# A Distributed Self-stabilizing Approach to Defeat Status Computation in Argumentation

**P. Baroni**
Dip. di Elettronica per l'Automazione
Università di Brescia,
Via Branze 38, I-25123 Brescia, Italy
baroni@ing.unibs.it

**M. Giacomin**
Dip. di Elettronica per l'Automazione
Università di Brescia,
Via Branze 38, I-25123 Brescia, Italy
giacomin@ing.unibs.it

## Abstract

Argumentation is an approach to practical uncertain reasoning which is receiving an increasing attention in the field of autonomous agents. Since decentralized agent architectures have been advocated by several authors, we propose a distributed approach to argumentation, in which independent processes carry out argumentation activity exploiting local information only. The issue of coordination has been explicitely tackled by devising a general self-stabilizing algorithm for the defeat status computation of arguments.

**Keywords:** Argumentation, Distributed Computing.

## 1 Introduction

In argumentation theory commonsense reasoning dealing with incomplete and uncertain information is modeled as the process of constructing and comparing arguments for propositions. Arguments are constructed from a given set of premises by chaining rules of inference to reach a conclusion, therefore they correspond to "proofs" in standard logic. However, while in standard logic rules of inference correspond to deductive reasons, which logically entail their conclusion, in defeasible reasoning they may represent just provisional reasons, that can be defeated in presence of new information. In such a framework, different arguments may contradict each other, and previously accepted propositions can be discarded in front of counterarguments. The core problem consists then in computing the "defeat status" of the arguments, i.e. determining which ones of them emerge undefeated: their conclusions are the most credible ones and should be believed.

Argumentation is receiving an increasing attention as a technique underlying the realization of intelligent autonomous agents [12, 13, 10]. Besides providing an integrated model for belief revision and defeasible reasoning [3], it takes into account the resource-bounded nature of a real agent, by providing the possibility of acting before the completion of the reasoning process [12]. In this paper, we focus on the use of argumentation within an autonomous agent, and we assume, as usual, that the agent is situated in a dynamic and uncertain world. As the external world changes, the agent is required to track down those features it is interested in, by managing several sensors, which are usually redundant and not completely reliable. In this case, it is reasonable to suppose that the agent will have to manage arguments generally based on multiple premises, having different degrees of reliability. In order to carry out argumentation, the agent is engaged in two main activities:

1. on one hand, it has to develop inferences on the basis of premises continuously updated by perceptual activity and of the conclusions previously drawn;

2. on the other hand, it has to revise the defeat status of its conclusions as a consequence of new conflicting arguments pro-

duced by the inferential activity.

While a decentralized organization has been widely recognized as an appropriate paradigm for autonomous agent architectural design, in all the argumentation approaches in the literature we are aware of, these two activities are delegated to a centralized algorithm exploiting complete global information. As a consequence, these approaches are inherently unable to exploit the potential advantages of a distributed organization, which have been advocated in several works about agent architectures (e.g. [1, 4]). In order to overcome this limitation, we define an approach where argumentation activity is distributed among different processes, which perform their computation asynchronously. For the sake of generality, we do not specify where or when information to make inferences is acquired, and we do not commit either to a specific form of the arguments or to a particular definition of defeat. Moreover, we adopt the "finest grain" model as far as distribution of arguments is concerned, in that each process puts forward exactly one argument. In this model, which approximately corresponds to our AMEs architecture [1], each process can be identified with its argument: in the following, we will refer to either of them without distinction.

Each process executes a program, in order to choose its own status of defeat on the basis of the current status assignments of its defeaters. Since there is no central locus of control, the status of its defeaters can only be acquired by exploiting local exchanges of information. Because of perceptual and inferential activity, the set of the arguments and their defeat relationship are dynamic: new processes can be generated and old processes can be removed corresponding to the addition of arguments (inferential activity) and the addition or removal of premises (perceptual activity). The overall status assignment has to adapt to this dynamics, therefore each process has to monitor its neighbors to track down their possible modifications, and it has to revise its own status accordingly.

Since computation is carried out by asynchronous processes which have only a limited, local view, the issue of coordinating their activity has to be explicitely taken into account. For this reason, we introduce the requirement that the system as a whole has to satisfy a well founded semantics, namely it must assign to the arguments (in a finite amount of time) the same defeat status they would be assigned by a sound centralized algorithm exploiting global information. This requirement corresponds to the property of *self-stabilization* [5]: regardless of the initial defeat statuses assigned to different arguments, each process eventually adjusts its assignment to the correct value. As a consequence, the algorithm can withstand both the addition or removal of processes and changes in the defeat relationship among arguments.

## 2 The Argumentation System

In this section, we present the argumentation system we will use in the paper.

### 2.1 The Argumentation Framework

For the sake of generality, we follow the approach adopted by Dung [6] which completely abstracts from both the internal structure of arguments and the specific notion of defeat between arguments. Arguments are simply conceived as the elements of a set, whose origin is not specified, partially ordered by a binary relation of defeat. Dung's primitive notion is that of argumentation framework:

**Definition 1** *An* argumentation framework *is a pair $AF =< \mathcal{A}, R >$ where $\mathcal{A}$ is a set of arguments, and $R$ is a binary relation on $\mathcal{A}$, i.e. $R \subseteq \mathcal{A} \times \mathcal{A}$.*

Given two arguments $\alpha$ and $\beta$, $(\alpha, \beta)$ is in the relation $R$ if $\alpha$ defeats $\beta$, namely if accepting $\alpha$ as justified prevents accepting $\beta$. However, this does not prevent $\beta$ to defeat $\alpha$ too.

### 2.2 The Semantics

As far as semantics is concerned, argumentation approaches can be distinguished in two classes [15], namely unique-status approaches and multiple-status approaches. The former class defines the defeat status of the arguments in such a way that there is always exactly one possible way to assign them a status, while the multiple-status approach encompasses a set of possible status assignments and considers an argument undefeated if it is justified in all of them.

We adopt the *grounded semantics* [6], which is based on the single status approach. This seems to be more viable in a distributed environment, since it might be difficult for several asynchronous processes to consider different global status assignments and subsequently to evaluate the justification of each argument on that basis. Given an argumentation framework AF $=<\mathcal{A}, R>$, the semantics is defined inductively[1] by means of the following definitions, taken from [12]:

**Definition 2**

- *All arguments of $\mathcal{A}$ are* in *at level* $0$.
- *An argument of $\mathcal{A}$ is* in *at level $n+1$ iff it is not defeated by any argument* in *at level $n$ (otherwise it is* out *at level $n+1$).*

**Definition 3**

- *An argument is* undefeated *(U) iff there is a level $m$ such that for every $n \geq m$, the argument is in at level $n$.*
- *An argument is* defeated *(D) iff there is a level $m$ such that for every $n \geq m$, the argument is out at level $n$.*
- *An argument is* provisionally defeated *(P) iff there is no level $m$ such that the argument is in at all higher levels or out at all higher levels.*

The underlying idea is that an undefeated argument is one which the agent should believe, a defeated argument is one which it should disbelieve, while a provisionally defeated argument is controversial, as for instance in the case in which there is an equally believable counter-argument.

If $\mathcal{A}$ is finite, we will represent an argumentation framework AF $=<\mathcal{A}, R>$ by means of a directed graph, called the *defeat graph* for AF, whose nodes are the arguments of $\mathcal{A}$ and whose edges represent the defeat relation. Moreover, we represent a status assignment to the arguments of $\mathcal{A}$ by means of a *labelled defeat graph*[2]:

**Definition 4** *Given an argumentation framework $AF =<\mathcal{A}, R>$, where $\mathcal{A}$ is finite, a* labelled defeat graph *for $\mathcal{A}$ is a triple $<\mathcal{A}, R, L>$ where $L$ is a function $L : \mathcal{A} \rightarrow \{U, D, P\}$.*

Since Definition 3 determines a univocal defeat status for every argument in $\mathcal{A}$, there is only one labelled defeat graph enforcing the status of defeat prescribed by the grounded semantics: this graph is called the *right* labelled defeat graph.

## 2.3 Some Semantic Properties

In this subsection, we recall some properties relevant to the defeat status of arguments, which will be useful to devise a self-stabilizing algorithm and to prove its correctness.

**Definition 5** [12] *An argument $\alpha$ becomes stably in (or out) at the nth level iff*

1. *either $n = 0$ or $\alpha$ is out (or in, respectively) at the $(n-1)$st level, and*
2. *for all $m \geq n$, $\alpha$ is in (or out, respectively) at the mth level.*

**Proposition 1** *If an argument becomes stably out at the nth level then it has a defeater which becomes stably in at the $(n-1)$st level.*

**Proposition 2** *If an argument $\alpha$ becomes stably in at level $0$ then it has no defeaters. If $\alpha$ becomes stably in at level $n$, where $n > 0$, then all its defeaters become stably out at lower than $n$ levels, and there is a defeater $\beta$ which becomes stably out at level $n-1$.*

An immediate consequence of the above properties is that the status assignment prescribed by the grounded semantics satisfies some constraints, that we call "coherence conditions":

[1]While the grounded semantics can be defined by means of a fixpoint definition [6], here we introduce it in Pollock's style [12], since this makes easier the description of the algorithm in Section 3.

[2]The idea of labelling the defeat graph has been studied, with a different purpose, also in [8].

**Definition 6** *A labelled defeat graph* $< \mathcal{A}, R, L >$ *is* coherent *iff every node* $\alpha \in \mathcal{A}$ *satisfies the following* coherence conditions*:*

- *If* $parents(\alpha) = \emptyset$ *then* $L(\alpha) = U$*.*
- *If* $\forall \beta \in parents(\alpha)\ L(\beta) = D$ *then* $L(\alpha) = U$*.*
- *If* $\exists \beta \in parents(\alpha) \mid L(\beta) = U$ *then* $L(\alpha) = D$*.*
- *If* $\exists \beta \in parents(\alpha) \mid L(\beta) = P$ *and* $\forall \beta \in parents(\alpha)\ L(\beta) \neq U$ *then* $L(\alpha) = P$*.*

**Proposition 3** *A right labelled defeat graph is coherent.*

## 3 Description of the Algorithm

Let us now turn to the problem of devising a distributed self-stabilizing algorithm for the defeat status computation of a dynamic defeat graph.
Given an argumentation framework AF $=< \mathcal{A}, R >$, we consider the defeat graph $\mathcal{G}$ for $\mathcal{A}$ and we assume that each vertex of $\mathcal{G}$ is a sequential process (or node for short). For each directed edge of $R$, we assume a unidirectional communication link. Each node has a set of local variables, including the representation of the status assignment of the associated argument, which can be updated by the node after evaluating its local variables and the local variables of its parents. Following the notation of [9], we will describe the program of each node as

$$^*\,[\mathrm{g}\,[1] \longrightarrow \mathrm{a}\,[1] \,\square\, \ldots \square\, \mathrm{g}\,[m] \longrightarrow \mathrm{a}\,[m]]$$

where each *guard* g [ ] is a boolean function of the variables of process $i$ and the variables of its parents, and each *action* (or *move*) a [ ] updates the variables of process $i$. The symbol $\square$ is called the nondeterminism symbol, and separates the guarded actions: at each iteration, one of the actions whose guards are true is selected for execution, and the algorithm terminates when all the guards are false.
A tentative approach might be arranged by simply letting the nodal processes react to the status of their defeaters according to the coherence conditions. As it can easily be verified, these conditions univocally determine the defeat status of a node given the defeat status of its parents, and, by Proposition 3, they must be satisfied by the right defeat status assignment. However it is easy to see that this approach does not work in case of a cyclic defeat graph: the right status assignment is not always enforced, and the system is not even stable, i.e. there are initial states and particularly "unlucky" process scheduling decisions, that prevent computation from terminating [7].

In order to enforce the right status assignment even in case of cyclic defeat graphs, we refine the basic algorithm by taking into account not just the defeat status, but also the notion of level. According to Proposition 1, if an argument $\alpha$ is defeated, then it becomes stably out at a level $k$, and it has a defeater $\beta$ which becomes stably in at level $k-1$: in a sense, $\beta$ is the 'cause' for $\alpha$ being defeated, therefore it will be denoted as the 'determinant node' for $\alpha$ (of course, there can be several determinant nodes). In case an argument $\alpha$ is undefeated, by Proposition 2 either it has no defeaters, or it becomes stably in at a level $k > 0$. In the latter case, all the defeaters of $\alpha$ become stably out at lower than $k$ levels, and there is a defeater $\beta$ which becomes stably out at level $k-1$: again, $\beta$ is the 'determinant node' for $\alpha$. We introduce for each node $\alpha$ a variable $\mathrm{d}\,[\alpha]$ aimed at representing the level at which $\alpha$ becomes stably in or stably out. If $\beta$ is a determinant node for $\alpha$, $\mathrm{d}\,[\alpha]$ has to be equal to $\mathrm{d}\,[\beta] + 1$.

According to the considerations above, in our approach every node $\alpha \in \mathcal{A}$ maintains two state variables: the first variable, $\mathrm{status}\,[\alpha] \in \{\mathrm{D}, \mathrm{U}, \mathrm{P}\}$, denotes the current defeat status of the node, while the second variable, $\mathrm{d}\,[\alpha] \in \mathbb{N}$, is meaningful only in case $\mathrm{status}\,[\alpha] \in \{\mathrm{D}, \mathrm{U}\}$. Sometimes, we will indicate the state of a node $\alpha$ as $\mathrm{s}\,[\alpha] = [\mathrm{D}\,(x) \mid \mathrm{U}\,(x) \mid \mathrm{P}]$.

In order to describe the algorithm, we introduce, for every node $\alpha$, the following notations:

$$\mathrm{PD}\,(\alpha) = \{\gamma \in \mathrm{parents}(\alpha) \mid \mathrm{status}\,[\gamma] = \mathrm{D}\}$$
$$\mathrm{PU}\,(\alpha) = \{\gamma \in \mathrm{parents}(\alpha) \mid \mathrm{status}\,[\gamma] = \mathrm{U}\}$$
$$\mathrm{PP}\,(\alpha) = \{\gamma \in \mathrm{parents}(\alpha) \mid \mathrm{status}\,[\gamma] = \mathrm{P}\}$$
$$\mathrm{PD}_d\,(\alpha) = \bigcup_{\gamma \in \mathrm{PD}(\alpha)} \{\mathrm{d}\,[\gamma]\}$$
$$\mathrm{PU}_d\,(\alpha) = \bigcup_{\gamma \in \mathrm{PU}(\alpha)} \{\mathrm{d}\,[\gamma]\}$$

```
{Program for node i}

M1: *[ $(\mathrm{parents}(i) = \emptyset) \wedge (\mathrm{s}[i] \neq \mathrm{U}(0))$
         $\rightarrow \mathrm{s}[i] := \mathrm{U}(0)$
M2:    $\Box\, (\mathrm{C}_1[i]) \wedge (\min[\mathrm{PU}_d(i)] < N) \wedge$
         $(\mathrm{s}[i] \neq \mathrm{D}(\min[\mathrm{PU}_d(i)] + 1))$
         $\rightarrow \mathrm{s}[i] := \mathrm{D}(\min[\mathrm{PU}_d(i)] + 1)$
M3:    $\Box\, (\mathrm{C}_1[i]) \wedge (\min[\mathrm{PU}_d(i)] \geq N) \wedge$
         $(\mathrm{s}[i] \neq \mathrm{P}) \rightarrow \mathrm{s}[i] := \mathrm{P}$
M4:    $\Box\, (\mathrm{C}_2[i]) \wedge (\mathrm{s}[i] \neq \mathrm{P}) \rightarrow \mathrm{s}[i] := \mathrm{P}$
M5:    $\Box\, (\mathrm{C}_3[i]) \wedge (\max[\mathrm{PD}_d(i)] < N) \wedge$
         $(\mathrm{s}[i] \neq \mathrm{U}(\max[\mathrm{PD}_d(i)] + 1))$
         $\rightarrow \mathrm{s}[i] := \mathrm{U}(\max[\mathrm{PD}_d(i)] + 1)$
M6:    $\Box\, (\mathrm{C}_3[i]) \wedge (\max[\mathrm{PD}_d(i)] \geq N) \wedge$
         $(\mathrm{s}[i] \neq \mathrm{P}) \rightarrow \mathrm{s}[i] := \mathrm{P}$
]
where

   $\mathrm{C}_1[i] \equiv (\mathrm{PU}(i) \neq \emptyset)$
   $\mathrm{C}_2[i] \equiv (\mathrm{PU}(i) = \emptyset) \wedge (\mathrm{PP}(i) \neq \emptyset)$
   $\mathrm{C}_3[i] \equiv (\mathrm{PU}(i) = \emptyset) \wedge (\mathrm{PP}(i) = \emptyset) \wedge$
              $(\mathrm{PD}(i) \neq \emptyset)$
```

Figure 1: The self-stabilizing algorithm for computing the defeat status of arguments

In order to simplify the description, we will use some simplifying assumptions on the model of computation. In particular, we assume that each node can examine the states of all its parents in a single atomic step. Moreover, we assume the presence of a central scheduler, that arbitrarily selects one of the enabled guards and allows the execution of the corresponding action to be completed, i.e. it allows a move of the node, before any guard is reevaluated. The results we obtain do not depend on these assumptions and can be easily extended to the case of a distributed scheduler and non-atomic examination of the states of the parents.

The algorithm is presented in Figure 1. In the following, we will indicate the moves of the algorithm as Mi, for $i = 1 \ldots 6$, and we will denote the corresponding guards as Gi. The basis of the algorithm is as follows: starting from an arbitrary initial state, each node $\alpha$ continuously examines the assignments of its parents and revises its own status according to the following conditions, which we call *supercoherence conditions*:

- If $\mathrm{parents}(\alpha) = \emptyset$ then $\mathrm{s}[\alpha] := \mathrm{U}(0)$
- If $\mathrm{PU}(\alpha) \neq \emptyset$ then
$$\mathrm{s}[\alpha] = \begin{cases} \mathrm{D}(\min[\mathrm{PU}_d(\alpha)] + 1) \\ \text{ if } \min[\mathrm{PU}_d(\alpha)] < N \\ \mathrm{P} \text{ otherwise} \end{cases}$$
- If $\mathrm{PU}(\alpha) = \emptyset$ and $\mathrm{PP}(\alpha) \neq \emptyset$ then $\mathrm{s}[\alpha] := \mathrm{P}$
- If $\mathrm{PU}(\alpha) = \mathrm{PP}(\alpha) = \emptyset$ and $\mathrm{PD}(\alpha) \neq \emptyset$ then
$$\mathrm{s}[\alpha] = \begin{cases} \mathrm{U}(\max[\mathrm{PD}_d(\alpha)] + 1) \\ \text{ if } \max[\mathrm{PD}_d(\alpha)] < N \\ \mathrm{P} \text{ otherwise} \end{cases}$$

where $N$ is a constant such that $N \geq 0$.

Basically, each node $\alpha$ updates its defeat status as it is prescribed by the coherence conditions, and it updates $\mathrm{d}[\alpha]$ to the proper value $\mathrm{d}[\beta] + 1$, where $\beta$ is the node which $\alpha$ recognizes as its determinant node. The constant $N$ plays a role when $\alpha$ belongs to a cycle of nodes which have to be provisionally defeated in the termination state. For the sake of clearness, let us refer to the simple graph of Figure 2, corresponding to the literature paradigmatic example called Nixon Diamond. We suppose that in the initial state of the algorithm $\mathrm{s}[\alpha] = \mathrm{U}(0)$ and $\mathrm{s}[\beta] = \mathrm{D}(1)$. In this state, only $\alpha$ can make a move: since $\mathrm{d}[\beta] = 1$ and $\beta$, which is the unique defeater of $\alpha$, is recognized as its determinant node, $\alpha$ updates $\mathrm{d}[\alpha]$ to 2. After this move, $\beta$ has guard G2 enabled, therefore it updates $\mathrm{d}[\beta]$ to 3 by M2, and so on. In order to stop this process, one of the nodes has to change its defeat status to P. In our approach, the constant $N$ represents the maximum level at which a node $\gamma$ can become stably in or stably out. As a consequence, $\gamma$ updates $\mathrm{d}[\gamma]$ to a value $x$ only if $x \leq N$, otherwise it changes its status to P by M3 or M6. Turning to the example above, this yields one of the two nodes, say $\alpha$, to be assigned the status of P. As a consequence, $\beta$ changes its status to $\mathrm{s}[\beta] = \mathrm{P}$ by M4, and the algorithm terminates with the right defeat status assignment.

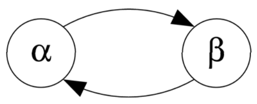

Figure 2: The Nixon diamond example

## 4 Correctness and Complexity

The correctness proof of the algorithm is organized in two parts. First, we prove that the algorithm is partially correct, namely that if it reaches a final state (i.e. eventually any node has its guards false and does not change its state) then the defeat status assignment of the nodes is the right one. Afterwards, we prove the total correctness of the algorithm by showing that, starting from an arbitrary initial state, it reaches its final state in a finite amount of time. This entails the desired self-stabilization property.

### 4.1 Partial Correctness

The proof of partial correctness is based on the following lemma, whose proof (omitted due to space limitations) can be obtained by induction on $m \leq N$.

**Lemma 1** *If the algorithm terminates on a defeat graph $\mathcal{G} =< \mathcal{A}, R >$, after termination the following holds: for every node $\alpha \in \mathcal{A}$ and for every $m \leq N$, $\alpha$ becomes stably in at level $m$ iff $s[\alpha] = U(m)$, and $\alpha$ becomes stably out at level $m$ iff $s[\alpha] = D(m)$.*

**Definition 7** *Given a defeat graph $\mathcal{G} =< \mathcal{A}, R >$, $MAXLEVEL(\mathcal{G})$ is the maximum level at which any argument of $\mathcal{A}$ becomes stably in or stably out.*

As it has been shown in the informal introduction to the algorithm, the constant $N$ has to be an upper bound to $\mathrm{MAXLEVEL}(\mathcal{G})$ in order to be used by a provisionally defeated node to recognize its right status. In fact, if the choice of $N$ is right, then the algorithm is partially correct, as it is proved by the following theorem.

**Theorem 1** *If the algorithm terminates on a defeat graph $\mathcal{G} =< \mathcal{A}, R >$, after termination the following holds. Let $L : \mathcal{A} \to \{U, D, P\}$ be the function such that $\forall \alpha \in \mathcal{A}\ L(\alpha) = status[\alpha]$. If $N \geq MAXLEVEL(\mathcal{G})$, then $< \mathcal{A}, R, L >$ is the right labelled defeat graph for $\mathcal{G}$. Moreover, for every node $\alpha \in \mathcal{A}$, if $status[\alpha] = D$ or $status[\alpha] = U$ then $\alpha$ becomes stably out or stably in, respectively, at level $d[\alpha]$.*

**Proof** If $\alpha$ is defeated or undefeated, it becomes stably in or stably out, respectively, at a level $m$. Since $N \geq \mathrm{MAXLEVEL}(\mathcal{G}) \geq m$, by Lemma 1 the conclusion follows. If $\alpha$ is provisionally defeated, by Lemma 1 $\mathrm{status}[\alpha] \neq \mathrm{D}$ and $\mathrm{status}[\alpha] \neq \mathrm{U}$, therefore it must be the case that $\mathrm{status}[\alpha] = \mathrm{P}$.

The following proposition gives an upper bound to $\mathrm{MAXLEVEL}(\mathcal{G})$, which is essential for the partial correctness of the algorithm.

**Proposition 4** *For every defeat graph $\mathcal{G} =< \mathcal{A}, R >$ with $n$ nodes, $MAXLEVEL(\mathcal{G}) < n$.*

**Proof** First, let us prove that if a node $\alpha$ becomes stably in or stably out at a level $k \geq 1$, then there are $k$ different nodes besides $\alpha$ which become stably in or stably out at different levels lower than $k$, i.e. $\exists\{\beta_0, \beta_1, \ldots \beta_{k-1}\}$ such that $\beta_i$ becomes stably in or stably out at level $i$, for $i = 0, \ldots, k-1$. This statement obviously holds for $k = 1$ and can easily be proved by induction on $k$, recalling Proposition 2 and Proposition 1.
Now, since $\mathcal{G}$ has $n$ nodes, there are no arguments which become stably in or stably out at a level $k \geq n$, therefore $\mathrm{MAXLEVEL}(\mathcal{G}) < n$.

The results of this section are summarized by the following theorem.

**Theorem 2** *Given a defeat graph $\mathcal{G}$ with $n$ nodes, if $N \geq n - 1$ then the algorithm on $\mathcal{G}$ is partially correct.*

### 4.2 Total Correctness

In this section, we show that the algorithm is totally correct, by proving that, starting from an arbitrary initial state, it reaches its final state in a finite amount of time.

In the following, we will say that a node $\alpha$

makes a *k-including move* if $\alpha$ changes its state to $\mathrm{U}(k)$ or $\mathrm{D}(k)$, and we will say that $\alpha$ makes a *k-removing move* if it changes its state from $\mathrm{U}(k)$ or $\mathrm{D}(k)$. We will denote by *k-move* a move which is either k-including or k-removing. It should be noticed that there are moves which are both k-including and k-removing, e.g. when $\alpha$ moves its state from $\mathrm{U}(k)$ to $\mathrm{D}(k)$.

**Definition 8** *At a given instant of time, the computation of the algorithm on a defeat graph $\mathcal{G} = <\mathcal{A}, R>$ is* stable at level m *iff for any $k \leq m$ there will be no more k-moves.*

The proof of the following lemma is omitted for the sake of brevity and can be obtained by induction on $k$.

**Lemma 2** *For every $k \leq N$, the computation becomes stable at level k in a finite amount of time.*

By the lemma above, we can immediately prove the termination of the algorithm.

**Theorem 3** *The algorithm reaches its final state in a finite amount of time.*

**Proof** By Lemma 2, after a finite amount of time the computation becomes stable at level $N$. Then, only those nodes $\gamma$ such that $\mathrm{d}[\gamma] > N$ can make moves, putting $\mathrm{s}[\gamma] = \mathrm{P}$. As a consequence, there are at most $n$ possible moves by all the nodes, which will be exhausted in a finite amount of time.

Combining the partial correctness and termination results, we have proved that our algorithm is self-stabilizing.

**Theorem 4** *Given a defeat graph $\mathcal{G}$ with $n$ nodes, if $N \geq n-1$ then the algorithm on $\mathcal{G}$ is correct.*

### 4.3 Computational Complexity

In order to analyze the computational complexity of the proposed algorithm, we have to introduce the notion of round. A *round* refers to a minimum execution sequence in which each enabled action is taken at least once [9]. Since in our algorithm the guards of a node are mutually exclusive, we define a round as a minimum execution sequence in which each node with an enabled guard is chosen by the centralized scheduler at least once.

The round complexity of our algorithm turns out to be $O(N)$:

**Proposition 5** *The algorithm terminates after at most $N+1$ rounds.*

## 5 Discussion and Conclusions

While algorithmic issues for argumentation have been investigated by means of centralized algorithms (e.g. [14]) or at a multi-agent level (e.g. [10]), in this paper we have considered a finer grain of distribution, and we have presented an argumentation framework in which arguments are constructed by asynchronous processes, which compute their own defeat status on the basis of local information only. We believe that our approach is a useful starting point to put argumentation in line with the evolution of agent architectures towards distribution.

In a previous paper [2], we have presented a different self-stabilizing algorithm for defeat status computation which can handle a limited form of defeat. In the literature, two kinds of defeaters (at least) can be found, namely rebutting defeaters and undercutting defeaters (see [11] for a discussion of their meaning and importance). If one restricts attention on rebutting defeaters, the defeat relation has specific properties, which have been exploited for the definition of the algorithm proposed in [2]. The approach proposed in the present paper is more general, since it does not rely on any specific notion of defeat between arguments and it does not constrain the topology of the defeat graph in any way. However, the present approach is not an 'extension' of [2]: if the underlying argumentation system is restricted to rebutting defeat, the more specific approach is advantageous in two respects:

1. While the algorithm proposed in the present paper terminates in at most $N+1$ rounds, which under the correctness constraint $N \geq n-1$ gives a worst-case

round complexity of $n$ rounds *at least*, the algorithm of [2] terminates after at most $n$ rounds, where $n$ is the number of nodes in the defeat graph.

2. While in the present approach the correctness of the self-stabilizing algorithm relies on a correct choice of the constant value of $N$, the algorithm of [2] does not require any 'guess' of this kind.

As for the second point, it can be proved that in case the number of processes exceeds $N$, the algorithm proposed in the present paper does not assign the status of defeated or undefeated incautiously, but it assigns a 'cautious status' of provisionally defeated to those arguments it can not handle.

A prototypical implementation of the proposed algorithm has been developed in C++ using a centralized scheduler. Work is currently underway to fulfill a distributed implementation, and to integrate it in a software agent architecture based on the Active Mental Entities model [1].

## References

[1] P. Baroni and D. Fogli. Modeling robot cognitive activity through active mental entities. *Robotics and Autonomous Systems*, 30(4):325–349, 2000.

[2] P. Baroni and M. Giacomin. A distributed self-stabilizing algorithm for argumentation. In *Proc. of 2001 International Parallel and Distributed Processing Symposium (IPDPS2001)*, San Francisco, CA, 2001. IEEE Press.

[3] P. Baroni, M. Giacomin, and G. Guida. Extending abstract argumentation systems theory. *Artificial Intelligence*, 120(2):251–270, 2000.

[4] R. Brooks. A robust layered control system for a mobile robot. *J. of Robotics and Automation*, 2(1):14–23, 1986.

[5] S. Dolev. *Self-Stabilization.* MIT Press, Cambridge, MA, 2000.

[6] P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming, and n-person games. *Artificial Intelligence*, 77(2):321–357, 1995.

[7] M. Giacomin. Self-stabilizing distributed algorithms for defeat status computation in argumentation. In *Multi-Agent Systems and Application II: 9th ECCAI-ACAI/EASSS 2001, AEMAS 2001, HoloMAS 2001 (LNAI 2322)*, pages 136–145. Springer-Verlag, Prague, Czech Republic, 2002.

[8] H. Jakobovits and D. Vermeir. Robust semantics for argumentation frameworks. *Journal of Logic and Computation*, 9(2):215–261, 1999.

[9] M. H. Karaata and F. Al-Anzi. A dynamic self-stabilizing algorithm for finding strongly connected components. In *Proc. of 8th ACM Symposium on Principles of Distributed Computing (PODC99)*, page 276, Atlanta, GA, 1999.

[10] S. Parsons, C. Sierra, and N. Jennings. Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3):261–292, 1998.

[11] J. L. Pollock. Defeasible reasoning. *Cognitive Science*, 11(4):481–518, 1987.

[12] J. L. Pollock. How to reason defeasibly. *Artificial Intelligence*, 57(1):1–42, 1992.

[13] J. L. Pollock. *Cognitive Carpentry: A Blueprint for How to Build a Person.* MIT Press, Cambridge, MA, 1995.

[14] H. Prakken. Dialectical proof theory for defeasible argumentation with defeasible priorities (preliminary report). In *Proc. of 4th ModelAge Workshop on Formal Model of Agent (LNAI 1760)*, pages 202–215. Springer-Verlag, 1998.

[15] H. Prakken and G. A. W. Vreeswijk. Logics for defeasible argumentation. In *Handbook of Philosophical Logic.* Kluwer Academic Publishers, Dordrecht, second edition. To appear.