

On the Role of Local Arguments in the (Timed) Concurrent Language for Argumentation

Stefano Bistarelli¹, Maria Chiara Meo² and Carlo Taticchi^{1,*}

¹Department of Mathematics and Computer Science, University of Perugia, Italy

²Department of Economics, University "G. d'Annunzio" of Chieti-Pescara, Italy

Abstract

Modelling the behaviour of concurrent agents that interact and reason in a dynamic environment is a difficult task. It requires tools that can effectively capture different types of interactions, such as persuasion and deliberation, while helping agents make decisions or reach agreements. This paper proposes a revised and extended version of the (timed) concurrent language for argumentation, better suited for modelling real-world scenarios. Our focus is on private information: we have given each agent a local argumentation store for reasoning with private knowledge. With this feature, agents can use the argumentation engine to implement courses of action based on their personal information and only disclose the bare minimum. Finally, we present an application example that models a privacy-preserving multi-agent decision-making process to demonstrate the capabilities of our language.

Keywords

Computational Argumentation, Concurrency, Locality

1. Introduction

Intelligent agents can exploit argumentation techniques to accomplish complex interactions like, for instance, negotiation [1, 2] and persuasion [3, 4]. The Timed Concurrent Language for Argumentation (TCLA) [5, 6] offers constructs to implement such interactions. Agents involved in the process share an argumentation store that serves as a knowledge base and where arguments and attacks represent the agreed beliefs. The framework can be changed via a set of primitives that allow adding and removing arguments and attacks. The language makes use of two kinds of expressions: a syntactic *check* that verifies if a given set of arguments and attacks is contained in the knowledge base, and semantic *test* operations that retrieve information about the acceptability of arguments in the knowledge base.

This paper introduces a number of new features aimed at facilitating the use of the language and making it more suitable for modelling real-world interaction scenes. The main contributions we will discuss are as follows:


- local stores that agents can use to enhance their reasoning with private information;


7th Workshop on Advances in Argumentation in Artificial Intelligence (AI³), November 06–09, 2023, Rome, Italy

*Corresponding author.

✉ stefano.bistarelli@unipg.it (S. Bistarelli); mariachiara.meo@unich.it (M. C. Meo); carlo.taticchi@unipg.it (C. Taticchi)

ORCID 0000-0001-7411-9678 (S. Bistarelli); 0000-0002-3700-3788 (M. C. Meo); 0000-0003-1260-4672 (C. Taticchi)

 © 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

- the possibility of optional timeouts to make some operators more flexible;
- syntactic sugar that simplifies the modelling of complex situations recurring in real cases.

Autonomous agents can benefit from local stores for hiding data they are unwilling to disclose and only making public the necessary information needed to reach the desired outcome. Think, for example, about intelligent agents negotiating over a shared resource. Some information owned by the agents might have a future strategic value or be subject to privacy concerns. While this kind of information should remain unavailable to other agents, the owner must be able to integrate it into the reasoning process. Our language allows modelling agents endowed with both a shared and a local store to realise the behaviour reported above. Timeouts are another fundamental component to enable proper interaction between agents. Usually, in the real world, one has limited time to make a decision on the next action to be taken. In some situations, however, especially when working with partial knowledge, agents must be able to wait for an indefinite amount of time until a certain condition is fulfilled. We also included an application example for showing the capabilities of τ CLA, with particular focus on using local stores. Contextually, we present syntactic sugar for realising frequently used complex operations.

2. Background

In this section, we recall some fundamental notions concerning Computational Argumentation. The first definition we need is that of Abstract Argumentation Framework (AF) [7].

Definition 1. Let U be the set of all possible arguments, which we refer to as the “universe”.¹ An Abstract Argumentation Framework is a pair $\langle Arg, R \rangle$ where $Arg \subseteq U$ is a set of adopted arguments and R is a binary relation on Arg (representing attacks among adopted arguments).

Given an AF, we want to identify subsets of *acceptable* arguments which are selected by applying criteria called argumentation semantics. Non-accepted arguments are rejected. Different kinds of semantics have been introduced that reflect desirable qualities for sets of arguments. Among the most studied ones, we find the admissible, complete, stable, semi-stable, preferred, and grounded semantics [8, 7] (denoted as *adm*, *com*, *stb*, *sst*, *prf* and *gde*, respectively). To identify acceptable arguments, we can resort to labelling-based semantics [9], an approach that associates with an AF a subset of all the possible labellings.

Definition 2. A labelling of an AF $F = \langle Arg, R \rangle$ is a total function $L : Arg \rightarrow \{in, out, undec\}$. Moreover, L is an admissible labelling for F when $\forall a \in Arg$

- $L(a) = in \implies \forall b \in Arg \mid (b, a) \in R. L(b) = out$;
- $L(a) = out \iff \exists b \in Arg \mid (b, a) \in R \wedge L(b) = in$.

In other words, an argument is labelled *in* only if all its attackers are labelled *out*, and it is labelled *out* when at least one *in* node attacks it. In all other cases, the argument is labelled *undec*. In particular, *in* arguments are acceptable, while the others will be rejected.

¹The set U is not present in the original definition by Dung and we introduce it for our convenience to distinguish all possible arguments from the adopted ones.

Similar criteria to that shown in Definition 2 can be used to capture other semantics [9]. In the following, we will write \mathcal{L}_σ^F to identify the set of all possible labellings of F with respect to the semantics σ . Besides computing the possible labellings with respect to a certain semantics σ , one of the most common tasks performed on AFs is to decide whether an argument a is accepted (labelled as *in*) in some labelling of \mathcal{L}_σ^F or in all labellings. In the former case, we say that a is *credulously* accepted with respect to σ ; in the latter, a is instead *sceptically* accepted with respect to σ .

3. Syntax and Operational Semantics

Communication between TCLA agents is implemented via shared memory, similarly to *cla* [10] and CC [11], and opposed to other languages (e.g. CSP [12] and CCS [13]) based on message passing. In TCLA, the shared memory consists of an AF which agents can access and modify. All agents are synchronised via a shared global clock, tracking the simultaneous execution of concurrent agents. We present the syntax of TCLA in Table 1, where P denotes a generic process, C a sequence of procedure declarations (or clauses), A a generic agent and E a generic guarded agent.

Table 1

TCLA syntax.

$$\begin{aligned}
P &::= \text{let } C \text{ in } A \\
C &::= p(x) :: A \mid C, C \\
A &::= \text{success} \mid \text{failure} \mid \text{add}(Arg, R) \rightarrow A \mid \text{rmv}(Arg, R) \rightarrow A \mid E \mid A \parallel A \mid \text{new } S \text{ in } A \mid p(x) \\
E &::= \text{check}_t(Arg, R) \rightarrow A \mid \text{c-test}_t(a, l, \sigma) \rightarrow A \mid \text{s-test}_t(a, l, \sigma) \rightarrow A \mid E + E \mid E +_P E
\end{aligned}$$

where Arg and S are supposed to be sets of arguments, R a set of attacks, a an argument, $l \in \{\text{in}, \text{out}, \text{undec}\}$, $\sigma \in \{\text{adm}, \text{com}, \text{stb}, \text{prf}, \text{gde}\}$.

In a process $P = \text{let } C \text{ in } A$, A is the initial agent to be executed in the context of the set of declarations C . A clause defined with C, C corresponds to the concatenation of more procedure declarations. An agent $\text{new } S \text{ in } A$ behaves like agent A where arguments in S and attacks built from S are local to A . Before describing in detail how the operators in Table 1 work, let us introduce the notion of timeout used by guarded agents (i.e., the *check*, *c-test* and *s-test* operations). Those agents verify if arguments and attacks in the shared memory meet certain properties and can be executed again in case they do not succeed. To specify how many times the execution can be repeated, guarded agents are endowed with a timeout $t \in \mathbb{N} \cup \{\infty\}$ stating after how many cycles of the global clock the operation will expire and terminate with failure.

The passing of time between all concurrent agents is handled on the same global clock via a timeout environment T as specified below. Let \mathcal{I} be a set of (timeout) identifiers. A timeout environment is a partial mapping $T : \mathcal{I} \rightarrow \mathbb{N} \cup \{\infty\}$ such that the set $\text{dom}(T) = \{I \in \mathcal{I} \mid T(I) \in \mathbb{N} \cup \{\infty\}\}$ (domain of T) is finite. T_0 is the empty timeout environment ($\text{dom}(T_0) = \emptyset$). As we need to manipulate T , for instance, to insert new timeouts or update a timer, we introduce some utility functions. If T_1 and T_2 are timeout environments such that

for each $I \in \text{dom}(T_1) \cap \text{dom}(T_2)$ we have that $T_1(I) = T_2(I)$, then $T_1 \cup T_2$ is the timeout environment such that $\text{dom}(T_1 \cup T_2) = \text{dom}(T_1) \cup \text{dom}(T_2)$ and for each $I \in \text{dom}(T_1 \cup T_2)$

$$(T_1 \cup T_2)(I) = \begin{cases} T_1(I) & \text{if } I \in \text{dom}(T_1) \\ T_2(I) & \text{otherwise} \end{cases}$$

We denote by $T[\bar{I} : \bar{t}]$ an update of T , with a possibly enlarged domain, namely

$$T[\bar{I} : \bar{t}](I) = \begin{cases} T(I) & \text{if } I \neq \bar{I} \\ \bar{t} & \text{otherwise} \end{cases}$$

The passing of time is marked by decreasing timers in a timeout environment T . At each step of the execution, all timers are decreased according to the function $\text{dec}(T)$ such that $\text{dom}(\text{dec}(T)) = \text{dom}(T)$ and

$$\text{dec}(T)(I) = \begin{cases} T(I) - 1 & \text{if } 0 < T(I) \in \mathbb{N} \\ T(I) & \text{if } T(I) = 0 \text{ or } T(I) = \infty \end{cases}$$

In this paper, we only consider the parallelism achieved with the true concurrency paradigm, i.e., all concurrent agents can be executed simultaneously as if an infinite number of processors were available. The adoption of a global clock defined via timeout environment allows time to elapse for all agents, not only in the case of true concurrency but also, for example, with interleaving (i.e., only one parallel agent is executed at a time) which could be introduced in future work. Below, we outline the functioning of all τCLA operators, focusing in particular on the management of the timeouts and the composition of concurrent agents.

Notation 1. Let $F = \langle \text{Arg}, R \rangle$ be an AF and S a set of arguments. We define:

- $R_{|S} = \{(a, b) \in R \mid a \in S \text{ or } b \in S\}$;
- $R_{\parallel S} = \{(a, b) \in R \mid a \in S \text{ and } b \in S\}$;
- $F \downarrow S = \langle \text{Arg} \cap S', R_{|S} \rangle$ where $S' = S \cup \{b \mid (b, c) \in R_{|S} \text{ or } (c, b) \in R_{|S}\}$;
- $F \uparrow S = \langle \text{Arg} \setminus S, R \setminus R_{|S} \rangle$.

The operational model of τCLA processes can be formally described by a transition system $T = (\text{Conf}, \longrightarrow)$, where we assume that each transition step takes exactly one time-unit. Configurations (in) Conf are triples, each triple consisting in a process P , an abstract argumentation framework AF representing a knowledge base and a timeout environment T . The transition relation $\longrightarrow \subseteq \text{Conf} \times \text{Conf}$ is the least relation satisfying the rules in Tables 2 - 8, and it characterises the (temporal) evolution of the system. So, $\langle A, F, T \rangle \longrightarrow \langle A', F', T' \rangle$ means that, if at time t we have the process A , the framework F and the time environment T , then at time $t + 1$ we have the process A' , the framework F' and the time environment T' . In the following, we will usually write a τCLA process $P = C.A$ as the corresponding agent A , omitting C when not required by the context.

Rules **Ad** and **Re** of Table 2 modify the store by adding and removing, respectively, arguments and attacks. Attacks can only be added between arguments that will be in the store after the

Table 2

Add and remove semantics.

$$\begin{aligned} \langle \text{add}(Arg', R') \rightarrow A, \langle Arg, R \rangle, T \rangle &\longrightarrow \langle A, \langle Arg \cup Arg', (R \cup R')_{\parallel (Arg \cup Arg')} \rangle, \text{dec}(T) \rangle & \mathbf{Ad} \\ \langle \text{rmv}(Arg', R') \rightarrow A, \langle Arg, R \rangle, T \rangle &\longrightarrow \langle A, \langle Arg \setminus Arg', (R \setminus R')_{\parallel (Arg \setminus Arg')} \rangle, \text{dec}(T) \rangle & \mathbf{Re} \end{aligned}$$

execution of the add operation. On the other hand, when an argument is removed, also the attacks involving that argument are removed.

Table 3 presents the semantics for the check operation. Whenever a rule is executed, all timers in the timeout environment T are decremented after the possible introduction of new timeouts. The first time a check is performed (rules **Ch1** and **Ch2**), a new timeout must be inserted in T . If the check is not successful on the first attempt, it will be repeated using the already existing timeout. In detail, rule **Ch1** triggers when the guard of the agent is satisfied, and the associated timer is positive. In this case, the agent proceeds to the subsequent action. Rule **Ch2** inserts a new timeout into T and repeats the check operation in the next step when the condition is not satisfied and the timer has not expired. The system will use the operator $check_I$ (rule **Ch4**) to execute a check on the store with a timeout I already present in T . In rule **Ch3**, the guard succeeds, and the timer associated with the check operation is not expired and already in T . Therefore, the program continues to the next action. Rule **Ch4** is the counterpart of **Ch2**, the only difference being that no new timeout is added to T . This is needed for the system to automatically repeat a check operation which did not succeed in the previous step. Note that if a timer t is set to ∞ and the timeouts are decreased with $\text{dec}(T)$, then t will be left

Table 3Check semantics. In the rules, $F = \langle Arg, R \rangle$.

$$\begin{aligned} &\frac{Arg' \subseteq Arg \wedge R' \subseteq R, t > 0}{\langle \text{check}_t(Arg', R') \rightarrow A, F, T \rangle \longrightarrow \langle A, F, \text{dec}(T) \rangle} & \mathbf{Ch1} \\ &\frac{\neg(Arg' \subseteq Arg \wedge R' \subseteq R), t > 0}{\langle \text{check}_t(Arg', R') \rightarrow A, F, T \rangle \longrightarrow \langle \text{check}_I(Arg', R') \rightarrow A, F, \text{dec}(T[I : t]) \rangle} & \mathbf{Ch2} \\ &\text{where } I \text{ is a fresh timeout identifier} \\ &\frac{Arg' \subseteq Arg \wedge R' \subseteq R, T(I) > 0}{\langle \text{check}_I(Arg', R') \rightarrow A, F, T \rangle \longrightarrow \langle A, F, \text{dec}(T) \rangle} & \mathbf{Ch3} \\ &\frac{\neg(Arg' \subseteq Arg \wedge R' \subseteq R), T(I) > 0}{\langle \text{check}_I(Arg', R') \rightarrow A, F, T \rangle \longrightarrow \langle \text{check}_I(Arg', R') \rightarrow A, F, \text{dec}(T) \rangle} & \mathbf{Ch4} \\ &\langle \text{check}_0(Arg', R') \rightarrow A, F, T \rangle \longrightarrow \langle \text{failure}, F, \text{dec}(T) \rangle & \mathbf{Ch5} \\ &\frac{T(I) = 0}{\langle \text{check}_I(Arg', R') \rightarrow A, F, T \rangle \longrightarrow \langle \text{failure}, F, \text{dec}(T) \rangle} & \mathbf{Ch6} \end{aligned}$$

unchanged. This behaviour is similar to that of the check with waiting in [10]. When the timer is 0, the agent fails as per rules **Ch5** and **Ch6**. This behaviour is similar to that of the check with failure in [10]. After the execution of rules **Ch3** and **Ch6**, i.e. when the check operation has terminated with success or failure, respectively, the timeout that was potentially entered in T becomes useless since no rule will inspect it afterwards. In this sense, a garbage collection procedure that removes from T the timeouts that are no longer used could be introduced.

Credulous and sceptical test operators perform a semantic verification on the acceptability of arguments in the store. Their rules follow the same idea as those for the check operation, with the only exception of the condition to satisfy, that is, respectively $\exists L \in \mathcal{L}_\sigma^F \mid L(a) = l$ for $c\text{-test}_I(a, l, \sigma)$ and $\forall L \in \mathcal{L}_\sigma^F. L(a) = l$ for $s\text{-test}_I(a, l, \sigma)$ instead of $Arg' \subseteq Arg \wedge R' \subseteq R$. The guard for the credulous test is satisfied when there exists at least one labelling L of F for a chosen semantics σ such that $L(a) = l$. The sceptical test, instead, demands a be assigned the label l by any labelling in \mathcal{L}_σ^F .

In Table 4, which concerns the rules for nondeterminism, and in Table 5 that realises the if-then-else, \mathcal{E} is the class of guarded agents, while \mathcal{E}_f a subset of the former class such that all outermost guards have either the associated timer $T(I) = 0$ or the timer on the check/test expression set to 0 ($t = 0$). In other words, the execution of agents in \mathcal{E}_f always leads to termination with failure. Rule **ND1** states that if two non-failing expressions E_1 and E_2 composed through the operator $+$ can transit into guarded expressions E'_1 and E'_2 , respectively, they will do so in the same execution step, producing the agent $E'_1 + E'_2$. According to rule **ND2**, the execution continues with E_1 if it can transit into a non-guarded agent. Finally, if one of the two agents fails, it is discarded, and the execution continues with the other agent (rule **ND3**).

Table 4

Nondeterminism semantics.

$$\frac{\langle E_1, F, T \rangle \longrightarrow \langle E'_1, F, T_1 \rangle, \langle E_2, F, T \rangle \longrightarrow \langle E'_2, F, T_2 \rangle, E_1, E_2 \notin \mathcal{E}_f, E'_1, E'_2 \in \mathcal{E}}{\langle E_1 + E_2, F, T \rangle \longrightarrow \langle E'_1 + E'_2, F, T_1 \cup T_2 \rangle} \quad \mathbf{ND1}$$

$$\frac{\langle E_1, F, T \rangle \longrightarrow \langle A_1, F, T' \rangle, E_1 \notin \mathcal{E}_f, A_1 \notin \mathcal{E}}{\langle E_1 + E_2, F, T \rangle \longrightarrow \langle A_1, F, T' \rangle} \quad \mathbf{ND2}$$

$$\frac{E_1 \in \mathcal{E}_f, \langle E_2, F, T \rangle \longrightarrow \langle A_2, F, T' \rangle}{\langle E_1 + E_2, F, T \rangle \longrightarrow \langle A_2, F, T' \rangle} \quad \mathbf{ND3}$$

The operator of Table 5 realises the (non-commutative) if-then-else construct: if we have $E_1 +_P E_2$ and E_1 can transit into E'_1 , like in rule **If1**, then the execution continues with $E'_1 +_P E_2$. If E_1 transits into an agent different from a guarded expression, the rightmost expression is discarded (rule **If2**). This behaviour is different from that obtainable with the $+$ operator for nondeterminism, which will arbitrarily execute one of the two branches, discarding the other. Finally, if E_1 fails, the first branch is discarded and the rightmost agent is executed as per **If3**. Rules **ND3** and **If3**, therefore, serve the same purpose and are identical, apart from the operator.

The procedure call defined in Table 6 only takes a single parameter x that can be an argument,

Table 5

If-then-else semantics.

$$\frac{\langle E_1, F, T \rangle \longrightarrow \langle E'_1, F, T' \rangle, E_1 \notin \mathcal{E}_f, E'_1 \in \mathcal{E}}{\langle E_1 +_P E_2, F, T \rangle \longrightarrow \langle E'_1 +_P E_2, F, T' \rangle} \quad \text{If1}$$

$$\frac{\langle E_1, F, T \rangle \longrightarrow \langle A_1, F, T' \rangle, E_1 \notin \mathcal{E}_f, A_1 \notin \mathcal{E}}{\langle E_1 +_P E_2, F, T \rangle \longrightarrow \langle A_1, F, T' \rangle} \quad \text{If2}$$

$$\frac{E_1 \in \mathcal{E}_f, \langle E_2, F, T \rangle \longrightarrow \langle A_2, F, T' \rangle}{\langle E_1 +_P E_2, F, T \rangle \longrightarrow \langle A_2, F, T' \rangle} \quad \text{If3}$$

a label among *in*, *out* and *undec*, a semantics σ , or an instant of time t . If necessary, this solution can be easily adjusted to accommodate additional parameters or consider parameterless procedures. Executing a procedure call p corresponds to replacing each instance of p with agent A in the execution, according to what is specified in the procedure declaration within the context of C .

Table 6

Procedure call semantics.

$$\langle p(y), F, T \rangle \longrightarrow \langle A[y/x], F, \text{dec}(T) \rangle \text{ with } p(x) :: A \text{ and } x \in \{a, l, \sigma, t\} \quad \text{PC}$$

Rule **TC** in Table 7 models the true concurrency operator. It only succeeds if all the agents composed through \parallel succeed. In our implementation, we use $*(F, F', F'') := (F' \cap F'') \cup ((F' \cup F'') \setminus F)$ to handle concurrent additions and removals of arguments.² If an argument a is added and removed in the same instant (e.g., through the process $\text{add}(\{a\}, \{\}) \rightarrow \text{success} \parallel \text{rmv}(\{a\}, \{\}) \rightarrow \text{success}$), we have two possible outcomes: if a was not present in the knowledge base, then it will be added since $a \in ((F' \cup F'') \setminus F)$; on the other hand, when a was already in the shared memory, we have that $a \notin ((F' \cup F'') \setminus F)$, and a is removed. In other words, we add the argument to shared memory if it is not present and remove it if it is already there. The operator $*$ can nevertheless be customized to obtain different behaviour and fit specific requirements.

Table 7

True concurrency semantics.

$$\frac{\langle A_1, F, T \rangle \longrightarrow \langle A'_1, F', T_1 \rangle, \langle A_2, F, T \rangle \longrightarrow \langle A'_2, F'', T_2 \rangle}{\langle A_1 \parallel A_2, F, T \rangle \longrightarrow \langle A'_1 \parallel A'_2, *(F, F', F''), T_1 \cup T_2 \rangle} \quad \text{TC}$$

In Tables 4 and 7, we have omitted the symmetric rules for the choice operator $+$ and the parallel composition \parallel . Indeed, $+$ is commutative, so $E_1 + E_2$ produces the same result as $E_2 + E_1$. The same is true for \parallel . Moreover, *success* and *failure* are the identity and the

²Union, intersection and difference between AFs are intended as the union, intersection and difference of their sets of arguments and attacks, respectively.

absorbing elements, respectively, under the parallel composition \parallel ; that is, for each agent A , we have that $A \parallel \text{success}$ and $A \parallel \text{failure}$ are the agents A and failure , respectively.

As specified by the rule in Table 8, the agent $\text{new } S \text{ in } A$ behaves like agent A where arguments in $S \subseteq \text{Arg}$ are considered local to A , i.e. the information on S provided by the external AF is hidden from A (therefore we consider the agent A executed in the argumentation framework $(AF \uparrow S) \cup AF_{loc}$) and, conversely, the information on S produced locally by A is hidden from external world (therefore the returned argumentation framework is $(AF' \uparrow S) \cup (AF'' \downarrow S)$).

Table 8
Locality semantics.

$$\frac{\langle A, (AF \uparrow S) \cup AF_{loc}, T \rangle \longrightarrow \langle B, AF', T' \rangle}{\langle \text{new } S \text{ in } A^{AF_{loc}}, AF, T \rangle \longrightarrow \langle \text{new } S \text{ in } B^{AF' \downarrow S}, (AF' \uparrow S) \cup (AF'' \downarrow S), T' \rangle} \quad \text{Loc}$$

where $AF = \langle \text{Arg}, R \rangle$, $AF' = \langle \text{Arg}', R' \rangle$ and $AF'' = \langle \text{Arg}, R_{\parallel \text{Arg}' \cup S} \rangle$

To describe locality, in Table 8, the syntax has been extended by an agent $\text{new } S \text{ in } A^{AF_{loc}}$, where AF_{loc} is a local AF of A containing information on S which is hidden from the external AF . When the computation starts, the local AF_{loc} is empty, i.e. $\text{new } S \text{ in } A = \text{new } S \text{ in } A^{(\emptyset, \emptyset)}$. Finally, for each set of arguments S , $\text{new } S \text{ in success}$ and $\text{new } S \text{ in failure}$ are the agents success and failure , respectively.

In the next section, we make use of some **syntactic sugar** to simplify the presentation of the results. Let $S = \{a_1, \dots, a_n\} \subseteq \text{Arg}$ and let G_1 and G_2 be guards of the form $\text{check}_t(\text{Arg}, R)$, $c\text{-test}_t(a, l, \sigma)$ or $s\text{-test}_t(a, l, \sigma)$:

$$\begin{aligned} (G_1 \wedge G_2) \rightarrow A &\text{ represents } (G_1 \rightarrow A) \parallel (G_2 \rightarrow A); \\ (G_1 \vee G_2) \rightarrow A &\text{ represents } (G_1 \rightarrow A) + (G_2 \rightarrow A); \\ \text{true} &\text{ represents a dummy } \text{check}_\infty(\{\}, \{\}); \\ \text{false} &\text{ represents a dummy } \text{check}_0(\{\}, \{\}); \\ \exists x \in S \mid \text{check}_t(\{x\}, R) \rightarrow A &\text{ represents } (\text{check}_t(\{a_1\}, R) \rightarrow A) + \\ &(\text{check}_t(\{a_2\}, R) \rightarrow A) + \dots + (\text{check}_t(\{a_n\}, R) \rightarrow A). \\ \text{Analogously for } \exists x \in S \mid c\text{-test}_t(x, l, \sigma) \rightarrow A &\text{ and} \\ \exists x \in S \mid s\text{-test}_t(x, l, \sigma) \rightarrow A; \\ \forall x \in S \mid \text{check}_t(\{x\}, R) \rightarrow A &\text{ represents } (\text{check}_t(\{a_1\}, R) \rightarrow A) \parallel \\ &(\text{check}_t(\{a_2\}, R) \rightarrow A) \parallel \dots \parallel (\text{check}_t(\{a_n\}, R) \rightarrow A). \\ \text{Analogously for } \forall x \in S \mid c\text{-test}_t(x, l, \sigma) \rightarrow A &\text{ and} \\ \forall x \in S \mid s\text{-test}_t(x, l, \sigma) \rightarrow A. \end{aligned}$$

4. Modelling Multi-Agent Decision Making with Privacy Preserved in TCLA

A possible use case for TCLA can be identified in modelling Multi-Agent Decision Making with Privacy Preserved (DMPP) in which agents need to communicate with other agents to make

socially optimal decisions but, at the same time, have some private information that they do not want to share. This problem can be instantiated as done in other works like [14] as a debate in a multi-agent environment where argumentation techniques are exploited for arriving at socially optimal outcomes by only revealing the necessary” and “disclosable” information. We start from (a slight modification of) the scenario proposed by [14], adapted from the Battle of the Sexes game in [15] and the meeting scheduling problem in [16]. In this example, we use the true concurrency operator from Table 7 to handle the parallel execution of agents. However, the illustrated methodology could be adjusted to also work with an interleaving approach.

Example 1. Alice and Bob are trying to agree on an activity to do together for the day. Alice is more interested in going to the ballet (we represent this statement with A:Ballet), while Bob would rather watch a football game (B:Football). Their beliefs can be modelled as in the AFs of Figure 1.

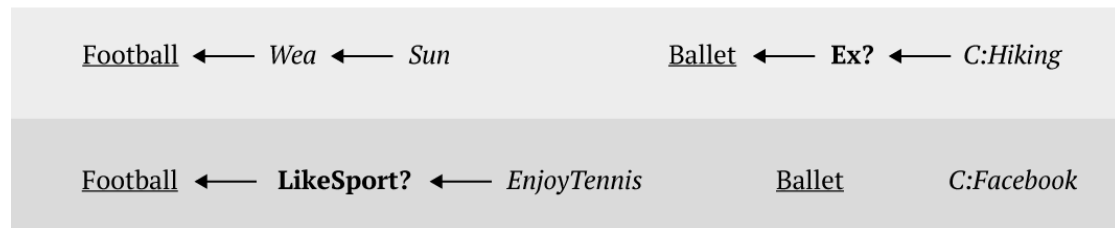


Figure 1: AFs representing Alice's (top) and Bob's (bottom) beliefs and observation [14].

Alice is worried that Bob's ex-wife might attend the ballet as well (denoted by **Ex?**), but she does not want Bob to know about this concern. However, Alice was informed by Caroline, Bob's ex-wife's daughter, that she had gone hiking with her mother earlier that day (*C:Hiking*). Alice would also be concerned about the weather (*Wea*) in case they decide to go watch the game (Football). However, she found from a forecast that it will be sunny (*Sun*). We also know that Alice does not mind sharing her concerns about the weather.

On the other hand, Bob forgot whether Alice enjoys sport or not (**LikeSport?**) and would prefer Alice not to be aware of this. However, they recently attended a tennis match, which Alice enjoyed (*EnjoyTennis*). Bob also came across a Facebook post by Caroline, mentioning that she was attending the ballet (*C:Facebook*). Note that *C:Facebook* is in conflict with *C:Hiking*, but this will only emerge when Alice and Bob will share their arguments.

A Multi-Agent Decision Making with Privacy Preserved problem is formalised as follows.

Definition 3. A Multi-Agent Decision Making with Privacy Preserved problem is a triple $DMPP = \langle Ag, Act, Sol \rangle$, where

- $Ag = \{Agent_1, \dots, Agent_N\}$ is a finite set of agents;
- $Act = \{a_1, \dots, a_M\}$ is a finite set of available actions for the agents;
- $SP = \{\langle Agent_1 : a^1, \dots, Agent_N : a^N \rangle \mid \{a^1, \dots, a^N\} \subseteq Act\}$ the set of all strategic profiles, namely the set of all the possible tuples of actions (one for each agent) and
- $Sol \subseteq SP$ is the set of the acceptable solutions of the problem.

Example 2. In Example 1, $N = 2$, since there are two agents: A (Alice) and B (Bob), $M = 2$ since there are two actions Ballet and Football (‘go to the ballet’ and ‘watch football’, respectively) and two acceptable solutions of the problem, namely $Sol = \{\langle A:\underline{\text{Ballet}}, B:\underline{\text{Ballet}} \rangle, \langle A : \underline{\text{Football}}, B:\underline{\text{Football}} \rangle\}$. Note that the two strategic profiles $\langle A:\underline{\text{Ballet}}, B:\underline{\text{Football}} \rangle$ and $\langle A : \underline{\text{Football}}, B:\underline{\text{Ballet}} \rangle$ are not acceptable solutions, because both agents want to attend these activities together (or not at all).

Differently from [14], we model agents’ actions and acceptable solutions, which are common and visible to all agents, as public arguments. Each agent $Agent_i$ can decide for its own actions and not for those of the other agents. Therefore, the actions of one agent are disjoint from those of the others and are of the form $Agent_i : a_j$, where $a_j \in Act$. We define the attacks between agents’ actions and acceptable solutions as follows:

$$NoG = \{(Agent_i : a, \langle Agent_1 : b_1, \dots, Agent_N : b_N \rangle \mid i \in \{1, \dots, N\}, a \in Act_i, \langle Agent_1 : b_1, \dots, Agent_N : b_N \rangle \in Sol, \text{ and } a \neq b_i\}$$

Example 3. In Example 1 we have that

$$NoG = \{ \begin{array}{l} (A:\underline{\text{Ballet}}, \langle A : \underline{\text{Football}}, B:\underline{\text{Football}} \rangle), \\ (A : \underline{\text{Football}}, \langle A:\underline{\text{Ballet}}, B:\underline{\text{Ballet}} \rangle), \\ (B:\underline{\text{Ballet}}, \langle A : \underline{\text{Football}}, B:\underline{\text{Football}} \rangle), \\ (B:\underline{\text{Football}}, \langle A:\underline{\text{Ballet}}, B:\underline{\text{Ballet}} \rangle) \end{array} \}.$$

Note that in this formalisation the arguments Ballet and Football are private arguments for Alice who uses them to privately decide what action to take, while A:Ballet and A:Football are public arguments with which Alice informs Bob of the action she would like to take. Analogously for Bob.

Now, let us consider a communication protocol inspired by Chronological synchronous backtracking (SBT), one of the simplest while most fundamental algorithms for distributed constraint satisfaction problems, which requires a static ordering of agents. Following this, agents try to make a social decision. Agents pass a token among them; the agent who has the token can check if the current solution (proposed by the first agent) is acceptable to him. In this case, send the token to the next agent; otherwise, send a message *ngd* (short for “not good”) to the previous agent. The communication terminates either because all the agents agree on a solution or because every solution proposed by the first agent has been discarded. In the former case, the partial solution constitutes a solution for the problem, while in the latter case, the problem is unsatisfiable. SBT is guaranteed to terminate and is sound and complete (i.e., it terminates only with correct answers and for all problems).

We can write a TCLA program emulating a multi-agent decision making with privacy preserved to model the Battle of Sexes game. We know that Alice’s preference is the sequence of actions Ballet · Football, while Bob’s preference is the sequence of actions Football · Ballet. Moreover, Alice and Bob’s private information is represented by the following two AFs, respectively:

$$\begin{aligned} AFp_A = & \langle \{ \underline{\text{Ballet}}, \underline{\text{Football}}, \mathbf{Ex?}, C:\underline{\text{Hiking}}, \underline{\text{Wea}} \}, \\ & \{ (\mathbf{Ex?}, \underline{\text{Ballet}}), (\underline{\text{Wea}}, \underline{\text{Football}}), (C:\underline{\text{Hiking}}, \mathbf{Ex?}) \}, \text{ and} \\ AFp_B = & \langle \{ \underline{\text{Ballet}}, \underline{\text{Football}}, \underline{\text{LikeSport?}}, \underline{\text{EnjoyTennis}} \}, \end{aligned}$$

$$\{(EnjoyTennis, LikeSport?), (LikeSport?, Football)\}.$$

The interaction that takes place between Alice and Bob in order to reach a jointly agreed decision can be modelled in TCLA as reported in Tables 9 and 10, respectively. The computation of $\mathcal{T}(A) \parallel \mathcal{T}(B)$ starts in an initial public shared framework

$$\begin{aligned} AFd = & \langle \{C:Hiking, Wea, Sun, EnjoyTennis, C:Facebook, \\ & \langle A:Ballet, B:Ballet \rangle, \langle A:Football, B:Football \rangle \}, \\ & \{(Sun, Wea), (C:Facebook, C:Hiking)\} \rangle. \end{aligned}$$

Suppose Alice is ranked higher in the static agent ordering; thus, she obtains the token first. Since a final agreement has to be reached, the order of the participants does not influence the final result in this example. Different outcomes may, however, be ob-

Table 9

Alice's behaviour in TCLA.

$\mathcal{T}(A) = new(\{A:Ballet, Football, Ex?\})$ in $T_A(Ballet \cdot Football)^{AFPA}$, where

$$\begin{aligned} T_A(Ballet \cdot Football) = & \\ & c-test_1(\underline{Ballet}, in, adm) \rightarrow \\ & (add(\{A:Ballet\}, \{(A:Ballet, \langle A:Football, B:Football \rangle)\}) \rightarrow \\ & ((c-test_1(\langle A:Ballet, B:Ballet \rangle, in, adm) \vee \\ & c-test_1(\langle A:Football, B:Football \rangle, in, adm)) \rightarrow \\ & (add(\{tok_B\}, \emptyset) \rightarrow \\ & (check_\infty(\{gd\}, \emptyset) \rightarrow success + \\ & check_\infty(\{ngd_A\}, \emptyset) \rightarrow rmv(\{ngd_A, A:Ballet\}, \emptyset) \rightarrow \\ & T_A(Football)))) \\ & +_P \\ & true \rightarrow rmv(\{A:Ballet\}, \emptyset) \rightarrow T_A(Football)) \\ & +_P \\ & true \rightarrow T_A(Football) \\ \text{and} \\ T_A(Football) = & \\ & c-test_1(\underline{Football}, in, adm) \rightarrow \\ & (add(\{A:Football\}, \{(A:Football, \langle A:Ballet, B:Ballet \rangle)\}) \rightarrow \\ & ((c-test_1(\langle A:Ballet, B:Ballet \rangle, in, adm) \vee \\ & c-test_1(\langle A:Football, B:Football \rangle, in, adm)) \rightarrow \\ & (add(\{tok_B\}, \emptyset) \rightarrow \\ & (check_\infty(\{gd\}, \emptyset) \rightarrow success + \\ & check_\infty(\{ngd_A\}, \emptyset) \rightarrow rmv(\{ngd_A, A:Football\}, \emptyset) \rightarrow \\ & failure))) \\ & +_P \\ & true \rightarrow rmv(\{A:Football\}, \emptyset) \rightarrow failure) \\ & +_P \\ & true \rightarrow failure \end{aligned}$$

proposing his own actions. He begins with his favourite action Football (“watching football”), and he finds that it is feasible by using $c\text{-test}_1(\text{Football}, in, adm)$ in $AFp_B \cup AFd'$. So he adds $\{B:\text{Football}\}, \{(A:\text{Football}, \langle A:\text{Ballet}, B:\text{Ballet} \rangle)\}$ (Bob, “watch football”) to AFd' . Then he analyses the consistency of B:Football with respect to the current partial solution (which now includes Alice’s proposal A:Football) and the acceptable solutions. Analogously to Alice, he executes $c\text{-test}_1(\langle A:\text{Ballet}, B:\text{Ballet} \rangle, in, adm) \vee c\text{-test}_1(\langle A:\text{Football}, B:\text{Football} \rangle, in, adm)$. Again, we can find an acceptable solution $\langle A:\text{Football}, B:\text{Football} \rangle$ in the global framework which is admissible. Since Bob is the last agent, he adds gd to the global framework and terminates with success. Finally, Alice verifies the presence of the token gd in the global framework and terminates with success as well. In the global framework, we find the solution to the problem, which is the only admissible acceptable solution $\langle A:\text{Football}, B:\text{Football} \rangle$.

Note that we can construct another model where Alice and Bob always have a private knowledge base, but the two agents simply work concurrently. However, the illustrated methodology could be easily adjusted to also model generic Multi-Agent Decision Making with Privacy Preserved, with any number of agents.

5. Conclusion

We have introduced a new version of TCLA with local stores to allow agents to keep important data private while still incorporating it into their decision-making processes. As a result, our language is well-suited to model real-life scenarios, and we have provided an example of how it can be used in a privacy-preserving setting to make decisions.

In the future, we plan to further extend TCLA to capture more aspects that may influence the behaviour of agents interacting through the exchange of arguments. For instance, we want to endow the agents with a notion of ownership to establish which actions can be performed on the shared arguments. In the current implementation, an autonomous agent could remove arguments added to the shared store by its opponents in order to win a debate. However, this is not an effective solution in practical cases and is also considered an illegal move in dialogue games. In addition, since the reasoning process of an agent involved in some form of interaction (e.g. persuasion) usually includes elaborating a winning strategy, we want to introduce TCLA constructs that a user can employ to identify the best sequence of actions to perform. To this end, we could resort either to a greedy approach, in which the agent identifies the best actions with respect to a given time instant (and configuration of the shared store), or to a more robust, optimised strategy that spans over a certain number of execution steps. Together with true concurrency, we also want to include an interleaving operator for handling parallel agents. The user may rely on both constructs depending on the purpose. Finally, we plan to use the language to model chatbot interactions [17] and explanation activities [18].

Acknowledgments

The authors are members of the INdAM Research group GNCS and of the Consorzio CINI. This work has been partially supported by: Project “Empowering Public Interest Communication with Argumentation (EPICA)”, (MIUR PRIN); INdAM - GNCS Project, CUP E53C22001930001;

Projects BLOCKCHAIN4FOODCHAIN, FICO, AIDMIX, “Civil Safety and Security for Society” (“Fondo Ricerca di Ateneo”, University of Perugia; years 2020, 2021, 2022); Project GIUSTIZIA AGILE, CUP J89J22000900005; Project VITALITY, CUP J97G22000170005 (NRRP-MUR).

References

- [1] A. C. Kakas, P. Moraitis, Adaptive agent negotiation via argumentation, in: H. Nakashima, M. P. Wellman, G. Weiss, P. Stone (Eds.), 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), Hakodate, Japan, May 8-12, 2006, ACM, 2006, pp. 384–391. URL: <https://doi.org/10.1145/1160633.1160701>. doi:10.1145/1160633.1160701.
- [2] I. Rahwan, S. D. Ramchurn, N. R. Jennings, P. McBurney, S. Parsons, L. Sonenberg, Argumentation-based negotiation, *Knowl. Eng. Rev.* 18 (2003) 343–375. URL: <https://doi.org/10.1017/S0269888904000098>. doi:10.1017/S0269888904000098.
- [3] K. Atkinson, T. J. M. Bench-Capon, P. McBurney, Multi-agent argumentation for edemocracy, in: M. Gleizes, G. A. Kaminka, A. Nowé, S. Ossowski, K. Tuyls, K. Verbeeck (Eds.), EUMAS 2005 - Proceedings of the Third European Workshop on Multi-Agent Systems, Brussels, Belgium, December 7-8, 2005, Koninklijke Vlaamse Academie van Belie voor Wetenschappen en Kunsten, 2005, pp. 35–46.
- [4] A. Rosenfeld, S. Kraus, Strategic argumentative agent for human persuasion, in: G. A. Kaminka, M. Fox, P. Bouquet, E. Hüllermeier, V. Dignum, F. Dignum, F. van Harmelen (Eds.), ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016), volume 285 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2016, pp. 320–328. URL: <https://doi.org/10.3233/978-1-61499-672-9-320>. doi:10.3233/978-1-61499-672-9-320.
- [5] S. Bistarelli, M. C. Meo, C. Taticchi, Timed concurrent language for argumentation with maximum parallelism, *Journal of Logic and Computation* 33 (2023) 712–737. URL: <https://doi.org/10.1093/logcom/exad009>. doi:10.1093/logcom/exad009.
- [6] S. Bistarelli, M. C. Meo, C. Taticchi, Timed concurrent language for argumentation: An interleaving approach, in: J. Cheney, S. Perri (Eds.), Practical Aspects of Declarative Languages - 24th International Symposium, PADL 2022, Philadelphia, PA, USA, January 17-18, 2022, Proceedings, volume 13165 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 101–116. URL: https://doi.org/10.1007/978-3-030-94479-7_7. doi:10.1007/978-3-030-94479-7_7.
- [7] P. M. Dung, On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, *Logic Programming and n-Person Games*, *Artif. Intell.* 77 (1995) 321–358. doi:10.1016/0004-3702(94)00041-X.
- [8] P. Baroni, M. Caminada, M. Giacomin, An introduction to argumentation semantics, *Knowl. Eng. Rev.* 26 (2011) 365–410. doi:10.1017/S0269888911000166.
- [9] M. Caminada, On the issue of reinstatement in argumentation, in: Proc. of JELIA 2006 - 10th European Conference on Logics in Artificial Intelligence, volume 4160 of *LNCS*, Springer, 2006, pp. 111–123. URL: https://doi.org/10.1007/11853886_11.

- [10] S. Bistarelli, C. Taticchi, A concurrent language for modelling agents arguing on a shared argumentation space, *Argument & Computation Pre-press* (2023) 1–28. doi:10.3233/AAC-210027.
- [11] V. A. Saraswat, M. Rinard, Concurrent constraint programming, in: *Proc. of POPL 1990 - 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ACM Press, 1990, pp. 232–245. doi:10.1145/96709.96733.
- [12] C. A. R. Hoare, Communicating sequential processes, *Commun. ACM* 21 (1978) 666–677. URL: <https://doi.org/10.1145/359576.359585>.
- [13] R. Milner, *A Calculus of Communicating Systems*, volume 92 of *LNCS*, Springer, 1980. URL: <https://doi.org/10.1007/3-540-10235-3>.
- [14] Y. Gao, F. Toni, H. Wang, F. Xu, Argumentation-based multi-agent decision making with privacy preserved, in: *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, Singapore, May 9-13, 2016, ACM, 2016, pp. 1153–1161.
- [15] I. Rahwan, K. Larson, Argumentation and game theory, in: *Argumentation in Artificial Intelligence*, Springer, 2009, pp. 321–339.
- [16] M. Silaghi, D. Mitra, Distributed constraint satisfaction and optimization with privacy enforcement, in: *2004 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT 2004)*, 20-24 September 2004, Beijing, China, IEEE Computer Society, 2004, pp. 531–535.
- [17] S. Bistarelli, C. Taticchi, F. Santini, A chatbot extended with argumentation, in: M. D’Agostino, F. A. D’Asaro, C. Larese (Eds.), *Proceedings of the 5th Workshop on Advances in Argumentation in Artificial Intelligence 2021 co-located with the 20th International Conference of the Italian Association for Artificial Intelligence (AIXIA 2021)*, Milan, Italy, November 29th, 2021, volume 3086 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021. URL: <https://ceur-ws.org/Vol-3086/short2.pdf>.
- [18] S. Bistarelli, A. Mancinelli, F. Santini, C. Taticchi, Arg-xai: a tool for explaining machine learning results, in: M. Z. Reformat, D. Zhang, N. G. Bourbakis (Eds.), *34th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2022*, Macao, China, October 31 - November 2, 2022, IEEE, 2022, pp. 205–212. URL: <https://doi.org/10.1109/ICTAI56018.2022.00037>. doi:10.1109/ICTAI56018.2022.00037.