Research Square

Preprints are preliminary reports that have not undergone peer review. They should not be considered conclusive, used to inform clinical practice, or referenced by the media as validated information.

Argumentation-Based Multi-Agent Distributed Reasoning in Dynamic and Open Environments

Helio Monte-Alto

heliohenrique@ufpr.br

Federal University of Paraná

Mariela Morveli-Espinoza

Federal University of Technology – Paraná

Cesar Tacla

Federal University of Technology – Paraná

Research Article

Keywords: multi-agent reasoning, distributed reasoning, contextual reasoning, defeasible logic, argumentation, open environments

Posted Date: July 31st, 2023

DOI: https://doi.org/10.21203/rs.3.rs-3204100/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License. Read Full License

Additional Declarations: Competing interest reported. This work was supported by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) under Grant Agreement CNPq 409523/2021-6.

Version of Record: A version of this preprint was published at Knowledge and Information Systems on April 15th, 2024. See the published version at https://doi.org/10.1007/s10115-024-02101-x.

Argumentation-Based Multi-Agent Distributed Reasoning in Dynamic and Open Environments

Helio Monte-Alto[1*], Mariela Morveli-Espinoza[2†] and Cesar Tacla[2†]

[1*]Federal University of Paraná, Jandaia do Sul, PR, Brazil.
[2*]Federal University of Technology - Paraná, Curitiba, PR, Brazil.

*Corresponding author(s). E-mail(s): heliohenrique@ufpr.br;
Contributing authors: morveli.espinoza@gmail.com; tacla@utfpr.edu.br;
†These authors contributed equally to this work.

Abstract

This work presents an approach for distributed and contextualized reasoning in multi-agent systems, considering environments in which agents may have incomplete, uncertain and inconsistent knowledge. Knowledge is represented by defeasible logic with mapping rules, which model the capability of agents to acquire knowledge from other agents during reasoning. Based on such knowledge representation, it is proposed an argumentation-based reasoning model that enables distributed building of reusable argument structures to support conclusions. Conflicts between arguments are resolved by an argument strength calculation that considers the trust among agents and the degree of similarity between knowledge of different agents, based on the intuition that greater similarity between knowledge defined by different agents implies in less uncertainty about the validity of the built argument. Contextualized reasoning is supported through sharing of relevant knowledge by an agent when issuing queries to other agents, which enable the cooperating agents to be aware of knowledge not known *a priori* but that is important to reach a reasonable conclusion given the context of the agent that issued the query. A distributed algorithm is presented and analytically and experimentally evaluated asserting its computational feasibility. Finally, our approach is compared to related work, highlighting the contributions presented, demonstrating its applicability in a broader range of scenarios, and presenting perspectives for future work.

Keywords: multi-agent reasoning, distributed reasoning, contextual reasoning, defeasible logic, argumentation, open environments

1 Introduction

Distributed reasoning is a central aspect required by many applications proposed in the last decades, including mobile and ubiquitous systems, smart spaces and Ambient Intelligence systems [1–3] – and more recently the Internet of Things (IoT) [4, 5] – and the Semantic Web [6]. In these types of systems, distributed entities, called agents, are able to capture or receive information from their environment and from other agents, and are often able to derive new knowledge or making decisions based on the available knowledge.

We consider multi-agent systems (MAS) in environments in which agents may have imperfect knowledge, which may be incomplete, uncertain or inconsistent knowledge [7]. Incomplete knowledge concerns the possibility of an agent not having a complete knowledge about the environment and thus may require knowledge defined by other agents in order to reach conclusions. For example: assume a robot agent a that can feel the wind but is blind, and that has knowledge expressed by the following rule: "I may conclude that *it is going to rain* (conclusion) if it is *windy* (premise 1) and *cloudy* (premise 2)". Agent a cannot reach the conclusion that it is going to rain by itself, because it cannot find the truth value of premise 2 by itself. But suppose there is another agent b. Then a can ask b if it is cloudy. If b answers positively, then a is able to conclude that it is going to rain.

An agent can also have uncertain and/or conflicting knowledge due to imperfect sensing capabilities or badly intentioned or unreliable agents, which can lead to a global inconsistent state of knowledge. For example, an agent c has a damaged visual sensor. If a asks c whether it is cloudy or not, it is possible that c cannot inform correctly, thus returning an answer that contradicts the answer from agent b. Therefore, a has to decide which agent to believe when receiving both answers. Such decision could be based on a trust degree – or preference – an agent has to another.

Another concern in distributed settings, especially open, dynamic and knowledge-intensive ones, is seeking knowledge from arbitrary sources. For example, suppose d and e entered the environment very recently, so that a does not know what kind of knowledge they hold. Suppose again that a wants to know whether it is going to rain, but agents b and c, which previously used to answer about whether it is cloudy or not, have left the system. Then, the only way to know about it is asking the newcomer agents, d and e, even not knowing whether they hold such knowledge or not.

Furthermore, in many settings, agents have to reason based on heterogeneous knowledge with different degrees of certainty regarding the equivalence of pieces of information coming from different sources. Suppose d answers that there are *clouds in the sky* but e answers that it is *overcast*. Considering that the premise *cloudy* of a's rule have a greater similarity to *overcast* (a sky covered with clouds) than with simply having *clouds in the sky*, a could give preference to e's answer based on the similarity of the information. Another interesting and easy to understand example: suppose a drone robot is in front of an open window, but it does not know if he is able to pass through it only by its own knowledge. Than it asks other robots in the environment, one of which, which is a terrestrial robot, knows that it can pass through an open door. Supposing the drone robot has learned beforehand that doors and windows are

quite similar, it can possibly use the other robot's knowledge to conclude that it can pass through that window.

Finally, a very important feature to have in distributed settings is enabling agents to share relevant information about their current situation, location, activity or object of interest when querying other agents. We call this kind of information *focus*. For example, suppose an agent f in a city distant from where a is. It looks at the sky and sees the clouds, and is also able to feel the wind, but it does not know if this means it is going to rain (it does not have the same rule as a). Thus, it asks other agents it knows. The only agent it is able to talk to at the moment is a, so it sends a message to a asking if it is going to rain. However, a, living in another city, is totally unaware of the weather conditions of f's location. Then, f must send two rules without premises (i.e., facts) stating that there are clouds in the sky and it is windy in the location f is in. Therefore, when a receives the query, it is able to conclude that it is going to rain in f's location, answering it to f. This is, in fact, a kind of contextual reasoning, as defined in classical work such as [8] and [9].

Structured argumentation is a prominent approach to formalize non-monotonic – or defeasible – reasoning by means of the construction and comparison of arguments for and against certain conclusions based on an underlying logic [10], enabling resolution of inconsistent knowledge between agents. Additionally, incomplete knowledge can be represented by means of mapping rules [1, 11], which enables agents to acquire knowledge from other agents in order to support a conclusion. However, there is no work based on argumentation that enables agents to acquire knowledge from arbitrary agents based on the similarity of different pieces of knowledge. Some related work, like [1], assume that agents always know the exact source of a given piece of knowledge and that there is no possibility of knowledge being accepted based on its similarity. Furthermore, no work was found that proposes a solution for the problem of reasoning with focus knowledge not known *a priori* by the cooperating agents in a fully distributed setting. More details about related work are discussed in Section 6.

This paper presents a distributed reasoning approach, including the formal model and an operational algorithm that enable agents to build rule-based structured arguments for and against some conclusion in an open and dynamic environment setting. The distributed algorithm, its analytical and experimental evaluation, as well as further discussions compared to related works, haven't been previously published. The formal model in an earlier stage of development was previously presented in [12], but this paper presents a revised version of it that defines some parts in a more straightforward and flexible way, in addition to useful definitions that enable a more natural connection with the presented algorithm. More specifically, the main differences regarding the formal model are the following: (i) the definitions of localization of focus rules and instantiation of *il-literals*; (ii) the definition of arguments has been simplified and now includes externally dependent arguments as external subarguments, as well as properties that are later used in other definitions; and (iii) the strength calculation formula now considers the concept of direct external subarguments and is based on a weighted average instead of summation. In summary, the previous work focused on the argumentation formalism and semantics, whereas the current work is focused on the operational and practical realization of distributed reasoning, by presenting a

distributed algorithm, complexity issues, optimizations, experimental evaluation and further discussions.

In the model presented in this paper, agents may receive a query about the validity of a given piece of knowledge, but have incomplete knowledge, requiring them to issue new queries to other agents in order to answer the query. Each agent builds arguments based on its local available knowledge augmented with knowledge acquired from other agents, and the strengths of the arguments are calculated based on the trust among agents and the certainty regarding the similarity of the knowledge used to build them, enabling agents to make conclusions based on the comparison of such arguments. This way, reasoning is distributed among agents that concurrently build their arguments and then combine them to support conclusions. We also propose knowledge sharing in the context of a query, which enables agents to cooperate in reasoning about knowledge which is not known *a priori* by them, but is relevant for the agent that issued the query, achieving a kind of contextual reasoning. We therefore demonstrate that our approach enables knowledge representation and reasoning in scenarios that are not possible in related work.

Another relevant contribution regards the way the proposed algorithm builds self-explanatory tree-like argumentation-based structures, which enables not only their use in resolving conflicts and reaching conclusions, but also enables reusing them in other reasoning processes and possibly taking advantage of them for possible future features, such as explaining and learning. Precise representation of reasoning in the form of arguments also enables the use of a broader range of conflict resolution strategies, such as those that take advantage of the tree-like structure of arguments.

We also demonstrate that our model can be viewed as a generalization of the proposal of Bikakis et al. [1] and that the conflict resolution strategy proposed in this work enables contextual reasoning that is more efficient than some of the strategies proposed in their approach [13].

Therefore, in summary, this paper presents four main contributions:

- Argumentation-based fully distributed reasoning model that takes advantage of knowledge acquired from arbitrary sources, relating pieces of knowledge based on their similarity, which enables its application in dynamic and open environments;
- Knowledge sharing in the scope of a query, enabling agents to be aware of relevant contextual knowledge from the point of view of the agent that issued the initial query.
- Strategy for resolving conflicts between arguments that takes into account the point of view of the agent that issued the query and the use of knowledge from indirectly queried agents.
- Operational self-explanatory argumentation-based structure building, which enables storing and reusing arguments for reasoning and possibly other future features such as explaining and learning.

Section 2 presents the formalization of our architecture, including agents and their knowledge representation formalism, as well as the formalization of query focuses and of the problem we are tackling. Section 3 presents the argumentation-based model of

reasoning. Section 4 presents the distributed query answering algorithm that operationally realizes this reasoning model, as well as some properties of the algorithm and optimizations. Section 5 presents some experimental evaluations. Section 6 presents related work and discussion. Finally, conclusions and future work are presented.

2 Architecture and Problem Formalization

Before presenting the argumentation-based model and algorithm, it is necessary to present a concise multi-agent architecture – considering only the features that are relevant for this work – together with the rule-based knowledge representation (Section 2.1) and the problem formalization, including the proposed structures for queries and focus knowledge (Section 2.2).

2.1 Multi-Agent Architecture and Rule-Based Knowledge Representation

The following presents the definition of a MAS in our approach.

Definition 1. *A Distributed Defeasible Reasoning-based Multi-Agent System (DDR-MAS) is defined as a tuple $\mathcal{S} = (Ags, F_Q)$, such that $Ags = \{a, b, c, ...\}$ is the set of agents at a given moment situated in the environment and $F_Q = \{\alpha, \beta, \gamma, ...\}$ is a set of query focuses at a given moment.*

△

The set of query focuses F_Q is necessary to enable the system to have a registry about the query focuses being considered at a given time. Query focuses will be detailed in Section 2.2.

The following presents the definition of an agent.

Definition 2. *An agent $a \in Ags$ is defined as a tuple of the form $(KB_a, P_a, \Theta_a, st_a)$, such that:*

- *KB_a is the agent's knowledge base;*
- *$P_a : Ags \rightarrow [0, 1]$ is a trust function, such that, given two agents b and c in Ags, if $P_a(b) > P_a(c)$, then a trusts b more than c. If $P_a(b) = P_a(c)$, then they are equally trustful. $P_a(a) = 1$, for every $a \in Ags$, i.e., an agent has total trust in itself;*
- *$\Theta_a : V^L \times V^L \rightarrow [0, 1]$, such that V^L is a vocabulary of literals, is a similarity function that maps pairs of literals to a number representing how similar they are;*
- *$st_a : [0, 1]$ is a similarity threshold, used together with Θ_a to decide whether two literals are similar enough or not.*

△

The trust function is used in the resolution of conflicts that may arise from the interaction between agents, as detailed in Section 3, and may result from various mechanisms, such as those related to Trust Theory [14] and dynamic preferences [15]. Therefore, the actual definition of such function is left open for the specific application developer.

The similarity function Θ_a and the similarity threshold st_a are required to enable the matching of knowledge from different agents, and will be formally defined in Definition 4. The process through which this similarity function is determined in each

agent is also out of the scope of this work, and may be defined by some learning process or expert knowledge.

As presented in Definition 3, knowledge bases are sets of rules composed of *labeled literals* (*l-literals*) p of the form $\langle D(p), L(p)\rangle$, where $D(p)$ is called the definer of the *l-literal* p, and $L(p)$ is a literal. The literal $L(p)$ represents atomic information (x) or the negation of atomic information ($\neg x$). For a literal x, its complementary literal is a literal corresponding to the strong negation of x, which can be denoted as $\sim x$. More precisely, $\sim x \equiv \neg x$ and $\sim (\neg x) \equiv x$. For any *l-literal* $p = \langle D(p), x\rangle$, the complementary *l-literal* is denoted as $\sim p = \langle D(\sim p), \sim x\rangle$.

The definer $D(p)$ can be either a direct reference to an agent $a \in Ags$ (in which case the *l-literal* is called a *concrete labeled literal*, or *cl-literal*) or a symbol @ meaning that any known agent in Ags may define the *l-literal* (in which case the *l-literal* is called a *schematic labeled literal*, or *sl-literal*).

Definition 3. *A knowledge base KB_a is a set of rules of the form*

$$r_{ai} : \langle a, x\rangle \hookleftarrow Body(r_{ai})$$

where r_{ai} is the rule identifier, $a \in Ags$, $i \in \mathbb{N}^+$, $\langle a, x\rangle$ is a cl-literal which is the head of the rule, and $Body(r)$ is a set of l-literals, also called body, representing a conjunction of l-literals. The symbol $\hookleftarrow$ is a placeholder for either $\leftarrow$, indicating a strict rule, or $\Leftarrow$, indicating a defeasible rule.

$\triangle$

A rule with empty body, i.e., when $|Body(r)| = 0$, is called a factual rule throughout this paper.

The set of *l-literals* in a KB is called its vocabulary, denoted $V^L = V^{CL} \cup V^{SL}$, where V^{CL} is the set of *cl-literals* and V^{SL} is the set of *sl-literals*, such that $V^{CL} \cap V^{SL} = \emptyset$. Such sets are also closed under negation, i.e., when $\langle D(p), x\rangle \in V^L$, then $\langle D(p), \sim x\rangle \in V^L$.

A *strict rule* is a rule $r_{ai} : \langle a, x\rangle \leftarrow Body(r_{ai})$ such that for every *l-literal* $\langle D(p), y\rangle \in Body(r_{ai})$, $D(p) = a$, i.e., all *l-literals* are local *cl-literals*, since they are defined locally by the very agent a. Similarly, a *local defeasible rule* is a rule $r_{ai} : \langle a, x\rangle \Leftarrow Body(r_{ai})$ such that for every *l-literal* $\langle D(p), y\rangle \in Body(r_{ai})$, $D(p) = a$. The difference is that strict rules cannot be defeated, i.e., they represent absolute truth in the system and are interpreted by classical logic.

An important assumption is that the subset of strict rules in a KB cannot have cycles nor inconsistencies. The maintenance of such consistency is left to a human specialist or by some belief revision/update mechanism, which is out of the scope of this work. In the other hand, defeasible rules have the interesting property of tolerating contradictory rules and chains of rules that lead to infinite cycles or self-defeating lines of reasoning. The way DDRMAS handle such cases is further explained in sections 3 and 4.

A mapping defeasible rule is a rule $r_{ai} : \langle a, x\rangle \Leftarrow Body(r_{ai})$ such that for every *l-literal* $\langle D(p), y\rangle \in Body(r_{ai})$, $D(p) \in Ags$ and there exists at least one *l-literal* $\langle D(p), y\rangle \in Body(r_{ai})$ s.t. $D(p) \in Ags \setminus \{a\}$, i.e., *l-literals* in the body may also be defined by other agents, in which case they are called *foreign cl-literals*. This is what enables agents to function with incomplete knowledge, requiring them to cooperate

with other agents to reach conclusions. The intuition of a mapping rule denoted by $r_{ai} : \langle a, x\rangle \Leftarrow \langle b, y\rangle$ is the following: "If a knows that agent b concludes y, then a considers it a valid premise to conclude x if there is not any adequate contrary evidence".

A schematic rule is a rule $r_{ai} : \langle a, x\rangle \Leftarrow Body(r_{ai})$ such that for every *l-literal* $\langle D(p), y\rangle \in Body(r), D(p) \in Ags \cup \{@\}$, and there exists at least one *l-literal* $\langle D(p), y\rangle \in Body(r)$ s.t. $D(p) = @$ i.e., *l-literals* may be defined by the own agent (local *cl-literals*), other specific agents (foreign *cl-literals*) or by an arbitrary agent (*sl-literal*). The intuition of a schematic rule denoted $r_{ai} : \langle a, x\rangle \Leftarrow \langle @, y\rangle$, is the following: "If a knows that some agent concludes a literal similar enough to y, then a considers it a valid premise to conclude x if there is not any adequate contrary evidence".

An *sl-literal* can be bound (or instantiated) to any *cl-literal* that is similar enough to it. This enables to support a broad range of scenarios, especially those involving approximate reasoning. For example, suppose a scenario in which knowledge is represented by means of an OWL ontology [16]. In this case, a similarity function could be based on axioms that use the `owl:sameAs` property. Other examples could include lexical similarity, such as WordNet [17], which could be used in scenarios involving speech or text recognition, and other syntactic and semantic matchmaking processes, such as Larks [18].

Therefore, we define a *similarity function* between *l-literals*, as well as the concepts of *similarity threshold*, *similar enough* and *similarity degree*, which are all related.

Definition 4. *A* ***similarity function*** $\Theta_a : V^L \times V^L \rightarrow [0, 1]$ *is such that a greater similarity between p and q results in a greater $\Theta_a(p, q)$ value. Given a* ***similarity threshold*** $st_a \in [0, 1]$*, two l-literals p and q are* ***similar enough*** *if $\Theta_a(p, q) \geq st_a$. Given two identical l-literals, the similarity between them will always be 1, i.e., $\forall p \in V^L, \Theta(p, p) = 1$.*

△

The similarity function can be used to define the concept of *instantiated l-literal* (*il-literal*), which includes the information of the similarity between two *l-literals*. The *il-literals* are particularly useful in the definition of arguments in Section 3, and are specially used to express that an *sl-literal* can be instantiated to a given *cl-literal* based on the similarity of their inner literal. Definition 5 defines an *il-literal*.

Definition 5. *Given an l-literal $q = \langle D(q), x\rangle$, a cl-literal $q' = \langle b, x'\rangle$, a similarity function Θ_a and a similarity threshold st_a, where $a, b \in Ags$, such that $\Theta_a(q, q') \geq st$ (i.e., q and q' are similar enough), the function $Inst_a(q, q')$ defines an instantiated l-literal (****il-literal****) as:*

$$Inst_a(q, q') = \langle b, x', \Theta_a(q, q')\rangle$$

△

Example 1. This is inspired by an example given by Anthony Bikakis in his thesis [19]. Suppose there are five mushroom hunters collecting mushrooms in a natural park, each one in possession of a mobile device with a personal agent: Alice's a, Barb's b, Charles's c, Dennis' d and Eric's e. The goal of mushroom hunting for every agent is to help their users to collect edible mushrooms. The mental states, including the knowledge bases, and the trust function of each of these agents are presented in Figure 1.

Alice: a	Barb: b
$\mathbf{KB_a} = \{r_{a1} : \langle a, \neg ed(M)\rangle \Leftarrow \langle a, dc(M)\rangle$ $r_{a2} : \langle a, col(M)\rangle \Leftarrow \langle @, ed(M)\rangle$ $r_{a3} : \langle a, \neg col(M)\rangle \Leftarrow \langle @, \neg ed(M)\rangle \}$ $P_a(b) = 0.4; P_a(c) = 0.6; P_a(d) = 0.2; P_a(e) = 0.8$	$\mathbf{KB_b} = \{r_{b1} : \langle b, \neg ed(M)\rangle \Leftarrow \langle b, hv(M)\rangle$ $r_{b2} : \langle b, col(M)\rangle \Leftarrow \langle @, ed(M)\rangle$ $r_{b3} : \langle b, \neg col(M)\rangle \Leftarrow \langle @, \neg ed(M)\rangle \}$ $P_b(a) = P_b(c) = P_b(d) = P_b(e) = 0.5$
Charles: c $\mathbf{KB_c} = \{r_{c1} : \langle c, ed(M)\rangle \Leftarrow \langle @, avl(M)\rangle \}$ $P_c(a) = P_c(e) = 1; P_c(b) = P_c(d) = 0.4$	**Dennis:** d $\mathbf{KB_d} = \{r_{d1} : \langle d, \neg ed(M)\rangle \Leftarrow \langle @, am(M)\rangle \}$ $P_d(c) = P_d(e) = 1; P_d(a) = P_d(b) = 0.4$
Eric: e $\mathbf{KB_e} = \{r_{e1} : \langle e, spa(M)\rangle \Leftarrow \langle e, hv(M)\rangle, \langle e, pbc(M)\rangle \}$ $P_e(a) = 0.4; P_e(b) = 0.6; P_e(c) = 0.2; P_e(d) = 0.8$	

Fig. 1: Agents' definitions for Example 1.

Alice's (a) has some knowledge about some species, such as the *death cap*, which it knows is not edible (rule r_{a1}). Suppose also that a and b share the same knowledge that, if a mushroom is edible, then they can collect it, and if it is not edible, they cannot. They are also willing to take into account the opinion of any known agent about the edibleness of a mushroom, i.e., if any agent states that a mushroom is edible, then they are willing to accept it to be true if there is no adequate contrary evidence. Such knowledge is thus presented both in KB_a (rules r_{a2} and r_{a3}) and in KB_b (rules r_{b2} and r_{b3}). Barb's (b) also believes that an object is not edible if it has a *volva* (rule r_{b1}).

Charles' (c) believes that a mushroom is edible if it is an *amanita velosa* (rule r_{c1}), but it cannot describe a mushroom of this species. Therefore, it is willing to accept that the mushroom is an *amanita velosa* if any other agent states that a mushroom is similar enough to an *amanita velosa*.

Dennis' (d) believes that an object is not edible if it is an *amanita* (rule r_{d1}). However, Dennis does not know anything about amanitas' characteristics, thus a *sl-literal* is also used in the body of the rule.

Finally, Eric's (e) believes that a mushroom is a *springtime amanita* if it has some properties, such as having a *volva* and a pale brownish cap (r_{e1}). Actually, *springtime amanita* is the same as *amanita velosa*, but we assume the agents in the system are not so certain about it. Therefore, the agents consider a similarity degree of 0.8 between them, thus the *il-literal* $\langle e, spa(M), 0.8\rangle$ can be instantiated based on $\langle @, avl(M)\rangle$ and $\langle e, spa(M)\rangle$. Similarly, a *springtime amanita* is a type of *amanita*, and the agents consider a similarity degree of 0.4 between them. Therefore, the *il-literal* $\langle e, spa(M), 0.4\rangle$ can be instantiated based on $\langle @, am(M)\rangle$ and $\langle e, spa(M)\rangle$. The similarity degree between every *l-literal* with lexically identical literals is defaulted to 1. Therefore, $\langle c, ed(M), 1\rangle$ can be instantiated based on $\langle @, ed(M)\rangle$ and $\langle c, ed(M)\rangle$, and in a similar way $\langle d, \neg ed(M), 1\rangle$ can be instantiated based on $\langle @, \neg ed(M)\rangle$ and $\langle d, \neg ed(M)\rangle$.

□

2.2 Query, Focus Knowledge and Problem Formalization

An agent $a_0 \in A$ (which we call *emitter agent*) is able to issue queries to itself or to other agents. When an agent emits an initial query, which results in subsequent queries to other agents (which we call *cooperating agents*), it is necessary that every such agent becomes aware of specific knowledge related to the focus of the initial query. This is a key feature to enable effective contextual reasoning, because it nourishes the reasoning ability of the cooperating agents with the possibility of considering relevant – and possibly not known *a priori* by these agents – information in the context of the emitter agent, which may be related to its current object of interest, activity, situation or location. The following define a *query focus*, as well as the concept of *focus knowledge*.

Definition 6. *A **query focus** for an l-literal p is a tuple $\alpha = (p, a_0, KB_\alpha^{\mathcal{F}})$, such that α is the unique identifier of the query, a is the agent that created the query and $KB_\alpha^{\mathcal{F}}$ is the focus knowledge base of α.*

△

Definition 7. *A **focus knowledge base** $KB_\alpha^{\mathcal{F}}$ is a set of defeasible rules and **focus rules** of the form $r_{\alpha i}^{\mathcal{F}} : Head(r_{\alpha i}^{\mathcal{F}}) \Leftarrow Body(r_{\alpha i}^{\mathcal{F}})$, s.t. $\exists p \in \{Head(r_{\alpha i}^{\mathcal{F}})\} \cup Body(r_{\alpha i}^{\mathcal{F}})$ s.t. $p = \langle \mathcal{F}, x \rangle$, i.e., at least one of the l-literals of the rule is a focus labeled literal (**fl-literal**).*

△

The idea of *fl-literals* in the format $\langle \mathcal{F}, x \rangle$ is that such knowledge must be temporarily interpreted by each cooperating agent as if it was a locally defined *cl-literal*, i.e., as if each agent defined it. For example, if an agent b receives a query with the following rule $r_1^{\mathcal{F}} : \langle \mathcal{F}, y_1 \rangle \Leftarrow$, it should interpret the rule as if it was $\langle b, y_1 \rangle \Leftarrow$. When the processing of the query is ended, the rule is discarded if it does not originally belong to the knowledge base of b. It is important to note that focus knowledge bases only have defeasible rules (which can also be mapping rules and schematic rules). This is because focus strict rules would require a mechanism to avoid introducing cycles and contradictions to the subset of strict rules of a KB.

It is important to note the difference between an *sl-literal* and an *fl-literal*. The first one is used to represent that an agent requires knowledge defined by an arbitrary external source, which has to be found by querying other agents. The former is used to represent knowledge that was shared with the agent in a way that the agent must consider it as a temporarily locally defined knowledge in order to answer a question in a better informed way.

Therefore, it is useful to define a localization function for focus rules, which requires defining the localization of *fl-literals*, as presented in definitions 8 and 9.

Definition 8 (Localization of *fl-literals*)**.** *Given an fl-literal $p = \langle \mathcal{F}, x \rangle$ and an agent a, the localization function of p in agent a is denoted $Loc(p, a) = \langle a, x \rangle$.*

△

Definition 9 (Localization of *focus rules*)**.** *Given a focus rule $r^{\mathcal{F}} : Head(r^{\mathcal{F}}) \Leftarrow Body(r^{\mathcal{F}})$, a function Loc_Rule that defines a localized rule for a given agent a based on $r^{\mathcal{F}}$ is defined as:*

$$Loc_Rule(r^{\mathcal{F}}, a) = r_a^{\mathcal{F}} : Loc(Head(r^{\mathcal{F}}), a) \Leftarrow \{Loc(q, a) \mid q \in Body(r^{\mathcal{F}})\}$$

$\triangle$

For convenience, it is useful to state some additional definitions. The *local extended knowledge base* of an agent a, denoted $KB_{a\alpha} = KB_a \cup KB^{\mathcal{F}}_{a\alpha}$, where $KB^{\mathcal{F}}_{a\alpha} = \{Loc_Rule(r^{\mathcal{F}}, a) \mid r^{\mathcal{F}} \in KB^{\mathcal{F}}_{\alpha}\}$, represents the local knowledge of a single agent augmented with the focus knowledge of a query focus α. Therefore, the *global extended knowledge* base given a current query focus α is defined as $KB_{\mathcal{S}\alpha} = \bigcup_a^{Ags} KB_{a\alpha}$. Similarly, $V^L_{a\alpha}$ and $V^L_{\mathcal{S}\alpha}$, respectively, denote the local extended vocabulary of an agent a and the global extended vocabulary.

Given these definitions, it is possible to formalize the problem we are solving: "Given an agent a with a query focus $\alpha = (p, a_0, KB^{\mathcal{F}}_{\alpha})$ for an *l-literal* p in a system $\mathcal{S}$, a wants to know whether *l-literal* p is a logical consequence of $KB_{\mathcal{S}\alpha}$ or not". This is a subproblem of the following problem: "Given a system $\mathcal{S}$ considering a query focus $\alpha = (p, a_0, KB^{\mathcal{F}}_{\alpha})$, which *l-literals* in $KB_{\mathcal{S}\alpha}$ are its logical consequences?". In our approach, as further explained in Section 3 and 4, it is possible that an *l-literal* p in $KB_{\mathcal{S}\alpha}$ may be a logical consequence ($KB_{\mathcal{S}\alpha} \models p$) – i.e., p has a $true$ truth-value – or may not be a logical consequence ($KB_{\mathcal{S}\alpha} \not\models p$) – i.e., p has a $false$ truth-value. However, it may be not possible to assert that p is a logical consequence or not, which happens specially when there are fallacious arguments (like circular and self-defeating arguments). In these cases, an *undec* truth-value is assigned to p, as presented in Section 4.

It is important to note that the definition of a global extended KB concerns only the formalization of the model and the problem: in practice, it is not necessary for agents to operationally perform a union of their KBs, as it would result in a centralized solution. The operational solution presented in this paper (Section 4) consists of each agent collaborating to construct arguments based on its own local extended KB, and then sending such arguments to other agents in order to generate larger arguments that enable answering the query presented.

Example 1.1. Given the scenario previously presented, suppose that, at some moment, Alice finds a mushroom m_1 with the following characteristics: It has a volva and a pale brownish cap. Thus, a asks itself a query, such that:

$$\begin{array}{c} p = \langle a, col(m_1)\rangle \quad \alpha = (p, a, KB^{\mathcal{F}}_{\alpha}) \quad KB^{\mathcal{F}}_{\alpha} = \{r^{\mathcal{F}}_{\alpha 1}, r^{\mathcal{F}}_{\alpha 2}\} \\ r^{\mathcal{F}}_{\alpha 1} : \langle \mathcal{F}, hv(m_1)\rangle \Leftarrow \qquad r^{\mathcal{F}}_{\alpha 2} : \langle \mathcal{F}, pbc(m_1)\rangle \Leftarrow \end{array} \tag{1}$$

where α is a new query focus whose focus knowledge base contains the rules that represent the mushroom's characteristics.

When receiving its own query, a cannot reach a conclusion based only on its own knowledge. Thus, it has to query other agents. In each query to another agent, it includes the same query focus α, in order to enable the agents to reason effectively about the mushroom perceived. Therefore, each agent applies the function *Loc_Rule* for every focus rule received. For example, when e receives the query, it temporarily defines internally the rules $\langle e, hv(m_1) \Leftarrow$ and $\langle e, pbc(m_1) \Leftarrow$. Given this focus, a will conclude that Alice can collect the mushroom m_1 because of the argument generated from b with the help of e (see r_{c1} and r_{e1}), though b alone (see r_{b1}) and d with e (see

r_{d1} and r_{e1}) state that it is not edible. The reason for this decision is based on the strength calculation of the arguments generated by each agent and is further discussed in Section 3.

□

Example 1.2. Suppose that, simultaneously to the query focus α from Example 1.1, Barb finds a mushroom (m_2) which she knows is a *death cap*, and wants to know if she should collect it or not. Therefore, its agent emits to itself a query with the following query focus:

$$p : \langle b, col(m_2) \rangle \quad \beta = (p, b, KB_\beta^{\mathcal{F}}) \quad KB_\beta^{\mathcal{F}} = \{r_{\beta 1}^{\mathcal{F}}\} \\ r_{\beta 1}^{\mathcal{F}} : \langle \mathcal{F}, dc(m_2) \rangle \Leftarrow \tag{2}$$

In this case, a new focus rule, β, is originated from Barb. It is important to note that, in this case, the focus knowledge from query focus α is not included. As presented in Section 3, Barb will decide not to collect m_2, since Alice's agent will answer the query indicating that m_2 is not edible.

□

3 Argumentation-based Model

This section is divided into two subsection. Section 3.1 presents the argumentation-based structures built as part of the reasoning process. Section 3.2 presents how different arguments are compared in order to enable conflict resolution.

3.1 Argumentation-based Structure

This section presents how arguments are derived from the local extended KB of each agent given a query focus. An *argument* $A \in Args_{\mathcal{S}\alpha}$, such that $Args_{\mathcal{S}\alpha}$ is the set of arguments that can be generated from a knowledge base $KB_{\mathcal{S}\alpha}$ is an n-ary tree derived from the chaining of rules of $KB_{\mathcal{S}\alpha}$.

Definition 10 formally presents an argument, as well as a set of functions that capture its elements and properties.

Definition 10 (Argument)**.** *Given a global extended knowledge base $KB_{\mathcal{S}\alpha}$, an argument $A \in Args_{\mathcal{S}\alpha}$ based on a rule $r : p \hookleftarrow Body(r) \in KB_{\mathcal{S}\alpha}$ such that $p = \langle a, x \rangle$, with $a \in Ags$, is a n-ary tree whose root node is labeled as p. Given $n = |Body(r)|$, if $n = 0$, then the tree has an only child node labeled $\top$, which is a leaf node. If $n > 0$, then the tree has n child nodes, each one corresponding to an l-literal $q = \langle D(q), y \rangle$ such that $q \in Body(r)$, and each of the child nodes is either labeled as:*

(I) q, if $D(q) = a$ (i.e., q is a local cl-literal) and there exists a rule r' s.t. $Head(r') = q' = \langle a, y \rangle$, or

(II) $Inst_a(q, q')$, if $D(q) \neq a$ (i.e., it is either a foreign cl-literal or an sl-literal) and there exists a rule r' s.t. $Head(r') = q' = \langle D(q'), y' \rangle$, where $D(q') \in Ags$ (i.e., q' is not an sl-literal) and $\Theta(q, q') > st$ (i.e., q and q' are similar enough, or identical).

A node is additionally labeled with a "!" mark in case it introduces a fallacious subargument – either a cyclic or self-defeating subargument. Therefore, given a potential node labeled q, the node is labeled $q!$ if there exists an ancestral node q_{orig} s.t. $D(q) = D(q_{orig})$ and either: $L(q) = L(q_{orig})$, or $L(q) = \sim L(q_{orig})$. A fallacious node is always a leaf node of an argument.

Each node not labeled with a "!" is the root of a subargument of A, which is another argument defined recursively according to this definition.

Each tree edge is labeled with the identifier of the ***rule*** *used to derive the argument, which can also be referred by $Rule(A)$.*

$Conc(A)$ refers to the ***conclusion*** *of an argument A, which is the l-literal that labels its root node.*

$Subs(A)$ refers to the set of ***proper subarguments*** *of an argument A. A proper subargument of an argument A, denoted A', is a proper subtree of the tree that represents the argument A. Proper subarguments are simply called* ***subarguments*** *throughout the text.*

$ExSubs(A) \subseteq Subs(A)$ refers to the set of ***external subarguments*** *of an argument A, i.e., the subarguments of A whose conclusions are il-literals. Formally: $B \in ExSubs(A)$ iff $B \in Subs(A)$ and $Conc(B) = \langle b, x, \theta \rangle$, s.t. $b \in Ags$, $x \in V$ and $\theta \in [0, 1]$.*

$DExSubs(A) \subseteq ExSubs(A)$ refers to the set of ***direct external subarguments*** *of an argument A, which are the external subarguments of A that are not external subarguments of another external subargument of A. Formally, $B \in DExSubs(A)$ iff $B \in ExSubs(A)$ and $\nexists C \in ExSubs(A)$ s.t. $B \in ExSubs(C)$.*

$Prem(A)$ refers to the set of ***premises*** *of an argument A, which are all the nodes of an argument except its root and the nodes labeled with $\top$. If $Prem(A) = \emptyset$, then A is a* ***base argument****. The premises can be expressed as the conclusions of the proper subarguments of an argument, i.e., $Prem(A) = \{q \mid q = Conc(B), B \in Subs(A)\}$.*

$Fall(A) \in \{true, false\}$ indicates whether an argument is ***fallacious*** *(true) or not (false). A not fallacious argument is called a* ***valid*** *argument. An argument is fallacious if any of its l-literals is a fallacious leaf node. Formally, $Fall(A) = true$ iff $\exists q! \in A$ s.t $q \in V^L_{\mathcal{S}\alpha}$.*

$Type(A) \in \{strict, defeasible\}$ indicates whether an argument is ***strict*** *or* ***defeasible****. A strict argument is based on a strict rule, and every of its subarguments are also strict. A defeasible argument is either based on a defeasible rule or at least one of its subarguments is defeasible. Formally, $Type(A) = strict$ iff $Type(Rule(A)) = strict$ and $\forall B \in Subs(A)$, $Type(Rule(B)) = strict$; and $Type(A) = defeasible$ iff $Type(Rule(A)) = defeasible$ or $\exists B \in Subs(A)$ s.t. $Type(Rule(B)) = defeasible$.*

$\triangle$

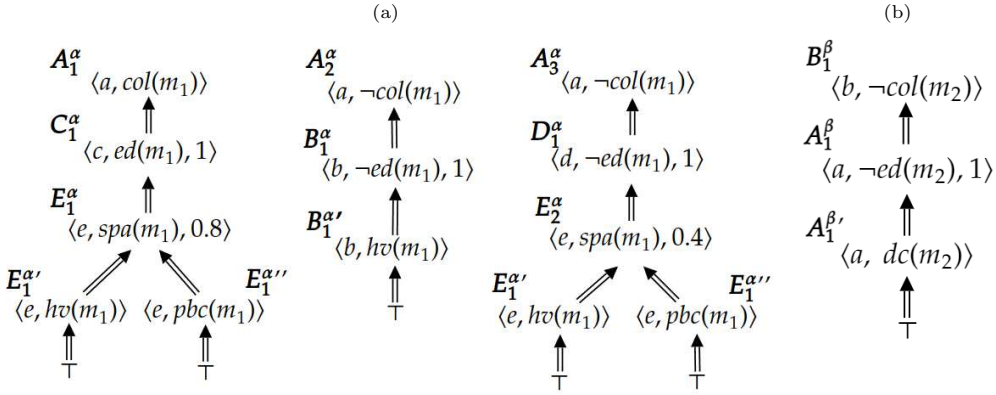
An argument is conventionally named based on the agent that defines it, and the query focus is used in the name as a superscript, given the possibility of having multiple query focuses. For example, if the conclusion of an argument is defined by an agent a in a query focus α, then the argument will be called A_i^α, with $i \in \mathbb{N}^+$. If it is an argument defined by an agent b in a query focus β, it will be called B_i^β.

A graphical representation is given in the examples to follow (see Figure 2). We opted to represent the edges as if they were directed to the conclusion of the argument,

so as to remind the rules' arrows. We also use doubled lines to represent edges from arguments based on defeasible rules in order to differentiate them from arguments based on strict rules.

Continuation of Example 1.1. Figure 2(a) shows the derived arguments that conclude $\langle a, col(m_1)\rangle$ and $\langle a, \neg col(m_1)\rangle$. In addition to these, arguments based on the rules r_{b2} e r_{b3} can also be derived, but they are omitted since they are almost identical to arguments A_1^α, A_2^α and A_3^α. The labels of the edges were also omitted since it is easy to see which arguments are derived from which rules (see Figure 1 and Equation 1).

Fig. 2: Set of arguments of Example 1.1 (a) and 1.2 (b).



Note that from schematic rule r_{a2} the argument A_1^α is formed by instantiating the *sl-literal* $\langle @, ed(m_1)\rangle$ with the conclusion of C_1^α. Similarly, A_2^α and A_3^α are formed from the rule r_{a3}, having its *sl-literals* instantiated with the conclusions of the subarguments B_1^α and D_1^α, respectively. The arguments B_1^α, E_1^α and E_2^α are simply originated from their local rules and the localized focus rules.

Argument C_1^α is based on r_{c1} and formed by instantiating the *sl-literal* $\langle @, avl(m_1)\rangle$ with the head of r_{e1}, from which the subargument E_1^α is created.

It is interesting to note that without sharing and localizing the focus knowledge of the query focus α, none of these arguments could be derived, since b and e would not be aware of the mushroom m_1 and its characteristics. This demonstrates how a kind of contextualized reasoning is allowed in DDRMAS.

□

Continuation of Example 1.2. From query focus β, the arguments derived are shown in Figure 2(b).

In this case, argument B_1^β is based on r_{b1} by instantiating the *sl-literal* $(@, \neg ed(m1))$ with the conclusion of A_1^β, which is based on r_{a1}. Once more, this is only possible because $r_{\beta 1}^{\mathcal{F}}$ is shared with the query focus β, making agent a aware of the fact that m_2 is a *death cap*.

It is interesting to note that this set of arguments is different from the arguments derived with query focus α and that both the queries do not interfere with each other.

□

3.2 Comparison of Arguments and Conflict Resolution

To resolve conflicts between arguments with contradictory conclusions, we use argumentation semantics. In this work, we use an adapted version of the argumentation semantic for Defeasible Logic (DL) proposed by [20] and extended by [1]. A *defeat* relation between arguments is proposed, from which the notions of acceptable, justified and rejected arguments are derived. This work extends the mentioned semantic specifically by defining an argument strength calculation which is used in the defeat relation. All other definition remain true to the original definitions and are only briefly presented in Section 4.

Definitions 11 and 12 present the attack and defeat relations:

Definition 11 (Attack)**.** *An argument A_i attacks a (local or distributed) defeasible argument A_j, denoted $A_i \mathcal{A} A_j$, iff their conclusions are complementary. Formally, let $\mathcal{A} \subset Args \times Args$ be the relation. Then $A_i \mathcal{A} A_j$ iff $Conc(A_i) = p$ and $Conc(A_j) = \sim p$.*

△

Definition 12 (Defeat)**.** *An argument A_i defeats an argument A_j, denoted $A_i \mathcal{D} A_j$, iff A_i attacks A_j, and A_j is not stronger than A_i. Formally, let $\mathcal{D} \subset Args \times Args$ be the relation and $StArg : Args \rightarrow [0, 1]$ be a argument strength function. Then $A_i \mathcal{D} A_j$ iff $A_i \mathcal{A} A_j$ and $StArg(A_i) \geq StArg(A_j)$.*

△

Definition 13 presents the definition of the strength of an *il-literal*, which is then used in the definition of strength of an argument (Definition 14).

Definition 13 (Strength of an *il-literal*)**.** *The strength of an il-literal $q_{inst} = (b, x, \theta)$ from the point of view of an agent a is $StIL(q_{inst}, a) = P_a(b) \times \theta$.*

△

Definition 14 (Argument Strength)**.** *$StArg : Args \rightarrow [0, 1]$ is a function on an argument A to values between 0 and 1, where 0 is the smallest possible strength value and 1 is the greatest possible strength value, defined as follows:*

$$StArg(A) = \begin{cases} 1 & \text{if } DExSubs(A) = \emptyset \\ \dfrac{\sum_{A' \in DExSubs(A)} StIL(Conc(A'), D(Conc(A))) \times StArg(A')}{|DExSubs(A)|} & \text{if } DExSubs(A) \neq \emptyset \end{cases} \tag{3}$$

△

The strength of an *il-literal* defines the basic element to calculate the strength of an argument. Intuitively, it can be said that the strengths of the *il-literals* represent the strengths of the bindings between the chained arguments coming from different agents, which are represented by external subarguments (see Definition 10 and Example 1.1). This strength consider not only the similarity degree (θ) between the knowledge defined by each different agent, but also the trust an agent has in each other ($P_a(b)$).

Considering that, the strength of an argument consists of a weighted average, by means of summing the strengths of its direct external subarguments multiplied by the

strength of the *il-literals* that bind the argument to each direct external subargument, all divided by the amount of direct external subarguments. This way, the strength of an argument is a function of the strengths of its direct external subarguments, which are arguments derived by other agents (see Definition 10). In other words, the similarity between knowledge from different agents and the trust between the agents is what defines the strength of an argument.

An interesting feature of this formula is that it "weakens" an argument when there is indirect trust between agents, which makes agents more skeptical to "third-party" arguments, i.e., arguments that are external subargument, but not direct external subarguments. This is because the calculus recursively depends on the strength of the direct external subarguments of the argument A ($DExSubs(A)$), inducing the multiplication of the strengths of the *il-literals* that are the conclusions of these subarguments. Since the range of the *il-literals* and argument strength function is $[0, 1]$, it then follows that the strength values corresponding to these indirect dependencies, when multiplied, result in reduced values when they are less than 1. This is demonstrated through the following example.

Continuation of Example 1.1. As illustrated in Figure 2, there are some defeasible arguments with contradictory conclusions: A_1^α vs. A_2^α and A_3^α. That means they attack each other: A_1^α attacks A_2^α and A_3^α and vice versa. One can therefore calculate the strengths of these arguments and conclude that A_1^α is stronger than A_2^α and A_3^α, thus defeating both, while A_2^α and A_3^α do not defeat A_1^α.

$$StArg(A_1^\alpha) = \frac{StIL(\langle c, ed(m_1), 1\rangle, a) \times StArg(C_1^\alpha)}{1}$$
$$= P_a(c) \times 1 \times \frac{StIL(\langle e, spa(m_1), 0.8\rangle, c) \times StArg(E_1^\alpha)}{1} = 0.48$$

$$StArg(A_2^\alpha) = \frac{StIL(\langle b, \neg ed(m_1), 1\rangle, a) \times StArg(B_1^\alpha)}{1} = P_a(b) \times 1 = 0.4$$

$$StArg(A_3^\alpha) = \frac{StIL(\langle d, \neg ed(m_1), 1\rangle, a) \times StArg(D_1^\alpha)}{1}$$
$$= P_a(d) \times 1 \times \frac{StIL(\langle e, spa(m_1), 0.4\rangle, d) \times StArg(E_1^\alpha)}{1} = 0.08$$

It is interesting to note, in this case, that although A_1^α was calculated as stronger, it still had a relatively low strength value due to the indirect dependence of A_1^α to E_1^α, as it was necessary to multiply the trust of Alice in Charles (0.6) to the similarity of 0.8 between springtime amanita ($spa(m_1)$) and *amanita velosa* ($avl(m_1)$. Charles' perfect trust in Eric (1) kept the strength from going even lower. In contrast, in the case of A_2^α, although Alice's trust in Barb is less than 0.4, the argument has not much less strength than A_1^α, since it only depends directly from a subargument defined by another agent.

Therefore, in this case, $A_1^\alpha \mathcal{D} A_2^\alpha$ and $A_1^\alpha \mathcal{D} A_3^\alpha$, but $A_2^\alpha \not\mathcal{D} A_1^\alpha$ and $A_3^\alpha \not\mathcal{D} A_1^\alpha$.

□

4 Distributed Query Answering Algorithm

This section presents a distributed algorithm for evaluating queries between agents based on the proposed knowledge representation and argumentation-based model.

This algorithm produces results based on the ambiguity-blocking semantics for Defeasible Logic (DL) proposed by [20] and extended by [1]. The reader is referred to these papers in order to have a deeper understanding of such semantics, but it is important to briefly define the concepts of "justified", "supported", "undercut" and "rejected" arguments. The *set of justified arguments* is incrementally and monotonically built, starting with strict arguments, then arguments that are not defeated or are defended by the already justified arguments, and so on until no further arguments can be justified. An argument A is *justified* if it is strict, or if it is supported by a set of already justified arguments J, and every argument that defeats A is undercut by the set of justified argument J (i.e., A is defended by J). An argument A is *supported* by a set of arguments J if every subargument of A is in J. An argument A is *undercut* by a set of arguments J if there is an argument B which is supported by J that defeats a subargument of A. Finally, an argument A is *rejected* if it is not strict and either a subargument of A is already rejected, or A is defeated by an argument supported by the set of justified arguments.

The specific reasoning problem that is solved by the algorithm is stated as follows: *Given a DDRMAS $\mathcal{S}$ and a query focus α for an l-literal p sent to agent a, compute the truth value of p based on whether justified arguments exist for some l-literal similar enough to p or not* . The algorithm produces an answer with three components: (i) a truth-value $tv_p \in \{true, false, undec\}$ for the *l-literal* p queried; (ii) a set of arguments that conclude *l-literals* similar enough to p; and (iii) a set of arguments that conclude *l-literals* similar enough to $\sim p$. This way, the agent that receives an answer also receives the arguments that support and refute p, so that it can perform conflict resolution from its own point of view.

The truth-value *true* implies that there exists an argument for a *l-literal* p' similar enough to p which is justified in $KB_{\mathcal{S}\alpha}$ (the global extended knowledge from system $\mathcal{S}$ with focus knowledge from the query α), $false$ implies that all arguments for every p' similar enough to p are rejected in $KB_{\mathcal{S}\alpha}$, and *undec* implies that there are not any argument for any p' similar enough to p that is justified, but there are arguments for some p' similar enough to p that are neither justified nor rejected in $KB_{\mathcal{S}\alpha}$. This last case occur when there are fallacious arguments involved.

The arguments in the algorithm have the same structure as the arguments presented in Section 3, having, in addition, three *boolean* labels: J, indicating that the argument has already been identified as justified; R, indicating that the argument has already been identified as rejected; and $SuppJ$, indicating that the argument is supported by a set of justified arguments. These three properties play a fundamental role in comparing the sets of arguments for and against p, in order to correctly implement the DL semantics [1, 20]. In fact, such labels (or flags) enable the implementation of the DL semantics in a labeling-based fashion, similar to [21], which proposes labeling semantics for abstract argumentation frameworks. In summary, instead of incrementally building the sets of justified and rejected arguments, this labeling-based approach traverses an argument in a post-order fashion labeling each argument as $SuppJ$, J

or R. When all arguments are labeled, we gather the set of J-labeled arguments as the justified arguments, and the set of R-labeled arguments as the set of rejected arguments.

It is also assumed that each argument has a numeric property St, referring to the strength of the argument, whose default value is 0, but that is modified throughout the execution of the algorithm by calculating the strength of the argument using the $StArg$ function.

The presentation of the algorithm will be divided into two sections. Section 4.1 presents the main procedure $Query$, and Section 4.2 presents the auxiliary function $Find_Def_Args$, which corresponds to a good part of the algorithm as a whole, and therefore is presented separately in order to facilitate reading and understanding. Finally, Section 4.3 presents some analytical evaluation of the algorithm as well an approach to optimize it.

4.1 Procedure *Query*

Each agent implements the same algorithm, which also defines a basic messaging protocol that allows them to collaborate effectively. The main procedure, $Query$, presented in Algorithm 1 in Appendix A, is called when an agent a receives a message of the form $Query(p, \alpha, hist_p)$ from an agent a_0. This can be considered an *"Ask"* speech act emitted by a_0, where: p is the *l-literal* queried; $\alpha = (p, a', KB_\alpha^{\mathcal{F}})$ is the query focus; and $hist_p$ is a list of *l-literals* already evaluated during the query processing, which allows to avoid infinite loops by detecting the occurrence of fallacious arguments. The query result is returned to a_0 as a message in the format $Ans(p, \alpha, tv_p, Args_p, Args_{\sim p})$, where: tv_p is the truth value of p; $Args_p$ is a set of arguments that support the conclusion p; and $Args_{\sim p}$ is a set of arguments that support the conclusion $\sim p$.

The algorithm can be divided into 6 (six) main steps:

1. Extended local KB creation given query focus α (line 3)
2. Search for *l-literals* p' similar to p in the extended local KB (lines 4 and 5)
3. Checking and handling of cycles and self-destructive rule chains (lines 6 and 7)
4. Local strict answer search for each similar *l-literal* q' (lines 8 to 15)
5. Local or distributed defeasible answer search for each similar *l-literal* q' (lines 16 to 22)
6. Comparison between arguments and calculation of the truth-value based on the strengths of defeasible sets of arguments for and against the conclusion of each similar *l-literal* q' (lines 23 to 26)

Figure 3 presents an activity diagram that illustrates the execution of these steps.

Step 1 defines the local extended KB, $KB_{a\alpha}$, as defined in Section 2. This enables the agent to consider not only its own knowledge, but also an augmented knowledge considering the focus knowledge received in the query.

Step 2 iterates over $KB_{a\alpha}$ searching for any rule with head p' such that p and p' are similar enough. This enables to consider literals by similarity, as described earlier.

Step 3 verifies whether p' or $\sim p'$ are not in $hist_p$. If any of these is the case, then a cycle or self-defeat is detected. The algorithm then creates a fallacious argument consisting of a unique leaf node labeled $p'!$.

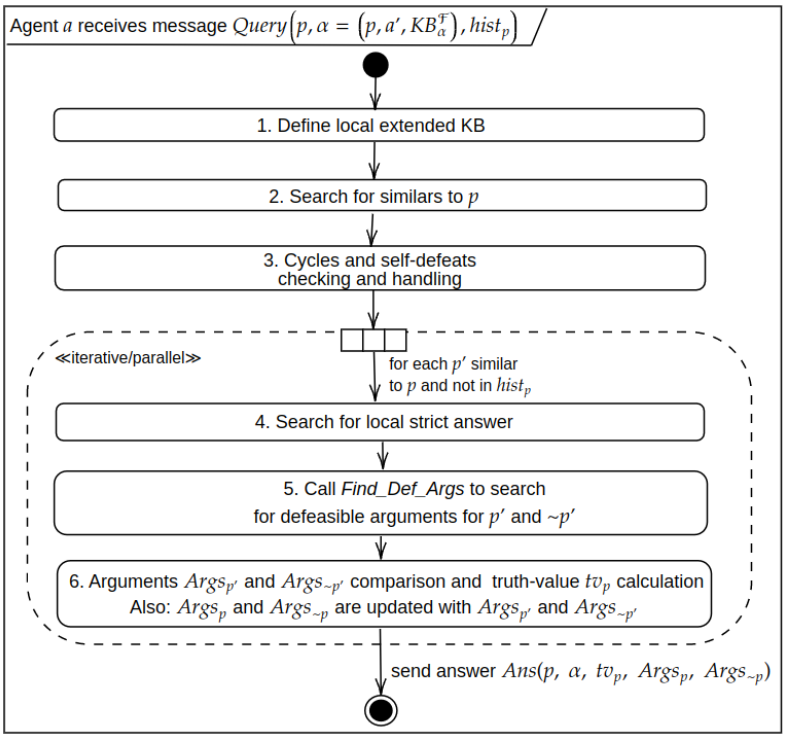
Step 4 to 6 are executed iteratively or in parallel, for each p'. It is interesting that it can be made in parallel because it mainly consist in querying other agents asynchronously.

Step 4 tries to build local strict arguments based on strict rules of the agent. This step works as a depth first algorithm traversing the strict rules of the agent trying to activate each rule in the chain of rules, as a classic logic proof finder. A pseudocode is presented in Algorithm 2 in Appendix A.

Step 5 tries to build defeasible arguments, both for p' and $\sim p'$. This is done by means of a helper function called $Find_Def_Args$, which is responsible for sending queries to other agents for each literal in the body of the rules with head p'. It is also responsible for incrementally building distributed arguments (arguments that have external subarguments). Section 4.2 details it a bit further. Furthermore, in step 5 it is verified whether local strict answers have been found (lines 19 to 22). If this is the case, every opposite argument is marked as rejected, since a strict argument defeats every opposite arguments and is not defeated by them.

Step 6 takes all the defeasible arguments generated for and against p' and compare them. Its pseudocode is presented in the $Compare_Def_Args$ function in Algorithm 3, Appendix A. This step implements most part of the DL semantics by checking the boolean flag $SuppJ$ and R and setting J and R accordingly, and, based on these

Fig. 3: Activity diagram for the $Query$ algorithm.



flags, decide the truth-value $tv_{p'}$. At this point, i.e., after $Find_Def_Args$ was called, $SuppJ$ – and also R for the case in which some subargument is marked as rejected, which means the argument has been undercut – have already been previously set in the $Find_Def_Args$ (step 4), when arguments are built and every subargument are checked whether they are justified or rejected, as explained in Section 4.2, step 4.

A brief explanation of step 6 is given as follows. The variable $tv_{p'}$ is set as *undec* as default. Then, a set of not undercut (already rejected by the existence of rejected subarguments) arguments for both p' and $\sim p'$ is defined (line 3). This is done because undercut arguments can't be used to prevent an argument from being justified: even if an undercut argument defeats another argument, this last one has already been defended by the argument that defeated the first one. Next, for each not undercut argument A:

(i) if A is supported by justified arguments and there doesn't exist any not undercut argument B that defeats A, then A is marked as justified (J) and the truth value is set to *true*, since a single justified argument for p' is sufficient for accepting such conclusion.

(ii) if there exists an opposite argument $B \in Args_{\sim p'}$ that is supported by justified arguments and defeats A, then A is marked as rejected (R).

Finally, if none of the arguments in $Args_{p'}$ have been found as justified and all of them have been rejected (or $Args_{p'}$ is empty, which simply means there is no argument for p'), then $tv_{p'}$ is set to false. However, it is possible that there are argument that are not justified nor rejected, in which case $tv_{p'}$ remains *undec* (this happens specifically when the argument is fallacious and is not defeated by an argument supported by the justified arguments).

4.2 Function *Find_Def_Args*

The function $Find_Def_Args$ is presented in Algorithm 4 in Appendix A. It receives a literal p', the extended knowledge base $KB_{a\alpha}$ created in the $Query$ procedure, the query context α and the history $hist_p$.

The procedure is split in 4 (four) steps:

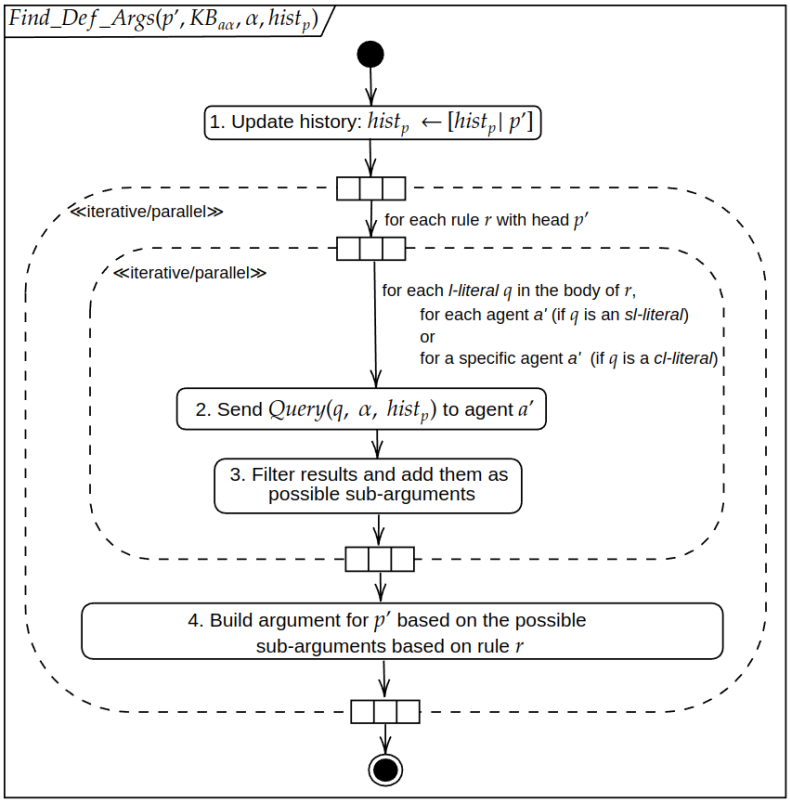
1. Update $hist_p$ by adding p' to it (line 3)
2. Send queries for every *l-literal* q that exists in the body of each r with head p' (lines 4 to 10)
3. Filter the results received for each q for the queries and add them as possible subarguments for an argument for p' (lines 11 to 16)
4. Build argument for p' based on the possible subarguments, for each different rule for p' (lines 17 to 28)

The Figure 4 presents an activity diagram that illustrates these steps.

Step 1 is self explanatory.

Step 2 consists of iterating over each rule r with head p' in $KB_{a\alpha}$, and for each rule r, iterate over each *l-literal* q in the body of r in order to send new queries, aiming to try to build argument for each q. This iteration can also be done in parallel, which is

Fig. 4: Activity diagram for the $Find_Def_Args$ algorithm.



interesting since the query messages are sent asynchronously. In this step, when an *sl-literal* is found, the query is sent in a broadcast manner to every agent in the system. Otherwise, if a *cl-literal* is found, the query is sent only to the agent $D(q)$. The code for querying agents is illustrated in Algorithm 5, Appendix A.

Step 3 is executed when answers are received. If no arguments are found for q, the algorithm goes to to the next rule and discards all the results for the current rule (lines 11 and 12). This is because an argument for p' based on a rule r can't be built if one of its body members can't be activated. Next, the arguments received are filtered in order to avoid an "explosion" of subarguments. This is done by eliminating fallacious arguments if valid ones were found (lines 13 and 14) and then getting the argument that has the greatest strength multiplied by the strength of the *il-literal* in its conclusion from the point of view of the current agent (a) – which is part of the argumentation strength calculation (line 15). This way, only the subargument that will contribute with the greatest strength possible is considered. This is reasonable when considering that the goal is to find the best arguments to support a conclusion, thereby avoiding the generation of a lot of irrelevant arguments that could impose unnecessary overhead over the MAS. This can also be viewed as a kind of pruning

of irrelevant arguments: fallacious subarguments are irrelevant when valid ones exist, and multiple subarguments for the same conclusion can be pruned by leaving only the one that provides the most benefit.

Step 4 is executed after the answers for each q in the body of a given rule r is received. It takes the possible subarguments and builds arguments for p' based on them. It is actually in this step that an an *sl-literal* is instantiated to an *il-literal* (see lines 20 and 21). Also, for each argument $A_{p'}$ built, if every subargument are marked as justified (J), then $A_{p'}$ is marked as supported by justified arguments ($SuppJ$, lines 14 and 15). In the other hand, if there exist any subargument which is marked as rejected (R), then $A_{p'}$ is also marked as rejected (lines 16 and 17). This also means $A_{p'}$ is considered as undercut, i.e., it can't be used to prevent another argument from being justified, even if it defeats the other argument, as explained in Section 4.1, step 6. Finally, is in this step that the strength of the argument is calculated by means of $StArg$ (Section 3), in line 18.

4.3 Complexity Issues and Optimization

It is important to note that the algorithm as presented thus far would have exponential complexity in the worst case. Suppose the following global KB:

$$r_{a1} : \langle a, x_1\rangle \Leftarrow \langle @, x_2\rangle, \langle @, x_3\rangle, \langle @, x_4\rangle; \qquad r_{b1} : \langle b, x_2\rangle \Leftarrow \langle @, x_3\rangle, \langle @, x_4\rangle$$

$$r_{c1} : \langle c, x_3\rangle \Leftarrow \langle @, x_4\rangle; \qquad r_{d1} : \langle d, x_4\rangle \Leftarrow$$

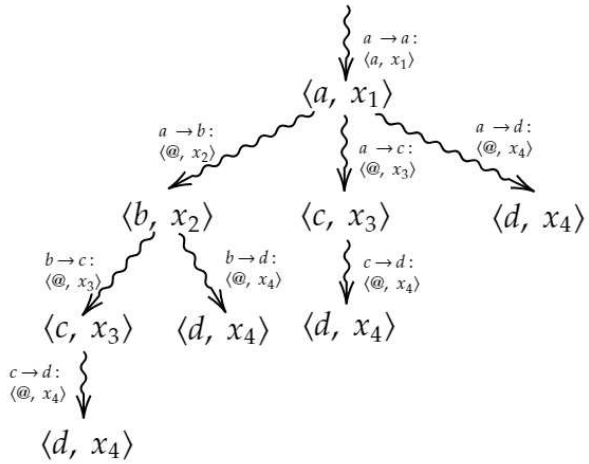
Figure 5 presents a call tree, considering only the queries that reach the $Find_Def_Args$ stage (step 5 of the $Query$ algorithm), which leads to recursive calls to the procedure $Query$. Cases in which the queries do not reach this stage do not induce recursion on the agents' knowledge base rules, namely: when the queried agent does not have any similar *l-literal* in its knowledge base, or when the *l-literal* which was queried or its complement is in the history $hist_p$. For simplicity, therefore, the complexity analysis criterion used is the number of calls to $Query$ that reach the $Find_Def_Args$ stage.

Each edge of the call tree presented in Figure 5 is labeled in the form $\langle a\rangle \rightarrow \langle b\rangle : \langle p\rangle$, where $\langle a\rangle$ is the agent that submitted the query, $\langle b\rangle$ is the agent that received the query, and $\langle p\rangle$ is the *l-literal* being queried. Each vertex is labeled as the *l-literal* $\langle p\rangle'$ that was found to be similar enough to p, and for which an answer will be sought.

It is easy to see that the *il-literals* $\langle b, x_2, 1\rangle$, $\langle c, x_3, 1\rangle$ and $\langle d, x_4, 1\rangle$ will be generated from this setup, constituting the set of premises of the arguments. Therefore, it is possible to induce that the number of queries that reach the stage of $Find_Def_Args$ for an arbitrary number of premises n in the arguments is asymptotically $O(2^n)$. The demonstration is straightforward and will not be presented for the sake of space and convenience.

As to the number of messages, complexity is even greater considering the number of messages that each agent sends for each *sl-literal* and to each known agent. This is because, in each of the queries, represented by the edges in the call tree, as shown in Figure 5, one must consider that the agent actually send query messages to each known

Fig. 5: Call tree for the worst case when $n = 3$.



agent. Therefore, the number of messages exchanged has an asymptotic complexity of $O(|Ags| \times 2^n)$ in the worst case.

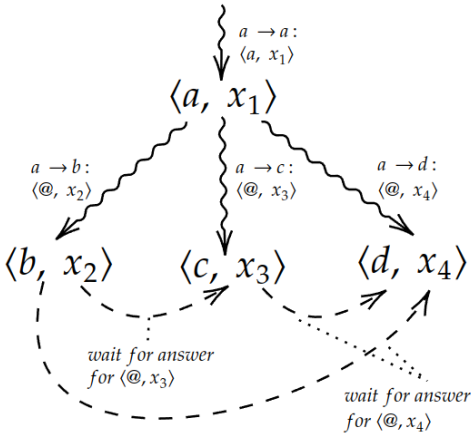
Therefore, it is clear that the algorithm as presented thus far is not computationally feasible. However, we present an optimization strategy that enables it to achieve polynomial time complexity: a *cache* memory that allows storing and reusing arguments previously generated in the context of the same query focus. This memory, referred as $C_\alpha[p']$, is individual and local to each agent, and mapped to each specific query focus α and by each *l-literal* p' that is similar enough to a queried *l-literal* p.

A record $C_\alpha[p']$ in agent a can store a null value ($\perp$), a *future* (as explained below), or the actual set of argument for and against p built in a previous query. Each time an agent a reaches steps 4 and 5 of the *Query* algorithm, it first checks whether p' (the *l-literal* which was found to be similar enough to the *l-literal* queried) have not been previously received in the same query focus by checking the value of $C_\alpha[p']$. If this is the case and the answer has already been returned, it proceeds to use that answer. If the argument is still not found – that is, the agent is still in the middle of processing to build it – the *thread* gets waiting until an answer is received. This is achieved through the concept of *future* (or *promise*) [22]. In simple terms, a future is an object that acts as a *proxy* for an initially unknown result because the calculation of its value has not yet been completed. Many parallel *threads* may wait until the value of this object is set. When this happens, they are notified, so they are able to capture the value and resume their execution.

The benefits of this simple caching mechanism are made clearer when observing what would happen in the case presented when using this caching mechanism. Figure 6 shows what would be the call tree for this case with the optimization presented.

Note that only $n+1 = 4$ queries make it to the $Find_Def_Args$ stage in this case, since there is no need to repeat the same query processing for $\langle @, x_3 \rangle$ and $\langle @, x_4 \rangle$, as in the version without *cache*. Therefore, it can be induced that the number of calls to *Query* that get to the $Find_Def_Args$ stage is now proportional to the number of

Fig. 6: Call tree for the worst case scenario when $n = 3$ with the *caching* mechanism.



premises, reducing the time complexity to $O(n)$. It is easy to think of the intuition for it: there will be as many queries as there are *l-literals* to query about as the algorithm traverse the agents' rules, similar to a depth-first tree traversal.

As to the number of messages, it is also reduced. For the worst case scenario with $n = 3$, only $3 \times 3 = 9$ messages are sent by a to each of the agents b, c and d ($|Ags| - 1 = 3$) about each of the *sl-literals* in the rule body r_{a1} ($n = 3$). Then there will be $2 \times 3 = 6$ messages, corresponding to the two *sl-literals* in the body of r_{b1}, which must also be sent to the 3 other agents. Then there will be $1 \times 3 = 3$ messages sent by c, corresponding to the single *sl-literal* in the body of r_{c1} sent to the 3 other agents. Therefore, $(3 \times (3 + 2 + 1)) \times 2 = 36$ messages are exchanged, including the query and reply messages. Generally speaking, the number of messages $m = [(|Ags|-1)\times\sum_{i=1}^{n} i)]\times 2 = [(|Ags|-1)\times(n\times(1+n)/2)]\times 2 = (|Ags|-1)\times(n^2+n)$. Therefore, the number of messages exchanged with optimization based on *cache* is $O(|Ags| \times n^2)$.

In an average case, however, the number of messages tends to be less. Let m be the number of *sl-literals* and k the number of foreign *cl-literals*. Assuming that each *sl-literal* occurs only once in the entire extended knowledge base – unlike the worst case, where they are repeated – then for each *sl-literal*, $(|Ags| - 1) \times 2$ messages are sent, counting query and reply messages. Furthermore, for each foreign *cl-literal*, 2 messages are sent, one query and one reply. Therefore, the total number of messages sent is $M = m \times (|Ags| - 1) \times 2 + k \times 2$. Therefore, the asymptotic complexity for the number of messages in this average case is $O(|Ags| \times m + k)$.

As to the total size of arguments generated, the cache mechanism enables it to be reduced, given that arguments already generated in memory can be just reused by reference. In fact, the maximum total size of arguments ends up being proportional to the amount of rules in the system combined with the similarity rate between literals, i.e., the amount of *sl-literal* in the body of the rules that are similar enough to the head of other rules. In other words, the more rules with premises that are similar enough to

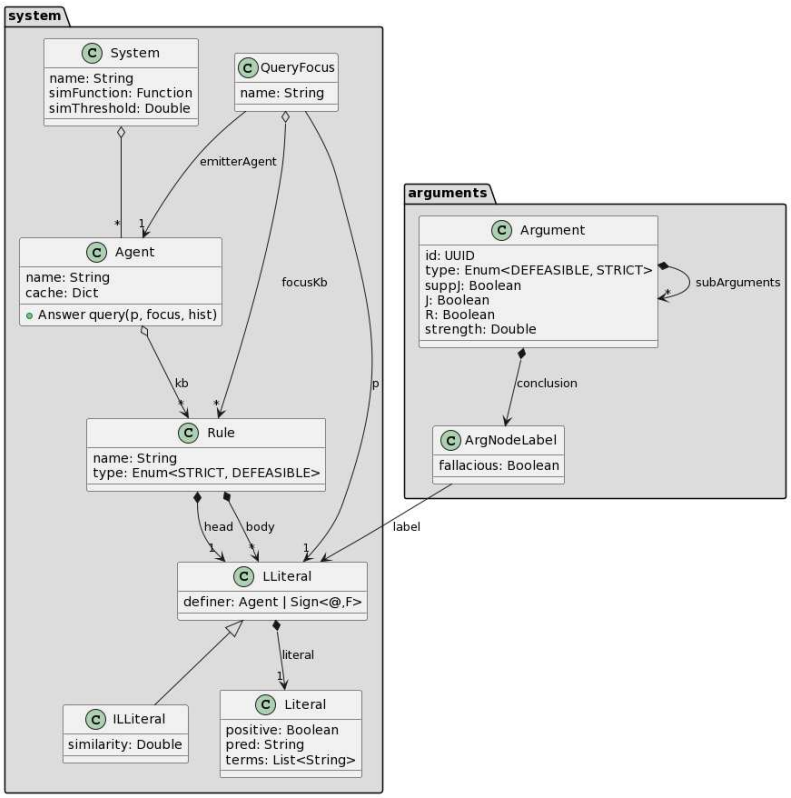
conclusions of other rules, the more arguments will be built, but no argument is built more than once.

Another point of interest of this caching mechanism is that it illustrates how DDRMAS allows reusing previously generated arguments. In this case, it was used to enable a better performance. However, the same arguments could be stored and reused in other features, such as those related to explanation and learning.

5 Experimental Evaluation

In order to ensure the computational feasibility and performance, as well as emphasize DDRMAS's potential for real applications, an implementation of the model was developed, as well as some testing scripts. The implementation was done using the Python programming language[1]. All the experiments were run using the Python 3.10.9 interpreter in an AMD Ryzen 3 5400u® in Ubuntu 22.04.2 LTS. A simple class diagram is presented in Figure 7.

Fig. 7: Class diagram for the implemented system.



[1] The code is available at https://github.com/helioh2/ddrmas-python

Three testing scripts were developed: (i) a script that creates and performs queries in randomized systems; (ii) a script that implements the worst case presented in Section 4.3; and (iii) a script that implements the worst case presented for the *Contextual Defeasible Logic* (CDL) model [13], in order to enable some comparison with DDRMAS.

The randomized script creates a random system taking the following parameters:

- Number of agents ($|Ags|$);
- Total number of distinct literals ($|L_{\mathcal{S}}|$);
- Total number of distinct rules ($|KB_{\mathcal{S}}|$);
- Maximum size of rules' body ($MaxB$);
- Number of focus rules ($FKBSize$);
- Percentage of *sl-literals* throughout the rules ($PerSL$);
- Percentage of pairs of literals with high similarity ($PerSim$);
- Whether the system has cycles or not ($AllowsCycles$);

The rules are generated based on random *l-literals* for their head and body. All rules are defeasible. Each rule has a body of size varying between and including 0 and $MaxB$, evenly distributed. The generated rules are randomly and uniformly distributed among the agents. Redundant or self-conflicting rules are discarded, since they do not add any new information, and such kinds of rules can be easily filtered on a real scenario. For example, rules like $B \Leftarrow A, A$, or $B \Leftarrow A, \sim A$ are removed; and when a rule $B \Leftarrow A$ exists, then $B \Leftarrow A, C$ is not relevant, therefore discarded. This does not impose any new assumption on the DDRMAS system, because mechanisms to remove such redundant rules can be executed in parallel and independently of the DDRMAS query system.

The intuition for this random script is getting statistical data that possibly gives insights about features and weaknesses of DDRMAS. Therefore, different settings were run by a number $TimesRun$ and the following data was captured from these executions: (i) total of arguments generated ($|Args_{\mathcal{S}\alpha}|$); (ii) total of messages exchanged ($|Msgs|$); (iii) sizes of messages exchanged ($SizesMsgs$); and (iv) times of execution (T).

Table 1 presents some results. Only the cases in which at least one argument was generated are accounted (note the $WithArgs$ line), in order to consider only the cases in which rules have been activated during reasoning.

It is interesting to note that the number of arguments, number of messages and sizes of messages are quite reasonable when compared to the amount of agents, literals and rules. Only the number of messages tends to be a little greater when there are more agents, but this is expected, as discussed in Section 4.3. It would be necessary to find out the impact of the number of message in real world applications in order to have further insights on the necessity of optimizing it and how it could be mitigated.

It is also evident that the allowance of cycles increase the amount of arguments generated, even with the existence of the mechanism that avoids infinite loops by creating fallacious arguments that behave as leaf nodes. This happens because, when the script is run with $AllowsCycles = False$, it discards rules that would introduce fallacious arguments. If there are no fallacious arguments, when trying to build some

Table 1: Results of the experiments with the randomized system generator.

Parameters	$TimesRun$	2000	2000	2000	2000	2000	2000
	$\|Ags\|$	5	5	20	20	20	20
	$\|L_{\mathcal{S}}\|$	5	5	20	20	20	20
	$\|KB_{\mathcal{S}}\|$	10	10	50	50	50	50
	$PerSL$	100%	100%	100%	100%	0%	0%
	$PerSim$	10%	10%	10%	10%	-	-
	$AllowsCycles$	Yes	No	Yes	No	Yes	No
	$FKBSize$	1	1	3	3	3	3
	$MaxB$	2	2	3	3	3	3
Outputs	$WithArgs$	579	581	727	707	671	613
	$\|Args_{\mathcal{S}\alpha}\|$ - Avg	2.41	1.85	12.49	3.35	1.28	1.22
	$\|Args_{\mathcal{S}\alpha}\|$ - Max	12	12	60	40	7	6
	$\|Msgs\|$ - Avg	9.76	5.39	418.15	74.76	0.68	0.64
	$\|Msgs\|$ - Max	72	56	2470	1444	12	12
	$SizesMsgs$ - Avg	0.91	1	0.60	0.32	2	3
	$SizesMsgs$ - Max	7	6	20	10	5	4
	T (ms) - Avg	0.69	0.48	17.23	2.06	0.20	0.43
	T (ms) - Max	2.60	2.15	75.35	30.56	1.80	1.79

arguments, the algorithm ends up not reaching neither a fallacious leaf node nor a factual rule, thus many arguments are never really built.

As for the script that implements the worst-case scenario presented in Section 4.3, it was executed with the following inputs and obtained the following results: $|Ags| = |L_{\mathcal{S}}| = 20 \rightarrow |Args_{\mathcal{S}\alpha}| = 20; |Msgs| = 7220; T = 143.96\ ms; Max(SizesMsgs) = 19; Avg(SizesMsgs) = 0.36$. These results confirm the analytical complexity analysis presented in Section 4.3.

In order to enable a comparison with the most similar related work, we modeled CDL's worst case presented in [13] in DDRMAS. This was done by considering the absence of *sl-literals*, i.e., all body members of rules are concrete *l-literals*. This setting consists of $n+1$ agents $\{a_0, a_1, ..., a_n\}$, such that a_0 has the following rules:

$$
\begin{aligned}
r_{a_0 1} &: \langle a_0, x_0\rangle \Leftarrow \langle a_2, x_2\rangle, \langle a_3, x_3\rangle, ..., \langle a_n, x_n\rangle \\
r_{a_0 2} &: \langle a_0, x_0\rangle \Leftarrow \langle a_1, x_1\rangle, \langle a_3, x_3\rangle, ..., \langle a_n, x_n\rangle \\
&\dots \\
r_{a_0[n/2]} &: \langle a_0, x_0\rangle \Leftarrow \langle a_1, x_1\rangle, ..., \langle a_{n/2-1}, x_{n/2-1}\rangle, \langle a_{n/2+1}, x_{n/2+1}\rangle, ..., \langle a_n, x_n\rangle \\
r_{a_0[n/2+1]} &: \langle a_0, \neg x_0\rangle \Leftarrow \langle a_1, x_1\rangle, ..., \langle a_{n/2}, x_{n/2}\rangle, \langle a_{n/2+2}, x_{n/2+2}\rangle, ..., \langle a_n, x_n\rangle \\
&\dots \\
r_{a_0 n} &: \langle a_0, \neg x_0\rangle \Leftarrow \langle a_1, x_1\rangle, \langle a_2, x_2\rangle, ..., \langle a_{n-1}, x_{n-1}\rangle
\end{aligned}
\tag{4}
$$

The other agents all have a single factual rule each. For example, a_1 has the rule $\langle a_1, x_1\rangle \Leftarrow$, a_2 has $\langle a_2, x_2\rangle \Leftarrow$, and so on.

The results for $|Ags| = |L_{\mathcal{S}}| = 100$ are the following: $|Args_{\mathcal{S}\alpha}| = 198; |Msgs| = 19404; T = 721.68\ ms; Max(SizesMsgs) = Max(SizesMsgs) = 1$. The number 100 for the agents/literals was chosen because 100 is the maximum amount of agents that Bikakis et al. tested with in their experiment [13]. It is interesting to note that,

in their case, they could not run their most complex strategy for conflict resolution (*Complex Mapping Sets*) – which uses all the context information possible – for more than 40 agents because it exceeded memory usage in their setup, and for 40 agents their experiment took 207828 ms to execute. This shows that our approach for conflict resolution enables an efficient context reasoning, with the addition of considering some nuances related to the indirect dependencies among agents, as explained in Section 3.2. The main differences between CDL and DDRMAS are further detailed in Section 6.

6 Related Work

This section presents some related work. The ones that are most similar are those based on Defeasible Logic (DL) and Multi-Context Systems (MCS). Governatori et al. [20] present an argumentation-based semantics for DL, enabling reasoning in the presence of possibly conflicting defeasible rules in a knowledge base. Bikakis et al. [1] extends this model, calling it *Contextual Defeasible Logic* (CDL), including the possibility of distributed knowledge bases with a querying algorithm to enable multiple agents to discover the truth-value of a literal. However, they do not propose ways to enable agents to use knowledge from arbitrary agents, i.e., it is assumed that all agents know which agent defines each knowledge, which is not realistic in many scenarios (the mushroom hunting scenario being an example, since it can't be modeled in CDL). There is also no proposal for handling knowledge from different agents based on similarity, assuming that all agents share the same vocabulary, which also assumes there is a centralized entity that standardize such vocabulary. Therefore, the *defeat* relation, which is the base for conflict resolution between arguments, is defined only in terms of preferences (trust) among agents. Furthermore, sharing contextual knowledge in queries, which is also a requirement for scenarios like the mushroom hunting one, is not proposed in their work.

In [13], Bikakis et al. propose different strategies for CDL in order to take advantage of the trust that multiple agents have in each other, which enables achieving similar results to the strength calculation proposed in this work. However, such strategies are not proposed using tree-like structures for arguments, therefore lacking the flexibility of proposing forms of conflict resolution based on the structure of arguments – for example, by means of a formula, as is done in DDRMAS. The most complete strategy proposed by them, called *Complex Mapping Sets* (*CS*), uses the trust among all the agents involved on achieving a given conclusion, but does not penalize indirect trust between them, as is done in the argument strength formula used in DDRMAS, which takes advantage of the tree-like structure of arguments. Furthermore, it was demonstrated in Section 5 that CDL systems can be modeled in DDRMAS – showing that this one is more general than that one – and that the worst case presented in [13] is feasible and efficient in DDRMAS for a great number of literals and agents. This is a relevant contribution, given that DDRMAS's default conflict resolution approach also uses all context information and relations between agents – except the irrelevant arguments which are discarded in step 3 of the $Find_Def_Args$ algorithm, as explained in Section 4.2 – which are materialized in the arguments built during reasoning.

Other relevant work in the field are DeLP (Defeasible Logic Programming) [23] and ASPIC+ [24]. None of them proposes distributed knowledge bases with a query algorithm in the same fashion as proposed in this work. Furthermore, there is no solution to deal with knowledge coming from arbitrary agents and matched by similarity. In [25], DeLP presents a similar idea to the query focus, called contextualized query, which is a query that includes knowledge relevant to help answering it. Such query must be issued to a node called *DeLP-Server*. However, such approach is restricted to a client-server query answering fashion, and does not propose a form of distributed reasoning with interrelated knowledge from different agents in a peer-to-peer manner as proposed in this work. As to ASPIC+, it proposes a framework for DL that can take advantage of extension-based semantics, like those proposed by Dung [26], but also does not propose an architecture for multi-agent reasoning.

The Argumentation-Based Multi-Agent System (ArgMAS) [27] presents a DeLP-based framework that supports collaboration between agents, which allows combining the knowledge bases of different agents in order to produce arguments. Therefore, this approach presents a way of reasoning with interrelated knowledge bases, although it does not use the concept of bridge/mapping rules. Instead, agents in an agent alliance construct partial arguments that contain sets of free literals, that is, sets of literals that are not yet supported by other arguments. From there, a meta-agent runs a backwards-chaining algorithm trying to construct a complete argument based on the partial arguments. However, this proposal has some relevant differences compared to the present work: (i) Collaboration between agents is restricted to a predefined set of agents belonging to the same alliance, while in DDRMAS an agent can collaborate with any other agent, without the need of an alliance being created; (ii) an agent can only participate in one alliance at a time, while our work allows an agent to participate in the reasoning of multiple query focuses simultaneously; (iii) the complete arguments are generated by a meta-agent, which centralizes a good part of the reasoning, and not by the individual agents in a totally distributed manner, as proposed in DDRMAS; and (iv) the correspondence between the literals of different agents is defined only by the exact correspondence, that is, a partial argument with a free literal x is chained to a literal that exactly concludes the value x, not being made a matching based on similarity. In addition, the work does not present ways of sharing focus in queries.

Similar work are also found in the field of MCS, which is a kind of formalization for systems composed of multiple knowledge bases that are interrelated by means of their vocabularies. However, no work was found that solved all the problems mentioned in the present work, and only one is based on argumentation-based semantics [1]. Furthermore, most of the approaches on MCS propose a conflict resolution based on the concept of *repair* [28], which modifies the knowledge bases to maintain a conflict-free global state. The present work, in the other hand, proposes conflict resolution at query time, i.e., conflicts are detected when arguments are built and then confronted to each other, which also do not necessarily imply in updating the knowledge bases, but only in giving the most reasonable answer given some criteria (in our case, trust among agents and knowledge similarity). Therefore, different from *repairing*, DDRMAS proposes maintaining a possibly inconsistent global state, resolving them when

conflicting knowledge is used, with the benefit of allowing agents to maintain different views about the environment and to be able to more faithfully execute a type of contextual reasoning in which there are conclusions that are valid in certain contexts but not in others.

Dao-Tran et al. [29] proposes the Dynamic Distributed Multi-Context System (DDMCS), which includes the use of schematic bridge (mapping) rules with a similarity function, and a *backtracking* algorithm to enumerate all possible substitutions (instantiations) of schematic bridge atoms (similar to the *sl-literals* of this work) in a distributed environment, in order to instantiate a concrete SMC, from which equilibria – possible sets of admissible beliefs – can be calculated. Therefore, DDMCS theoretically requires the pre-instantiation of a concrete SMC for each new change in the system, which may involve each existing knowledge base, as there are dependencies between them. This could result in unnecessary overhead in highly dynamic environments. Furthermore, some sort of centralized entity would be needed to instantiate and assign these "instantiated knowledge bases" to each agent. The approach of the present work (DDRMAS), in the other hand, does not require any centralized action in response to changes. When an agent needs to query other agents about an *sl-literal*, it simply sends a message in *broadcast* to the other agents, without even needing to know each existing agent in the system specifically. Agents arriving at the environment can simply send a *broadcast* message to introduce themselves. Agents leaving the environment can also send this type of message to let other agents know of their absence. In cases in which this is not possible (for example, an agent abruptly removed from the system), when such an agent is queried by another agent, some communication service or broker could simply return a communication error. Changes in agents' knowledge also do not require any intervention in DDRMAS. When an agent with updated knowledge is queried, it will simply use its current knowledge.

Conclusions and Future Work

A comprehensive and flexible model, which allows modeling and implementing scenarios with distributed agents, with the possibility of incomplete and conflicting knowledge bases, as well as a form of contextualized reasoning, was presented. A formalization of knowledge bases and queries, as well as the formalization of arguments that can be generated from them, is presented. It is also proposed a distributed algorithm that allows the processing of queries according to the underlying argumentation semantics. It was analytically and experimentally demonstrated that the model is computational feasible and that it presents contributions and advantages when compared to related work.

Future work include proposing models for explanation and learning based on the argument structures generated in the reasoning process. This way, agents could explain the decisions they made, as well as adapt their rules based on the arguments received from other agents.

Alternative forms of argument strength calculation can also be proposed and tested. Some ideas in this respect include: (i) *joint argument strength* for a given literal, i.e., considering the sum of strengths of all the arguments for and against a given

literal; (ii) a naiver strength calculation that considers all arguments, including external subarguments of subarguments, equal to direct external subarguments; (iii) a more arrogant strength calculation that does not take into account the strength calculation using the trust function of the agent that generates the argument, but only the trust function of the argument that issued the query; and (iv) a more skeptical strength calculation, that uses the *min* operation instead of summation, such that only the strength of the weaker subargument is used to derive the strength of an argument. More details about these different strength calculations, as well as comparisons and properties, will be presented in future work.

Further investigation on the relation of DDRMAS to existing argumentation models could also be realized. In fact, it is possible to prove that a DDRMAS system can be transformed to a Governatori's DL system, but not the other way around, demonstrating that DDRMAS inherits DL properties, but also extends it. This demonstration was not included in this paper on account of its already considerable length. Furthermore, it could also be fruitful to compare DDRMAS to other models, such as DeLP and ASPIC+, and investigating whether it is possible to adapt DDRMAS to use different argumentation semantics.

Another interesting line of future work would be to implement applications based on DDRMAS in order to demonstrate its usefulness in real world scenarios, as well as software frameworks in different programming languages to facilitate the development of such applications.

Statements and Declarations

Competing interests

This work was supported by *Conselho Nacional de Desenvolvimento Científico e Tecnológico* (CNPq) under Grant Agreement CNPq 409523/2021-6.

Author contributions

All authors contributed to the study conception and design. Modeling, definitions, analysis and implementation were performed by Helio Monte-Alto with the aid, guidance and suggestions of all co-authors. All authors commented on previous versions of the manuscript and all authors read and approved the final manuscript.

Data availability

The implementation code presented in this paper, as well as the test scripts and results of the executions presented, are available at https://github.com/helioh2/ddrmas-python.

References

[1] Bikakis, A., Antoniou, G.: Defeasible contextual reasoning with arguments in ambient intelligence. IEEE Transactions on Knowledge and Data Engineering **22**(11), 1492–1506 (2010)

[2] Rakib, A., Uddin, I.: An efficient rule-based distributed reasoning framework for resource-bounded systems. Mobile Networks and Applications **24**(1), 82–99 (2019)

[3] Mekuria, D.N., Sernani, P., Falcionelli, N., Dragoni, A.F.: Smart home reasoning systems: a systematic literature review. Journal of Ambient Intelligence and Humanized Computing, 1–18 (2019)

[4] Maarala, A.I., Su, X., Riekki, J.: Semantic reasoning for context-aware internet of things applications. IEEE Internet of Things Journal **4**(2), 461–473 (2017)

[5] Su, X., Li, P., Riekki, J., Liu, X., Kiljander, J., Soininen, J.-P., Prehofer, C., Flores, H., Li, Y.: Distribution of semantic reasoning on the edge of internet of things. In: Proceedings of the 2018 IEEE International Conference on Pervasive Computing and Communications (PerCom), Athens, Greece, pp. 1–9 (2018). IEEE

[6] Adjiman, P., Chatalic, P., Goasdoué, F., Rousset, M.-C., Simon, L.: Distributed reasoning in a peer-to-peer setting: Application to the semantic web. Journal of Artificial Intelligence Research **25**, 269–314 (2006)

[7] Levesque, H.J.: Knowledge representation and reasoning. Annual review of computer science **1**(1), 255–287 (1986)

[8] McCarthy, J.: Generality in artificial intelligence. Commun. ACM **30**(12), 1030–1035 (1987)

[9] Benerecetti, M., Bouquet, P., Ghidini, C.: Contextual reasoning distilled. Journal of Experimental & Theoretical Artificial Intelligence **12**(3), 279–305 (2000)

[10] Besnard, P., Garcia, A., Hunter, A., Modgil, S., Prakken, H., Simari, G., Toni, F.: Introduction to structured argumentation. Argument & Computation **5**(1), 1–4 (2014)

[11] Brewka, G., Roelofsen, F., Serafini, L.: Contextual default reasoning. In: Proceedings of the 20th International Joint Conference on Artifical Intelligence. IJCAI'07, pp. 268–273. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2007)

[12] Monte-Alto, H.H.L.C., Possebom, A.T., Morveli-Espinoza, M.M.M., Tacla, C.A.: A rule-based argumentation framework for distributed contextual reasoning in dynamic environments. DYNA **88**(217), 120–130 (2021)

[13] Bikakis, A., Antoniou, G., Hasapis, P.: Strategies for contextual reasoning with conflicts in ambient intelligence. Knowledge and Information Systems **27**(1), 45–84 (2011)

[14] Castelfranchi, C., Falcone, R.: Trust Theory: A Socio-cognitive and Computational Model vol. 18. John Wiley & Sons, New York (2010)

[15] Modgil, S.: Reasoning about preferences in argumentation frameworks. Artificial intelligence **173**(9-10), 901–934 (2009)

[16] Bechhofer, S., Van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A., et al.: Owl web ontology language reference. W3C Recommendation **10**(02) (2004)

[17] Miller, G.A.: Wordnet: a lexical database for english. Communications of the ACM **38**(11), 39–41 (1995)

[18] Sycara, K., Widoff, S., Klusch, M., Lu, J.: Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. Autonomous Agents and Multi-Agent Systems **5**(2), 173–203 (2002)

[19] Bikakis, A., Antoniou, G.: Defeasible contextual reasoning in ambient intelligence. PhD thesis, Computer Science Department, University of Crete (2009)

[20] Governatori, G., Maher, M.J., Antoniou, G., Billington, D.: Argumentation semantics for defeasible logic. Journal of Logic and Computation **14**(5), 675–702 (2004)

[21] Baroni, P., Caminada, M., Giacomin, M.: An introduction to argumentation semantics. The knowledge engineering review **26**(4), 365–410 (2011)

[22] Baker, H.C., Hewitt, C.: The incremental garbage collection of processes, vol. 12, pp. 55–59. Association for Computing Machinery, New York, NY, USA (1977)

[23] García, A.J., Simari, G.R.: Defeasible logic programming: An argumentative approach. Theory Pract. Log. Program. **4**(2), 95–138 (2004)

[24] Modgil, S., Prakken, H.: The aspic+ framework for structured argumentation: a tutorial. Argument & Computation **5**(1), 31–62 (2014)

[25] García, A.J., Simari, G.R.: Defeasible logic programming: Delp-servers, contextual queries, and explanations for answers. Argument & Computation **5**(1), 63–88 (2014)

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

[27] Thimm, M., Garcia, A.J., Kern-Isberner, G., Simari, G.R.: Using collaborations for distributed argumentation with defeasible logic programming. In: Proceedings of the 12th International Workshop on Non-Monotonic Reasoning (NMR'08), pp. 179–188 (2008)

[28] Brewka, G., Ellmauthaler, S., Gonçalves, R., Knorr, M., Leite, J., Pührer, J.: Reactive multi-context systems: Heterogeneous reasoning in dynamic environments. Artificial Intelligence **256**, 68–104 (2018)

[29] Dao-Tran, M., Eiter, T., Fink, M., Krennwallner, T.: Dynamic distributed nonmonotonic multi-context systems. Nonmonotonic Reasoning, Essays Celebrating its 30th Anniversary, Studies in Logic **31** (2010)

Appendix A Algorithm Pseudocodes

when a **receives message** $Query(p, \alpha = (p, a', KB_\alpha^{\mathcal{F}}), hist_p)$ **from** a_0
  thread-safe vars: $tv_p \leftarrow false$; $Args_p \leftarrow \emptyset$; $Args_{\sim p} \leftarrow \emptyset$
  $KB_{a\alpha} \leftarrow KB_a \cup \{Loc_Rule(r^{\mathcal{F}}, a) \mid r^{\mathcal{F}} \in KB_\alpha^{\mathcal{F}}\}$;
  $rlits \leftarrow Find_Similar_RLs(p, KB_{a\alpha})$
  if $rlits = \emptyset$ **then** send $Ans(p, \alpha, false, \emptyset, \emptyset)$ to a_0 and terminate
  $Args_p \leftarrow \{p'! \mid p' \in rlits, \{p', \sim p'\} \cap hist_p \neq \emptyset\}$
  if $Args_p \neq \emptyset$ **then** $tv_p \leftarrow undec$
  executing in parallel for each $p' \in rlits$ **s.t.** $\{p', \sim p'\} \cap hist_p = \emptyset$
    $has_strict_answer_{p'} \leftarrow false$
    if $Local_Strict_Ans(p') = (true, A_{p'})$ **then**
      $Args_p \leftarrow Args_p \cup \{A_{p'}\}$
      $tv_p \leftarrow true$; $has_strict_answer_{p'} \leftarrow true$
    else if $Local_Strict_Ans(\sim p') = (true, A_{\sim p'})$ **then**
      $Args_{\sim p} \leftarrow Args_{\sim p} \cup \{A_{\sim p'}\}$
      $tv_p \leftarrow false$; $has_strict_answer_{p'} \leftarrow true$
    executing commands in parallel and waiting for all to finish
      $Args_{p'} \leftarrow Find_Def_Args(p', KB_{a\alpha}, \alpha, hist_p)$
      $Args_{\sim p'} \leftarrow Find_Def_Args(\sim p', KB_{a\alpha}, \alpha, hist_p)$
    if $has_strict_answer_{p'} = true$ *and* $tv_p = false$ **then**
      for $A_{p'} \in Args_{p'}$ **do** $A_{p'}.R \leftarrow true$
    else if $has_strict_answer_{p'} = true$ *and* $tv_p = true$ **then**
      for $A_{\sim p'} \in Args_{\sim p'}$ **do** $A_{\sim p'}.R \leftarrow true$
    else
      $tv_{p'} \leftarrow Compare_Def_Args(Args_{p'}, Args_{\sim p'})$
      if $tv_{p'} = true$ **then** $tv_p = true$
      else if $tv_{p'} = undec$ *and* $tv_p \neq true$ **then** $tv_p \leftarrow undec$
    $Args_p \leftarrow Args_p \cup Args_{p'}$
    $Args_{\sim p} \leftarrow Args_{\sim p} \cup Args_{\sim p'}$
  send $Ans(p, \alpha, tv_p, Args_p, Args_{\sim p})$ to a_0 and terminate

Algorithm 1: Pseudocode for the *Query* procedure.

```
function Local_Strict_Ans(p'):
    for r ∈ KB_a^s and Head(r) = p' do
        A_p' ← New_Arg(p', [], strict, J = true, St = 1 * Θ(p, p'))
        for q ∈ Body(r) do
            (tv_q, A_q) ← Local_Strict_Ans(q)
            if tv_q = false then stop and go to next rule
            Add_Sub_Arg(A_p', A_q)
        return (true, A_p')
    return (false, ⊥)
```

Algorithm 2: Pseudocode for the *Local_Strict_Ans* function.

```
function Compare_Def_Args(Args_p', Args_~p'):
    tv_p' ← undec
    Args^~U ← {A | A ∈ Args_p' ∪ Args_~p', A.R = false}
    executing in parallel for each A ∈ Args^~U
        if A.SuppJ = true and ∄B ∈ Args^~U s.t. BDA then
            A.J ← true
            if Conc(A) = p' then tv_p' ← true
        else if ∃B ∈ Args^~U s.t. BDA and B.SuppJ = true then
            A.R ← true
    if tv_p' ≠ true and (Args_p' = ∅ or ∀A_p' ∈ Args_p', A_p'.R = true) then
        tv_p' ← false
    return tv_p'
```

Algorithm 3: Pseudocode for the *Compare_Def_Args* function.

```
function Find_Def_Args(p', KB_aα, α, hist_p):
    thread-safe var: Args_p' ← ∅
    hist_p' ← [hist_p|p']
    executing in parallel for each r ∈ KB_aα s.t. Head(r) = p'
        thread-safe var: possible_subargs_r ← ∅
        executing in parallel for each q ∈ Body(r)
            if D(q) = @ then
                Args_q' ← Query_Agents(Ags, q, α, hist_p')
            else if D(q) ∈ Ags then
                Args_q' ← Query_Agents({D(q)}, q, α, hist_p')
            if Args_q' = ∅ then
                stop and discard all processing for rule r
            if ∃A ∈ Args_q' s.t. Fall(A) = false then
                Args_q' ← {A ∈ Args_q' | Fall(A) = false}
            A_q' ← argmax_{A∈Args_q'}(A.St * StIL(Conc(A), a)) //– getting the
              argument for q which has the greatest strength multiplied by the
              strength of its conclusion from the point of view of the current
              agent (see formula for StArg)
            possible_subargs_r ← possible_subargs_r ∪ {(q, A_q')}
        A_p' ← New_Arg(p', [], defeasible)
        for (q, A_q') ∈ possible_subargs_r do
            q' ← Conc(A_q')
            if D(q) = @ or Θ(q, q') < 1 then
                Conc(A_q') ← Inst_a(q, q')        //– instantiating l-literal and
                  setting it into the conclusion of the argument
            Add_Sub_Arg(A_p', A_q')
        if ∀A_q' ∈ Subs(A_p'), A_q'.J = true then
            A_p'.SuppJ ← true
        else if ∃A_q' ∈ Subs(A_p') s.t. A_q'.R = true then
            A_p'.R ← true        //– this also means that A_p' is undercut,
              because some of its subarguments are rejected (defeated by
              arguments supported by justified ones)
        A_p'.St ← StArg(A_p')
        Args_p' ← Args_p' ∪ {A_p'}
    return Args_p'
```

Algorithm 4: Pseudocode for the $Find_Def_Args$ function.

```
function Query_Agents(agents, q, α, hist_q):
    thread-safe var: Args_q ← ∅
    executing in parallel for each a ∈ agents do
        send to a: Query(q, α, hist_q) and wait for answer:
          Ans(q, α, tv'_q, Args'_q, Args'_~q), else if timeout reached then discard
        Args_q ← Args_q ∪ Args'_q
    return Args_q
```

Algorithm 5: Pseudocode for the *Query_Agents* procedure.