : Design of the Library

We have chosen the language C++ to implement CoQuiAAS to take advantage of the Object Oriented Programming (OOP) paradigm and its good computational efficiency. First, the use of OOP allows us to give CoQuiAAS an elegant conception, which is well suited to maintain and upgrade the software. Moreover, C++ ensures having high computing performances, which is not the case of some other OOP languages. At last, it makes easier the integration of coMSSExtractor, a C++ underlying tool we used to solve the problems under consideration.

coMSSExtractor [17] is a software dedicated to extract MSS/coMSS pairs from a Partial Max-SAT instance. As coMSSExtractor integrates the Minisat SAT solver [19] – which is used as a black box to compute MSSes – the API provided by coMSSExtractor allows us to use the API provided by Minisat to handle the requests that require a simple SAT solver. This way, CoQuiAAS does not need a second solver to compute the whole set of requests it is attended to deal with.

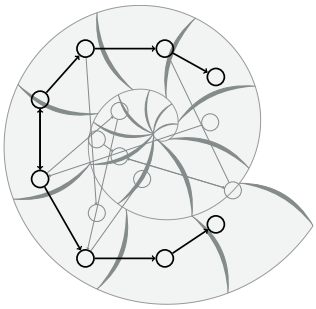

Fig. 2. CoQuiAAS Logo

The core of our library is the interface *Solver*, which contains the high-level methods required to solve the problems. The method `initProblem` makes every required initialization given the input datas. In the case of our approaches, it initializes the SAT solver or the coMSS extractor with the logical encoding corresponding to the argumentation framework, the semantics and the reasoning task to perform. The initialization step depends on the concrete realization of the *Solver* interface returned by the **SolverFactory** class, given the command-line parameters of CoQuiAAS. The method `computeProblem` is used to compute the result of the problem, and `displaySolution` prints the result into the dedicated output stream using the format expected by the *First International Competition on Computational Models of Argumentation*.

The abstract class *SATBasedSolver* (respectively *CoMSSBasedSolver*) gathers the features and initialization common to each solver based on a SAT solver (respectively a coMSS extractor), for instance the method `hasAModel` which returns a Boolean indicating if the SAT instance built from the argumentation problem is consistent or not. Among the subclasses of *SATBasedSolver*, we built *DefaultSATBasedSolver* and its subclasses, which are dedicated to use the API of coMSSExtractor to take advantage of its SAT solver features, inherited from Minisat, to solve the problems. If the user wants to call any other SAT solver rather than coMSSExtractor – as soon as the given semantics is compatible with SAT encodings – a command-line option leads the **SolverFactory** to generate an instance of the class *ExternalSATBasedSolver*, which also extends *SATBasedSolver*. This solver class is initialized with a command to execute so as to call any external software able to read a CNF formula written in the DIMACS format, and to print a solution using the format of SAT solvers competitions. This class allows to execute the command provided to CoQuiAAS to perform the computation related to the problem. This feature enables, for instance, the comparison between the relative efficiency of several SAT solvers on the argumentation instances. The same pattern is present in the coMSS-based part of the library, with the class *CoMSSBasedSolver*, which can be instantiated *via* the default solver *DefaultCoMSSBasedSolver*, which uses coMSSExtractor, or *via* the class *ExternalCoMSSBasedSolver*

to use any external software whose input and output correspond to coMSSExtractor ones, for the pairs request/semantics corresponding to our coMSS-based approaches.

Our design is flexible enough to make CoQuiAAS evolutive. For instance, it is simple to create a solver based on the API of another SAT solver than coMSSExtractor: creating a new class **MySolver** which extends *SATBasedSolver* (and also, the interface, *Solver* which is the root of each solver) and implementing the required abstract methods (`initProblem`, `hasAModel`, `getModel` and `addBlockingClause`) is the only work needed. It is also possible to extend directly the class *Solver* and to implement its methods `initProblem`, `computeProblem` and `displaySolution` to create any kind of new solver. For instance, if we want to develop a CSP-based approach for argumentation-based reasoning, using encodings such that those from [20], we just need to add a new class *CSPBasedSolver* which implements the interface *Solver*, and to reproduce the process which lead to the conception of the SAT-based solvers, but using this time the API of a CSP solver (or an external CSP solver).

Once the solver written, we just need to give an option to the command-line which executes CoQuiAAS, and to update the method `getSolverInstance` in the **SolverFactory**, which knows the set of the command-line parameters (stored in the map `opt`). For instance, the parameter `-solver MySolver` can be linked to the use of the class *MySolver* dedicated to the new solver. The code given below is sufficient to do that.

```
if(!opt["-solver"].compare("MySolver"))
  return new MySolver(...);
```

In the way we conceived the interface *Solver*, it is supposed that a solver is dedicated to a single problem and a single semantics. Thus, it is possible to implement a class which executes a unique algorithm, suited to a single pair (problem,semantics). For instance, [21] describes a procedure which determines if a given argument belongs to the grounded extension of an argumentation framework. We can consider the possibility to implement a class **GroundedDiscussion** which realizes the interface **Solver** to solve the skeptical decision problem under the grounded semantics using this dedicated algorithm-.

This default behaviour of CoQuiAAS does not prevent the implementation of classes able to deal with several request for a given semantics, as soon as the **SolverFactory** returns an instance of the right solver for the considered semantics. Thus, since to the possibility to tackle each problem for a given semantics through a SAT instance (or a MSS problem), we have simplified the design of our solvers using a single class for each semantics, taking advantage of the *template* design pattern. For instance, the method `computeProblem` in the class **CompleteSemanticSolver** is implemented as described at Algorithm 1.

Algorithm 1: `computeProblem`

Data: An argumentation framework F, a problem P, an argument a_i

switch P **do**
- **case** *Give some extension*
  - computeOneExtension(F)
- **case** *Enumerate the extensions*
  - computeAllExtensions(F)
- **case** *Credulous decision*
  - checkCredulousAcceptance(F,a_i)
- **case** *Skeptical decision*
  - checkSkepticalAcceptance(F,a_i)

V. Experimental Results

We have lead some experiments on the benchmarks provided by the organizers of the *First International Competition on Computational Models of Argumentation* to test the solvers before the competition. The first set of test cases contains a family of 20 instances said `real`, whose the number of arguments vary between 5000 and 100 000, and 79 random instances whose the number of arguments vary between 20 and 1000. The second set of test cases contains random instance whose the number of arguments vary between 200 and 400. CoQuiAAS has been executed on computers equipped with 3.0 GHz Intel Xeon processors, with 2.0 GB RAM, and the GNU/Linux distribution CentOS 6.0 (64 bits). The timeout for each instance was set to 900 seconds.

We have given some priority to the study of practical efficiency of our approach for enumeration problems, since the time to enumerate the extensions of an argumentation framework is an upper bound for the time required by the other problems. The results are given at Table II. We have agregated the times by family of instances, and we present here the average times. The symbol "−" indicates that the whole family has reached the timeout ; the other families have been completely solved. These results correspond to the average runtime to initialize the problem (read the AF instance and translate it into a CNF formula) and to solve it.

TABLE II. Average Time by Family to Enumerate the Extensions for each Semantics. Time are given rounded to 10^{-2} seconds. "−" means CoQuiAAS reached the timeout for all instances in the family.

Family	#Inst.	Gr	St	Pr	Co
`rdm20`	25	< 0.01	0.01	< 0.01	< 0.01
`rdm50`	24	< 0.01	< 0.01	< 0.01	< 0.01
`rdm200`	24	< 0.01	0.5	5.32	1.57
`rdm1000`	6	0.25	−	−	−
`real`	20	7.25	7.51	8.55	6.88
`XXX200`	4	< 0.01	0.08	0.04	0.03
`XXX300`	64	0.02	12.53	34.8	21.36
`XXX400`	22	0.01	0.12	0.13	0.08

Our experiments show the efficiency of our approach on the competition instances, except for the instances `rdm1000` which have reached the timeout without being solved, for the stable, complete and preferred semantics. The average time to solve `XXX300` instances with these three semantics is particularly high compared to the average time required by the `XXX400` instances for the same semantics. However, this global comparison hides the fact the difference between these two families are explained by the presence of some particularly hard instances in the family `XXX300`. Some of

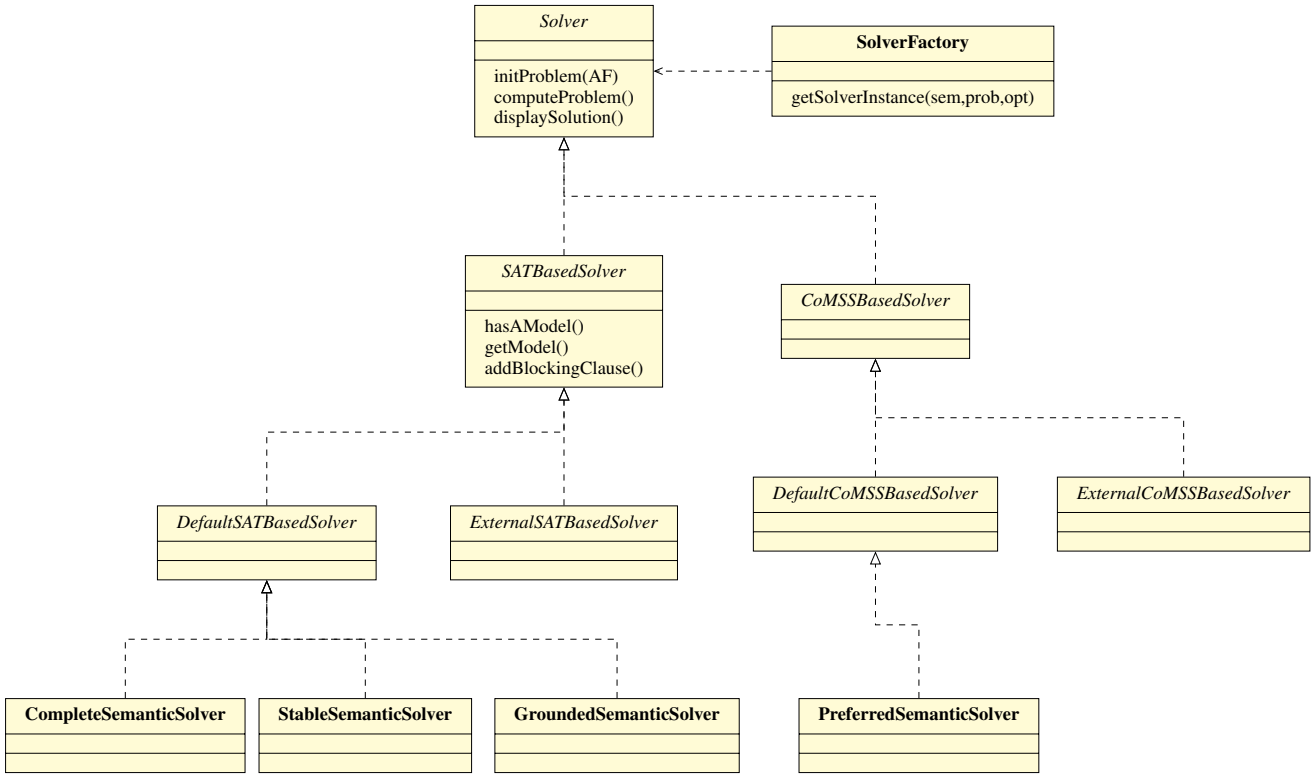

Fig. 3. Simplified class diagram of the solver part of CoQuiAAS

them require several dozen of seconds, and even 280 seconds for the complete semantics and 653 seconds for the preferred semantics; however, the large majority of the instances are solved within a few seconds – 41 instances are solved in strictly less than one second for the complete semantics, and 37 instances for the preferred semantics.

VI. Related Works

The first experimental results, described in the previous section, show that CoQuiAAS is able to deal with large instances. The second interesting question about CoQuiAAS' efficiency is "how does it behave compared to the other existing softwares".

Indeed, several similar approaches have been developped in the recent years. ASPARTIX [22], first, proposes an implementation based on ASP techniques to compute the extensions of an argumentation framework, for many semantics. It does not provide directly a possibility to tackle skeptical and credulous decision. CEGARTIX [23], based on SAT techniques, focuses on the request whose complexity is at the second level of the polynomial hierarchy. Though it is efficient, the current version of this software is far less general than our library, which allows to tackle every usual request for every usual semantics, and which can be easily exteded to work with other semantics. At last, ArgSemSAT [24] is also a SAT-based software, which allows the enumeration of the extension for the usual semantics. As far as we know, none of these softwares has been conceived with an easy integration of other kind of constraint solvers in mind.
These three pieces of software were the most efficient ones to tackle argumentation issues before the competition ICCMA 2015.

Recently, the argumentation community has been interested in developping numerous approaches to solve argumentation problems. This has been motivated by the organisation of the ICCMA 2015: eighteen different solvers participated to the competition (including CoQuiAAS), tackling more or less tracks among the sixteen pairs (semantics,problem). Some of them were updated versions of the existing pieces of software, while many new solvers have been developped for the competition.
Among these solvers, eight are able to tackle the whole range of problems (semantics,tasks). After the aggregation of these eight solvers performances, CoQuiAAS received the award "First Place", thanks to its computational efficiency and its capacity to deal with each semantics and each inference problem of the competition. Roughly speaking, CoQuiAAS is the most efficient software among those which can deal with any argumentation problem. More details can be found on the website of the competition [5].
An interesting remark about the results of the competition is that the three awarded pieces of software are based on SAT technology (CoQuiAAS, ArgSemSAT and LabSATSolver [25]). This confirms that studying logical encodings of argumentation semantics is a promising approach to solve argumentation problems.

VII. Conclusion

In this paper, we present our approaches based on SAT and MSS extraction to solve the most usual inference problems

from an abstract argumentation framework. We put forward the elegant design of our software library CoQuiAAS, which has been developed to be easily maintainable and upgradeable. A first version of our software is available on-line[2]. We also presented some preliminary experimentation results showing that our library seems to be very efficient to solve the benchmarks proposed by the argumentation community.

Several research tracks are planned as future work. First, concerning the inference from an abstract argumentation framework, there exists some semantics which have been proposed after the four "classical" ones of Dung. Developing some approaches for these semantics is an interesting challenge, in particular for the semi-stable [26] and the stage [27] semantics, for which even the credulous acceptance is at the second level of the polynomial hierarchy.
We also plan to extend our work to different extensions of Dung's framework, such as Weighted Argumentation Frameworks [28], Preference-based Argumentation Frameworks [29] or Value-based Argumentation Frameworks [30]. Adding the possibility to work with labellings [31] rather than extensions is also a natural future work.
At last, we want to improve the user's experience. It would be interesting to have a vizualisation tool to work with argumentation frameworks, and to see the result of the requests performed on these frameworks. We plan to develop a Graphical User Interface, based on CoQuiAAS computing engine, to improve the quality of the interactions between the user and the system.

ACKNOWLEDGMENT

This work benefited from the support of the project AMANDE ANR-13-BS02-0004 of the French National Research Agency (ANR).
This work is founded in part by the *Conseil Régional Nord-Pas de Calais* and the FEDER program.

REFERENCES

[1] P. M. Dung, "On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming, and n-person games," *Artificial Intelligence*, vol. 77, no. 2, pp. 321–357, 1995.

[2] P. Besnard and A. Hunter, *Elements of Argumentation*. MIT Press, 2008.

[3] L. Amgoud and N. Hameurlain, "An argumentation-based approach for dialog move selection," in *ArgMAS'06*, 2006, pp. 128–141.

[4] J. Leite and J. Martins, "Social abstract argumentation," in *IJCAI'11*, 2011, pp. 2287–2292.

[5] M. Thimm and S. Villata, "First International Competition on Computational Models of Argumentation (ICCMA'15)," 2015, see http://argumentationcompetition.org/2015/.

[6] S. Coste-Marquis, C. Devred, and P. Marquis, "Symmetric argumentation frameworks," in *ECSQARU'05*, 2005, pp. 317–328.

[7] P. E. Dunne and M. Wooldridge, "Complexity of abstract argumentation," in *Argumentation in Artificial Intelligence*, I. Rahwan and G. R. Simari, Eds. Springer, 2009, ch. 5, pp. 85–104.

[8] W. Dvořák and S. Woltran, "On the intertranslatability of argumentation semantics," *Journal of Artificial Intelligence Research*, vol. 41, no. 2, pp. 445–475, 2011.

[9] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the 3rd annual ACM symposium on Theory of computing*. ACM, 1971, pp. 151–158.

[2]see http://www.cril.univ-artois.fr/coquiaas

[10] A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds., *Handbook of Satisfiability*, ser. Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press, 2009.

[11] M. H. Liffiton and K. A. Sakallah, "Algorithms for computing minimal unsatisfiable subsets of constraints," *Journal of Automated Reasoning*, vol. 40, no. 1, pp. 1–33, 2008.

[12] J. Marques-Silva, F. Heras, M. Janota, A. Previti, and A. Belov, "On computing minimal correction subsets," in *IJCAI'13*, 2013.

[13] G. Audemard, J. Lagniez, and L. Simon, "Improving glucose for incremental SAT solving with assumptions: Application to MUS extraction," in *SAT'13*, 2013, pp. 309–317.

[14] N. Eén and N. Sörensson, "Temporal induction by incremental SAT solving," *Electr. Notes Theor. Comput. Sci.*, vol. 89, no. 4, pp. 543–560, 2003.

[15] J. Lagniez and A. Biere, "Factoring out assumptions to speed up MUS extraction," in *SAT'13*, 2013, pp. 276–292.

[16] J. Bailey and P. J. Stuckey, "Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization," in *PADL'05*, 2005, pp. 174–186.

[17] É. Grégoire, J. Lagniez, and B. Mazure, "An experimentally efficient method for (MSS, CoMSS) partitioning," in *AAAI'14*, 2014, pp. 2666–2673.

[18] P. Besnard and S. Doutre, "Checking the acceptability of a set of arguments," in *NMR'04*, 2004, pp. 59–64.

[19] N. Eén and N. Sörensson, "An extensible sat-solver," in *SAT'03*, 2003, pp. 502–518.

[20] L. Amgoud and C. Devred, "Argumentation frameworks as constraint satisfaction problems," *Annals of Mathematics and Artificial Intelligence*, vol. 69, no. 1, pp. 131–148, 2013.

[21] M. Caminada and M. Podlaszewski, "Grounded semantics as persuasion dialogue," in *COMMA'12*, 2012, pp. 478–485.

[22] U. Egly, S. A. Gaggl, and S. Woltran, "Aspartix: Implementing argumentation frameworks using answer-set programming," in *ICLP'08*, 2008.

[23] W. Dvořák, M. Järvisalo, J. P. Wallner, and S. Woltran, "CEGARTIX: A SAT-Based Argumentation System," 2012, PoS'12.

[24] F. Cerutti, M. Giacomin, and M. Vallati, "Argsemsat: Solving argumentation problems using SAT," in *COMMA'14*, 2014, pp. 455–456.

[25] F. Brons, "LabSAT-Solver: Utilizing caminada's labelling approach as a boolean satisfiability problem," ICCMA'15, 2015.

[26] M. W. A. Caminada, W. A. Carnielli, and P. E. Dunne, "Semi-stable semantics," *Journal of Logic and Computation*, vol. 22, no. 5, pp. 1207–1254, 2012.

[27] B. Verheij, "Two approaches to dialectical argumentation: Admissible sets and argumentation stages," in *FAPR'96*. Universiteit, 1996, pp. 357–368.

[28] P. E. Dunne, A. Hunter, P. McBurney, S. Parsons, and M. Wooldridge, "Weighted argument systems: Basic definitions, algorithms, and complexity results," *Artificial Intelligence*, vol. 175, no. 2, pp. 457–486, 2011.

[29] L. Amgoud and C. Cayrol, "A reasoning model based on the production of acceptable arguments," *Annals of Mathematics and Artificial Intelligence*, vol. 34, no. 1-3, pp. 197–215, 2002.

[30] T. J. M. Bench-Capon, "Value-based argumentation frameworks," in *NMR'02*, 2002, pp. 443–454.

[31] M. Caminada, "On the issue of reinstatement in argumentation," in *JELIA'06*, 2006, pp. 111–123.