Timed Concurrent Language for Argumentation*

Stefano Bistarelli[1], Maria Chiara Meo[2], and Carlo Taticchi[1]

[1] Università degli Studi di Perugia, Italy
[stefano.bistarelli,carlo.taticchi]@unipg.it
[2] Università degli Studi G. d'Annunzio di Chieti-Pescara, Italy
mariachiara.meo@unich.it

Abstract. Argumentation Theory offers formalisms for the study of reasoning processes taking place between intelligent entities. In this context, time is a crucial factor: in a real-world environment, activities have a determined temporal duration and the behaviour of agents is influenced by the actions previously taken. While agent-based modelling languages naturally implement concurrency and time constraints, the currently available languages for argumentation do not allow to explicitly model this type of behaviours. In this paper, we propose a language for modelling concurrent interaction between agents that also allows the specification of temporal intervals in which particular actions occur. Such a language, that we call Timed Concurrent Language for Argumentation, allows agents to communicate with each other and to reason on the acceptability of their beliefs with respect to a given time interval. We also show how Timed Abstract Argumentation Frameworks can be modelled by combining time and concurrency.

Keywords: Argumentation Theory · Timed concurrent language · Semantics.

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 [16], AFs in short, are used to study the acceptability of arguments according to given selection criteria. Formalisms like Timed Abstract Argumentation Frameworks (TAFs) [10,14,15] 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 be even interrupted) and the coordination of such activities has to take into consideration this timeliness property. The interacting actors are mutually

influenced by their actions, meaning that agent A reacts accordingly to the timing and quantitative aspects related to the behaviour of agent B, and vice versa. Moreover, the information brought forward by debating agents that interact in a dynamic environment can be affected by time constraints, limiting, for instance, the influence of some arguments in the system to a certain time lapse. A mechanism for handling time is therefore required to better model the behaviour of intelligent agents involved in argumentation processes.

With this paper, we introduce the Timed Concurrent language for Argumentation (*tcla*), a timed extension of *cla* [5,6], which models dynamic interactions between agents and uses notions from Argumentation Theory to reason about shared knowledge. We provide both the syntax and the operational semantics. Coordinating agents that need to take decisions both on arguments and attacks and time events may benefit from this language. Such agents can be used, for instance, to implement TAFs, namely extended AFs in which arguments are only available for a given temporal interval. The language we present is well suited for this kind of tasks, as Section 4 shows with examples. More complex dynamics could also be considered: for example, an argument could be made available only if specific conditions are met, and agents could be given priorities in introducing arguments for certain time intervals.

tcla is based on the hypothesis of *bounded asynchrony* [25]: any computation takes a bounded period of time rather than being instantaneous as in the concurrent synchronous languages `esterel` [3], `lustre` [17], `signal` [20] and `Statecharts` [18]. Time itself is measured by a discrete global clock, i.e., the internal clock of the *tcla* process. In *tcla*, we directly introduce a timed interpretation of the programming constructs by identifying a time-unit with the time needed for the execution of a basic action (that can be an addition, a removal or a check of arguments/attacks, or a semantical test of arguments), and by interpreting action prefixing ($\rightarrow$) as the next-time operator. An explicit timing primitive is also introduced in order to allow for the specification of timeouts. Note that we consider a paradigm where parallel operations are expressed in term of maximal parallelism. According to the maximal parallelism policy, at each moment, every enabled agent of the system is activated, while in the interleaving paradigm only one of the enabled agents is executed instead. By using maximal parallelism, time passes for all the parallel processes involved in a computation. This approach, analogous to the one adopted in [8,4,25], is different from the one of [9], where time elapsing is interpreted in terms of interleaving, although assuming maximal parallelism for actions depending on time. Our proposal is also different from the one in [11], where time does not elapse for timeout constructs.

The rest of the paper is organised as follows: in Section 2 we summarise the most important background notions and frameworks from which *tcla* derives. In Section 3 we present *tcla* and the operational semantics of its agents. Section 4 exemplifies the use of timed paradigms in *tcla* and shows an application example of how a TAF can be dynamically instantiated, highlighting the relation between

tcla and TAFs. Section 5 describes some related work, and Section 6 concludes the paper by also indicating future research lines.

2 Background

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 [16], Dung defines the building blocks of abstract argumentation.

Definition 1 (AFs). *Let U be the set of all available arguments*[3]*, which we refer to as the "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. We define the sets of arguments that attack (and that are attacked by) another argument as follows.

Definition 2 (Attacks). *Let $F = \langle Arg, R\rangle$ be an AF, $a \in Arg$ and $S \subseteq Arg$. We define the sets $a_F^+ = \{b \in Arg \mid (a, b) \in R\}$, $a_F^- = \{b \in Arg \mid (b, a) \in R\}$, $S_F^+ = \bigcup_{a\in S} a_F^+$ and $S_F^- = \bigcup_{a\in S} a_F^-$ (we will omit the subscript F when it is clear from the context).*

Definition 3 (Acceptable Argument). *Given an AF $F = \langle Arg, R\rangle$, an argument $a \in Arg$ is acceptable with respect to $D \subseteq Arg$ if and only if $\forall b \in Arg$ such that $(b, a) \in R$, $\exists c \in D$ such that $(c, b) \in R$, and we say that a is **defended** from D.*

By using the notion of defence as a criterion for distinguishing acceptable sets of arguments (extensions) in the framework, one can further refine the set of selected "good" arguments. The goal is to establish which are the acceptable arguments according to a certain semantics, namely a selection criterion. Non-accepted arguments are rejected. Different kinds of semantics have been introduced [16,1] that reflect qualities which are likely to be desirable. We first recall definitions for the extension-based semantics [16], namely admissible, complete, stable, semi-stable, preferred, and grounded semantics (denoted with *adm*, *com*, *stb*, *sst*, *prf* and *gde*, respectively, and generically with σ).

Definition 4 (Extension-based semantics). *Let $F = \langle Arg, R\rangle$ be an AF. A set $E \subseteq Arg$ is conflict-free in F, denoted $E \in S_{cf}(F)$, when there are no $a, b \in E$ such that $(a, b) \in R$. For $E \in S_{cf}(F)$ we have that:*

- *$E \in S_{adm}(F)$ when each $a \in E$ is defended by E*[4]*;*

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

[4] If $E \in S_{adm}(F)$ we say that E is an admissible extension of F. Analogously for the other semantics.

- $E \in S_{com}(F)$ *when* $E \in S_{adm}(F)$ *and* $\forall a \in Arg$ *defended by* E, $a \in E$;
- $E \in S_{stb}(F)$ *when* $\forall a \in Arg \setminus E$, $\exists b \in E$ *such that* $(b, a) \in R$;
- $E \in S_{sst}(F)$ *when* $E \in S_{com}(F)$ *and* $E \cup E^{+}$ *is maximal;*
- $E \in S_{prf}(F)$ *when* $E \in S_{adm}(F)$ *and* E *is maximal;*
- $E \in S_{gde}(F)$ *when* $E \in S_{com}(F)$ *and* E *is minimal.*

Besides enumerating the extensions for a certain semantics σ, one of the most common tasks performed on AFs is to decide whether an argument a is accepted in some extension of $S_\sigma(F)$ or in all extensions of $S_\sigma(F)$. 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.

Example 1. In Figure 1 we provide an example of AF where sets of extensions are given for all the mentioned semantics: $S_{cf}(F) = \{\{\}, \{a\}, \{b\}, \{c\}, \{d\}, \{a, c\}, \{a, d\}, \{b, d\}\}$, $S_{adm}(F) = \{\{\}, \{a\}, \{c\}, \{d\}, \{a, c\}, \{a, d\}\}$, $S_{com}(F) = \{\{a\}, \{a, c\}, \{a, d\}\}$, $S_{prf}(F) = \{\{a, c\}, \{a, d\}\}$, $S_{stb}(F) = S_{sst}(F) = \{\{a, d\}\}$, and $S_{gde}(F) = \{\{a\}\}$. To conclude the example, we want to point out that argument a is sceptically accepted with respect to the complete semantics, since it appears in all three subsets of $S_{com}(F)$. On the other hand, arguments c, that is in just one complete extension, is credulously accepted with respect to the complete semantics.

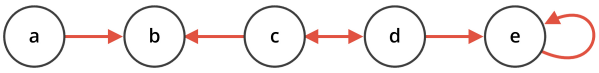

Fig. 1. Example of abstract argumentation framework.

Many of the above-mentioned semantics (such as the admissible and the complete ones) exploit the notion of defence in order to decide whether an argument is part of an extension or not. The phenomenon for which an argument is accepted in some extension because it is defended by another argument belonging to that extension is known as *reinstatement* [12]. In the same paper, Caminada also gives a definition for a reinstatement labelling.

Definition 5 (Reinstatement labelling). *Let* $F = \langle Arg, R \rangle$ *be an AF and* $\mathbb{L} = \{\mathsf{in}, \mathsf{out}, \mathsf{undec}\}$. *A labelling of* F *is a total function* $L : Arg \to \mathbb{L}$. *We define* $in(L) = \{a \in Arg \mid L(a) = \mathsf{in}\}$, $out(L) = \{a \in Arg \mid L(a) = \mathsf{out}\}$ *and* $undec(L) = \{a \in Arg \mid L(a) = \mathsf{undec}\}$. *We say that* L *is a reinstatement labelling if and only if it satisfies the following:*

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

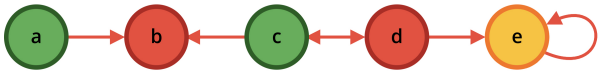

Fig. 2. Example of reinstatement labelling.

In other words, an argument is labelled in if all its attackers are labelled out, and it is labelled out if at least an in node attacks it. In all other cases, the argument is labelled undec. A labelling-based semantics [1] associates with an AF a subset of all the possible labellings. In Figure 2 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 the yellow argument e (that attacks itself) is undec.

Table 1. Reinstatement labelling vs semantics.

Labelling restrictions	Semantics
no restrictions	complete
empty undec	stable
minimal undec	semi-stable
maximal in	preferred
maximal out	preferred
maximal undec	grounded
minimal in	grounded
minimal out	grounded

Given a labelling L, it is possible to identify a correspondence with the extension-based semantics [1]. In particular, the set of in arguments coincides with a complete extension, while other semantics can be obtained through restrictions on the labelling as shown in Table 1.

The notion of time can be included into the reasoning model of AFs by considering temporal intervals [15] defined as follows.

Definition 6 (Temporal Interval). *A temporal interval is a pair built from $t_1, t_2 \in \mathbb{Z}$ in one of the following ways:*

- $[t_1, t_1]$*, denoting the moment* t_1*;*
- $[t_1, \infty)$*, a set of moments formed by all the numbers in* $\mathbb{Z}$ *greater than or equal to* t_1*;*
- $(-\infty, t_2]$*, a set of moments formed by all the numbers in* $\mathbb{Z}$ *smaller than or equal to* t_2*;*
- $[t_1, t_2]$*, a set of moments formed by all the numbers in* $\mathbb{Z}$ *from* t_1 *to* t_2 *(both included);*
- $(-\infty, \infty)$*, denoting a set of moments formed by all the numbers in* $\mathbb{Z}$*.*

The set of all the intervals defined over $\mathbb{Z} \cup \{-\infty, \infty\}$ *is denoted* Υ.

In Timed Argumentation Frameworks [14,15], each argument is associated with temporal intervals that express the period of time in which the argument is available.

Definition 7 (TAFs). *A timed abstract argumentation framework is a tuple* $\langle Arg, R, Av \rangle$ *where* $Arg \subseteq U$ *is a set of adopted arguments belonging to the universe* U, R[5] *is a binary relation on* Arg, *and* $Av \colon Arg \to \wp(\Upsilon)$ *is the availability function for timed arguments.*

We illustrate in Figure 3 an example of the behaviour of timed arguments taken from [15], with availability function defined as $Av(a) = \{[10, 40], [60, 75]\}$, $Av(b) = \{[30, 50]\}$, $Av(c) = \{[20, 40], [45, 55], [60, 70]\}$, $Av(d) = \{[47, 65]\}$ and $Av(e) = \{(-\infty, 44]\}$. Note that attack relations in a TAF never change, but are only considered when availability of both the attacker and the attacked arguments overlaps (alternatively, we can consider attacks to blink together with the arguments they connect). We can imagine the modifications at a given instant t taking place simultaneously, as if every argument of the TAF was handled (made available/not available) by a different agent.

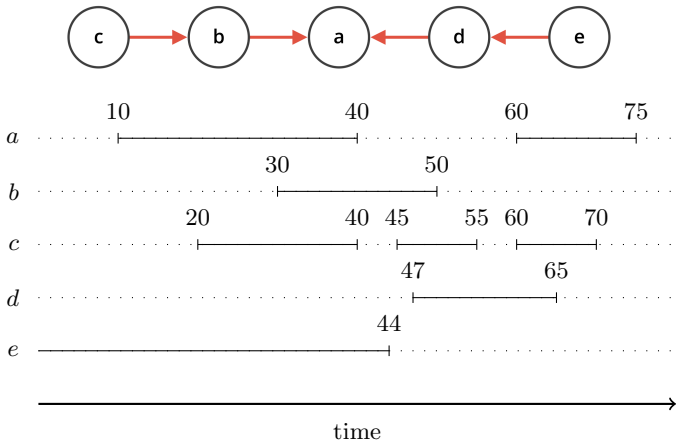

Fig. 3. Example of timed abstract argumentation framework.

Given a semantics σ, the set of acceptable arguments in a TAF can be computed with respect to a certain moment t by only considering arguments (and attacks) available at t. For example, in the TAF of Figure 3, the singleton $\{b\}$ is an admissible extension for $t \in (41, 44)$, that is when b itself is available and its attacker c is not.

[5] Note that in certain time intervals, attacks could be defined between arguments not currently available in the TAF.

3 *tcla* Syntax and Semantics

The syntax of our Timed Concurrent Language for Argumentation, *tcla*, is presented in Table 2, where, as usual, P, C, A and E denote a generic process, a sequence of procedure declarations (or clauses), a generic agent and a generic guarded agent, respectively. Moreover $t \in \mathbb{N} \cup \{+\infty\}$. In Tables 3 and 4, then, we give the definitions for the transition rules.

$$
\begin{aligned}
P &::= C.A \\
C &::= p(a, l, \sigma, i) :: A \mid C.C \\
A &::= \mathit{success} \mid \mathit{failure} \mid \mathit{add}(\mathit{Arg}, R) \rightarrow A \mid \mathit{rmv}(\mathit{Arg}, R) \rightarrow A \mid E \mid A \| A \mid \exists_x A \mid \\
&\quad\; p(a, l, \sigma, i) \\
E &::= \mathit{test}_{c,t}(a, l, \sigma) \rightarrow A \mid \mathit{test}_{s,t}(a, l, \sigma) \rightarrow A \mid \mathit{check}_t(\mathit{Arg}, R) \rightarrow A \mid E + E \mid \\
&\quad\; E +_P E \mid E \|_G E
\end{aligned}
$$

Table 2. *tcla* syntax.

Communication between *tcla* agents is implemented via shared memory, similarly to *cla* [5] and CC [26], and opposed to other languages (e.g., CSP [19] and CCS [23]) based on message passing. 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 $t = 0$ (note that a Boolean syntactic category could be introduced in replacement of $\mathcal{E}_0$ to better handle guards and allow for finer distinctions). In a *tcla process* $P = C.A$, A is the initial agent, to be executed in the context of the set of declarations C.

The operational model of *tcla* processes can be formally described by a transition system $T = (\mathit{Conf}, \longrightarrow)$, where we assume that each transition step exactly takes one time-unit. Configurations (in) Conf are pairs consisting of a process and an AF $F = \langle \mathit{Arg}, R \rangle$ representing the common knowledge base. The transition relation $\longrightarrow \subseteq \mathit{Conf} \times \mathit{Conf}$ is the least relation satisfying the rules in Tables 3 and 4, and it characterises the (temporal) evolution of the system. So, $\langle A, F \rangle \longrightarrow \langle A', F' \rangle$ means that, if at time t we have the process A and the framework F, then at time $t + 1$ we have the process A' and the framework F'.

In Tables 3 and 4 we have omitted the symmetric rules for the choice operator $+$ and for the two parallel composition operators $\|$ and $\|_G$. Indeed, $+$ is commutative, so $E_1 + E_2$ produces the same result as (that is, is congruent to) $E_2 + E_1$. The same is also true for $\|$ and $\|_G$. Note that $+$, $\|$ and $\|_G$ are also associative. Moreover $\mathit{success}$ and $\mathit{failure}$ are the identity and the absorbing elements under the parallel composition $\|$, respectively (namely for each agent A, we have that $A \| \mathit{success}$ and $A \| \mathit{failure}$ are the agents A and $\mathit{failure}$, respectively). In the following, we will usually write a *tcla* process $P = C.A$ as the corresponding agent A, omitting C when not required by the context.

$$\langle add(Arg', R') \to A, \langle Arg, R\rangle\rangle \longrightarrow \langle A, \langle Arg \cup Arg', R \cup R''\rangle\rangle \quad \text{where } R'' = \{(a,b) \in R' \mid a, b \in Arg \cup Arg'\} \qquad \text{Ad}$$

$$\langle rmv(Arg', R') \to A, \langle Arg, R\rangle\rangle \longrightarrow \langle A, \langle Arg \setminus Arg', R \setminus \{R' \cup R''\}\rangle\rangle \quad \text{where } R'' = \{(a,b) \in R \mid a \in Arg' \vee b \in Arg'\} \qquad \text{Re}$$

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

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

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

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

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

$$\langle test_{c,0}(a, l, \sigma) \to A, F\rangle \longrightarrow \langle failure, F\rangle \qquad \text{CT (3)}$$

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

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

$$\langle test_{s,0}(a, l, \sigma) \longrightarrow A, F\rangle \longrightarrow \langle failure, F\rangle \qquad \text{ST (3)}$$

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

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

Table 3. *tcla* operational semantics (part 1/2).

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

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

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

$$\frac{\langle A_1, F\rangle \longrightarrow \langle A_1', F'\rangle,\quad \langle A_2, F\rangle \longrightarrow \langle A_2', F''\rangle}{\langle A_1 \| A_2, F\rangle \longrightarrow \langle A_1' \| A_2', *(F, F', F'')\rangle} \quad \text{Pa}$$

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

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

$$\frac{\langle A[y/x], F\rangle \longrightarrow \langle A', F'\rangle}{\langle \exists_x A, F\rangle \longrightarrow \langle A', F'\rangle} \text{ with } y \in U \setminus Arg \quad \text{HA}$$

$$\langle p(b, m, \gamma, j), F\rangle \longrightarrow \langle A[b/a, m/l, \gamma/\sigma, j/i], F\rangle \text{ when } p(a, l, \sigma, i) :: A \quad \text{PC}$$

Table 4. *tcla* operational semantics (part 2/2).

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 the AF F. When performing an addition, (possibly) new arguments are taken from $U \setminus Arg$. We want to make clear that the tuple (Arg', R') is not an AF, indeed it is possible to have $Arg' = \emptyset$ and $R' \neq \emptyset$. However, the structure of the shared store after an *add* operation is guaranteed to be an AF compliant with Definition 1, since only attacks between arguments in $Arg \cup Args'$ are added to R.

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. Trying to remove an argument (or an attack) which does not exist in F will have no consequences.

The $test_{c,t}(a, l, \sigma) \rightarrow A$, $test_{s,t}(a, l, \sigma) \rightarrow A$ and $check_t(Arg, R) \rightarrow A$ constructs are explicit timing primitives that allow for the specification of timeouts. In fact, in timed applications often one cannot wait indefinitely for an event. Thus, a timed language should allow us to specify that, in case a given time

bound is exceeded (i.e. a *timeout* occurs), the wait is interrupted and an alternative action is taken.

The operator $check_t(Arg', R')$ (rules Ch (1)-(3) in Table 3) realises a timed construct and is used to verify whether, in a given time interval, the specified arguments and attack relations are contained in the set of arguments and attacks of 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 an action in the agent A (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 SC (1)-(3) in Table 3 are similar to rules Ch (1)-(3) described before. Observe that we have two distinct test operations, both requiring the specification of an argument $a \in Arg$, a label $l \in \{\mathsf{in}, \mathsf{out}, \mathsf{undec}\}$[6] and a semantics $\sigma \in \{adm, com, stb, sst, prf, gde\}$. The credulous $test_c(a, l, \sigma)$ succeeds if there exists at least one extension of $S_\sigma(F)$ whose corresponding labelling L is such that $L(a) = l$. Similarly, the sceptical $test_s(a, l, \sigma)$ succeeds if a is labelled l in all possible labellings $L \in S_\sigma(F)$.

The operator $+_P$ (If (1)-(2) in Table 3) 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 will be always chosen over E_2, even if also the guards of E_2 succeed. So, 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 $E_1' +_P E_2 \in \mathcal{E}$.

The guarded parallelism GP (1)-(3) in Table 4 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 remaining operators are classical concurrency compositions: the rule parallelism Pa in Table 4 models the parallel composition operator in terms of *maximal parallelism*. By transition rules, an agent in a parallel composition obtained through $\|$ succeeds only if all the agents succeed. In our implementation of rule Pa, we use $*(F, F', F'') := (F' \cap F'') \cup ((F' \cup F'') \setminus F)$ to handle parallel additions and removals of arguments[7]. In particular, if an argument a is added and removed at the same moment (e.g., trough the program $add(\{a\}, \{\}) \rightarrow success \parallel rmv(\{a\}, \{\}) \rightarrow success$), we have two possible outcomes: if a was not

[6] Other labelling semantics could be considered where undec arguments are further divided into "don't know" and "don't care" [7].

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

present in the knowledge base, then the *add* operation gains priority over the *rmv* one, 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.

Any agent composed through + (rules ND (1)-(2)) is chosen if its guards succeed; the existential quantifier $\exists_x A$ of rule HA behaves like agent A where variables (arguments) in x are local to A. The procedure call (rule PC) has four parameters that allow the implementation of operators which takes into account an argument, a semantic label (in/out/undec), a semantics σ and a time interval i, respectively.

In the following we provide the definition for the observables of the language, which considers the traces of successful or failed terminating computations that the agent A can perform for each *tcla* process $P = C.A$. Given a transition relation $\longrightarrow$, we denote by $\longrightarrow^*$ its reflexive and transitive closure.

Definition 8 (Observables for *tcla*). *Let $P = C.A$ be a tcla process. We define*

$$\mathcal{O}_{io}(P) = \{F_1 \cdots F_n \cdot ss \mid \langle A, F_1\rangle \longrightarrow^* \langle success, F_n\rangle\} \cup \{F_1 \cdots F_n \cdot ff \mid \langle A, F_1\rangle \longrightarrow^* \langle failure, F_n\rangle\}.$$

4 Modelling TAFs

In the context of argumentation, the possibility to include time in the reasoning processes conducted by intelligent agents allows for modelling dynamic behaviours, such as, for instance, the addition and the removal of information at precise moments from a knowledge base.

Agents using *tcla* constructs and availability functions in TAFs regulate the existence of arguments in a similar way; in particular, we can imagine each agent having control over an argument of the TAF. This result is showed by providing explicitly a translation from a TAF into a *tcla* process.

In the following, given $I \subseteq \Upsilon$, I is ordered if there is no $[t_1, t_2] \in I$ such that $t_2 < t_1$ and for each $i_1, i_2 \in I$, we have that i_1 and i_2 are disjoint and non-consecutive. Without loss of generality, we consider only ordered sets of intervals. Moreover, given $I \neq \emptyset$ we denote by

$$first(I) = \begin{cases} (-\infty, +\infty) & \text{if } (-\infty, +\infty) \in I \\ (-\infty, t] & \text{if } (-\infty, t] \in I \\ [t, +\infty) & \text{if } \{[t, +\infty)\} = I \\ [t_1, t_2] & \text{otherwise} \end{cases}$$

where $[t_1, t_2] \in I$ is such that for each $T' \in I$, where $T' = [t'_1, t'_2]$ or $T' = [t'_1 + \infty)$, $t_1 < t'_1$. Moreover, we denote by $continue(I) = I \setminus first(I)$.

Finally, given $a \in Arg$ and a binary relation R defined over Arg, we denote by $R_{|a} = \{(a, b) \mid (a, b) \in R\} \cup \{(b, a) \mid (b, a) \in R\}$. To ease the translation, we use $sleep(t) \rightarrow A$, with $t \geq 0$, as a shortcut for

$$\begin{cases} A & \text{if } t \leq 0 \\ check_1(\{\}, \{\}) \rightarrow (sleep(t-1) \rightarrow A) & \text{otherwise} \end{cases}$$

The agent $sleep(t) \rightarrow A$, with $t \geq 0$, executes the check operation for exactly t times and then the agent A is executed. Practically, $sleep(t)$ lets the agent A wait for t instants of time.

In the following we assume that for each $[t_1, t_2] \in S$, $t_1 > 0$.

Definition 9 (Translation). *The translation of a TAF* $\langle \{a_1, \ldots, a_n\}, R, Av \rangle$ *is*

$$\mathcal{T}(\langle \{a_1, \ldots, a_n\}, R, Av \rangle) = \mathcal{T}_{add}(0, a_1, R_{|a_1}, Av(a_1)) \| \cdots \| \mathcal{T}_{add}(0, a_n, R_{|a_n}, Av(a_n))$$

where

$$\mathcal{T}_{add}(t, a, R, S) = \begin{cases} success & \text{if } S = \emptyset \\ add(\{a\}, R) \rightarrow success & \text{if } S = \{(-\infty, +\infty)\} \\ sleep(t_{in} - t - 1) \rightarrow & \\ \quad (add(\{a\}, R) \rightarrow \mathcal{T}_{rmv}(t, a, R, S)) & otherwise \end{cases}$$

and

$$\mathcal{T}_{rmv}(t, a, R, S) = \begin{cases} success & \text{if } S = \{[t_1, +\infty)\} \\ sleep(t_{fin} - t_{in}) \rightarrow & \\ \quad (rmv(\{a\}, \{\}) \rightarrow & \\ \qquad \mathcal{T}_{add}(t_{fin} + 1, a, R, continue(S))) & otherwise \end{cases}$$

where

$$t_{in} = \begin{cases} 1 & \text{if } first(S) = (-\infty, t_2] \\ t_1 & \text{if } first(S) = [t_1, t_2] \text{ or } first(S) = [t_1, +\infty) \end{cases}$$

and $t_{fin} = t_2$ *if* $first(S) = (-\infty, t_2]$ *or* $first(S) = [t_1, t_2]$.

Theorem 1. *The translation* $\mathcal{T}$ *of a TAF produces a tcla program such that, at any instant of time* $t > 0$*, the arguments in the store are all and only the arguments available at* t *in the original TAF.*

Corollary 1. *Given a tcla program produced by a translation* $\mathcal{T}$ *of a TAF, the set of arguments in the store accepted with respect to a semantics* σ *at a given instant of time* $t > 0$ *coincides with the set of extensions identified by* σ *at moment* t *on the original TAF .*

Example 2. Consider the TAF of Figure 3. By following previous translation $\mathcal{T}'$, the availability of its timed arguments for each time instant $t > 0$ can be simulated by a *tcla* process that modifies the store by adding and removing part of the underlying framework. We use five agents in parallel (see Table 5), each of which is in charge of handling one argument of the TAF. Synchronisation between these agents is provided through the use of sleep constructs that handle time lapsing. In the initial configuration, the shared memory is empty ($F = \langle \{\}, \{\} \rangle$).

$sleep(9) \rightarrow (add(\{a\}, \{(b,a),(d,a)\}) \rightarrow (sleep(30) \rightarrow (rmv(\{a\}, \{\}) \rightarrow$
$(sleep(18) \rightarrow (add(\{a\}, \{(b,a),(d,a)\}) \rightarrow (sleep(15) \rightarrow (rmv(\{a\}, \{\}) \rightarrow success)))))))) \parallel$
$sleep(29) \rightarrow (add(\{b\}, \{(b,a),(c,b)\}) \rightarrow (sleep(20) \rightarrow (rmv(\{b\}, \{\}) \rightarrow success))) \parallel$
$sleep(19) \rightarrow (add(\{c\}, \{(c,b)\}) \rightarrow (sleep(20) \rightarrow (rmv(\{c\}, \{\}) \rightarrow$
$(sleep(3) \rightarrow (add(\{c\}, \{(c,b)\}) \rightarrow (sleep(10) \rightarrow (rmv(\{c\}, \{\}) \rightarrow$
$(sleep(3) \rightarrow (add(\{c\}, \{(c,b)\}) \rightarrow (sleep(10) \rightarrow (rmv(\{c\}, \{\}) \rightarrow success)))))))))))) \parallel$
$sleep(46) \rightarrow (add(\{d\}, \{(d,a),(e,d)\}) \rightarrow (sleep(18) \rightarrow (rmv(\{d\}, \{\}) \rightarrow success))) \parallel$
$add(\{e\}, \{(e,d)\}) \rightarrow (sleep(43) \rightarrow (rmv(\{e\}, \{\}) \rightarrow success))$

Table 5. *tcla* program realising the TAF of Figure 3 where each agent handles one argument.

5 Related Work

This kind of frameworks has been studied from the point of view of dynamics in works like [13,24], where the authors investigate the consequences brought by modifications on AFs. Given the very nature of argumentation, it is reasonable to assume that the interaction between interacting entities (being them people or synthetic agents) are regulated by the passing of time [21,22].

Timed Abstract Dialectical Frameworks (tADFs) are introduced in [2] to cope with acceptance conditions changing over time. Differently form TAFs, which extend classical AFs with the notion of time intervals, tADFs use generic statements that subsumes arguments and relations (not only including attacks, but also, for instance, supports). Therefore, tADFs are not suitable for being modelled through *tcla* processes, which, instead, relies on AFs.

6 Conclusion and Future Work

In this paper, we introduced *tcla*, an extension of *cla* in which time is taken into account in the form of timeouts on expressions. We used maximal parallelism to realise simultaneous execution of actions carried forward by different agents. We presented the operational semantics, and showed examples of how to instantiate TAF by using our language. As future work, we would like to study time-dependent notions of acceptability. We could introduce operators for checking if a given argument is acceptable in all (or in some) instants of time. Quantitative temporal operators could, then, be defined to check acceptability only in given intervals. We are also planning to provide a working implementation of *tcla* able to aid research and serve for application purposes.

References

1. Baroni, P., Caminada, M., Giacomin, M.: An introduction to argumentation semantics. Knowl. Eng. Rev. **26**(4), 365–410 (2011), https://doi.org/10.1017/S0269888911000166
2. Baumann, R., Heinrich, M.: Timed abstract dialectical frameworks: A simple translation-based approach. In: Prakken, H., Bistarelli, S., Santini, F., Taticchi, C. (eds.) Computational Models of Argument - Proceedings of COMMA 2020, Perugia, Italy, September 4-11, 2020. Frontiers in Artificial Intelligence and Applications, vol. 326, pp. 103–110. IOS Press (2020), https://doi.org/10.3233/FAIA200496
3. Berry, G., Gonthier, G.: The esterel synchronous programming language: Design, semantics, implementation. Sci. Comput. Program. **19**(2), 87–152 (1992), https://doi.org/10.1016/0167-6423(92)90005-V
4. Bistarelli, S., Gabbrielli, M., Meo, M.C., Santini, F.: Timed soft concurrent constraint programs: An interleaved and a parallel approach. Theory Pract. Log. Program. **15**(6), 743–782 (2015), https://doi.org/10.1017/S1471068414000106
5. Bistarelli, S., Taticchi, C.: A concurrent language for argumentation. In: Fazzinga, B., Furfaro, F., Parisi, F. (eds.) Proceedings of the Workshop on Advances In Argumentation In Artificial Intelligence 2020, Online, November 25-26, 2020. CEUR Workshop Proceedings, vol. 2777, pp. 75–89. CEUR-WS.org (2020), http://ceur-ws.org/Vol-2777/paper67.pdf
6. Bistarelli, S., Taticchi, C.: Introducing a tool for concurrent argumentation. In: Faber, W., Friedrich, G., Gebser, M., Morak, M. (eds.) Logics in Artificial Intelligence - 17th European Conference, JELIA 2021, Virtual Event, May 17-20, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12678, pp. 18–24. Springer (2021), https://doi.org/10.1007/978-3-030-75775-5_2
7. Bistarelli, S., Taticchi, C.: A unifying four-state labelling semantics for bridging abstract argumentation frameworks and belief revision. In: Proceedings of the 22nd Italian Conference on Theoretical Computer Science, Bologna, Italy, September 13-15, 2021. CEUR Workshop Proceedings, CEUR-WS.org (2021), to appear
8. de Boer, F.S., Gabbrielli, M., Meo, M.C.: A timed concurrent constraint language. Inf. Comput. **161**(1), 45–83 (2000), https://doi.org/10.1006/inco.1999.2879
9. de Boer, F.S., Gabbrielli, M., Meo, M.C.: Proving correctness of timed concurrent constraint programs. ACM Trans. Comput. Log. **5**(4), 706–731 (2004), https://doi.org/10.1145/1024922.1024926
10. Budán, M.C., Lucero, M.J.G., Chesñevar, C.I., Simari, G.R.: Modeling time and valuation in structured argumentation frameworks. Inf. Sci. **290**, 22–44 (2015), https://doi.org/10.1016/j.ins.2014.07.056
11. Busi, N., Gorrieri, R., Zavattaro, G.: Process calculi for coordination: From linda to javaspaces. In: Rus, T. (ed.) Algebraic Methodology and Software Technology. 8th International Conference, AMAST 2000, Iowa City, Iowa, USA, May 20-27, 2000, Proceedings. Lecture Notes in Computer Science, vol. 1816, pp. 198–212. Springer (2000), https://doi.org/10.1007/3-540-45499-3_16
12. Caminada, M.: On the issue of reinstatement in argumentation. In: Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A. (eds.) Logics in Artificial Intelligence, 10th European Conference, JELIA 2006, Liverpool, UK, September 13-15, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4160, pp. 111–123. Springer (2006), https://doi.org/10.1007/11853886_11

13. Cayrol, C., de Saint-Cyr, F.D., Lagasquie-Schiex, M.: Change in abstract argumentation frameworks: Adding an argument. J. Artif. Intell. Res. **38**, 49–84 (2010), https://doi.org/10.1613/jair.2965
14. Cobo, M.L., Martínez, D.C., Simari, G.R.: On admissibility in timed abstract argumentation frameworks. In: Coelho, H., Studer, R., Wooldridge, M.J. (eds.) ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings. Frontiers in Artificial Intelligence and Applications, vol. 215, pp. 1007–1008. IOS Press (2010), https://doi.org/10.3233/978-1-60750-606-5-1007
15. Cobo, M.L., Martínez, D.C., Simari, G.R.: On semantics in dynamic argumentation frameworks. In: XVIII Congreso Argentino de Ciencias de la Computación (2013), http://sedici.unlp.edu.ar/handle/10915/31428
16. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artif. Intell. **77**(2), 321–358 (1995), https://doi.org/10.1016/0004-3702(94)00041-X
17. Halbwachs, N., Lagnier, F., Ratel, C.: Programming and verifying real-time systems by means of the synchronous data-flow language LUSTRE. IEEE Trans. Software Eng. **18**(9), 785–793 (1992), https://doi.org/10.1109/32.159839
18. Harel, D.: Statecharts: A visual formalism for complex systems. Sci. Comput. Program. **8**(3), 231–274 (1987), https://doi.org/10.1016/0167-6423(87)90035-9
19. Hoare, C.A.R.: Communicating sequential processes. Commun. ACM **21**(8), 666–677 (1978), https://doi.org/10.1145/359576.359585
20. Le Guernic, P., Gautier, T., Le Borgne, M., Le Maire, C.: Programming real-time applications with signal. Proc. IEEE **79**(9), 1321–1336 (1991), https://doi.org/10.1109/5.97301
21. Mann, N., Hunter, A.: Argumentation using temporal knowledge. In: Besnard, P., Doutre, S., Hunter, A. (eds.) Computational Models of Argument: Proceedings of COMMA 2008, Toulouse, France, May 28-30, 2008. Frontiers in Artificial Intelligence and Applications, vol. 172, pp. 204–215. IOS Press (2008), http://www.booksonline.iospress.nl/Content/View.aspx?piid=9281
22. Marcos, M.J., Falappa, M.A., Simari, G.R.: Dynamic argumentation in abstract dialogue frameworks. In: McBurney, P., Rahwan, I., Parsons, S. (eds.) Argumentation in Multi-Agent Systems - 7th International Workshop, ArgMAS 2010, Toronto, ON, Canada, May 10, 2010 Revised, Selected and Invited Papers. Lecture Notes in Computer Science, vol. 6614, pp. 228–247. Springer (2010), https://doi.org/10.1007/978-3-642-21940-5_14
23. Milner, R.: A Calculus of Communicating Systems, Lecture Notes in Computer Science, vol. 92. Springer (1980), https://doi.org/10.1007/3-540-10235-3
24. Rotstein, N.D., Moguillansky, M.O., Falappa, M.A., García, A.J., Simari, G.R.: Argument theory change: Revision upon warrant. In: Besnard, P., Doutre, S., Hunter, A. (eds.) Computational Models of Argument: Proceedings of COMMA 2008, Toulouse, France, May 28-30, 2008. Frontiers in Artificial Intelligence and Applications, vol. 172, pp. 336–347. IOS Press (2008), http://www.booksonline.iospress.nl/Content/View.aspx?piid=9292
25. Saraswat, V.A., Jagadeesan, R., Gupta, V.: Timed default concurrent constraint programming. J. Symb. Comput. **22**(5/6), 475–520 (1996), https://doi.org/10.1006/jsco.1996.0064
26. Saraswat, V.A., Rinard, M.C.: Concurrent constraint programming. In: Allen, F.E. (ed.) Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages, San Francisco, California, USA, 1990. pp. 232–245. ACM Press (1990), https://doi.org/10.1145/96709.96733