Conditional labelling for abstract argumentation

Guido Boella[1], Dov M. Gabbay[2], Alan Perotti[1],
Leendert van der Torre[3], Serena Villata[4]

[1] Dipartimento di Informatica, Università di Torino {guido,perotti}@di.unito.it
[2] King's College London dov.gabbay@kcl.ac.uk
[3] ICR, University of Luxembourg leon.vandertorre@uni.lu
[4] INRIA, Sophia Antipolis serena.villata@inria.fr

Abstract. Agents engage in dialogues having as goals to make some arguments acceptable or unacceptable. To do so they may put forward arguments, adding them to the argumentation framework. Argumentation semantics can relate a change in the framework to the resulting extensions but it is not clear, given an argumentation framework and a desired acceptance state for a given set of arguments, which further arguments should be added in order to achieve those justification statuses. Our methodology, called *conditional labelling*, is based on argument labelling and assigns to each argument three propositional formulae. These formulae describe which arguments should be attacked by the agent in order to get a particular argument *in*, *out*, or *undecided*, respectively. Given a conditional labelling, the agents have a full knowledge about the consequences of the attacks they may raise on the acceptability of each argument without having to recompute the overall labelling of the framework for each possible set of attack they may raise.

1 Introduction

Agents engage in dialogues having as goals to make some arguments acceptable or unacceptable: for instance, *agent A wins the auction* or *agent B is proven guilty.* At each turn, an agent owns a set of possible arguments she can add to the framework: each addiction of further arguments to the framework is called a *move.* Argumentation semantics allow us to relate the introduction of a new argument (a *move*) to the resulting justification status of an argument (the *goal*): for instance, *if you defeat argument α then argument β will be labeled undec.* What is missing is a mechanism for making inferences from goals to moves: suppose an agent wants to make an argument β *undec.* How can she compute which arguments to add in order to achieve this goal? What she can do is to try and simulate the introduction of every possible argument she owns to the framework and then compute β's resulting label, comparing it to her goal. Beside this exhaustive approach there is no way, so far, for an agent to know which move to make in order to achieve her goal. Since reaching a goal may require the insertion of several arguments, the complexity of the exhaustive approach is exponential (cardinality of the powerset) over the number of arguments an agent can add to the framework.

The research question of the paper is:

– How to change an abstract argumentation framework, by introducing new arguments and their associated attacks, in order to have one or more arguments accepted or rejected?

Suppose that two agents, Ag_1 and Ag_2, initiate a dialogue. Ag_1 proposes argument a, as depicted in Figure 1.1. Assume that Ag_2 wants to defeat Ag_1's argument but we have that argument a is *in*, and the only way to have it labelled *out* is to attack it. Thus, Ag_2 attacks a with her new argument b, defeating it. At this turn, as shown in Figure 1.2, it is up to Ag_1 to decide how to proceed in the dialogue. She wants to have her argument a accepted, so she puts forward argument c which attacks b, obtaining the framework in Figure 1.3. In this basic framework, it is straightforward to see which arguments the agents should attack in order to get their arguments accepted. In more complex argumentation frameworks, where also cycles are involved, it is less simple to detect these arguments; consider the framework depicted in Figure 2: it contains loops and multiple attacks. Suppose that an agent wants to defend argument i: it is not intuitive at all to see which potential modifications of the framework allow her to do that. Moreover, if she has a set A^{ag} of arguments she may add to the framework, she may have to run $2^{|A^{ag}|}$ tests in order to find out whether she can defend i, thus making this process' complexity dependent on the number of possible moves she has.

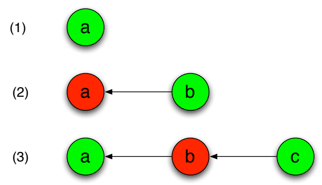

Fig. 1: An argumentation framework with a basic reinstatement.

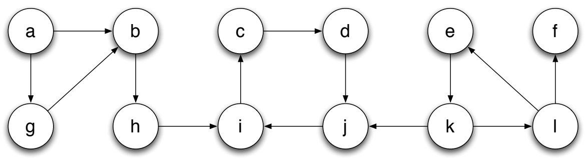

Fig. 2: A more complex argumentation framework.

Thus, the research question breaks down into the following subquestions:

1. What kind of information can we associate to each argument concerning its possible justification statuses depending on the acceptability of other arguments in the framework?
2. How to compute this information in an efficient way?

We deal with abstract argumentation frameworks [3], where the internal structure of the arguments is left unspecified. We are inspired by Caminada's labelling [2], which assigns to each argument a label *in*, *out*, *undec*, and we extend this idea by assigning a triple of propositional formulae, called *conditional lables*, to every argument in the framework. These formulae are a guide in the dialogic process and suggest which move should be made next. For instance, considering the framework of Figure 1.3, the conditional label of a for making it accepted is the emptyset because a is already *in* and no "move" is needed to get it accepted. The conditional label, instead, for making a unaccepted is $a \vee c$, because a can be defeated by defeating a itself or c. Depending on the further arguments at her disposal, an agent may not be able to directly defeat an argument and therefore giving all alternatives is required. Conditional labelling assigns a conditional label to each abstract argument in the framework, even if the framework involves one or more cycles.

The implementation of the algorithm of conditional labelling deals with a number of complexity issues, mostly due to loops in the argumentation frameworks: some preprocessing techniques allow to speed up the performances displayed by a straightforward implementation of the conditional labels' theoretical definition.

In this paper, we are interested in introducing the basic ideas of the conditional labelling and explain it using a number of examples. We do not treat belief revision, and we restrict our examples to grounded semantics.

The paper is organized as follows: Section 2 provides the basic concepts of argumentation theory, Section 3 introduces the conditional evaluation of arguments, Section 4 discusses an algorithmical definition of the conditional labelling and some possible optimizations for implementation. Finally, some conclusions are drawn.

2 Background

We provide the basic concepts and insights of Dung's abstract argumentation [3].

Definition 1. *(Abstract argumentation framework) An abstract argumentation framework is a pair* $\langle \mathcal{A}, \rightarrow \rangle$. $\mathcal{A}$ *is a set of elements called arguments and* $\rightarrow \subseteq \mathcal{A} \times \mathcal{A}$ *is a binary relation called attack. We say that an argument* A_i *attacks an argument* A_j *if and only if* $(A_i, A_j) \in \rightarrow$.

Definition 2. *(Conflict-free, Defence) Let* $C \subseteq \mathcal{A}$. *A set* C *is conflict-free if and only if there exist no* $A_i, A_j \in C$ *such that* $A_i \rightarrow A_j$. *A set* C *defends an argument* A_i *if and only if for each argument* $A_j \in A$ *if* A_j *attacks* A_i *then there exists* $A_k \in C$ *such that* A_k *attacks* A_j.

Definition 3. *(Acceptability semantics) Let C be a conflict-free set of arguments, and let $\mathcal{D} : 2^{\mathcal{A}} \mapsto 2^{\mathcal{A}}$ be a function such that $\mathcal{D}(C) = \{A | C\ defends\ A\}$.*

- *C is admissible if and only if $C \subseteq \mathcal{D}(C)$.*
- *C is a complete extension if and only if $C = \mathcal{D}(C)$.*
- *C is a grounded extension if and only if it is the smallest (w.r.t. set inclusion) complete extension.*
- *C is a preferred extension if and only if it is a maximal (w.r.t. set inclusion) complete extension.*
- *C is a stable extension if and only if it is a preferred extension that attacks all arguments in $\mathcal{A} \setminus C$.*

The concepts of admissibility, as well as those of Dung's semantics are originally stated in terms of sets of arguments. It is equal to express these concepts using argument *labeling*. This approach has been proposed firstly by Jakobovits and Vermeir [4] and then by Caminada [2] with the aim to provide quality postulates for dealing with the reinstatement of arguments. The simplest example of reinstatement is: argument A_1 attacks argument A_2 and argument A_2 attacks argument A_3. We have that argument A_1 reinstates argument A_3, i.e., it makes argument A_3 accepted by attacking the attacker of A_3. In a reinstatement labeling [2], an argument is labeled "in" if all its attackers are labeled "out" and it is labeled "out" if it has at least an attacker which is labeled "in".

Definition 4. *(AF-labeling) Let $\langle \mathcal{A}, \rightarrow \rangle$ be an abstract argumentation framework. An AF-labeling is a total function $lab : \mathcal{A} \rightarrow \{in,\ out,\ undec\}$. We define $in(lab) = \{A_i \in \mathcal{A} | lab(A_i) = in\}$, $out(lab) = \{A_i \in \mathcal{A} | lab(A_i) = out\}$, $undec(lab) = \{A_i \in \mathcal{A} | lab(A_i) = undec\}$.*

Definition 5. *(Reinstatement labeling) Let lab be an AF-labeling. We say that lab is a reinstatement labeling if and only if it satisfies the following:*

- $\forall A_i \in \mathcal{A} : (lab(A_i) = out \equiv \exists A_j \in \mathcal{A} : (A_j \rightarrow A_i \wedge lab(A_j) = in))$ *and*
- $\forall A_i \in \mathcal{A} : (lab(A_i) = in \equiv \forall A_j \in \mathcal{A} : (A_j \rightarrow A_i \supset lab(A_j) = out))$ *and*
- $\forall A_i \in \mathcal{A} : (lab(A_i) = undec \equiv \exists A_j \in \mathcal{A} : (A_j \rightarrow A_i \wedge \neg(lab(A_j) = out)) \wedge \nexists A_k \in \mathcal{A} : (A_k \rightarrow A_i \wedge lab(A_k) = in)$.

3 Conditional labels

Our goal is to enrich each argument with some information about his *vulnerability*, i.e., we want to know how this argument could be successfully (even if indirectly) attacked, defended or made undecided. Our proposal is to attach **three** formulae to each argument, meaning respectively

- Which arguments should I attack in order to have this argument labelled *in*?
- Which arguments should I attack in order to have this argument labelled *out*?

– Which arguments should I attack in order to have this argument labelled *undec*?

Given an argumentation framework $\langle A, R\rangle$, we associate to each argument α three formulae: α^+, α^-, $\alpha^?$. We indicate a generic formula associated to argument α as α^*. The language of the formulae is the same:

Definition 6. *(Language of conditional labels)*

– *if* $\beta \in A$, β° *is a formula.*
– $\top$ *and* $\bot$ *are formulae*
– *if* α_1^* *and* α_2^* *are formulae, also* $\alpha_1^* \wedge \alpha_2^*$ *and* $\alpha_1^* \vee \alpha_2^*$ *are.*

We will refer to α^+ (respectively: α^-, $\alpha^?$) formulae as *green* (*red*, *grey*) formulae.

The interpretation of the formulae is: a *green* formula α^+, if satisfied, guarantees that the related argument α is accepted (labelled *in*). The same holds for *red* formulae for *out* labels and *grey* formulae for *undec* labels respectively. The atoms of those formulae are argument names β° or the special values $\top, \bot$.

– β° means *you have to defeat argument* β (to reach your goal)
– $\top$ means *you do not need to do anything* (to reach your goal)
– $\bot$ means *you can not do anything* (to reach your goal)

Figure 3 provides a simple example of a framework with conditional labels.

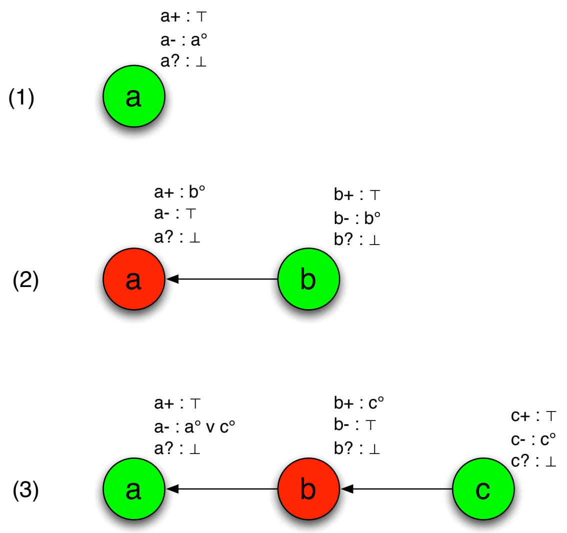

Fig. 3: An argumentation framework with a basic reinstatement and conditional labels

- Figure 3.1: There's no need to modify the framework in order to achieve a's acceptability (it is already labelled *in*) ($a^+ : \top$); to defeat a you have to defeat a ($a^- : a^\circ$), you can not make a undecidable by defeating any combination of the arguments of the framework ($a^? : \perp$).
- Figure 3.2: a can be reinstated defeating b ($a^+ : b^\circ$), a is already *out* ($a^- : \top$); b is already *in* ($b^+ : \top$) and can only be defeated by being directly defeated ($b^- : b^\circ$); no argument can be made undecidable by defeating any combination of the arguments of the framework ($a^?, b^? : \perp$).
- Figure 3.3: a is *in* ($a^+ : \top$) and can be defeated by defeating a itself **or** c ($a^- : a^\circ \vee c^\circ$); b is *out* ($b^- : \top$) and can be reinstated defeating c ($b^+ : c^\circ$); c is in ($c^+ : \top$) and can only be defeated by direct (and successful) attack ($c^- : c^\circ$); no argument can be made undecidable by defeating any combination of the arguments of the framework ($a^?, b^?, c^? : \perp$).

Now we can introduce a more formal definition of what conditional labels are and what can be used for.

Definition 7. *(dnf,makeset)*
Let Γ be a propositional formula. dnf(Γ) is the normalization of Γ in Disjunctive Normal Form.
Let makeset($\bigvee_i \bigwedge_j \alpha_j^i$) $= \{\{\alpha_1^1, \alpha_2^1, .., \alpha_p^1\}, \{\alpha_1^2, \alpha_2^2, .., \alpha_q^2\}, .., \{\alpha_1^n, \alpha_2^n, .., \alpha_m^n\}\}$.

This function translates a dnf-formula in a set of sets of atoms, where each set corresponds to a conjunctive subformula of the dnf-formula in input. For instance, *dnf*$(a^\circ \wedge (b^\circ \vee c^\circ)) = (a^\circ \wedge b^\circ) \vee (a^\circ \wedge c^\circ)$ and *makeset*$((a^\circ \wedge b^\circ) \vee (a^\circ \wedge c^\circ)) = \{\{a, b\}, \{a, c\}\}$.

Definition 8. *($\mathbb{L}(\alpha, \langle A, R\rangle)$)*
Let $\mathbb{L}(\alpha, \langle A, R\rangle)$ be the label of argument α in the framework $\langle A, R\rangle$.

Definition 9. *(defeat)*
Let U be the universe of arguments and $A \subset U$, let $AF = \langle A, R\rangle$ be an abstract argumentation framework and let $\alpha \in U \backslash A$. defeat($\alpha$) $= \{\beta \mid \beta \in A, \mathbb{L}(\beta, \langle A \cup \{\alpha\}, R\rangle) = out\}$.

This function gives information about which arguments β of a framework are defeated inserting a new argument α into it. For instance, considering the framework in Figure 1.c with $A = a$, *defeat*(b)=$\{a\}$, *defeat*(c)=$\{\emptyset\}$. The definition of *defeat* can be easily extended for sets of arguments: it will point out which arguments of a framework are defeated by inserting a set of new arguments. For instance, considering the framework in Figure 1.c with $A = \{a\}$, *defeat*($\{b, c\}$)=$\{\emptyset\}$

A *move* M is the insertion of a set of arguments into the framework: in the previous example, $\emptyset$, $\{b\}$, $\{c\}$ and $\{b, c\}$ are (possible) moves. Applying a move

$M = \{\alpha_1, .., \alpha_n\}$ to a framework $AF = \langle A, R\rangle$ transforms it into a new framework $AF^M = \langle\{A \cup M\}, R\rangle$.

Definition 10. *(js)*
Let js be this function: $js(+) : in$, $js(-) : out$, $js(?) : undec$.

Definition 11. *(Conditional labels' structure)*
A conditional label $\alpha^i : body^i_\alpha$ (where α is an argument, $i \in \{+, -, ?\}$ and $body^i_\alpha$ is a propositional formula) is a relation between a justification status and a set of targets.

- *The justification status is expressed by the head of the label: α^i means that α is labelled $js(i)$.*
- *The set of targets is expressed by the body of the label, and it consists in a set of sets of argument to defeat.*

Definition 12. *(Conditional labels' use)*
Given a framework $AF = \langle A, R\rangle$, an argument $\alpha \in A$ with label $\alpha^i : body^i_\alpha$ and a move M,

$$(defeat(M) \in makeset(dnf(body^i_\alpha))) \Rightarrow \mathbb{L}(\alpha, AF^M) = js(i)$$

This means: when we modify a framework via a move M we can defeat a set of arguments *defeat*(M). If this set is one of the allowed target sets for the conditional label of an argument α (that is, if this set belongs to *makeset*$(dnf(body^i_\alpha))$ for some α, i), then the labelling of α in the resulting framework will be the one expressed by the head of the label α^i (that is, $js(i)$).

In the next sections we will explain how to associate labels to arguments. Problems arise when cycles (loops) are introduced in the framework, since they introduce *undecided* labels and the same argument could be given different labels according to different semantics. In this paper we focus on the *grounded* semantics, since it always allows to compute one single labelling.

4 Creating conditional labels

The formal definition of conditional labels we gave is not constructive and therefore the issue about how to actually compute conditional labels has to be addressed. One of the key aspects of argumentation frameworks is the possibility for arguments to influence their own justification status through loops: this is a *global* property of the framework which is hard to instantiate on a single argument. A first approach could be considering the unfolding of the graph, but this can not be done for two main reasons: first of all, *breaking* the loops causes an irreparable loss of information (and therefore one could end up computing conditional labels for a completely different framework); secondly, the number of unfolding could be exponential over the number of arguments: in this case, the overall complexity is the same of the exhaustive approach (*try all combinations of attacks and see what happens*), thus making the whole process pointless.

Our approach is to assign to each argument a triple of **local** labels (that is, labels created by only taking into account the attackers of the argument) and then use a substitution mechanism to generate the final labels.
The local labels correspond to:

$$a^+ = \bigwedge_{b \ s.t. \ (b,a)\in R} b^-$$

The meaning of this formula is: *in order to ensure a's acceptance, all of a's attackers must be out.*

$$a^- = a^\circ \vee \bigvee_{b \ s.t. \ (b,a)\in R} b^+$$

The meaning of this formula is: *in order to ensure a's rejection, either a is defeated or one of a's attacker is accepted.*

$$a^? = \left(\bigvee_{b \ s.t. \ (b,a)\in R} b^? \right) \wedge \left(\bigwedge_{b \ s.t. \ (b,a)\in R} b^- \vee b^? \right)$$

The meaning of this formula is: *in order to have an argument undecided, at least one of its attackers has to be undecided and all of them must be in or undecided.* Note that this definition of grounded semantics mirrors Dung's original formulation.
The a° in the second formula means *a has to be defeated* and no substitution is required; obviously b^+, b^- and $b^?$ refer to other formulae and have to be substituted to the actual formulae they refer to.

After this initial definition, the substitution process takes place. It consists in substituting the references to other labels to those labels' actual values.

Simplifications need to be specified:

- $\top \vee \alpha \rightsquigarrow \top$ (you either do nothing or do α: doing nothing is more convenient)
- $\bot \vee \alpha \rightsquigarrow \alpha$ (you can either fail or do α: in order to succeed you have to do α)
- $\top \wedge \alpha \rightsquigarrow \alpha$ (you have to both do nothing and α, therefore α)
- $\bot \wedge \alpha \rightsquigarrow \bot$ (you fail and you have to do α: you still fail)
- $\alpha \wedge \alpha \rightsquigarrow \alpha$
- $\alpha \vee \alpha \rightsquigarrow \alpha$
- $\alpha \vee (\alpha \wedge \beta) \rightsquigarrow \alpha$
- $\alpha \wedge (\alpha \vee \beta) \rightsquigarrow \alpha$

Consider again the framework in Figure 1.3 (reinstatement a-b-c). The initial conditional labels are:

- a^+: $\top$, a^-: a°, $a^?$: $\bot$

- $b^+\colon a^-,\ \ b^-\colon a^+ \vee b^\circ,\ \ b^?\colon a^? \wedge (a^? \vee a^-)$
- $c^+\colon b^-,\ \ c^-\colon b^+ \vee c^\circ,\ \ c^?\colon b^? \wedge (b^? \vee b^-)$

Substituting in b^* we get:

- $b^+\colon a^\circ,\ \ b^-\colon \top \vee b^\circ,\ \ b^?\colon \bot \wedge (\bot \vee a^\circ)$

And after simplifying:

- $b^+\colon a^\circ,\ \ b^-\colon \top,\ \ b^?\colon \bot$

Doing the same for c^* we get the conditional labels:

- $a^+\colon \top,\ \ a^-\colon a^\circ,\ \ a^?\colon \bot$
- $b^+\colon a^\circ,\ \ b^-\colon \top,\ \ b^?\colon \bot$
- $c^+\colon \top,\ \ c^-\colon a^\circ \vee c^\circ,\ \ c^?\colon \bot$

The conditional labels give us information about the 'static' Caminada labelling of the arguments and also provides us information about what minimal set of arguments we should defeat in order to assign a certain label to a certain argument.

In case of loops, new problems arise: the substitution mechanism can end up visiting the same node multiple times, so some termination techniques have to be addressed. Consider, for instance, the framework $AF = \langle \{a\}, \{(a,a)\} \rangle$. Computing the conditional labels without termination techniques we obtain:

- $a^+ : a^- = \underline{a^+ \vee a^\circ} = a^- \vee a^\circ = a^+ \vee a^\circ \vee a^\circ \rightsquigarrow \underline{a^+ \vee a^\circ} = ...$
- $a^? : \underline{a^? \wedge (a^? \vee a^-)} \rightsquigarrow a^? = \underline{a^? \wedge (a^? \vee a^-)} \rightsquigarrow ...$

Simplification rules keep the size of formulae under control, but both in a^+ and $a^?$ we end up cycling among the same set of labels without termination. The main consideration is that, according to the definition we have given so far, the substitution process goes on until it reaches unattacked arguments (the only ones which do not require further substitution). But if a framework's component is a loop with no ingoing arcs, this will never happen. Moreover, considering the a^+ label in the previous example, one could notice that both a^+ and a^- appear in the label: this is, intuitively, an unsatisfiable request. Therefore, termination rules have to be applied.

Let $i, j \in \{+, -, ?\}$. If α^i appears in the body of α^j:

- if $i = j = ?$, $\ \alpha^i \rightsquigarrow \top$
- else, $\ \alpha^i \rightsquigarrow \bot$

We express our termination conditions as simplification rules; the meaning is the following: if, substituting in the body of a conditional formula for an argument α, I reach a conditional formula over the same argument, I know α belongs to a loop. So in this case the $a^?$ label is satisfied while a^+, a^- are not: if I found no way to give this argument a *in-out* label navigating whole loop, It is pointless to go through the whole loop again.
Applying these rules to the previous example we get:

- $a^+ : a^- \rightsquigarrow \bot$
- $a^- : a^+ \vee a^\circ \rightsquigarrow \bot \vee a^\circ \rightsquigarrow a^\circ$
- $a^? : a^? \wedge (a^? \vee a^-) \rightsquigarrow a^? \rightsquigarrow \top$

which is exactly what we want to obtain: there's no way to make a *in* ($a^+ : \bot$), a can be directly defeated ($a^- : a^\circ$), a is already *undec* so there's no need to do anything in order to make it *undec* ($a^? : \top$).

Now a few simple examples.
Consider Figure 4.1. The basic labels are:

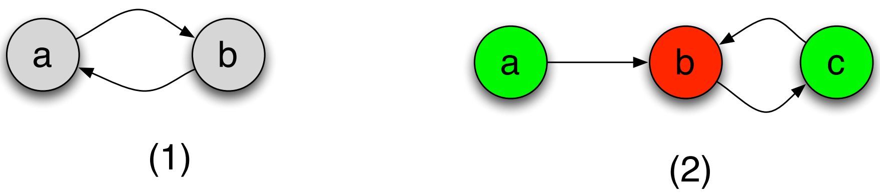

Fig. 4: Basic frameworks

- $a^+ : b^-,\ \ a^- : b^+ \vee a^\circ,\ \ a^? : b^?$
- $b^+ : a^-,\ \ b^- : a^+ \vee b^\circ,\ \ b^? : a^?$

Solving the labels, for a we get $a^+ : b^\circ$, $a^- : a^\circ$, $a^? : \top$, and this is exactly what we want to obtain.
Consider Figure 4.2. The basic labels are:

- $a^+ : \top,\ \ a^- : a^\circ,\ \ a^? : \bot$
- $b^+ : a^- \wedge c^-,\ \ b^- : a^+ \vee c^+ \vee b^\circ,\ \ b^? : (a^? \vee c^?) \wedge (a^- \vee a^?) \wedge (c^- \vee c^?)$
- $c^+ : b^-,\ \ c^- : b^+ \vee c^\circ,\ \ c^? : b^?$

Consider argument b: it is *out*, but can be labelled *in* if we attack both a and c or *undec* if we attack a (thus activating the $b-c$ loop). We compute the conditional labels in the following way:

$$\begin{aligned}
b^+ &: a^- \wedge c^- \\
&= a \wedge (b^+ \vee c^\circ) \\
&= a^\circ \wedge (\bot \vee c^\circ) \\
&\rightsquigarrow a^\circ \wedge c^\circ \ \text{(}b\text{ can be labelled }\textit{in}\text{ by defeating }a\text{ and }c\text{)} \\
b^- &: a^+ \vee c^+ \vee b^\circ \\
&= \top \vee b^- \vee b^\circ \\
&= \top \vee \bot \vee b^\circ \\
&\rightsquigarrow \top \ \text{(no move is required in order to label }b\ \textit{out}\text{)}
\end{aligned}$$

$$
\begin{aligned}
b^? \ &: (a^? \vee c^?) \wedge (a^- \vee a^?) \wedge (c^- \vee c^?) \\
&= \ (\bot \vee b^?) \wedge (a^\circ \vee \bot) \wedge ((b^+ \vee c^\circ) \vee b^?) \\
&\rightsquigarrow (b^?) \wedge (a^\circ) \wedge ((b^+ \vee c^\circ) \vee b^?)) \\
&= \ (b^?) \wedge (a^\circ) \wedge ((\bot \vee c^\circ) \vee \top) \\
&\rightsquigarrow (b^?) \wedge (a^\circ) \wedge (\top) \\
&= \ (\top) \wedge (a^\circ) \wedge (\top) \\
&\rightsquigarrow a^\circ \ \text{(b can be labelled \textit{undec} by defeating a)}
\end{aligned}
$$

Our approach can be decomposed in four phases:

1. associate each argument to three base labels
2. compute conditional labels by substitution
3. find target sets (for instance, by dnf-normalizing the formulae)
4. find a move such that it satisfies a target set of the goal formula.

The biggest challenge lies in step (2), because the substitution process for each formula has the size of the framework as upper bound and the same substitutions take place several times, especially in highly connected frameworks. A support for implementation can be a preprocessing phase of loop detection: loops are the main cause of complexity in label substitution, and knowing which loops an argument belongs to can help propagating activation-deactivation conditions. We call *active* a loop of arguments such that all arguments are labelled *undec* under grounded semantics, *not active* otherwise. Attacking some argument in order to make the arguments of the loop switch from *undec* to *in* or *out* is what we call *deactivating* the loop; we call the opposite process *activating* the loop. For instance, in the framework in Figure 5.1, the *b*-*c*-*e*-*f* loop is active.

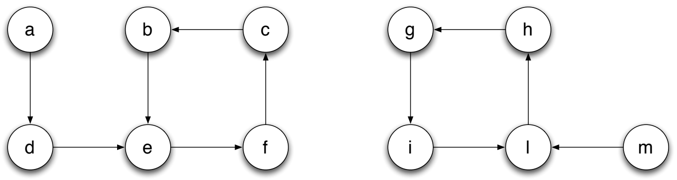

Fig. 5: Frameworks with loops

Some conditional labels are:

- $b^+ : c^\circ \vee e^\circ \vee a^\circ = f^+ = c^- = e^-$
- $b^- : b^\circ \vee f^\circ = f^- = c^+ = e^+$
- $b^? : \top = f^? = c^? = e^?$

According to the definition and the substitution algorithm, all those labels would be computed sequentially. But they just mirror the possible deactivations of the cycle, splitted in two sets according to the position (even/odd) of arguments along the cycle. So detecting the cycle one could compute the conditional labels

of a single argument and then copy them (alternating from green to red formulae according to even/odd path) for each argument in the loop. This also holds for activation conditions for not active loops (like the one in Figure 5.2) and can be easily extended to odd-length cycles.
Therefore, loop detection can be a major improvement for our algorithm's performances.
We developed a methodology based on computing the powers of the adjacency matrix of the graph. Consider the framework in Figure 6: there is a single loop (a) and two bigger ones (b-c and d-e-f), plus edges which do not belong to loops. The adjacency matrix of the framework is represented in Table 1 (let it be m^1).

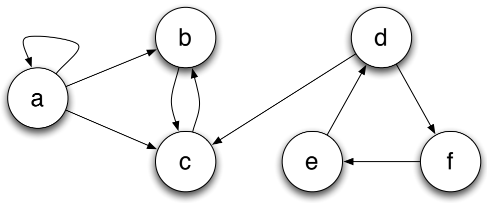

Fig. 6: Frameworks with loops

	a	b	c	d	e	f
a	1	1	1	0	0	0
b	0	0	1	0	0	0
c	0	1	0	0	0	0
d	0	0	1	0	0	1
e	0	0	0	1	0	0
f	0	0	0	0	1	0

Table 1: Adjacency matrix for the framework in Figure 6

In fact, m^1 gives us information about self-loops in the framework: the arguments attacking themselves correspond to the 1 on the main diagonal of m^1: in our example, a. We express this property as $a \in diagonal(m^1)$
But what happens if we compute $m^2 = m^1 * m^1$? On the n-th element of the main diagonal of m^2 we will have a 1 iff the n-th argument is able to reach itself in n steps: that is, if it belongs to a n-deep loop. This is easy to constructively prove by showing how a multiplication between matrices is made. So, for a framework $AF = \langle A, R \rangle$ with $\mid A \mid = n$ we can just compute m^2, m^3, .. m^n to detect all loops.

Table 2: powers of adjacency matrix

m^2	a	b	c	d	e	f
a	1	1	1	0	0	0
b	0	1	0	0	0	0
c	0	0	1	0	0	0
d	0	1	0	0	1	0
e	0	0	1	0	0	1
f	0	0	0	1	0	0

m^3	a	b	c	d	e	f
a	1	1	1	0	0	0
b	0	0	1	0	0	0
c	0	1	0	0	0	0
d	0	0	1	1	0	0
e	0	1	0	0	1	0
f	0	0	1	0	0	1

m^4	a	b	c	d	e	f
a	1	1	1	0	0	0
b	0	1	0	0	0	0
c	0	0	1	0	0	0
d	0	1	1	0	0	1
e	0	0	1	1	0	0
f	0	1	0	0	1	0

Note that in m^1 we detect a, in m^2 b and c, in m^3 d,e,f and no new loop is detected in m^4. Notice that $\alpha \in diagonal(m^p) \Rightarrow \alpha \in diagonal(m^{k*p}), \forall k \in \mathbb{N}$. For instance, in the previous example, $a \in diagonal(m^p) \forall p > 0$ and $b, c \in diagonal(m^2), diagonal(m^4)$.. This redundancy of information can be overcome by cross-checking or by removing the elements on the diagonal of the adjacency matrix before multiplying it again.
The number of arguments is the upper bound for loop depths but can be narrowed down in several ways, for instance by detecting connected components or pruning siphons and traps: in the first case, the deepest loop consist of the maximal values over the number of arguments of each connected component; in the second one, siphons and traps obviously can not be part of loops, thus allowing the lowering of the upper bound.

5 Related Work

Conditional labelling is closely related to the dialogues games [5, 1]. In argumentation theory, such games regulate dialogues where two parties argue about the tenability of one or more claims or arguments, each trying to persuade the other participant to adopt their point of view. Such dialogues are often called persuasion dialogues. Among others, Prakken [5] presents a formal framework for a class of argumentation dialogues, where each dialogue move either attacks or surrenders to a preceding move of the other participant. For instance, each claim, why and since move is viewed as an attacking and each concede move is a surrendering reply.

Amgoud and Hameurlain [1] argue that a strategy is a two steps decision process: i) to select the type of act to utter at a given step of a dialogue, and ii) to select the content which will accompany the act. The first step consists of selecting among all the acts allowed by the protocol, the best option which according to some strategic beliefs of the agent will at least satisfy the most important strategic goals of the agent. The second step consists of selecting among different alternatives, the best one which, according to some basic.

Roth et al. [6] start from two principles: i) the outcome of a dispute depends on the strategies actually adopted by parties, but ii) this does not mean that the outcome can never be predicted because by using game theoretical solution concepts, the actions themselves can often be found. They use defeasible logic in combination with standard probability calculus in order to prove that a defeasible proof holds, on the basis of the probabilities assigned to the premises. This probability of a claim was then interpreted in the game theoretical sense as the payoff for the proponent of the claim.

In comparison with this kind of frameworks, we share the idea that the first step consists in choosing the next move depending on the strategies of the agents. The differences are that we are not interested in providing a complete framework for argumentation dialogues games, we aim at providing a tool which can be used in those systems and which can be integrated with strategies. We do not restrict our framework to deal with two agents, and we extend the well-known argumentation labelling in order to provide a complete information about the argumentation framework on which it is applied.

6 Summary

In this paper, we present a new kind of argument labelling, called conditional labelling. Conditional labelling allows to associate to each argument the information concerning its possible justification statuses, depending on the changes in the framework. In particular, we express this information by means of propositional formulae which express which arguments should be attacked in order to get the desired argument accepted, not accepted, or undecided. While it is quite straightforward to assign those conditional labels in argumentation frameworks without cycles and multiple attacks, it is rather complicated in the general case. When an argumentation framework with cycles is considered, it is possible to have in the conditional label α^* of an argument another α^* because the conditional labelling algorithm, using substitution, looks for all the attackers of the node until it finds the node itself. The conditional labelling allows the agents to avoid the exhaustive search of all the possible combinations in adding new arguments, and decreases exponential complexity this search requires. Loop detection via powers of the adjacency matrix is proposed as a preprocessing mechanism to compute common labels among the arguments in a loop.
Future work addresses several issues: from a purely argumentative perspective, it would be nice to find out how conditional labels can be useful *after* a move: that is, if the previous information can be used to compute new conditional labels after the framework has been modified. Associating a cost concept to moves, our labelling lets agents link action costs to goals' outcomes, and can therefore be used as an underlying mechanism to develop strategies in a game theoretical context.

References

1. L. Amgoud and N. Hameurlain. A formal model for designing dialogue strategies. In H. Nakashima, M. P. Wellman, G. Weiss, and P. Stone, editors, *AAMAS*, pages 414–416. ACM, 2006.
2. M. Caminada. On the issue of reinstatement in argumentation. In M. Fisher, W. van der Hoek, B. Konev, and A. Lisitsa, editors, *JELIA*, volume 4160 of *Lecture Notes in Computer Science*, pages 111–123. Springer, 2006.
3. P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77(2):321–358, 1995.
4. H. Jakobovits and D. Vermeir. Robust semantics for argumentation frameworks. *J. Log. Comput.*, 9(2):215–261, 1999.
5. H. Prakken. Coherence and flexibility in dialogue games for argumentation. *J. Log. Comput.*, 15(6):1009–1040, 2005.
6. B. Roth, R. Riveret, A. Rotolo, and G. Governatori. Strategic argumentation: a game theoretical investigation. In *ICAIL*, pages 81–90. ACM, 2007.