Concurrent Argumentation with Time: an Overview

Stefano Bistarelli¹, Maria Chiara Meo² and Carlo Taticchi¹

¹University of Perugia, Perugia, Italy

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

Abstract

The Timed Concurrent Language for Argumentation (tcla) is a framework to model concurrent interactions between communicating agents that reason and take decisions through argumentation processes, also taking into account the temporal duration of the performed actions. Time is, indeed, a crucial factor when dealing with dynamic environments in real-world applications, where agents need to act in a coordinated fashion to reach their own goals. In this paper, we discuss the syntax and the operational semantics of tcla, providing insights on how its constructs can be used to realise complex interactions between agents.

Keywords

argumentation, time, concurrency

1. Introduction

Argumentation Theory pursues the objective of studying how conclusions can be reached, starting from a set of assumptions, through a process of logical reasoning. This process is very similar to the human way of thinking and involves features which can be traced to a form of dialogue between two (or more) people. Abstract Argumentation Frameworks [1], AFs in short, are used to study the acceptability of arguments according to given selection criteria. Frameworks like the Timed Abstract Argumentation Framework (TAF) [2] have been proposed to meet the need for including the notion of time into argumentation processes. Time is a particularly important aspect of cooperative environments: in many "real-life" applications the activities have a temporal duration (that can even be interrupted) and the coordination of such activities has to take into consideration this time dependence. A mechanism for handling time is therefore required to better model the behaviour of intelligent agents involved in argumentation processes.

In this paper, we introduce the Timed Concurrent language for Argumentation (tcla) [3], a timed extension of cla [4, 5, 6], which models dynamic interactions between agents and uses notions from Argumentation Theory to reason about shared knowledge. We consider a paradigm where parallel operations are expressed in term of maximal parallelism.

 $^{5^{}th}$ Workshop on Advances In Argumentation In Artificial Intelligence (Al³ 2021)

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

^{© 2021} Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

2. Argumentation Theory

Argumentation Theory aims to understand and model the human natural fashion of reasoning, allowing one to deal with uncertainty in non-monotonic (defeasible) reasoning. In his seminal paper [1], Dung defines the building blocks of abstract argumentation.

Definition 1 (AFs). Let U be the set of all available arguments¹, that we call "universe". 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.

For two arguments $a, b \in Arg$, the notation $(a, b) \in R$ represents an attack directed from a against b. Acceptability of arguments is then computed through methodologies like the reinstatement labelling presented in [7], starting from attack relations in the framework.

Definition 2 (Reinstatement labelling). Let $F = \langle Arg, R \rangle$ be an AF and consider the set of labels $\mathbb{L} = \{$ in, out, undec $\}$. A labelling of F is a total function $L : Arg \to \mathbb{L}$. We define $in(L) = \{a \in Arg \mid L(a) = in\}, out(L) = \{a \in Arg \mid L(a) = out\}$ and $undec(L) = \{a \in Arg \mid L(a) = undec\}$. We say that L is a reinstatement labelling if and only if it satisfies the following:

- $\forall a \in Arg : a \in in(L) \iff \forall b \in Arg \mid (b,a) \in R : b \in out(L)$
- $\forall a \in Arg : a \in out(L) \iff \exists b \in Arg \mid (b,a) \in R \land b \in in(L)$

A labelling-based semantics [8] associates with an AF a subset of all the possible labellings. In Figure 1 we show an example of reinstatement labelling on an AF in which arguments a and c highlighted in green are in, red ones (b and d) are out, and the yellow argument e (that attacks itself) is undec.

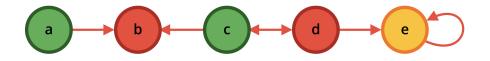


Figure 1: Example of reinstatement labelling.

Given a labelling L, it is possible to identify a correspondence with the extension-based semantics [8]. In particular, the set of in arguments coincides with a complete extension, while other semantics can be obtained through restrictions (as shown in Table 1 of [7]). In the following, we use $L \in S_{\sigma}(F)$ to denote a labelling L corresponding to an extension of the semantics σ . Besides computing a labelling for a certain semantics σ , one of the most common tasks performed on AFs is to verify whether an argument a has a certain label l in some labelling (credulous test) or in all labellings (sceptical test).

¹The set U is not present in the original definition by Dung and we introduce it for our convenience.

3. tcla Syntax and Semantics

The syntax of *tcla* is presented in Table 1, where P, C, A and E denote a generic process, a sequence of procedure declarations (or clauses)², a generic agent and a generic guarded agent, respectively. In Table 2, then, we give the definitions for the transition rules.

Table 1 tcla syntax.

$P ::= C.A, \ C ::= C.C$
$A ::= success \mid failure \mid add(Arg, R) \rightarrow A \mid rmv(Arg, R) \rightarrow A \mid E \mid A \parallel A$
$E ::= check_t(Arg, R) \to A \mid c\text{-}test_t(a, l, \sigma) \to A \mid s\text{-}test_t(a, l, \sigma) \to A \mid E + E \mid E +_P E \mid E \parallel_G E$

Communication between *tcla* agents is implemented via shared memory, similarly to *cla* [5] and CC [9]. In the following, we denote by \mathcal{E} the class of guarded agents and by \mathcal{E}_0 the class of guarded agents such that all outermost guards have t = 0. Suppose we have an agent A whose knowledge base is represented by a framework $F = \langle Arg, R \rangle$. An add(Arg', R') action performed by the agent results in the addition of a set of arguments $Arg' \subseteq U$ (where U is the universe) and a set of relations R' to F. When performing an addition, (possibly) new arguments are taken from $U \setminus Arg$. Intuitively, rmv(Arg, R) allows to specify arguments and/or attacks to be removed from the knowledge base. The removal of an argument from an AF also involves removing all attack relations outgoing from and incoming to that argument, thus making F preserve the structure of an AF.

The $c\text{-}test_t(a, l, \sigma) \rightarrow A$, $s\text{-}test_t(a, l, \sigma) \rightarrow A$ and $check_t(Arg, R) \rightarrow A$ constructs are explicit timing primitives that allow for the specification of timeouts. The operator $check_t(Arg', R')$ (rules Ch (1)-(3) in Table 2) realises a timed construct and is used to verify whether, in a given time interval, the specified arguments and attacks are contained in the knowledge base, without introducing any further change. If t > 0 and the check is positive, the operation succeeds and the agent $check_t(Arg', R') \rightarrow A$ can perform the subsequent action (rule Ch (1)). If t > 0 but the check is not satisfied, then the control is repeated at the next time instant and the value of the counter t is decreased (rule Ch (2)). Axiom Ch (3) shows that, if the timeout is exceeded, i.e., the counter t has reached the value of 0, then the process $check_t(Arg', R') \rightarrow A$ fails. The rules for credulous tests CT (1)-(3) and sceptical tests ST (1)-(3) in Table 2 require the specification of an argument $a \in Arg$, a label $l \in \{in, out, undec\}$ and a semantics σ [1]. $c\text{-}test_t(a, l, \sigma)$ succeeds if there exists at least one extension of $S_{\sigma}(F)$ whose labelling L is such that L(a) = l, while $s\text{-}test_t(a, l, \sigma)$ succeeds if a is labelled l in all possible labellings $L \in S_{\sigma}(F)$.

The operator $+_P$ (If (1)-(2) in Table 2) is left-associative and realises an if-then-else construct: if we have $E_1 +_P E_2$ (with $E_1, E_2 \in \mathcal{E}$) and the guards of E_1 succeed, than E_1 is chosen over E_2 .In order for E_2 to be selected, it has to be the only one such that its guards succeed and will be selected only after E_1 fails. If the guards of E_1 do not fail, the execution can either move to any consequent agent A_1 which does not belong to \mathcal{E} , or proceed with $E'_1 +_P E_2 \in \mathcal{E}$.

²Note that infinite sequences of procedure declarations are also allowed.

Table 2

tcla operational semantics.

$$\begin{split} \langle add(Arg', R') \to A, \langle Arg, R \rangle \rangle &\longrightarrow \langle A, \langle Arg \cup Arg', R \cup R'' \rangle \rangle \\ \text{with } R'' = \{(a, b) \in R' \mid a, b \in Arg \cup Arg'\} \end{split}$$

$$\begin{array}{l} \langle rmv(Arg', R') \to A, \langle Arg, R \rangle \rangle \longrightarrow \langle A, \langle Arg \setminus Arg', R \setminus \{R' \cup R''\} \rangle \rangle \\ \text{with } R'' = \{(a, b) \in R \mid a \in Arg' \lor b \in Arg'\} \end{array}$$

$$\frac{Arg' \subseteq Arg \land R' \subseteq R \quad t > 0}{\langle check_t(Arg', R') \to A, \langle Arg, R \rangle \rangle \longrightarrow \langle A, \langle Arg, R \rangle \rangle}$$
Ch (1)

$$\frac{Arg' \not\subseteq Arg \lor R' \not\subseteq R \quad t > 0}{\langle check_t(Arg', R') \to A, \langle Arg, R \rangle \rangle \longrightarrow \langle check_{t-1}(Arg', R') \to A, \langle Arg, R \rangle \rangle} \qquad \mathsf{Ch} (2)$$

$$\langle check_0(Arg', R') \to A, F \rangle \longrightarrow \langle failure, F \rangle$$
 Ch (3)

$$\frac{\exists L \in S_{\sigma}(F) \mid l = L(a) \quad t > 0}{\langle c\text{-}test_t(a, l, \sigma) \to A, F \rangle \longrightarrow \langle A, F \rangle}$$
CT (1)

$$\frac{\forall L \in S_{\sigma}(F).l \neq L(a) \quad t > 0}{\langle c\text{-}test_t(a,l,\sigma) \to A, F \rangle \longrightarrow \langle c\text{-}test_{t-1}(a,l,\sigma) \to A, F \rangle}$$
CT (2)

$$\langle c\text{-}test_0(a,l,\sigma) \to A, F \rangle \longrightarrow \langle failure, F \rangle$$
 CT (3)

$$\frac{\forall L \in S_{\sigma}(F).l = L(a) \quad t > 0}{\langle s \cdot test_t(a, l, \sigma) \to A, F \rangle \longrightarrow \langle A, F \rangle}$$
 ST (1)

$$\frac{\exists L \in S_{\sigma}(F) \mid l \neq L(a) \quad t > 0}{\langle s \text{-}test_t(a, l, \sigma) \to A, F \rangle \longrightarrow \langle s \text{-}test_{t-1}(a, l, \sigma) \to A, F \rangle}$$
 ST (2)

$$\langle s\text{-}test_0(a,l,\sigma) \longrightarrow A, F \rangle \longrightarrow \langle failure, F \rangle$$
 ST (3)

$$\frac{\langle E_1, F \rangle \longrightarrow \langle A_1, F \rangle, \quad E_1 \notin \mathcal{E}_0, \quad A_1 \notin \mathcal{E}}{\langle E_1 + P E_2, F \rangle \longrightarrow \langle A_1, F \rangle}$$
 If (1)

$$\frac{\langle E_1, F \rangle \longrightarrow \langle E'_1, F \rangle, \quad E_1 \notin \mathcal{E}_0, \quad E'_1 \in \mathcal{E}}{\langle E_1 +_P E_2, F \rangle \longrightarrow \langle E'_1 +_P E_2, F \rangle} \qquad \frac{E_1 \in \mathcal{E}_0, \langle E_2, F \rangle \longrightarrow \langle A_2, F \rangle}{\langle E_1 +_P E_2, F \rangle \longrightarrow \langle A_2, F \rangle} \qquad \text{If (2)}$$

$$\frac{\langle E_1, F \rangle \longrightarrow \langle A_1, F \rangle, \ \langle E_2, F \rangle \longrightarrow \langle A_2, F \rangle, \ E_1, E_2 \notin \mathcal{E}_0, \ A_1 \notin \mathcal{E}}{\langle E_1 \|_G E_2, F \rangle \longrightarrow \langle A_1 \| A_2, F \rangle}$$
GP (1)

$$\frac{\langle E_1, F \rangle \longrightarrow \langle E'_1, F \rangle, \ \langle E_2, F \rangle \longrightarrow \langle E'_2, F \rangle, \quad E_1, E_2 \notin \mathcal{E}_0, \quad E'_1, E'_2 \in \mathcal{E}}{\langle E_1 \|_G E_2, F \rangle \longrightarrow \langle E'_1 \|_G E'_2, F \rangle}$$
GP (2)

$$\frac{E_1 \in \mathcal{E}_0, \langle E_2, F \rangle \longrightarrow \langle A_2, F \rangle}{\langle E_1 \|_G E_2, F \rangle \longrightarrow \langle A_2, F \rangle}$$
GP (3)

$$\frac{\langle A_1, F \rangle \longrightarrow \langle A'_1, F' \rangle, \ \langle A_2, F \rangle \longrightarrow \langle A'_2, F'' \rangle}{\langle A_1 \| A_2, F \rangle \longrightarrow \langle A'_1 \| A'_2, *(F, F', F'') \rangle}$$
 Pa

$$\frac{\langle E_1, F \rangle \longrightarrow \langle A_1, F \rangle, \quad E_1 \notin \mathcal{E}_0, \quad A_1 \notin \mathcal{E}}{\langle E_1 + E_2, F \rangle \longrightarrow \langle A_1, F \rangle} \qquad \frac{E_1 \in \mathcal{E}_0, \langle E_2, F \rangle \longrightarrow \langle A_2, F \rangle}{\langle E_1 + E_2, F \rangle \longrightarrow \langle A_2, F \rangle} \qquad \mathsf{ND} (1)$$

$$\frac{\langle E_1, F \rangle \longrightarrow \langle E'_1, F \rangle, \ \langle E_2, F \rangle \longrightarrow \langle E'_2, F \rangle, \quad E_1, E_2 \notin \mathcal{E}_0, \quad E'_1, E'_2 \in \mathcal{E}}{\langle E_1 + E_2, F \rangle \longrightarrow \langle E'_1 + E'_2, F \rangle}$$
 ND (2)

The guarded parallelism GP (1)-(3) in Table 2 is designed to allow all the operations for which the guards in the inner expression are satisfied. In more detail, the guards of $E_1||_G E_2$ succeed when either the guards of E_1 , E_2 or both succeed and all the operations that can be executed are executed. This behaviour is different both from classical parallelism (for which all the agents have to succeed in order for the parallel agent to succeed) and from nondeterminism (that only selects one branch). The rule parallelism Pa in Table 2 models the parallel composition operator in terms of *maximal parallelism*: we use $*(F, F', F'') := (F' \cap F'') \cup ((F' \cup F'') \setminus F)$ to handle parallel additions and removals of arguments³. Finally, any agent composed through + (rules ND (1)-(2)) is chosen if its guard succeeds.

4. Conclusion and Future Work

In this paper, we introduced *tcla*, a concurrent argumentation language for modelling interacting agents in which also time is taken into account. We presented both the syntax and the operational semantics. Maximal parallelism has been used to realise simultaneous execution of actions carried forward by different agents. As a future work, we want to adopt an interleaving approach for handling this kind of processes: we plan to have a different transition system in which time elapsing and computational steps are realised by two distinct types of actions.

References

- 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.
- [2] M. L. Cobo, D. C. Martínez, G. R. Simari, On admissibility in timed abstract argumentation frameworks, in: ECAI, volume 215 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2010, pp. 1007–1008.
- [3] S. Bistarelli, M. C. Meo, C. Taticchi, Timed concurrent language for argumentation, in: CILC, volume 3002 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021, pp. 1–15.
- [4] S. Bistarelli, C. Taticchi, Towards an implementation of a concurrent language for argumentation, in: AI*IA, volume 12414 of *LNCS*, Springer, 2020, pp. 154–171.
- [5] S. Bistarelli, C. Taticchi, A concurrent language for argumentation, in: AI³@AI*IA, volume 2777 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2020, pp. 75–89.
- [6] S. Bistarelli, C. Taticchi, Introducing a tool for concurrent argumentation, in: JELIA, volume 12678 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 18–24.
- [7] M. Caminada, On the issue of reinstatement in argumentation, in: JELIA, volume 4160 of Lecture Notes in Computer Science, Springer, 2006, pp. 111–123.
- [8] P. Baroni, M. Caminada, M. Giacomin, An introduction to argumentation semantics, Knowl. Eng. Rev. 26 (2011) 365–410.
- [9] V. A. Saraswat, M. C. Rinard, Concurrent constraint programming, in: POPL, ACM Press, 1990, pp. 232–245.

³Union, intersection and difference between AFs are intended as the union, intersection and difference of their sets of arguments and attacks, respectively.