A Non-interleaving Timed Process Algebra and a Process Logic for Verifying Composition of Agents

Yoshinao ISOBE and Kazuhito OHMAKI

Information Technology Research Institute
National Institute of Advanced Industrial Science and Technology (AIST)
Tsukuba Central 2, 1-1-1, Umezono, Tsukuba, Ibaraki, 305-8568 Japan
{ y-isobe | k.ohmaki }@aist.go.jp

ABSTRACT

We present formal frameworks *tCCA*, *tLCA*, and *tICCA* for verifying composition of agents. Behaviors of composite agents are described in *tCCA* and specifications for them are described in *tLCA*. Since consistency between specifications in *tLCA* is undecidable as proven in this paper, we propose to use intermediate specifications described in *tICCA* instead of checking the consistency, and then give useful propositions for verifying composition of agents in *tICCA*.

Keywords: process algebra, process logic, time, satisfiability, non-interleaving

1 INTRODUCTION

We are members of a project called ESP (Evolutionary System Project)[1]. The purpose of ESP is to develop an agent-system where agents can evolve by spontaneously moving over networks, combining with other agents, and communicating with them. In such systems, unexpected behavior may be caused by composition of agents. To avoid unexpected behavior, the members of ESP are discussing an agent-system where each agent has specifications, and if specifications between two agents are not consistent, then they cannot combine. These agents are modeled as shown in Figure 1. In ESP, we are studying how to formally verify consistency between specifications. This paper defines a process algebra *tCCA*, a process logic *tLCA*, and a process algebra *tICCA*.

tCCA (a timed Calculus of Composite Agents) defined in Section 2 is a non-interleaving timed process algebra. Many non-interleaving process algebras have been proposed (e.g. [1, 3, 5]) and timed process algebras have been proposed (e.g. [4, 9, 13]). The features of *tCCA* are summarized as follows: (1) it is possible to observe which agents have performed actions and (2) the passage of time is needed for executing any action. By the feature (1), concurrent behavior is explicitly distinguished from interleaving behavior. The feature (2) removes unnatural situations such that actions can infinitely be performed in a finite time. Note that each feature is not new. For example, (1) is expressed in Distributed CCS[5] and (2) is discussed in [2]. The purpose of this paper is not to propose a new process algebra, but to show relations between a process algebra and a process logic with considering both of (1) and (2).

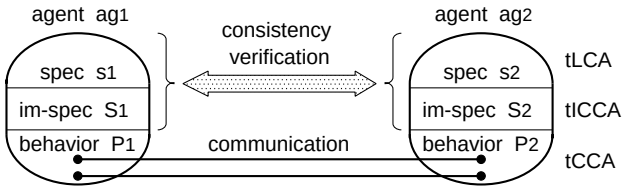

Figure 1: The image of agents with specifications

tLCA (a timed Logic for Composite Agents) defined in Section 3 is a process logic for describing specifications of agent-behaviors described in *tCCA*. As shown in Figure 1, if each agent ag_i has a specification s_i (i.e. each behavior P_i satisfies s_i), then it is important to verify the consistency between s_1 and s_2 before composition of ag_1 and ag_2. However, we prove that the consistency is undecidable in Section 3.

tICCA (a timed Intermediate CCA) defined in Section 4 is a basic process algebra. Since the consistency in *tLCA* is undecidable, we use *intermediate specifications* (also called *im-specifications*) between behaviors (*tCCA*) and specifications (*tLCA*). Im-specifications express *abstract* behaviors of agents by hiding uninteresting actions because verification often considers some actions (not all). Therefore, the number of states in im-specifications decreases. In Section 4, we show how to verify composition of agents in *tICCA* instead of in *tCCA*.

2 PROCESS ALGEBRA *tCCA*

In this section, *tCCA* is defined for describing behaviors of composite agents. At first, it is assumed that a set $\mathcal{N}_{ac}$, ranged over by $a, b, \cdots$, of *action-names* are given. Then, the set Act, ranged over by $\alpha, \beta, \cdots$, of *actions* is defined as: $Act = \mathcal{N}_{ac} \cup \{\mathtt{1}\}$, where the *time action* '$\mathtt{1}$' represents the passage of *one time-unit*, where a time-unit is an abstract unit of time, and it may be a second,

[1]The project ESP has been done with the support of JSPS(Japan Society for the Promotion of Science) Grants-in-Aid for Scientific Research

a minute, or a clock of CPU. Also, it is assumed that a set $\mathcal{N}_{ag}$, ranged over by $\psi, \varphi, \cdots$, of *agent-names* are given. Then, the set Act_S of *single-actions* is defined as: $Act_S = \{a@\psi : a \in \mathcal{N}_{ac}, \psi \in \mathcal{N}_{ag}\}$, and subsets of Act_S are represented by $\mu, \nu, \cdots$. Then, the set Act_M of *multi-actions* is defined as: $Act_M = \{\mu : \mu \subseteq Act_S, \forall a@\psi \in \mu.\ \forall b@\varphi \in \mu - \{a@\psi\}.\ \psi \neq \varphi\}$. A multi-action $\{a_i@\psi_i : i \in I\} \in Act_M$ represents that agents-ψ_i $(i \in I)$ simultaneously perform actions-a_i, respectively. Thus, the number of actions which an agent can simultaneously perform is one at most. Subsets of the set Act_M of multi-actions are represented by $M, N, \cdots$.

It is also assumed that a set $\mathcal{K}_{bh}$, ranged over by $K, \cdots$, of *behavior-constants* is given. Then, the set Bh, ranged over by $P, Q, \cdots$, of *behaviors* is the smallest set which contains the following expressions:

$$\begin{array}{rr} K : \text{Recursion } (K \in \mathcal{K}_{bh}), & \mathbf{I} : \text{Idling}, \\ a.P : \text{Prefix } (a \in \mathcal{N}_{ac}), & P \triangleright Q : \text{Timeout}, \\ \sum_{i \in I} P_i : \text{Summation}, & \end{array}$$

where P, P_i, Q are already in Bh. Also, the set CBh, ranged over by $C, D, \cdots$, of *concurrent behaviors* is the smallest set which contains the following expressions:

$$\begin{array}{rcl} P@\psi & : & \text{Naming } (P \in Bh, \psi \in \mathcal{N}_{ag}), \\ C|D & : & \text{Composition } (Agn(C) \cap Agn(D) = \emptyset), \\ C \backslash M & : & \text{Restriction } (M \subseteq Act_M), \end{array}$$

where C, D are already in CBh. The function $Agn : CBh \to 2^{\mathcal{N}_{ag}}$ is defined as follows: $Agn(P@\psi) = \{\psi\}$, $Agn(C|D) = Agn(C) \cup Agn(D)$, and $Agn(C \setminus M) = Agn(C)$. These operators have the binding power such that: Prefix > Timeout > Summation > Naming > Restriction > Composition. Notation $C \equiv D$ represents that C and D are syntactically identical.

Each operator is briefly explained as follows. The behavior $a.P$ represents that an agent can perform action-a and thereafter behaves like P. Here, it is important that the passage of one time-unit is always needed for executing any action. An action $a(n)$ which needs n time-units $(n \geq 1)$ for its execution is inductively defined as: $a(1).P \equiv a.P$ and $a(n+1).P \equiv a.a(n).P$,

$\sum_{i \in I} P_i$ behaves like P_j for some $j \in I$, and the choice is made by the first action of P_j except for the time action $\mathtt{1}$. The passage of time must be made by all behaviors P_i $(i \in I)$. This operator corresponds to 'strong choice' in TCCS[9]. We also use a short notation for binary choice as follows: $P_1 + P_2 \equiv \sum_{i \in \{1,2\}} P_i$.

Meaning of each behavior-constant is given by a defining equation. It is assumed that for every behavior-constant $K \in \mathcal{K}_{bh}$, there is a defining equation $K \stackrel{\text{def}}{=} P$.

$P \triangleright Q$ *now* behaves like P and behaves like Q *after one time-unit*. Thus, $P \triangleright_{(n)} Q$ which behaves like P until n time-units pass and behaves like Q after that is defined as: $P \triangleright_{(0)} Q \equiv Q$ and $P \triangleright_{(n+1)} Q \equiv P \triangleright (P \triangleright_{(n)} Q)$.

Name	Hypothesis	$\Rightarrow$	Conclusion
Id		$\Rightarrow$	$\mathbf{I} \xrightarrow{\mathtt{1}} \mathbf{I}$
$\mathbf{Act}_1$		$\Rightarrow$	$a.P \xrightarrow{a} P$
$\mathbf{Act}_2$		$\Rightarrow$	$a.P \xrightarrow{\mathtt{1}} a.P$
$\mathbf{Sum}_1$	$\exists i \in I.\ P_i \xrightarrow{a} P'$	$\Rightarrow$	$\sum_{i \in I} P_i \xrightarrow{a} P'$
$\mathbf{Sum}_2$	$\forall i \in I.\ P_i \xrightarrow{\mathtt{1}} P_i'$	$\Rightarrow$	$\sum_{i \in I} P_i \xrightarrow{\mathtt{1}} \sum_{i \in I} P_i'$
$\mathbf{TO}_1$	$P \xrightarrow{a} P'$	$\Rightarrow$	$P \triangleright Q \xrightarrow{a} P'$
$\mathbf{TO}_2$		$\Rightarrow$	$P \triangleright Q \xrightarrow{\mathtt{1}} Q$
Rec	$K \stackrel{\text{def}}{=} P,\ P \xrightarrow{\alpha} P'$	$\Rightarrow$	$K \xrightarrow{\alpha} P'$
$\mathbf{Name}_1$	$P \xrightarrow{a} P'$	$\Rightarrow$	$P@\psi \xrightarrow{\{a@\psi\}} P'@\psi$
$\mathbf{Name}_2$	$P \xrightarrow{\mathtt{1}} P'$	$\Rightarrow$	$P@\psi \xrightarrow{\emptyset} P'@\psi$
Com	$C \xrightarrow{\mu} C', D \xrightarrow{\nu} D'$	$\Rightarrow$	$C\|D \xrightarrow{\mu \cup \nu} C'\|D'$
Res	$C \xrightarrow{\mu} C', \mu \in M \cup \{\emptyset\}$	$\Rightarrow$	$C \backslash M \xrightarrow{\mu} C' \backslash M$

Figure 2: Inference rules for $\xrightarrow{\alpha}$ and $\xrightarrow{\mu}$

Furthermore, $(n).P$ which behaves like P after n time-units are defined as: $(n).P \equiv \mathbf{I} \triangleright_{(n)} P$.

The Naming operator @ defines that an agent named ψ behaves like P by $P@\psi$. The function Agn is used for uniquely assigning an agent-name. The concurrent behavior $(C|D)$ concurrently executes C and D. $C \backslash M$ can execute only multi-actions contained in $M \cup \{\emptyset\}$.

Semantics of behaviors and concurrent behaviors is given by the labelled transition systems $\langle Bh, Act, \{\xrightarrow{\alpha} : \alpha \in Act\}\rangle$ and $\langle CBh, Act_M, \{\xrightarrow{\mu} : \mu \in Act_M\}\rangle$, respectively, where the sets $\xrightarrow{\alpha}$ and $\xrightarrow{\mu}$ are the smallest relations satisfying the inference rules in Figure 2.

Id, $\mathbf{Act}_2$, $\mathbf{Sum}_2$, and **Rec** show that behavior does not alter by the passage of time without the Timeout $\triangleright$. $\mathbf{TO}_{1,2}$ show that the timeout process of P is ignored in $P \triangleright Q$, and the timeout process of $P \triangleright Q$ is Q. $\mathbf{Name}_2$ shows that the passage of time is hidden. Thus, the passage of one time-unit for behaviors is $\xrightarrow{\mathtt{1}}$, while one for concurrent behaviors is $\xrightarrow{\emptyset}$.

We often use a behavior-constant $P \rangle Q$ defined as:

$$P \rangle Q \stackrel{\text{def}}{=} \sum\{a.P' \rangle Q' : P \xrightarrow{a} P', Q \xrightarrow{\mathtt{1}} Q'\} \cup \{a.Q' : Q \xrightarrow{a} Q'\} \\ \triangleright \sum\{P' \rangle Q' : P \xrightarrow{\mathtt{1}} P', Q \xrightarrow{\mathtt{1}} Q'\}$$

for each $P, Q \in Bh$. Thus, $P \rangle Q$ usually behaves like P but it may be *interrupted* by an action (not $\mathtt{1}$) of Q.

In the rest of this section, we give an example in *tCCA*. It is interaction between a researcher and a pizza-worker. In this case, a time-unit is a minute. At first, the behavior *RES* of the researcher is defined as follows:

$$RES \stackrel{\text{def}}{=} order.(study(30).\mathbf{I} \\ \rangle\ (receive.eat(20).\mathbf{I} \triangleright_{(60)} cancel.angry(10).\mathbf{I}))$$

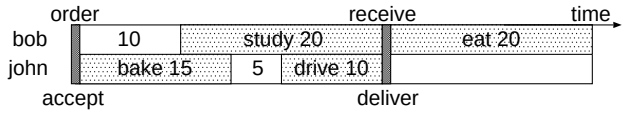

Figure 3: An example of transition-sequences in *RP*

The researcher *orders* a pizza, then *studies* for 30 minutes, but if he *receives* the pizza, then he stops studying and *eats* it for 20 minutes. However, if 60 minutes have passed, he *cancels* the order and is *angry* for 10 minutes. For example, the following transition is possible: $RES \xrightarrow{order} (\xrightarrow{study})^{15} \xrightarrow{receive} (\xrightarrow{eat})^{20}\,\mathbf{I}$, where $(\xrightarrow{\alpha})^n$ represents $\xrightarrow{\alpha}\cdots\xrightarrow{\alpha}$ with n occurrences of $\xrightarrow{\alpha}$. And, since the researcher can rest during study, he may cancel the order before finishing the study as follows: $RES \xrightarrow{order} (\xrightarrow{study})^{10} (\xrightarrow{1})^{45} (\xrightarrow{study})^{5} \xrightarrow{cancel} (\xrightarrow{angry})^{10}\,\mathbf{I}$, where $(\xrightarrow{1})^{45}$ means that he is idling for 45 minutes. This idle time ($study(20).\mathbf{I} \xrightarrow{1} study(20).\mathbf{I}$) is inferred by $\mathbf{Act}_2$. Note n of $\triangleright_{(n)}$ decreases even while idling.

Next, the behavior *PIZ* of the pizza-worker is defined:

$$PIZ \stackrel{\text{def}}{=} accept.(bake(15).drive(10).deliver.\mathbf{I} \rangle\, canceled.\mathbf{I})$$

If the pizza-worker *accepts* an order, then he *bakes* a pizza for 15 minutes, then *drives* for 10 minutes, and finally *delivers* the pizza. However, the order may always be *canceled*. In the fastest case, the pizza-worker can deliver the pizza after 25 minutes ($bake(15)$ and $drive(10)$), but the order may be canceled if he is idle as follows: $PIZ \xrightarrow{accept} (\xrightarrow{1})^{10} (\xrightarrow{bake})^{15} (\xrightarrow{1})^{30} (\xrightarrow{drive})^{5} \xrightarrow{canceled} \mathbf{I}$.

Finally, *RES* and *PIZ* are concurrently combined.

$$RP \equiv (RES@bob | PIZ@john) \setminus M_{rp}$$

where *bob* and *john* are names of the researcher and the pizza-worker, respectively. And M_{rp} is the set of feasible multi-actions in *RP* and is given as follows: $M_{rp} = M_s \cup M_r \cup M_p \cup \{\mu\cup\nu : \mu \in M_r, \nu \in M_p\}$, $M_s = \{\{order, accept\}, \{receive, deliver\}, \{cancel, canceled\}\}$, $M_r = \{\{study\}, \{eat\}, \{angry\}\}$, $M_p = \{\{bake\}, \{drive\}\}$, where agent-names $@bob$ and $@john$ appended to each action are omitted (e.g. $\{study\}$ is the abbreviation of $\{study@bob\}$). M_s is the set of multi-actions which must synchronize between the researcher and the pizza-worker. M_r and M_p are the sets of multi-actions which can independently or simultaneously be performed by the researcher and the pizza-worker, respectively. For example, an order must synchronize with an acceptance, and the pizza-worker can bake independently. For example, $RP \xrightarrow{\{order,accept\}} (\xrightarrow{\{bake\}})^{10} (\xrightarrow{\{study,bake\}})^{5} (\xrightarrow{\{study\}})^{5} (\xrightarrow{\{study,drive\}})^{10} \xrightarrow{\{receive,deliver\}} (\xrightarrow{\{eat\}})^{20} Done$ can be inferred, where $Done \equiv (\mathbf{I}@bob | \mathbf{I}@john) \setminus M_{rp}$. It means that the researcher rests for 10 minutes after the order, and then studies for 20 minutes, then receives the pizza, then eats it, as shown in Figure 3.

3 A PROCESS LOGIC *tLCA*

In this section, a process logic *tLCA* is defined to describe specifications for behaviors in *tCCA*.

It is assumed that a set $\mathcal{K}_{sp}$, ranged over by $K, \cdots$, of *Constants* is given. Then, the set *Sp*, ranged over by $s, t, \cdots$, of *specifications* is the smallest set which contains the following expressions:

$\langle\mu\rangle s$: Possibility ($\mu \in Act_M$), $\quad \bigwedge_{i\in I} s_i$: Conjunction,
$[\mu]s$: Necessity ($\mu \in Act_M$), $\quad \bigvee_{i\in I} s_i$: Disjunction,
K : Recursion ($K \in \mathcal{K}_{sp}$),

where s, s_i are already in *Sp*. And subsets of *Sp* are represented by $\sigma, \rho, \cdots$. These operators have the binding power such that: {Possibility, Necessity} > Conjunction > Disjunction. The following short notations are also used for the true $\mathtt{tt}$, the false $\mathtt{ff}$, and so on: $\mathtt{tt} \equiv \bigwedge_{i\in\emptyset} s_i$, $\mathtt{ff} \equiv \bigvee_{i\in\emptyset} s_i$, $s_1 \wedge s_2 \equiv \bigwedge_{i\in\{1,2\}} s_i$, $s_1 \vee s_2 \equiv \bigvee_{i\in\{1,2\}} s_i$, $\langle\mu\rangle^1 s \equiv \langle\mu\rangle s$, $\langle\mu\rangle^{n+1} s \equiv \langle\mu\rangle\langle\mu\rangle^n s$, $[\mu]^1 s \equiv [\mu]s$, $[\mu]^{n+1} s \equiv [\mu][\mu]^n s$, $\langle a_i@\psi_i : i \in I\rangle s \equiv \langle\{a_i@\psi_i : i \in I\}\rangle s$, $[a_i@\psi_i : i \in I]s \equiv [\{a_i@\psi_i : i \in I\}]s$.

The Possibility $\langle a_i@\psi_i : i \in I\rangle s$ requires that all agents-ψ_i ($i \in I$) can simultaneously perform a_i respectively, and s can be satisfied after that. The Necessity $[a_i@\psi_i : i \in I]s$ requires that if all agents-ψ_i simultaneously perform a_i respectively, then s is always satisfied after that. It is assumed that for every $K \in \mathcal{K}_{sp}$, there is a defining equation $K \stackrel{\text{def}}{=} s$, and each Constant is guarded by Possibilities or Necessities. For example, $K \stackrel{\text{def}}{=} \langle\mu\rangle K$ is guarded and $K \stackrel{\text{def}}{=} K \wedge \langle\mu\rangle K$ is not guarded. This recursion corresponds to maximum fixpoint.

Next, satisfaction relation of a concurrent behavior (in *tCCA*) for a specification (in *tLCA*) is defined by the function $DC : Sp \to 2^{2^{Sp}}$ as: $DC(\langle\mu\rangle s) = \{\{\langle\mu\rangle s\}\}$, $DC([\mu]s) = \{\{[\mu]s\}\}$, $DC(\bigwedge_{i\in I} s_i) = \{\bigcup_{i\in I} \sigma_i : \forall i \in I.\ \sigma_i \in DC(s_i)\}$, $DC(\bigvee_{i\in I} s_i) = \bigcup_{i\in I} DC(s_i)$, and $DC(K) = DC(s)$, if $K \stackrel{\text{def}}{=} s$. This function is used for translating any specification s into a disjunctive normal form $(\bigvee\{\bigwedge\{s' : s' \in \sigma\} : \sigma \in DC(s)\})$.

Definition 3.1 *Let $\nu \subseteq Act_S$. A set $\mathcal{R} \subseteq CBh \times Sp$ is a (ν)-satisfaction-subset, if $(C, s) \in \mathcal{R}$ implies that for some $\sigma \in DC(s)$, the following conditions hold for every $\mu, \mu'' \in Act_M$, $s' \in Sp$, and $C'' \in CBh$,*

(i) $\langle\mu\rangle s' \in \sigma \;\Rightarrow\; \exists (C', \mu').\, C \xrightarrow{\mu'} C', \mu \doteq_\nu \mu', (C', s') \in \mathcal{R}$,
(ii) $[\mu]s' \in \sigma, C \xrightarrow{\mu''} C'', \mu \doteq_\nu \mu'' \;\Rightarrow\; (C'', s') \in \mathcal{R}$,

where the (ν)-restricted equivalence relation $\doteq_\nu$ over multi-actions is defined as follows: $\mu_1 \doteq_\nu \mu_2$ if and only if $\mu_1 \cap \nu = \mu_2 \cap \nu$. Then, if $(C, s) \in \mathcal{R}$ for some (ν)-satisfaction-subset $\mathcal{R}$, then C satisfies *s with respect to ν, written $C \models_\nu s$. For the special case $\nu = Act_S$, $C \models_{Act_S} s$ is abbreviated to $C \models s$.* ∎

The parameter ν in $\models_\nu$ is used for *partial verification* and the set ν of single-actions are called an *available set*. The satisfaction relation $\models_\nu$ is the largest (ν)-satisfaction-subset and such definition style is similar to one of bisimilarity[8]. This satisfaction relation can be verified by algorithms similar to one for bisimilarity.

The following properties can be easily proven.

Proposition 3.1 *Let $\nu \subseteq Act_S$, $\mu \in Act_M$, $C \in CBh$, $s, s_i \in Sp$. Then,* (1) $C \models_\nu \langle\mu\rangle s$ *iff* $\exists(\mu', C').\ (C \xrightarrow{\mu'} C',\ \mu \doteq_\nu \mu',\ C' \models_\nu s)$, (2) $C \models_\nu [\mu]s$ *iff* $\forall(\mu', C').\ ((C \xrightarrow{\mu'} C',\ \mu \doteq_\nu \mu') \Rightarrow C' \models_\nu s)$, (3) $C \models_\nu \bigwedge_{i\in I} s_i$ *iff* $\forall i \in I.\ C \models_\nu s_i$, (4) $C \models_\nu \bigvee_{i\in I} s_i$ *iff* $\exists i \in I.\ C \models_\nu s_i$, (5) $C \models_\nu K$ *iff* $\exists s.\ (C \models_\nu s,\ K \stackrel{\rm def}{=} s)$. ∎

The following short notations are useful for expressing requirements for the passage of time:

$$s \vee_{\langle 0\rangle} t \equiv t,\ \ s \vee_{\langle n+1\rangle} t \equiv s \vee \bigvee\{\langle\mu\rangle(s \vee_{\langle n\rangle} t) : \mu \in Act_M\},$$
$$s \wedge_{[0]} t \equiv t,\ \ s \wedge_{[n+1]} t \equiv s \wedge \bigwedge\{[\mu](s \wedge_{[n]} t) : \mu \in Act_M\}.$$

$s \vee_{\langle n\rangle} t$ requires that s can be satisfied by n time-units pass or t can be satisfied after n time-units, for some execution path. On the other hand, $s \wedge_{[n]} t$ requires that s is always satisfied until n time-units pass and t is satisfied after n time-units, for any execution path.

The example of the researcher '*bob*' and the pizza-worker '*john*' is used again. Since *john* can bake a pizza for 15 minutes and can drive and deliver for 10+1 minutes, *bob* can begin to eat in 26 minutes as follows:

$$RP \models \langle order@bob, accept@john\rangle(\langle eat@bob\rangle \mathtt{tt} \vee_{\langle 27\rangle} \mathtt{ff}),$$

where $(\langle eat@bob\rangle \mathtt{tt} \vee_{\langle 27\rangle} \mathtt{ff})$ requires that *bob* can eat in 26 minutes because ff cannot be satisfied. Furthermore, *bob* never eats until 26 minutes pass as follows:

$$RP \models \langle order@bob, accept@john\rangle([eat@bob]\mathtt{ff} \wedge_{[26]} \mathtt{tt}).$$

Local specifications for the researcher *bob* can be easily verified by $\models_{\nu_b}$, where $\nu_b = \{a@bob : a \in \mathcal{N}_{ac}\}$. For example, it can be verified that the researcher can receive a pizza just after study for 30 minutes as follows:

$$RP \models_{\nu_b} s_b \equiv \langle order@bob\rangle\langle study@bob\rangle^{30}\langle receive@bob\rangle\mathtt{tt}.$$

In the rest of this section, we consider *satisfiability*. Before two agents combine, it is useful for avoiding unexpected behavior to verify consistency between their specifications. In *tLCA*, the consistency between s and t is replaced to satisfiability of $s \wedge t$ defined as follows.

Definition 3.2 *Let $s \in Sp$. s is* satisfiable *if and only if for some $C \in CBh$, $C \models s$.* ∎

We show that satisfiability in *tLCA* is undecidable by translating the membership problem in unrestricted grammars [6] into the satisfiability problem in *tLCA*. At first, action-sequence grammars G are defined.

Definition 3.3 *A tuple $G = \langle \mathcal{A}, a_0, \lambda\rangle$ is an* action-sequence grammar, *where $\mathcal{A} \subseteq \mathcal{N}_{ac}$ is a finite set of action-names, $a_0 \in \mathcal{N}_{ac}$ is the* initial action-name, *and λ is a finite set of* rewriting rules *such that: $\lambda \subseteq (\mathcal{A}^* - \{\varepsilon\}) \times \mathcal{A}^*$, where $\mathcal{A}^*$, ranged over by $\tilde{a}, \tilde{b}, \cdots$, is the set of finite action-sequences obtained by concatenating zero or more action-names in $\mathcal{A}$. The symbol ε is the empty action-sequence, thus $\tilde{a}\varepsilon = \varepsilon\tilde{a} = \tilde{a}$.* ∎

Definition 3.4 *Let $G{=}\langle \mathcal{A}, a_0, \lambda\rangle$ be an action-sequence grammar. The G-*language *$\mathcal{L}(G)$ is defined as follows: $\mathcal{L}(G) = \bigcup_{n\geq 0} \mathcal{L}^{(n)}(G)$, $\mathcal{L}^{(0)}(G) = \{a_0\}$, and $\mathcal{L}^{(n+1)}(G)$ $= \{\tilde{a}_1\tilde{b}\tilde{a}_2 : \exists\tilde{a}.\ \tilde{a}_1\tilde{a}\tilde{a}_2 \in \mathcal{L}^{(n)}(G), (\tilde{a}, \tilde{b}) \in \lambda\}$.* ∎

By Definition 3.3, action-sequence grammars are unrestricted grammars because there is no restriction on rewriting rules. It is known that the membership problem ($\tilde{a} \in \mathcal{L}(G)$?) in such grammars is undecidable.

Then, the following result is obtained.

Theorem 3.2 *Satisfiability in tLCA is undecidable.*

Proof Let $G = \langle \mathcal{A}, a_0, \lambda\rangle$ be an action-sequence grammar. Then, we can show that the following relation ($*$) holds. It implies that the satisfiability is undecidable because $\tilde{a} \notin \mathcal{L}(G)$ is undecidable. The details are omitted because of lack of space.

$$(GR_{(G)} \wedge [\tilde{a}\,\mathtt{end@p}]^*\mathtt{ff}) \text{ is satisfiable} \iff \tilde{a} \notin \mathcal{L}(G) \quad (*)$$

where the specification $GR_{(G)}$ is defined from G as:

$$GR_{(G)} \stackrel{\rm def}{=} RW_{(G)} \wedge EQ^{(\mathtt{q},\mathtt{p})}_{(G)} \wedge \partial\langle a_0@\mathtt{p}\rangle\partial\langle\mathtt{end@p}\rangle\mathtt{tt}$$
$$RW_{(G)} \stackrel{\rm def}{=} \bigwedge\{[\tilde{a}@\mathtt{p}]^*\langle\tilde{b}@\mathtt{q}\rangle^* EQ^{(\mathtt{p},\mathtt{q})}_{(G)} : (\tilde{a}, \tilde{b}) \in \lambda\}$$
$$\cup\{\partial[a@\mathtt{p}]\partial\langle a@\mathtt{q}\rangle RW_{(G)} : a \in \mathcal{A} \cup \{\mathtt{end}\}\}$$
$$EQ^{(\psi,\varphi)}_{(G)} \stackrel{\rm def}{=} \bigwedge\{\partial[a@\psi]\partial\langle a@\varphi\rangle EQ^{(\psi,\varphi)}_{(G)} : a \in \mathcal{A} \cup \{\mathtt{end}\}\}$$

where $\mathtt{end} \in \mathcal{N}_{ac}$, $\mathtt{end} \notin \mathcal{A}$, and specifications $\langle\tilde{a}@\psi\rangle^* s$ and $[\tilde{a}@\psi]^* s$ are short notations defined as follows: $\langle\varepsilon@\psi\rangle^* s \equiv s$, $\langle a\tilde{a}'@\psi\rangle^* s \equiv \partial\langle a@\psi\rangle\langle\tilde{a}'@\psi\rangle^* s$, $[\varepsilon@\psi]^* s \equiv s$, $[a\tilde{a}'@\psi]^* s \equiv \partial[a@\psi][\tilde{a}'@\psi]^* s$. Furthermore, for any $s \in Sp$, a Constant ∂s is defined as: $\partial s \stackrel{\rm def}{=} s \wedge [\emptyset]\partial s$. ∎

4 A PROCESS ALGEBRA *tICCA*

In this paper, our purpose is to show how to avoid unexpected behavior caused by composition of agents. For example, suppose that agent-ψ and agent-φ behave like P and Q (in *tCCA*), and must satisfy s and t (in *tLCA*), respectively. Then, it is a useful method for avoiding unexpected behavior to verify the consistency between s and t before agents-ψ and φ combine. However, there is no algorithm for verifying the consistency as shown in Section 3. To solve this problem, although it is expected to use a decidable and useful subclass of *tLCA*, we have not been able to define such subclass yet.

Alternate method is to verify whether $P@\psi|Q@\varphi$ satisfies $s \wedge t$, or not. There are algorithms for verifying the satisfaction relation $\models$. However, the number of reachable states of concurrent behavior explosively increases with concurrency level. To decrease the number of states, it is useful to hide useless actions because verification often considers only some actions (not all). Therefore we define a process algebra *tICCA* for expressing *intermediate specifications* (also called *im-specifications*) between concurrent behaviors (*tCCA*) and specifications (*tLCA*). In *tICCA*, states of behavior can decreases by hiding uninteresting actions. In this section, we show which actions can be hidden and how to verify composition of agents in *tICCA*.

It is assumed that a set $\mathcal{K}_{im}$, ranged over by $K, \cdots$, of *im-Constants* is given. Then, the set *ISp*, ranged over by $S, T, \cdots$, of *im-specifications* is the smallest set which contains the following expressions:

$$\begin{array}{rr} \mathbf{0} : \text{Stop}, & \mu;S : \text{(Insistent) Prefix}, \\ K : \text{Recursion}, & \bigoplus_{i\in I} S_i : \text{(Weak) Summation}, \end{array}$$

where $\mu \in Act_M$, $K \in \mathcal{K}_{im}$, and S, S_i are already in *ISp*.

It is important that the multi-action μ is directly prefixed to S, and the execution of μ is not delayed (i.e. $\mu;S \stackrel{\emptyset}{\not\longmapsto} \mu;S$). For $\bigoplus_{i\in I} S_i$, the choice is made even by a time passage. This operator is similar to 'weak choice' in TCCS[9]. We use a short notation for binary selection as follows: $S_1 \oplus S_2 \equiv \bigoplus_{i\in\{1,2\}} S_i$. It is assumed that for every im-Constant $K \in \mathcal{K}_{im}$, there is a defining equation $K \stackrel{\text{def}}{=} S$, and also that an idling im-Constant for each S is defined as follows: $\partial S \stackrel{\text{def}}{=} S \oplus \emptyset;\partial S$.

Semantics of im-specifications is given by the LTS $\langle ISp, Act_M, \{\stackrel{\mu}{\longmapsto} : \mu \in Act_M\}\rangle$, where the set $\stackrel{\mu}{\longmapsto}$ is the smallest relation satisfying the rules in Figure 4.

Then, we define im-Constants $S|T$, $S\backslash M$, and S/ν, for each $S, T \in ISp$, $M \subseteq Act_M$, and $\nu \subseteq Act_S$ as:

$$\begin{array}{rl} S|T & \stackrel{\text{def}}{=} \bigoplus\{(\mu\cup\nu);(S'|T') : \ S \stackrel{\mu}{\longmapsto} S',\ T \stackrel{\nu}{\longmapsto} T'\} \\ S\backslash M & \stackrel{\text{def}}{=} \bigoplus\{\mu;(S'\backslash M) : \ S \stackrel{\mu}{\longmapsto} S',\ \mu \in M\cup\{\emptyset\}\} \\ S/\nu & \stackrel{\text{def}}{=} \bigoplus\{(\mu\cap\nu);(S'/\nu) : \ S \stackrel{\mu}{\longmapsto} S'\} \end{array}$$

The satisfaction relation between a concurrent behavior and an im-specification is defined.

Definition 4.1 *Let $\nu \subseteq Act_S$. A set $\mathcal{R} \subseteq CBh \times ISp$ is a (ν)-*im-satisfaction-subset*, if $(C, S) \in \mathcal{R}$ implies the following conditions hold for every $\mu \in Act_M$,*

(i) $C \stackrel{\mu}{\longrightarrow} C' \Rightarrow \exists S'.\ S \stackrel{\mu\cap\nu}{\longmapsto} S', (C', S') \in \mathcal{R}$,

(ii) $S \stackrel{\mu}{\longmapsto} S' \Rightarrow \exists(\mu', S').\ \mu = \mu' \cap \nu, C \stackrel{\mu'}{\longrightarrow} C', (C', S') \in \mathcal{R}$.

Then, if $(C, S) \in \mathcal{R}$ for some (ν)-im-satisfaction-subset $\mathcal{R}$, then C satisfies *S with respect to ν, written $C \vdash_\nu S$. Especially, $C \vdash_{Act_S} S$ is written $C \vdash S$. The bisimilarity $S \sim T$ is defined in the same way as $C \vdash S$.* ∎

Name	Hypothesis	⇒	Conclusion
I.Act		⇒	$\mu;S \stackrel{\mu}{\longmapsto} S$
W.Sum	$\exists i \in I.\ S_i \stackrel{\mu}{\longmapsto} S'$	⇒	$\bigoplus_{i\in I} S_i \stackrel{\mu}{\longmapsto} S'$
Rec	$K \stackrel{\text{def}}{=} S,\ S \stackrel{\mu}{\longmapsto} S'$	⇒	$K \stackrel{\mu}{\longmapsto} S'$

Figure 4: Inference rules for $\stackrel{\mu}{\longmapsto} \subseteq ISp \times ISp$

By hiding uninteresting single-actions $a@\psi \notin \nu$, the number of states of im-specifications may decrease. The set ν in $\vdash_\nu$ is also called an *available set*.

The following Propositions 4.1 and 4.2 show how $\vdash_\nu$ is preserved by the operators $|$ and $\backslash$ of *tCCA*. Intuitively, the condition (`res`) in Proposition 4.2 requires that restricted single-actions must be included in ν.

Proposition 4.1 *If $Agn(C_1) \cap Agn(C_2) = \emptyset$ and $C_1 \vdash_{\nu_1} S_1$ and $C_2 \vdash_{\nu_2} S_2$, then $C_1|C_2 \vdash_\nu S_1|S_2$, where $\nu = (\nu_1\downarrow Agn(C_1)) \cup (\nu_2\downarrow Agn(C_2))$, and $\nu\downarrow\Psi$ is a subset of ν defined as follows: $\nu\downarrow\Psi = \{a@\psi \in \nu : \psi \in \Psi\}$.* ∎

Proposition 4.2 *If $\mathtt{res}(act(C), M, \nu) \subseteq M$ and $C \vdash_\nu S$, then $C\backslash M \vdash_\nu S\backslash M'$, where $M' = \{\mu \cap \nu : \mu \in M\}$ and $\mathtt{res} : 2^{Act_M} \times 2^{Act_M} \times 2^{Act_S} \to 2^{Act_M}$ is defined as:*

$$\mathtt{res}(M_0, M, \nu) = \{\mu_0 \in M_0 : \exists \mu \in M \cup \{\emptyset\}.\ \emptyset \neq \mu_0 \doteq_\nu \mu\},$$

and $act(C)$ is the set of feasible multi-actions in C at most, and is defined as: $act(P@\psi) = \{\{a@\psi\} : a \in acn(P)\} \cup \{\emptyset\}$, $act(C_1|C_2) = \{\mu_1 \cup \mu_2 : \mu_1 \in act(C_1), \mu_2 \in act(C_2)\}$, $act(C\backslash M) = (act(C) \cap M) \cup \{\emptyset\}$, where $acn(P)$ is the set of action-names occurring in P. ∎

The following Proposition 4.3 is used for hiding uninteresting actions. Furthermore, Proposition 4.4 shows that $C \models_\nu s$ can be verified by $S \models_\nu s$ if $C \vdash_\nu S$, where $S \models_\nu s$ is defined in the same way as $C \models_\nu s$.

Proposition 4.3 *If $C \vdash_\nu S$, then for any $\mu \subseteq Act_S$, $C \vdash_{\nu\cap\mu} S/\mu$.* ∎

Proposition 4.4 *Assume that $C \vdash_\nu S$. Then, $C \models_\nu s$ if and only if $S \models_\nu s$.* ∎

The example of the researcher and the pizza-worker is used again. Now, assume that the available sets are given as: $\nu_r = \{order@bob, receive@bob, cancel@bob\}$ and $\nu_p = \{accept@john, deliver@john, canceled@john\}$. Then, im-specifications S_r and S_p such that $RES@bob \vdash_{\nu_r} S_r$ and $PIZ@john \vdash_{\nu_p} S_p$ are given as follows:

$$\begin{array}{rl} S_r & \equiv \partial\{order@bob\};S_{wait}^{(60)}, \\ S_{wait}^{(n+1)} & \equiv \emptyset;S_{wait}^{(n)} \oplus \{receive@bob\};\partial\mathbf{0}, \\ S_{wait}^{(0)} & \equiv \partial\{cancel@bob\};\partial\mathbf{0}, \\ \\ S_p & \equiv \partial\{accept@john\};S_{work}^{(25)}, \\ S_{work}^{(n+1)} & \equiv \partial(\emptyset;S_{work}^{(n)} \oplus S_c), \\ S_{work}^{(0)} & \equiv \partial(\{deliver@john\};\partial S_c \oplus S_c), \\ S_c & \equiv \{canceled@john\};\partial\mathbf{0}. \end{array}$$

Then, by Proposition 4.1, we obtain that $RP \equiv (RES@bob|PIZ@john)\backslash M_{rp} \vdash_{\nu_r \cup \nu_p} (S_r|S_p)\backslash M'_{rp}$ because $\mathtt{res}(act(RES@bob|PIZ@john), M_{rp}, \nu_r \cup \nu_p) \subseteq M_{rp}$, where $M'_{rp} = M_s \cup \{\emptyset\}$. Furthermore, by Proposition 4.3, $RP \vdash_{\nu_r} S'_r$, where $S'_r \equiv (S_r|S_p)\backslash M'_{rp}/\nu_r$.

Now, let $s_r \equiv \langle order@bob\rangle(\langle receive@bob\rangle \mathtt{tt} \vee_{\langle 26\rangle} \mathtt{ff})$, then we can prove that $S'_r \models_{\nu_r} s_r$ according to Definition 3.1. Hence, by Proposition 4.4, $RP \models_{\nu_r} s_r$ because $RP \vdash_{\nu_r} S'_r$. The verification-cost of $S'_r \models_{\nu_r} s_r$ is lower than one of $RP \models_{\nu_r} s_r$ because the numbers of states of S'_r and RP are $2,491$ and $34,991$, respectively.

5 RELATED WORK

Many process algebras which have non-interleaving semantics have been proposed (e.g. in [1, 3, 5]) by considering locality or causality between actions, and a process logic with locality has also been given in [1]. Also, many timed process algebras have been discussed (e.g. in [9, 4, 13]). Furthermore, a timed process algebra with durational actions has been proposed and a process logic for it has been presented[2]. However, satisfiability in such process logics has not been discussed.

Many temporal logics which consider non-interleaving traces have been proposed (e.g. in [7, 12]), and satisfiability has been studied. However they have not considered the passage of time. For example, two specifications $\langle a@\psi\rangle\langle b@\varphi\rangle\mathtt{tt}$ and $\langle b@\varphi\rangle\langle a@\psi\rangle\mathtt{tt}$ are equal[2] in these temporal logics, but are not equal in *tLCA* because agent-ψ consumes one time-unit for performing the action-a and agent-φ can alter its own state in the passage of time. Hence, although it is proven that satisfiability in several non-interleaving temporal logics is undecidable in [7, 10], undecidability in *tLCA* cannot be directly proven from the results in [7, 10].

6 CONCLUSION

We have defined *tCCA* for describing concurrent behaviors of composite agents. *tLCA* has also been defined for describing specifications and it has been proven that satisfiability in *tLCA* is undecidable. Then we have proposed to use im-specifications described in *tICCA* instead of using a concurrent behavior, and have shown useful propositions for verifying composition of agents. An im-specification of a concurrent behavior C can be automatically generated from C by Proposition 4.3.

We have introduced non-interleaving for expressing concurrency in specifications because there is not always a *concurrent* system to satisfy a given specification even if there is a *sequential* system to satisfy it. For example, if non-interleaving is not considered, then a specification $s \equiv \langle a\rangle\langle b\rangle\mathtt{tt} \wedge [\emptyset][b]\mathtt{ff}$ is satisfiable (e.g. $a.b.\mathbf{I} \models s$). However, if it is required that a and b are performed by different agents like $s' \equiv \langle a@\psi\rangle\langle b@\varphi\rangle\mathtt{tt} \wedge [\emptyset][b@\varphi]\mathtt{ff}$, then s' is not satisfiable. Non-interleaving is necessary for correctly checking whether there is a *concurrent* system to satisfy a given specification, or not.

There are two reasons why we assume that the passage of one time-unit is always needed for executing any action. One is to remove unnatural situations such that actions can infinitely be performed in a finite time. Another is that we have expected that *tLCA* may be decidable by the assumption (i.e. the restriction), although we now know that *tLCA* is undecidable. It is a future work to define a decidable and useful subclass of *tLCA*.

[2] This equality is the key for proving undecidability in [7, 10].

ACKNOWLEDGMENTS

The authors wish to express our gratitude to Professor Shinichi Honiden of National Institute of Informatics, and to express our gratitude to Instructor Tadashi Iijima of Keio University. They are also members of ESP.

REFERENCES

[1] G.Boudol, I.Castellani, M. Hennessy, and A.Kiehn: Observing localities, *Theoretical Computer Science*, Vol.114, pp.31-61, 1993.

[2] X.J.Chen, F.Corradini: On the Specification and Verification of Performance Properties for a Timed Process Algebra, LNCS 1349, pp.123-137, 1997.

[3] P.Degano and C.Priami: Non-interleaving semantics for mobile processes, *Theoretical Computer Science*, Vol.216, pp.237-270, 1999.

[4] M.Hennessy, T.Regan: A process algebra for timed systems, *Info. and comp.*, Vol.117,pp.221-239,1995.

[5] P.Krishnan: Distributed CCS, CONCUR'91, LNCS 527, pp.393-407, 1991.

[6] P.Linz: *An Introduction to Formal Languages and Automata*, Jones and Bartlett Publishers, 1990.

[7] K.Lodaya, R.Parikh, R.Ramanujam, P.S. Thiagarahan: A Logical Study of Distributed Transition Systems, *Info. and comp.*, Vol.119, pp.91-118, 1995.

[8] R.Milner: *Communication and Concurrency*, Prentice-Hall, 1989.

[9] F.Moller and C.Tofts: A Temporal Calculus of Communicating Systems, CONCUR'90, LNCS 458, Springer-Verlag, pp.401-415, 1990.

[10] W.Penczek: Undecidability of Propositional Temporal Logics on Trace Systems, Information Processing Letters 43, pp.147-153, 1992.

[11] C.Stirling: An Introduction to Modal and Temporal Logics for CCS, LNCS 491, pp.2-20, 1989.

[12] P.S.Thiagarahan: A Trace Based Extension of Linear Time Temporal Logic, LICS, pp.438-447, 1994.

[13] I.Ulidowski and S.Yuen: Extending Process Languages with Time, LNCS 1349 pp.525-538, 1997.