Eventuality in LOTOS with a Disjunction Operator

Yoshinao ISOBE, Yutaka SATO, Kazuhito OHMAKI

Electrotechnical Laboratory
1-1-4 Umezono, Tsukuba, Ibaraki 305-8568, Japan
E-mail:{ isobe|ysato|ohmaki }@etl.go.jp

Abstract. *LOTOS* is a formal specification language, designed for the precise description of open distributed systems and protocols. Our purpose is to introduce the operators of logics (for example, disjunction, conjunction, greatest fixpoint, least fixpoint in μ-calculus) into (basic) LOTOS, in order to describe *flexible* specifications. *Disjunction operators* $\vee$ have been already proposed for expressing two or more implementations in a flexible pecification. In this paper, we propose an extended LOTOS with *two state operators*. They can control recursive behavior, in order to express *eventuality*. The eventuality is useful for *liveness properties* that something good *must* eventually happen. Then, we present a method for checking the consistency of a number of flexible specifications, and a method for producing a *conjunction specification* of them.

1 Introduction

The design of large scale distributed systems is known to be a complex task. In order to support the design, formal description techniques (FDTs) are used for verifying that a realized system conforms to its specification. Process algebra such as CCS[12], CSP[4], and LOTOS[17] is one of FDTs, and especially LOTOS is standardized by ISO.

In practise, *flexible specifications* are often given to a system instead of its complete specification in the first design step, and the flexible specifications are refined step by step, for reducing the number of possible implementations. In this case, a flexible specification represents two or more various implementations, however a specification described in process algebra usually represents only one implementation except equivalent implementations with it.

In order to describe such flexible specifications, disjunction operators $\vee$ have been proposed by Steen et al.[14] for LOTOS and independently by us[5] for Basic CCS. These operators are similar to a disjunction operator in logic, and if P_1 is an implementation of a specification S_1 and P_2 is an implementation of a specification S_2, then the specification $S_1 \vee S_2$ can be implemented by either P_1 or P_2, where an implementation is formally an specification expression which does not contain disjunction operators (i.e. it is executable). It is important to note that non-determinism of CSP can not always play the disjunction instead of $\vee$, because specifications can contain non-determinism, such as for gambling machines or timeout (see [14]).

For example, the following specification AB represents implementations which can iteratively perform the action a or can stop after the action b.

$$AB := a; AB \vee b; \mathbf{stop}$$

where ; is a prefix operator, thus $a; AB$ requires its implementations that they can perform a and thereafter conform to the specification AB. The symbol := is used for defining the left *Constant* AB as the right specification, thus it is a recursive definition. In this case, the disjunction $\vee$ is recursively resolved. Therefore, all the following implementations satisfy the specification AB.

$$A_\infty := a; A_\infty, \quad AB_0 := b; \mathbf{stop}, \quad AB_2 := a; a; b; \mathbf{stop}$$

In the above example, the action b can not be always performed in implementations satisfying AB, because A_∞ satisfies AB.

Designers often require that something good *must* eventually happen, namely a *liveness property*. For example, if the above action b must eventually happen, then how is AB modified? An answer is to use an *infinite* disjunction (intuitively, like $\bigvee_{(n>0)} a^n; b; \mathbf{stop}$), but the infinity complicates integration, verification, et al. of flexible specifications.

In this paper, we propose to use two kinds of stats, called *stable states* and *unstable states*, in order to express eventuality. Intuitively, disjunction operators must be resolved so that a stable state is eventually selected. For example, the following specification AB' represents implementations which can perform finite a and *must eventually* stop after b.

$$AB' := \triangleleft a; AB' \vee \circ b; \mathbf{stop}$$

where $\triangleleft$ and $\circ$ are called an *un-stabilizer* and a *stabilizer*, and they make an unstable stable sate ($\triangleleft a; AB$) and a stable state ($\circ b; \mathbf{stop}$), respectively. Thus, ($\triangleleft a; AB$) makes it impossible to infinitely select the action a. Consequently, the above AB_0 and AB_2 satisfy AB', but A_∞ does not satisfy AB'.

The outline of this paper is as follows. In Section 2, we propose an extended labelled transition system called μLTS, by introducing unstable states into the ALTS[14]. The ALTS is an labelled transition system (LTS) extended by adding *unlabeled transitions* for disjunction operators. Then, we define a specification language called μLOTOS based on the μLTS. In Section 3, a satisfaction relation between an implementation and a specification is defined, and the properties of unstable states are shown. In Section 4, we present a method for checking the consistency of a number of specifications, and a method for producing a conjunction specification of them. In Section 5, we discuss related works. In Appendix, a table of the notations used in this paper is given.

2 Definition of specifications

In this section, we present a specification language called μLOTOS for describing flexible specifications. In order to concisely explain our main ideas, we will only consider a small subset of the operators of LOTOS in this paper, but it is not difficult to introduce the other operators into μLOTOS.

In Subsection 2.1, the syntax of μLOTOS is defined. In Subsection 2.2, a μLTS is given, and then the semantics of μLOTOS is defined.

2.1 Syntax

We assume that a finite set of *names* $\mathcal{N}$ is given. The set of actions Act is defined as $Act = \mathcal{N} \cup \{i\}$ and $\alpha, \beta, \cdots$ are used to range over Act, where i is a special action called an *internal action* ($i \notin \mathcal{N}$). We give a set of *state operators* $\Psi = \{\circ, \triangleleft\}$, where $\circ$ is called a *stabilizer* and $\triangleleft$ is called an *un-stabilizer*. The set Ψ is ranged over by $\psi, \phi, \cdots$.

We also assume that a set of *specification constants* (also called *Constants*) $\mathcal{K}$ and a set of *specification variables* (also called *Variables*) $\mathcal{X}$ are given. The set $\mathcal{K}$ is ranged over by $A, B, \cdots$, and the set $\mathcal{X}$ is ranged over by $X, Y, \cdots$.

Then, the syntax of μLOTOS is defined.

Definition 2.1 *We define* specification expressions M *with the following syntax:*

$$M ::= A \mid X \mid \mathbf{stop} \mid \psi\alpha; M \mid M \,[\!]\, M \mid M \,|[G]|\, M \mid M \vee M$$

where $A \in \mathcal{K}$*,* $X \in \mathcal{X}$*,* $\psi \in \Psi$*,* $\alpha \in Act$*, and* $G \subseteq \mathcal{N}$*. The set of all the specification expressions is denoted by* $\mathcal{M}$ *and* $M, N, \cdots$ *range over* $\mathcal{M}$*. The operators* ; *,* $[\!]$*,* $|[G]|$*, and* $\vee$ *are called a* Prefix*, a* Choice*, a* Parallel*, and a* Disjunction*, respectively.* □

The difference between the Choice operator $[\!]$ and the Disjunction operator $\vee$ is intuitively explained as follows. For the Choice, users decide whether $M \,[\!]\, N$ behaves like either M or N at run time, i.e. a dynamic choice. For the Disjunction, designers decide whether $M \vee N$ is implemented by either M or N in specification phase, i.e. a static choice. Thus, Disjunctions are used only in specifications and does not remain in implementations.

The Parallel operator $|[G]|$ of LOTOS synchronizes actions included in G and independently performs the other actions. This can synchronize three or more specifications.

We write $Var(M)$ for the set of Variables occurring in the specification expression M, and it is inductively defined as follows :

$$\begin{array}{ll} Var(A) = \emptyset, & Var(\psi\alpha; M) = Var(M), \\ Var(\mathbf{stop}) = \emptyset, & Var(M \; op \; N) = Var(M) \cup Var(N), \\ Var(X) = \{X\}, & \end{array}$$

where op is $[\!]$ or $|[G]|$ or $\vee$. A specification expression M is called a *specification*, if it contains no Variables (i.e. $Var(M) = \emptyset$). The set of specifications is denoted by $\mathcal{S}$, and it is ranged over by $S, T, U, \cdots$.

A Constant is a specification whose meaning is given by a defining equation. We assume that for every Constant $A \in \mathcal{K}$, there is a defining equation of the form $A := S$, where S is a specification which can contain Constants again. Thus, it is a recursive definition. We assume that recursion must be guarded by Prefixes, such as $A := \circ a; A$. For example, we do not consider $A := A \,[\!]\, \circ a.\mathbf{stop}$.

The state operators $\circ$ and $\triangleleft$ make stable states and unstable states. A stable state corresponds to a state in standard LOTOS. If every un-stabilizer $\triangleleft$ is replaced with a stabilizer $\circ$, then μLOTOS is the same as the language of [14]. Note that stabilizer $\circ$ is often omitted. For example, $\circ\alpha; M$ is written as $\alpha; M$.

A specification which neither contains Disjunctions nor un-stabilizers, is called a *process* or an *implementation*. Thus, the set of processes $\mathcal{P}$ is a subset of $\mathcal{S}$, and the syntax is defined in terms of the following BNF expression:

$$P ::= A \mid \mathbf{stop} \mid \circ\alpha ; P \mid P \,[\,]\, P \mid P\,|[G]|\,P$$

where $A \in \mathcal{K}_P \subseteq \mathcal{K}$, $\alpha \in \mathit{Act}$, and $G \subseteq \mathcal{N}$. We assume that for every Constant $A \in \mathcal{K}_P$, there is a defining equation of the form $A := P$, where $P \in \mathcal{P}$. The set $\mathcal{P}$ is ranged over by $P, Q, \cdots$.

In order to avoid too many parentheses, operators have binding power in the following order: Prefix > Parallel > Choice > Disjunction. We also use the following short notations:

$$\sum \mathcal{C} \equiv \begin{cases} \mathbf{stop} & (\mathcal{C} = \emptyset) \\ M_1 \,[\,]\, M_2 \,[\,]\, \cdots \,[\,]\, M_n & (\mathcal{C} = \{M_1, \cdots, M_n\}) \end{cases}$$
$$\bigvee \mathcal{C} \equiv \begin{cases} \mathbf{F} & (\mathcal{C} = \emptyset) \\ M_1 \vee M_2 \vee \cdots \vee M_n & (\mathcal{C} = \{M_1, \cdots, M_n\}) \end{cases}$$

where $\mathcal{C}$ is a finite subset of specifications and the relation $\equiv$ represents syntactic identity. $\mathbf{F}$ is a specification constant defined as follows:

$$\mathbf{F} := \lhd i ; \mathbf{F}$$

where i is an internal action. Intuitively, no process satisfies $\mathbf{F}$, because $\mathbf{F}$ has only one unstable state. On the other hand, a specification constant $\mathbf{T}$ which is satisfied by all the processes is defined as follows:

$$\mathbf{T} := \bigvee \{ \sum \{ \circ\alpha ; \mathbf{T} \ : \ \alpha \in \mathcal{A} \} \ : \ \mathcal{A} \subseteq \mathit{Act} \}$$

The formal properties of $\mathbf{F}$ and $\mathbf{T}$ are shown in Proposition 3.2 in Section 3.

2.2 Semantics

At first, we propose an extended labelled transition system called μLTS, by introducing unstable states into the ALTS[14]. The ALTS is a labelled transition system (LTS) with unlabeled transitions. The difference between the μLTS and the ALTS is that the μLTS has the set $\bigcirc$ of stable states. In other words, if $\bigcirc$ is the set of all the states, then the μLTS is the same as the ALTS.

Definition 2.2 *A* μLTS *is a structure* $\langle \mathit{ST}, L, \rightarrow, \mapsto, \bigcirc \rangle$*, where* ST *is a set of states,* L *is a set of labels,* $\rightarrow \subseteq \mathit{ST} \times L \times \mathit{ST}$ *is a set of labelled transitions,* $\mapsto \subseteq \mathit{ST} \times \mathit{ST}$ *is a set of unlabelled transitions, and* $\bigcirc \subseteq \mathit{ST}$ *is a* set of stable states. *As a notational convention, we write* $s \xrightarrow{\alpha} s'$ *for* $(s, \alpha, s') \in \rightarrow$ *and* $s \mapsto s'$ *for* $(s, s') \in \mapsto$. $\square$

In μLTS, stable states are defined as follows: a state $s \in \mathit{ST}$ is *stable* if and only if either $s \in \bigcirc$ or $s' \not\rightarrow$ for all s' such that $s \mapsto s'$, where $s \not\rightarrow$ means that there is no pair (α, s') such that $s \xrightarrow{\alpha} s'$. So, a stop s is stable, even if $s \notin \bigcirc$.

The semantics of μLOTOS is given by the μLTS $\langle \mathcal{M}, \mathit{Act}, \rightarrow, \mapsto, \bigcirc \rangle$, where $\rightarrow$ is defined in Definition 2.3, $\mapsto$ is defined in Definition 2.4, and $\bigcirc$ is defined in Definition 2.5. *In this paper, we consider only specification expressions with finite states.*

Definition 2.3 *The labelled transition relation* $\rightarrow \subseteq \mathcal{M} \times Act \times \mathcal{M}$ *is the smallest relation satisfying the following inference rules.*

Name	*Hypothesis*	$\vdash$	*Conclusion*
Act		$\vdash$	$\psi\alpha; M \xrightarrow{\alpha} M$
Con	$S \xrightarrow{\alpha} S',\ A := S$	$\vdash$	$A \xrightarrow{\alpha} S'$
Ch$_1$	$M \xrightarrow{\alpha} M'$	$\vdash$	$M \mathbin{[]} N \xrightarrow{\alpha} M'$
Ch$_2$	$N \xrightarrow{\alpha} N'$	$\vdash$	$M \mathbin{[]} N \xrightarrow{\alpha} N'$
Par$_1$	$M \xrightarrow{\alpha} M',\ \alpha \notin G$	$\vdash$	$M \mathbin{\vert[G]\vert} N \xrightarrow{\alpha} M' \mathbin{\vert[G]\vert} N$
Par$_2$	$N \xrightarrow{\alpha} N',\ \alpha \notin G$	$\vdash$	$M \mathbin{\vert[G]\vert} N \xrightarrow{\alpha} M \mathbin{\vert[G]\vert} N'$
Par$_3$	$M \xrightarrow{\alpha} M',\ N \xrightarrow{\alpha} N',\ \alpha \in G$	$\vdash$	$M \mathbin{\vert[G]\vert} N \xrightarrow{\alpha} M' \mathbin{\vert[G]\vert} N'$

$\square$

Definition 2.4 *The unlabelled transition relation* $\mapsto\ \subseteq \mathcal{M} \times \mathcal{M}$ *is the smallest relation satisfying the following inference rules.*

Name	*Hypothesis*	$\vdash$	*Conclusion*
Act$_\vee$		$\vdash$	$\psi\alpha; M \mapsto \psi\alpha; M$
Stop$_\vee$		$\vdash$	**stop** $\mapsto$ **stop**
Con$_\vee$	$S \mapsto S',\ A := S$	$\vdash$	$A \mapsto S'$
Ch$_\vee$	$M \mapsto M',\ N \mapsto N'$	$\vdash$	$M \mathbin{[]} N \mapsto M' \mathbin{[]} N'$
Par$_\vee$	$M \mapsto M',\ N \mapsto N'$	$\vdash$	$M \mathbin{\vert[G]\vert} N \mapsto M' \mathbin{\vert[G]\vert} N'$
Dis$_1$	$M \mapsto M'$	$\vdash$	$M \vee N \mapsto M'$
Dis$_2$	$N \mapsto N'$	$\vdash$	$M \vee N \mapsto N'$

$\square$

Definition 2.5 *The set of stable states* $\bigcirc \subseteq \mathcal{M}$ *is the smallest relation satisfying the following inference rules.*

Name	*Hypothesis*	$\vdash$	*Conclusion*
Act$_\circ$		$\vdash$	$\circ\alpha; M \in \bigcirc$
Stop$_\circ$		$\vdash$	**stop** $\in \bigcirc$
Con$_\circ$	$S \in \bigcirc,\ A := S$	$\vdash$	$A \in \bigcirc$
Ch$_\circ$	$M \in \bigcirc,\ N \in \bigcirc$	$\vdash$	$M \mathbin{[]} N \in \bigcirc$
Par$_\circ$	$M \in \bigcirc,\ N \in \bigcirc$	$\vdash$	$M \mathbin{\vert[G]\vert} N \in \bigcirc$
Dis$_\circ$	$M \in \bigcirc,\ N \in \bigcirc$	$\vdash$	$M \vee N \in \bigcirc$

$\square$

The rules for labelled transitions $\longrightarrow$ is exactly same as the rules in standard LOTOS, except ψ in **Act**. The state operator ψ does not affect $\longrightarrow$.

Unstable states can be made from un-stabilizers $\triangleleft$, because there is no rule for $\triangleleft\alpha; M$ in Definition 2.5. It is noted that there are stable state M, even if $M \notin \bigcirc$. For example, the specification $S \equiv (\triangleleft a; S_1 \mathbin{|[a,b]|} \circ b; S_2)$ is stable, because $S \mapsto S' \not\longrightarrow$ for all S', although $S \notin \bigcirc$.

Unlabelled transitions $\mapsto$ [14, 5] are used for resolving disjunction operators, as shown in the rules **Dis**$_{1,2}$. Intuitively, a process P satisfies a specification S, if and only if $S \mapsto S'$ and P satisfies S' for some specification S'. For example, the following specification VM of a vending machine can be implemented by either $(coin; coffee; \mathbf{stop})$ or $(coin; tea; \mathbf{stop})$,

$$VM := (coin; coffee; \mathbf{stop}) \vee (coin; tea; \mathbf{stop})$$

because $VM \mapsto (coin; coffee; \mathbf{stop})$ and $VM \mapsto (coin; tea; \mathbf{stop})$.

The definition of $\mapsto$ is slightly changed from [14]. In our definition, all the specification can perform unlabelled transitions, and it is not necessary to *successively* perform unlabelled transitions twice. Formally, the following proposition holds, where $\mathcal{M}_0$ is the set of specification expressions which do not change states by unlabelled transitions, thus $\mathcal{M}_0 = \{M : \{M' : M \mapsto M'\} = \{M\}\}$.

Proposition 2.1 *If $M \mapsto M'$, then $M' \in \mathcal{M}_0$.*
Proof By induction on the length of the inference of $M \mapsto M'$. We show only one case by $\mathbf{Par}_\vee$, here. By $\mathbf{Par}_\vee$, $M \mapsto M'$ implies that for some M_1, M_2, M_1', M_2', and G, $M \equiv M_1 \,|[G]|\, M_2$, $M' \equiv M_1' \,|[G]|\, M_2'$, $M_1 \mapsto M_1'$, and $M_2 \mapsto M_2'$. Thus, by induction, $M_1' \in \mathcal{M}_0$ and $M_2' \in \mathcal{M}_0$. These imply that if $M' \equiv M_1' \,|[G]|\, M_2' \mapsto M''$, then $M'' \equiv M_1' \,|[G]|\, M_2'$ by $\mathbf{Par}_\vee$. Hence, $M' \in \mathcal{M}_0$. $\square$

In the rest of this paper, $\mathcal{M}_0$ (which is a subset of $\mathcal{M}$) is ranged over by $M_0, N_0, \cdots$. And also, we use $\mathcal{S}_0$ to denote the set of M_0 which contain no Variables, thus $\mathcal{S}_0 = \mathcal{S} \cap \mathcal{M}_0$, and $\mathcal{S}_0$ is ranged over by $S_0, T_0, \cdots$.

3 Satisfaction

In this section, we define a satisfaction $P \models S$ of a process P for a specification S as an extension of the satisfaction $P \models_{[14]} S$ in [14]. The definition of $\models_{[14]}$ has been given as follows: the satisfaction $\models_{[14]}$ is the largest relation such that, $P \models_{[14]} S$ implies that for some S_0, $S \mapsto S_0$ and for all $\alpha \in Act$ the following two conditions hold:

$(i.[14])$ if $P \xrightarrow{\alpha} P'$ then, for some S', $S_0 \xrightarrow{\alpha} S'$ and $P' \models_{[14]} S'$,
$(ii.[14])$ if $S_0 \xrightarrow{\alpha} S'$ then, for some P', $P \xrightarrow{\alpha} P'$ and $P' \models_{[14]} S'$.

This requires that there exists an S_0 which satisfies $(i.[14])$ and $(ii.[14])$. This makes it possible that a specification can be satisfied by two or more various processes. As shown in Proposition 2.1, S_0 can not be resolved any more by $\mapsto$, but it may be resolved again after an labelled transition $S_0 \xrightarrow{\alpha} S'$. Therefore, the definition of the satisfaction is inductive. For example, the specification $(a; (b; \mathbf{stop} \vee c; \mathbf{stop}) \vee d; \mathbf{stop})$ can be implemented by either $(a; b; \mathbf{stop})$, or $(a; c; \mathbf{stop})$, or $(a; b; \mathbf{stop} \,[]\, a; c; \mathbf{stop})$, or $(d; \mathbf{stop})$.

In the definition of $\models_{[14]}$, the specification S_0 can be freely selected from $\{S_0 : S \mapsto S_0\}$. On the other hand, we can control the selection by state operators $\circ$ and $\lhd$. The key point is that S must eventually reach a stable state. Then, our satisfaction is defined as follows.

Definition 3.1 *A relation $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{S}$ is a* satisfaction relation, *if $(P, S) \in \mathcal{R}$ implies $(P, S) \in \theta(\mathcal{R})$, where $\theta(\mathcal{R}) \subseteq \mathcal{P} \times \mathcal{S}$ is inductively defined for any relation $\mathcal{R}$, as follows:*

• $(P, S) \in \theta^{(0)}(\mathcal{R})$ *iff for some S_0, $S \mapsto S_0$, $S_0 \in \bigcirc$, and for all $\alpha \in Act$,*

$$\begin{array}{l} (i)\ \textit{if } P \xrightarrow{\alpha} P' \textit{ then, for some } S',\ S_0 \xrightarrow{\alpha} S' \textit{ and } (P', S') \in \mathcal{R}, \\ (ii)\ \textit{if } S_0 \xrightarrow{\alpha} S' \textit{ then, for some } P',\ P \xrightarrow{\alpha} P' \textit{ and } (P', S') \in \mathcal{R}, \end{array}$$

• $(P, S) \in \theta^{(n+1)}(\mathcal{R})$ *iff for some S_0, $S \mapsto S_0$ and for all $\alpha \in Act$,*

$$\begin{array}{l} (i)\ \textit{if } P \xrightarrow{\alpha} P' \textit{ then, for some } (m, S'),\ S_0 \xrightarrow{\alpha} S',\ (P', S') \in \theta^{(m)}(\mathcal{R}),\ m \le n, \\ (ii)\ \textit{if } S_0 \xrightarrow{\alpha} S' \textit{ then, for some } (m, P'),\ P \xrightarrow{\alpha} P',\ (P', S') \in \theta^{(m)}(\mathcal{R}),\ m \le n, \end{array}$$

• $(P, S) \in \theta(\mathcal{R})$ *iff $(P, S) \in \theta^{(n)}(\mathcal{R})$, for some n.* □

Definition 3.2 *P* satisfies *S, written $P \models S$, if $(P, S) \in \mathcal{R}$, for some satisfaction relation $\mathcal{R}$.* (i.e. $\models$ is the relation $\bigcup\{\mathcal{R}\ :\ \mathcal{R}$ is a satisfaction relation$\}$). *We use the notation $Proc(S)$ for the set of all the processes which satisfy the specification S* (i.e. $Proc(S) = \{P\ :\ P \models S\}$). □

The relation $\models$ is the largest satisfaction relation, and we can prove that $P \models S$ if and only if $(P, S) \in \theta(\models)$. $P \models S$ requires that S must reach a stable state S_0 after finite transitions, where P and S must keep the relation $\models$. It is noted that if $P \not\longrightarrow$ and $S \mapsto S_0 \not\longrightarrow$ for some S_0, then $(P, S) \in \theta^{(1)}(\models)$, even if $S_0 \notin \bigcirc$, because S_0 is stable. For example, $(\mathbf{stop}, \triangleleft a; S_1 \,|[a, b]|\, \circ b; S_2) \in \theta^{(1)}(\models)$.

The following relations between $\models$ and $\models_{[14]}$ can be easily shown.

– if $P \models S$, then $P \models_{[14]} S$.
– if $P \models_{[14]} S$ and S has only stable states, then $P \models S$

The important proposition for the Disjunction $\vee$ shown in [14] holds also for our satisfaction.

Proposition 3.1 *Let $S, T \in \mathcal{S}$. Then $Proc(S \vee T) = Proc(S) \cup Proc(T)$.*
Proof This is similar to the proof of Proposition 7 in [14]. □

In Subsection 2.1, we defined two special specifications $\mathbf{T}$ and $\mathbf{F}$. Two propositions for $\mathbf{T}$ and $\mathbf{F}$ are given: Proposition 3.2 shows that all the processes satisfy $\mathbf{T}$ and no process satisfies $\mathbf{F}$. Proposition 3.3 shows the properties for substitution, where the notation $M\{N/X\}$ indicates the substitution of N for every occurrence of the Variable X in M, and the notation $\tilde{M}$ is an indexed set $M_1, \cdots, M_n$. For example, $\{\tilde{S}/\tilde{X}\}$ represents $\{S_1/X_1, S_2/X_2, \cdots, S_n/X_n\}$, and $\{\mathbf{T}/\tilde{X}\}$ represents $\{\mathbf{T}/X_1, \mathbf{T}/X_2, \cdots, \mathbf{T}/X_n\}$.

Proposition 3.2 (1) *$Proc(\mathbf{T}) = \mathcal{S}$ and* (2) *$Proc(\mathbf{F}) = \emptyset$.*
Proof (1) $\mathbf{T}$ has only stable states and any combination of actions ($\mathcal{A} \subseteq Act$) can be selected by $\mapsto$ from $\mathbf{T}$. (2) $\mathbf{F}$ has only one unstable state, because $\triangleleft i; \mathbf{F} \notin \bigcirc$ and $\mathbf{F}$ has always a transition by i. □

Proposition 3.3 *Let M contain Variables $\tilde{X}$ at most. For any $\tilde{S} \in \mathcal{S}$, the following relations hold.*

1. $\mathit{Proc}(M\{\tilde{S}/\tilde{X}\}) \subseteq \mathit{Proc}(M\{\mathbf{T}/\tilde{X}\})$.
2. $\mathit{Proc}(M\{\mathbf{F}/\tilde{X}\}) \subseteq \mathit{Proc}(M\{\tilde{S}/\tilde{X}\})$.

Proof (outline)

1. We can show that the following $\mathcal{R}$ is a satisfaction relation.
$$\mathcal{R} = \{(P,T) : \exists S, P \models S, T \in \mathit{TR}(S), P \in \mathcal{P}, S \in \mathcal{S}\} \cup \models$$
where $\mathit{TR}(S)$ is the set of all the specifications obtained from the specification S by replacing some subexpressions of S by $\mathbf{T}$.
2. We can show that the following $\mathcal{R}$ is a satisfaction relation.
$$\mathcal{R} = \{(P, M\{\tilde{S}/\tilde{X}\}) : P \models M\{\mathbf{F}/\tilde{X}\}, P \in \mathcal{P}, S \in \mathcal{S}\} \cup \models$$
For this proof, the following property is used : If $M\{\mathbf{F}/\tilde{X}\} \mapsto T_0$ and $P \models T_0$, then for some M_0, $M \mapsto M_0$, $T_0 \equiv M_0\{\mathbf{F}/\tilde{X}\}$, M_0 is guarded, and for any $\tilde{S}$, $M\{\tilde{S}/\tilde{X}\} \mapsto M_0\{\tilde{S}/\tilde{X}\}$ $\Box$

Next, we show examples of $P \models S$. At first, consider the following process *PAB* and the specification *SAB*:

$$\mathit{PAB} := a; a; b; \mathit{PAB} \qquad \mathit{SAB} := \triangleleft a; \mathit{SAB} \vee \circ b; \mathit{SAB}$$

In the specification *SAB*, only $(\circ b; \mathit{SAB})$ is stable. Thus, *SAB* requires that the action b must always eventually be performed, although the action a may be performed zero or more times before b. In this case, we can show that $\mathit{PAB} \models \mathit{SAB}$, because the following $\mathcal{R}$ is a satisfaction relation,

$$\mathcal{R} = \{(\mathit{PAB}, \mathit{SAB}), (a; b; \mathit{PAB}, \mathit{SAB}), (b; \mathit{PAB}, \mathit{SAB})\}$$

because $(\mathit{PAB}, \mathit{SAB}) \in \theta^{(2)}(\mathcal{R})$, $(a; b; \mathit{PAB}, \mathit{SAB}) \in \theta^{(1)}(\mathcal{R})$, and $(b; \mathit{PAB}, \mathit{SAB}) \in \theta^{(0)}(\mathcal{R})$.

Secondly, the following specification *FILE* is considered.

$$\begin{aligned} \mathit{FILE} &:= \mathit{open}; \mathit{OPENED} \times \mathit{creat}; \mathit{FILE} \\ \mathit{OPENED} &:= \triangleleft \mathit{write}; \mathit{OPENED} \times \triangleleft \mathit{read}; \mathit{OPENED} \times \mathit{close}; \mathit{FILE} \end{aligned}$$

where $M \times N$ is the short notation defined as follows.

$$M \times N \equiv M \vee N \vee (M \,[]\, N)$$

The specification *OPENED* requires that the action *close* must be eventually performed, because of the un-stabilizers of $\triangleleft$*write* and $\triangleleft$*read*. Thus, this specification *FILE* requires that a file must be eventually closed by the action *close* after opened by the action *open*, and/or that a file can be created by the action *creat*. The subexpression $(\triangleleft \mathit{write}; \mathit{OPENED} \times \triangleleft \mathit{read}; \mathit{OPENED})$ permits that actions *write* and *read* are inserted after *open* and before *close*. For example, *FILE* can be implemented by the following processes *READ* or *UPDATE* (i.e. $\mathit{READ} \models \mathit{FILE}$ and $\mathit{UPDATE} \models \mathit{FILE}$).

$$\begin{aligned} \mathit{READ} &:= \mathit{open}; \mathit{read}; \mathit{close}; \mathit{READ} \,[]\, \mathit{creat}; \mathit{READ} \\ \mathit{UPDATE} &:= \mathit{open}; \mathit{read}; (\mathit{close}; \mathit{UPDATE} \,[]\, \mathit{write}; \mathit{close}; \mathit{UPDATE}) \end{aligned}$$

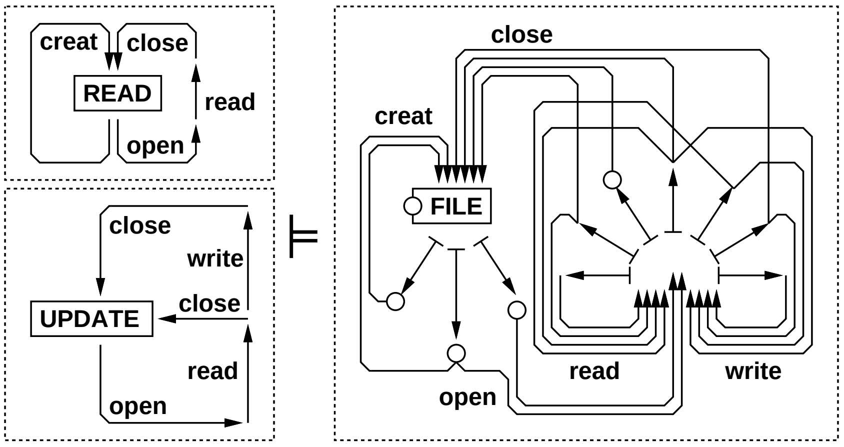

Fig. 1. The transition graphs of *READ*, *UPDATE*, and *FILE* ($\circ$: a stable state)

The transition graphs of *READ*, *UPDATE*, and *FILE* are shown in Fig. 1, where each circle in *FILE* means a stable state, and unlabelled transitions which do not change states are omitted.

The un-stabilizers $\triangleleft$ in *OPENED* guarantee that the action *close* must be eventually performed. If the un-stabilizer of $\triangleleft$*read* in *OPENED* is replaced by a stabilizer $\circ$, then *FILE* can be also implemented by the following unexpected process *READLOOP*.

$$\mathit{READLOOP} := \mathit{open}; \mathit{LOOP}, \qquad \text{where } \mathit{LOOP} := \mathit{read}; \mathit{LOOP}$$

As another example, a special case $I_S := \triangleleft i; I_S \vee S$ is interesting, where i is an internal action. This means that zero or more finite internal actions can be performed before S. Although the internal action is not distinguished from any other actions in Definition 3.1 like *strong bisimilarity*[12], it is possible by I_S to ignore finite internal actions like in *weak branching bisimilarity*[3] for convergent specifications (no internal action cycles (p.148 in [12])).

In the rest of this section, important properties of state operators $\circ$ and $\triangleleft$ are shown. At first, we define two subsets $\mathcal{M}_\nu$ and $\mathcal{M}_\mu$ of $\mathcal{M}$ as follows.

Definition 3.3 *The specification expressions M in $\mathcal{M}_\nu$ are defined with the following syntax:*

$$M ::= A \mid X \mid \mathbf{stop} \mid \circ\alpha; M \mid M \,[]\, M \mid M\,|[G]|\,M \mid M \vee M$$

where $A \in \mathcal{K}_\nu \subseteq \mathcal{K}$, $X \in \mathcal{X}$, $\alpha \in Act$, and $G \subseteq \mathcal{N}$. We assume that for every $A \in \mathcal{K}_\nu$, there is a defining equation of the form $A := P$ and $P \in \mathcal{M}_\nu$. □

Definition 3.4 *The specification expressions M in $\mathcal{M}_\mu$ are defined with the following syntax:*

$$M ::= X \mid S \mid \triangleleft\alpha; M \mid M \,[]\, M \mid M\,|[G]|\,M \mid M \vee M$$

where $S \in \mathcal{S}$, $\alpha \in Act$, and $G \subseteq \mathcal{N}$. □

It is important to note that differences between $\mathcal{M}_\nu$ and $\mathcal{M}_\mu$ are not only $\circ\alpha; M$ and $\lhd\alpha; M$. Every specification in $\mathcal{M}_\nu$ contains no unstable states, and $\mathcal{M}_\nu$ is the same as the language in [14]. On the other hand, $\mathcal{M}_\mu$ can contain the specification $\circ\alpha; M$, if M contains no Variable, because $\mathcal{M}_\mu$ contains $S \in \mathcal{S}$.

Then, Theorem 3.4 holds, where the indexed definition $\tilde{A} := \tilde{M}\{\tilde{A}/\tilde{X}\}$ represents $A_i := M_i\{A_1/X_1, \cdots, A_n/X_n\}$ for each $i \in \{1, \cdots, n\}$.

Theorem 3.4 *Let $\tilde{M}$ be guarded by Prefixes and contain Variables $\tilde{X}$ at most, and let $\tilde{A} := \tilde{M}\{\tilde{A}/\tilde{X}\}$.*

1. *Let $\tilde{M} \in \mathcal{M}_\nu$. $P \models M_i\{\tilde{A}/\tilde{X}\}$ if and only if $P \models M_i^{\langle n\rangle}\{\mathbf{T}/\tilde{X}\}$ for any n.*
2. *Let $\tilde{M} \in \mathcal{M}_\mu$. $P \models M_i\{\tilde{A}/\tilde{X}\}$ if and only if $P \models M_i^{\langle n\rangle}\{\mathbf{F}/\tilde{X}\}$ for some n.*

where $M_i^{\langle n\rangle}$ is the specification expression defined inductively as follows:

$$M_i^{\langle 0\rangle} \equiv X_i, \qquad M_i^{\langle n+1\rangle} \equiv M_i\{\tilde{M}^{\langle n\rangle}/\tilde{X}\}$$

Proof (outline)

1. The 'only if part' is directly shown by Proposition 3.3(1). For the 'if part', we use another inductive definition $\bigcap_{i\geq 0} \models_{(i)}$ of $\models$, where $\models_{(0)} = \mathcal{P} \times \mathcal{S}$ and $\models_{(n+1)} = \theta(\models_{(n)})$. Then we can show that for any n, if $P \models_{(n)} N\{\tilde{M}^{\langle n\rangle}/\tilde{X}\}\{\mathbf{T}/\tilde{X}\}$, then $P \models_{(n)} N\{\tilde{M}^{\langle n\rangle}/\tilde{X}\}\{\tilde{A}/\tilde{X}\}$, where $N \in \mathcal{M}_\nu$ and N contain Variables $\tilde{X}$ at most. This is not difficult, because N and $\tilde{M}$ have no unstable states. Finally, we can set $N \equiv X_i$, then the result follows.
2. The 'if part' is directly shown by Proposition 3.3(2). For the 'only if part', we show that the following $\mathcal{R}$ is a satisfaction relation.

$$\begin{aligned}\mathcal{R} = \{(P, N\{\tilde{M}^{\langle n\rangle}/\tilde{X}\}\{\mathbf{F}/\tilde{X}\}) : \tilde{A} := \tilde{M}\{\tilde{A}/\tilde{X}\}, (P, N\{\tilde{A}/\tilde{X}\}) \in \theta^{(m)}(\models),\\ n \geq m, N \text{ and } \tilde{M} \text{ are guarded and contain Variables } \tilde{X} \text{ at most},\\ N \in \mathcal{M}_\mu, \tilde{M} \in \mathcal{M}_\mu\} \cup \models\end{aligned}$$

The key points are that (1) $N\{\tilde{M}^{\langle n\rangle}/\tilde{X}\}$ is still guarded after n transitions, (2) $N\{\tilde{A}/\tilde{X}\}$ must reach a stable state after m' transitions for some $m' \leq m$, because $(P, N\{\tilde{A}/\tilde{X}\}) \in \theta^{(m)}(\models)$, and (3) if $N'\{\mathbf{F}/\tilde{X}\}$ is stable and $N' \in \mathcal{M}_\mu$, then N' contains no Variables (i.e. $N'\{\mathbf{F}/\tilde{X}\} \equiv N'\{\tilde{A}/\tilde{X}\}$). □

Since we consider only finite state specifications, Theorem 3.4 shows that if $\tilde{M} \in \mathcal{M}_\nu$ then $\tilde{A}$ is the greatest fixpoint of recursive equations $\tilde{X} = \tilde{M}$, and if $\tilde{M} \in \mathcal{M}_\mu$ then $\tilde{A}$ is the least fixpoint of them. For example, if $M \equiv a; X \vee b; \mathbf{T}$, $A := M\{A/X\}$, and $P \models A$, then P may not perform b (i.e. may infinitely perform a), because $M \in \mathcal{M}_\nu$. On the other hand, if $M \equiv \lhd a; X \vee b; \mathbf{T}$, $A := M\{A/X\}$, and $P \models A$, then P must eventually perform b, because $M \in \mathcal{M}_\mu$.

4 Integration of specifications

A number of flexible specifications are sometimes given to a large system instead of its complete specification, because many designers work on the same system design in parallel, and it is not easy for each designer to know the whole system. Such design method decreases responsibility of each designer, but it raises two important issues: *consistency check* of the flexible specifications and *integration* of them. In general, since the integrated specification satisfies all of them, it corresponds to a *conjunction specification* of them. The consistency and the conjunction specification are defined as follows.

Definition 4.1 *Let $S_i \in \mathcal{S}$. The specifications $S_1, \cdots, S_n$ are* consistent *with each other, if $\bigcap_{1 \leq i \leq n} Proc(S_i) \neq \emptyset$. And the specification S is a* conjunction specification *of $S_1, \cdots, S_n$, if $Proc(S) = \bigcap_{1 \leq i \leq n} Proc(S_i) \neq \emptyset$.* □

In Subsection 4.1, a relation $\sim$ is given for checking the consistency between two specifications. In Subsection 4.2, a method called the $\wedge$-method is given for producing a conjunction specification of two specifications. A conjunction specification of three or more specifications can be produced by iteratively using $\sim$ and the $\wedge$-method.

4.1 Consistency check

In this subsection, we consider the consistency of *two* specifications. At first, a relation $\sim$ is defined as a generalized relation from the satisfaction $\models$.

Definition 4.2 *A relation $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ is a* consistent relation, *if $(S, T) \in \mathcal{R}$ implies $(S, T) \in \Theta(\mathcal{R})$, where $\Theta(\mathcal{R}) \subseteq \mathcal{S} \times \mathcal{S}$ is inductively defined for any relation $\mathcal{R}$, as follows:*

- $(S, T) \in \Theta^{(0)}(\mathcal{R})$ *iff for some S_0 and T_0, $S \mapsto S_0$, $T \mapsto T_0$, $S_0 \in \bigcirc$, $T_0 \in \bigcirc$,*

$$(i)\ \textit{if } S_0 \xrightarrow{\alpha} S' \textit{ then, for some } T',\ T_0 \xrightarrow{\alpha} T' \textit{ and } (S', T') \in \mathcal{R},$$
$$(ii)\ \textit{if } T_0 \xrightarrow{\alpha} T' \textit{ then, for some } S',\ S_0 \xrightarrow{\alpha} S' \textit{ and } (S', T') \in \mathcal{R},$$

- $(S, T) \in \Theta^{(n+1)}(\mathcal{R})$ *iff for some S_0 and T_0, $S \mapsto S_0$, $T \mapsto T_0$,*

$$(i)\ \textit{if } S_0 \xrightarrow{\alpha} S' \textit{ then, for some } (m, T'),\ T_0 \xrightarrow{\alpha} T',\ (S', T') \in \Theta^{(m)}(\mathcal{R}),\ m \leq n,$$
$$(ii)\ \textit{if } T_0 \xrightarrow{\alpha} T' \textit{ then, for some } (m, S'),\ S_0 \xrightarrow{\alpha} S',\ (S', T') \in \Theta^{(m)}(\mathcal{R}),\ m \leq n,$$

- $(S, T) \in \Theta(\mathcal{R})$ *iff $(S, T) \in \Theta^{(n)}(\mathcal{R})$, for some n.* □

Definition 4.3 *$S \sim T$, if $(S, T) \in \mathcal{R}$ for some consistent relation $\mathcal{R}$.* □

The relation $\Theta(\mathcal{R})$ is an extension of $\theta(\mathcal{R})$ in Definition 3.2 to over $\mathcal{S} \times \mathcal{S}$, and we can show that $S \sim T$ if and only if $(S, T) \in \Theta(\sim)$. The relation $S \sim T$ requires that S and T must eventually reach stable states at the same time.

It is important to note that the relation $\sim$ is too strong to check the consistency between two specifications. For example, the following two specifications *SAB* and *SBA* (*SAB* was also used in Section 3) are consistent with each other,

$$SAB := \triangleleft a; SAB \vee \circ b; SAB, \qquad SBA := \circ a; SBA \vee \triangleleft b; SBA$$

because there exist processes P such that $P \models SAB$ and $P \models SBA$, for example $PAB := a; a; b; PAB$. On the other hand, $S \not\sim T$, because *SAB* and *SBA* can not reach stable states *at the same time.*

In this paper, we present a method for checking the consistency after transforming a specification into a *standard form.* At first, the following set is defined in order to define the standard form, where $Dri(S_0) = \{S' : \exists \alpha, S_0 \xrightarrow{\alpha} S'\}$.

Definition 4.4 *Let $\mathcal{U} \subseteq \mathcal{S}$. A set $\mathcal{V} \subseteq \mathcal{S}$ is a* pre-$\mathcal{U}$ set*, if $S \in \mathcal{V}$ implies that,*

$$(pre1)\ \text{if } S \mapsto S_0 \notin \bigcirc, \text{ then } Dri(S_0) \subseteq \mathcal{V},$$
$$(pre2)\ \text{if } S \mapsto S_0 \in \bigcirc, \text{ then } Dri(S_0) \subseteq \mathcal{U}.$$

Then, $Pre(\mathcal{U}) = \bigcup\{\mathcal{V} : \mathcal{V}$ is a pre-$\mathcal{U}$ set$\}$. □

Then, the standard form is defined as follows, where $S \simeq T$ represents $Proc(S) = Proc(T)$.

Definition 4.5 *A set $\mathcal{U} \subseteq \mathcal{S}$ is a* standard set*, if $S \in \mathcal{U}$ implies that,*

(1) *if $S \mapsto S_0 \notin \bigcirc$, then for some $S_0' \in \bigcirc$, $S \mapsto S_0' \simeq S_0$, $Dri(S_0') \subseteq Pre(\mathcal{U})$,*
(2) *if $S \mapsto S_0$, then for some S_0', $S \mapsto S_0' \simeq S_0$, $Dri(S_0') \subseteq \mathcal{U}$,*
(3) *if $S \mapsto S_0$, then either $Dri(S_0) \subseteq \mathcal{U}$ or $Dri(S_0) \subseteq Pre(\mathcal{U})$.*

Then, $STD = \bigcup\{\mathcal{U} : \mathcal{U}$ is a standard set$\}$. □

Definition 4.6 *Let $S \in \mathcal{S}$.*

1. *S is in* standard form*, if $S \in STD$.*
2. *S is in* pre-standard form*, if $S \in Pre(STD)$.* □

As shown in Definition 4.5, if $S \in STD$ then for any S_0 such that $S \mapsto S_0$, for some $S_0' \in \bigcirc$ such that $S \mapsto S_0'$, $Proc(S_0) = Proc(S_0')$. And furthermore, for every derivation S' such that $S_0 \xrightarrow{\alpha} S'$, $S' \in Pre(STD)$. This condition (1) makes it possible to *immediately* reach a stable state if S is in standard form, and thereafter S' must be in pre-standard form. In order to return to be in standard form, S' must eventually reach a stable state. The condition (2) requires that S *can* keep in standard form, and (3) requires that S *must* keep in either standard form or pre-standard form.

Then, a specification $\mathbf{ST}(S)$ produced form S is defined as follows.

Definition 4.7 *Let $S \in \mathcal{S}$. The specification $\mathbf{ST}(S)$ is defined as follows.*

$$\begin{aligned}
\mathbf{ST}(S) &:= \bigvee\{ST_0(S_0) : S \mapsto S_0\} \vee \bigvee\{SS_0(S_0) : S \mapsto S_0 \notin \bigcirc\} \\
\mathbf{PST}(S) &:= \bigvee\{ST_0(S_0) : S \mapsto S_0 \in \bigcirc\} \vee \bigvee\{PST_0(S_0) : S \mapsto S_0 \notin \bigcirc\} \\
ST_0(S_0) &\equiv \sum\{\psi\alpha; \mathbf{ST}(S') : S_0 \xrightarrow{\alpha} S', St(S_0) = \psi\} \\
PST_0(S_0) &\equiv \sum\{\psi\alpha; \mathbf{PST}(S') : S_0 \xrightarrow{\alpha} S', St(S_0) = \psi\} \\
SS_0(S_0) &\equiv \sum\{\circ\alpha; \mathbf{PST}(S') : S_0 \xrightarrow{\alpha} S'\}
\end{aligned}$$

where $St : \mathcal{S}_0 \to \Psi$ is a state function defined as : if $S_0 \in \bigcirc$ then $St(S_0) = \circ$, otherwise $St(S_0) = \lhd$. □

The specifications $\mathbf{ST}(S)$ and $\mathbf{PST}(S)$ are Constants. Since we consider only specifications S with finite states, the number of states of $\mathbf{ST}(S)$ is also finite. The key point is that $\mathbf{ST}(S)$ contains a stable state $SS_0(S_0)$ if $S \mapsto S_0 \notin \bigcirc$. It is important to note that the derivation of $SS_0(S_0)$ is $\mathbf{PST}(S')$ instead of $\mathbf{ST}(S')$. In order to return $\mathbf{ST}$, S' must eventually reach a stable state (This is similar to the requirement of STD).

Proposition 4.1 shows that the set of processes which satisfy S is not changed by the transformation $\mathbf{ST}$. And Proposition 4.2 shows that $\mathbf{ST}(S)$ is in standard form for any S. Therefore, for any specification S, we can transform S into a standard form S' such that $S \simeq S'$ by $\mathbf{ST}$.

Proposition 4.1 *Let $S \in \mathcal{S}$ and $S_0 \in \mathcal{S}_0$. Then*

$$\begin{aligned} S &\simeq \mathbf{ST}(S) \simeq \mathbf{PST}(S), \\ S_0 &\simeq ST_0(S_0) \simeq PST_0(S_0) \simeq SS_0(S_0) \end{aligned}$$

Proof (outline) For $S \simeq \mathbf{ST}(S) \simeq \mathbf{PST}(S)$, we can show that the following $\mathcal{R}_{1,2}$ are satisfaction relations.

$$\begin{aligned} \mathcal{R}_1 &= \{(P, \mathbf{ST}(S)) \ : \ P \models S\} \cup \{(P, \mathbf{PST}(S)) \ : \ P \models S\} \\ \mathcal{R}_2 &= \{(P, S) \ : \ P \models \mathbf{ST}(S)\} \cup \{(P, S) \ : \ P \models \mathbf{PST}(S)\} \end{aligned}$$

The relation $S_0 \simeq ST_0(S_0) \simeq PST_0(S_0) \simeq SS_0(S_0)$ can be shown by similar satisfaction relations. □

Proposition 4.2 *Let $S \in \mathcal{S}$. Then $\mathbf{ST}(S) \in STD$ and $\mathbf{PST}(S) \in Pre(STD)$.*
Proof (outline) We can show that the following $\mathcal{V}$ and $\mathcal{U}$ are a pre-$\mathcal{U}$ set and a standard set, respectively : $\mathcal{V} = \{\mathbf{PST}(S) : S \in \mathcal{S}\}$ and $\mathcal{U} = \{\mathbf{ST}(S) : S \in \mathcal{S}\}$. □

The above examples *SAB* and *SBA* are used, again. The specification *SAB* is transformed by **ST** into the following specification:

$$\begin{aligned} \mathbf{ST}(SAB) &:= \triangleleft a; \mathbf{ST}(SAB) \vee \circ b; \mathbf{ST}(SAB) \vee \circ a; \mathbf{PST}(SAB) \\ \mathbf{PST}(SAB) &:= \triangleleft a; \mathbf{PST}(SAB) \vee \circ b; \mathbf{ST}(SAB) \end{aligned}$$

The specification $\mathbf{ST}(SBA)$ is symmetrical with $\mathbf{ST}(SAB)$ for a and b. The state $(\circ a; \mathbf{PST}(SAB))$ is important, thus $\mathbf{ST}(SAB)$ contains a stable state which can perform the action a. This implies that $\mathbf{ST}(SAB)$ and $\mathbf{ST}(SBA)$ can reach stable states at the same time. In fact, we can prove that $\mathbf{ST}(SAB) \sim \mathbf{ST}(SBA)$.

Now, the relation $\sim$ can be used for checking the consistency of two specifications as shown in Proposition 4.3. The relation $\sim$ can be automatically checked by a similar algorithm to one for bisimilarity [6].

Proposition 4.3 *Let $S, T \in STD$. Then $S \sim T$ iff $Proc(S) \cap Proc(T) \neq \emptyset$.*
Proof ('if' part) We show that the following $\mathcal{R}$ is a consistent relation.

$$\mathcal{R} = \{(S, T) : P \models S, P \models T, S \in STD \cup Pre(STD), T \in STD \cup Pre(STD)\}$$

Let $P \models S$, $P \models T$, $S \in STD$, and $T \in STD$. Since $P \models S$, there exists S_0 such that $S \mapsto S_0$ and $P \models S_0$. Here, by Definition 4.5, for some $S_0' \in \bigcirc$, $S \mapsto S_0'$ and $P \models S_0'$. Similarly, for some $T_0' \in \bigcirc$, $T \mapsto T_0'$ and $P \models T_0'$.

For (i), let $S_0' \xrightarrow{\alpha} S'$. Since $P \models S_0' \in \mathcal{S}_0$, for some P', $P \xrightarrow{\alpha} P'$ and $P' \models S'$. Furthermore, since $P \models T_0'$, for some T', $T_0' \xrightarrow{\alpha} T'$ and $P' \models T'$. Here, by Definition 4.5, $S' \in STD \cup Pre(STD)$ and $T' \in STD \cup Pre(STD)$. Thus, $(S', T') \in \mathcal{R}$. For (ii), it is symmetrical. Consequently, $(S, T) \in \Theta^{(0)}(\mathcal{R})$.

For the other cases such that $S \in Pre(STD)$ and $T \in STD$, S can reach either a state $S' \in STD$ or a stop, because $P \models S$ (i.e. S must reach a stable state). Hence, these cases can be shown by induction on n of $(P, S) \in \Theta^{(n)}(\models)$.

('only if' part) Assume that $S \sim T$. By the definition of $\sim$, there exist S_0 and T_0 such that $S_0 \sim T_0$, $S \mapsto S_0$, and $T \mapsto T_0$. Then, it can be proven that the following process $\mathbf{CP}^{(n)}(S_0, T_0)$ satisfies both S_0 and T_0, where $n = |S_0, T_0| = \min\{n : (S_0, T_0) \in \Theta^{(n)}(\sim)\}$. The detail is omitted.

$$\begin{aligned}\mathbf{CP}^{(n)}(S_0, T_0) := \textstyle\sum\{\circ\alpha; \mathbf{CP}^{(n')}(S_0', T_0') : \exists (S', T'), |S_0', T_0'| = n', n = 0,&\\ S_0 \xrightarrow{\alpha} S' \mapsto S_0', T_0 \xrightarrow{\alpha} T' \mapsto T_0', S_0' \sim T_0'\} \,[]&\\ \textstyle\sum\{\circ\alpha; \mathbf{CP}^{(n')}(S_0', T_0') : \exists (S', T'), |S_0', T_0'| = n' \leq n-1,&\\ S_0 \xrightarrow{\alpha} S' \mapsto S_0', T_0 \xrightarrow{\alpha} T' \mapsto T_0', S_0' \sim T_0'\}&\end{aligned} \qquad \square$$

By Proposition 4.3, the above relation $\mathbf{ST}(SAB) \sim \mathbf{ST}(SBA)$ implies that $\mathbf{ST}(SAB)$ and $\mathbf{ST}(SBA)$ have common processes. Furthermore, this implies SAB and SBA have common processes by Proposition 4.1, thus they are consistent.

Proposition 4.3 shows a method to produce a common process $\mathbf{CP}^{(n)}(S_0, T_0)$ of two specifications S and T (i.e. $\mathbf{CP}^{(n)}(S_0, T_0) \in \mathit{Proc}(S) \cap \mathit{Proc}(T)$). We can also use $\mathbf{CP}^{(n)}(S_0, S_0)$ for producing an executable process P from a specification S such that $P \models S$, where $S \mapsto S_0$.

In the rest of this section, we give a relation $\cong$ which implies $\simeq$.

Definition 4.8 *A relation $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ is a* full consistent relation*, if $(S, T) \in \mathcal{R}$ implies that the following conditions* (1) *and* (2) *hold:*

(1) *for all S_0 such that $S \mapsto S_0$, for some T_0, $T \mapsto T_0$, and (i), (ii), (iii) hold,*
(2) *for all T_0 such that $T \mapsto T_0$, for some S_0, $S \mapsto S_0$, and (i), (ii), (iii) hold,*
(i) *for all α and S', if $S_0 \xrightarrow{\alpha} S'$ then, for some T', $T_0 \xrightarrow{\alpha} T'$, and $(S', T') \in \mathcal{R}$,*
(ii) *for all α and T', if $T_0 \xrightarrow{\alpha} T'$ then, for some S', $S_0 \xrightarrow{\alpha} S'$, and $(S', T') \in \mathcal{R}$,*
(iii) *either $S_0 \not\longrightarrow$ or $S_0 \in \bigcirc$ if and only if either $T_0 \not\longrightarrow$ or $T_0 \in \bigcirc$* $\square$

Definition 4.9 *S and T are* fully consistent*, written $S \cong T$, if $(S, T) \in \mathcal{R}$ for some full consistent relation $\mathcal{R}$.* $\square$

The condition (iii) requires that S_0 is stable if and only if T_0 is stable. For example, $(S_1 \equiv \triangleleft a; S \,[]\, a; S)$ and $(S_2 \equiv \triangleleft a; S)$ are fully consistent, because both S_1 and S_2 are unstable.

The full consistency is an equivalence relation. And the full consistency implies that two specifications have the same processes as follows.

Proposition 4.4 *Let $S, T \in \mathcal{S}$. If $S \cong T$,then $S \simeq T$ (i.e. $\mathit{Proc}(S) = \mathit{Proc}(T)$).*
Proof It can be shown that the relation $\{(P, T) : \exists S, P \models S, S \cong T\}$ is a satisfaction relation. This proof is not difficult. $\square$

The opposite direction of Proposition 4.4 (i.e. if $S \simeq T$, then $S \cong T$) does not always hold. For example, $A_1 := a; A_1$ and $A_2 := \triangleleft a; a; A_2$ have the same processes, but $A_1 \not\cong A_2$. The relation $\cong$ is a simple sufficient condition for $\simeq$.

4.2 Conjunction specification

In this subsection, we present a method for producing a conjunction specification from two specifications. The pair of this method and the relation $\sim$ allows to check the consistency of three or more specifications, and to produce a conjunction specification of them.

The key idea for producing conjunction specifications is the standard form defined in Definition 4.6. If two specifications are not in standard form, then *eventualities* of them will be confused with each other. Another important point of our method is that *non-determinism* of Choices $[]$ and Disjunctions $\vee$ is considered. By this non-determinism, each state in a specification can consist with a number of various states in another specification. By considering the non-determinism, we present the $\wedge$-*method* which produces a specification constant $S \wedge T$ from two specifications S and T. Since we consider only finite state specifications S and T, the number of states of $S \wedge T$ is also finite.

Definition 4.10 *Let $S, T \in \mathcal{S}$. The specification $S \wedge T$ is defined as follows.*

$$\begin{aligned} S \wedge T &:= \bigvee\{S_0 \triangle T_0 : S \mapsto S_0, T \mapsto T_0, S_0 \sim T_0\} \\ S_0 \triangle T_0 &\equiv \textstyle\sum\{\psi\alpha; \bigvee\{S' \wedge T' : T_0 \xrightarrow{\alpha} T', S' \sim T'\} : S_0 \xrightarrow{\alpha} S', St(S_0) = \psi\} \,[]\, \\ &\quad \textstyle\sum\{\psi\alpha; \bigvee\{S' \wedge T' : S_0 \xrightarrow{\alpha} S', S' \sim T'\} : T_0 \xrightarrow{\alpha} T', St(T_0) = \psi\} \quad \square \end{aligned}$$

The specification $S \wedge T$ performs common actions of S and T, like $S\,|[G]|\,T$. The main difference from $S\,|[G]|\,T$ is that $S \wedge T$ keeps the relation $\sim$ between S and T. For example, compare the following $S_{ab} \wedge S_{ac}$ and $S_{ab}\,|[a,b,c]|\,S_{ac}$.

$$S_{ab} \wedge S_{ac} \cong a;\mathbf{stop} \quad , \qquad S_{ab}\,|[a,b,c]|\,S_{ac} \cong a;\mathbf{stop} \vee \mathbf{stop}$$

where $S_{ab} \equiv a;\mathbf{stop} \vee b;\mathbf{stop}$ and $S_{ac} \equiv a;\mathbf{stop} \vee c;\mathbf{stop}$. $S_{ab} \wedge S_{ac}$ is rightly the common specification $a;\mathbf{stop}$ of S_{ab} and S_{ac}. On the other hand, $S_{ab}\,|[a,b,c]|\,S_{ac}$ contains also the specification $\mathbf{stop}$. The $\mathbf{stop}$ arises from non-determinism of Disjunctions $\vee$, for example, by unlabelled transitions $S_{ab} \mapsto b;\mathbf{stop}$ and $S_{ac} \mapsto c;\mathbf{stop}$, where $b;\mathbf{stop}\,|[a,b,c]|\,c;\mathbf{stop} \not\longrightarrow$. The condition $S_0 \sim T_0$ in Definition 4.10 avoids mismatches by non-determinism, such as $b;\mathbf{stop}$ and $c;\mathbf{stop}$.

Furthermore, by non-determinism of Choices $[]$, there may exist two or more specifications T'_i such that $T_0 \xrightarrow{\alpha} T'_i$, $S' \sim T'_i$, and $T'_i \not\sim T'_j$, for each $S_0 \xrightarrow{\alpha} S'$. For example, if $S_0 \equiv a;(b;\mathbf{stop} \vee c;\mathbf{stop})$ and $T_0 \equiv a;b;\mathbf{stop}\,[]\,a;c;\mathbf{stop}$, then $S_0 \xrightarrow{a} S' \equiv b;\mathbf{stop} \vee c;\mathbf{stop}$, $T_0 \xrightarrow{a} T'_1 \equiv b;\mathbf{stop}$, $T_0 \xrightarrow{a} T'_2 \equiv c;\mathbf{stop}$, $S' \sim T'_1$, $S' \sim T'_2$, and $T'_1 \not\sim T'_2$. In this case, a consistent pair such as (S', T'_i) can not be uniquely decided. Therefore, all the consistent pairs are flexibly combined by Disjunctions, as shown in the part $\bigvee\{S' \wedge T' \mid T_0 \xrightarrow{\alpha} T', S' \sim T'\}$ in Definition 4.10.

Then, we give an expected proposition for the $\wedge$-method.

Proposition 4.5 *Let $S, T \in STD$. If $S \sim T$, then $S \wedge T$ is a conjunction specification of the specifications S and T, thus $Proc(S \wedge T) = Proc(S) \cap Proc(T)$.*
Proof We can show that the following $\mathcal{R}_1$ and $\mathcal{R}_2$ are satisfaction relations.

$$\begin{aligned} \mathcal{R}_1 &= \{(P,S) : \exists (S_0,T_0), S \mapsto S_0, S_0 \sim T_0, P \models S_0 \triangle T_0\} \\ \mathcal{R}_2 &= \{(P,U) : \exists (S_0,T_0), U \mapsto S_0 \triangle T_0, P \models S_0, P \models T_0, \\ &\qquad (Dri(S_0), Dri(T_0) \subseteq \text{either } STD \text{ or } Pre(STD))\} \end{aligned}$$

For $\mathcal{R}_2$, the key point is that S_0 and T_0 can reach stable states at the same time, because all the derivations of them are in (pre-)standard form. This means that $S_0 \triangle T_0$ can also reach a stable state. The detail is omitted. $\square$

The two specifications $\mathbf{ST}(\mathit{SAB})$ and $\mathbf{ST}(\mathit{SBA})$ in Subsection 4.1 are used, again. By the $\wedge$-method, the following specifications are produced.

$$\begin{aligned}
C_1 &\equiv \mathbf{ST}(\mathit{SAB}) \wedge \mathbf{ST}(\mathit{SBA}) := \triangleleft a; C_1 \vee \triangleleft b; C_1 \vee \circ a; C_2 \vee \circ b; C_3 \\
C_2 &\equiv \mathbf{PST}(\mathit{SAB}) \wedge \mathbf{ST}(\mathit{SBA}) := \triangleleft b; C_1 \vee \triangleleft a; C_2 \vee \circ b; C_3 \\
C_3 &\equiv \mathbf{ST}(\mathit{SAB}) \wedge \mathbf{PST}(\mathit{SBA}) := \triangleleft a; C_1 \vee \circ a; C_2 \vee \triangleleft b; C_3
\end{aligned}$$

Then, By Proposition 4.1 and Proposition 4.5,

$$\mathit{Proc}(C_1) = \mathit{Proc}(\mathbf{ST}(\mathit{SAB})) \cap \mathit{Proc}(\mathbf{ST}(\mathit{SBA})) = \mathit{Proc}(\mathit{SAB}) \cap \mathit{Proc}(\mathit{SBA}).$$

In order to reach a stable state from C_1, either $(\circ a; C_2)$ or $(\circ b; C_3)$ must be eventually selected. If $(\circ a; C_2)$ is selected, then C_2 requires that b must be eventually performed. Thus, C_1 requires that a and b must be always eventually performed.

Finally, we give a theorem to produce a conjunction specification of three or more specifications by iteratively using $\sim$ and the $\wedge$-method (and the transformation by $\mathbf{ST}(S)$). This theorem also shows how to check their consistency.

Theorem 4.6 *Let $T_1, T_2 \in \mathit{STD}$, T_1 be a conjunction specification of $S_1, \cdots, S_m$, and T_2 be a conjunction specification of $S_{m+1}, \cdots, S_n$.*

(1) *If $T_1 \sim T_2$, then $T_1 \wedge T_2$ is a conjunction specification of $S_1, \cdots, S_n$.*

(2) *If $T_1 \not\sim T_2$, then $S_1, \cdots, S_n$ are not consistent.*

Proof This is easily proven by Proposition 4.3, Proposition 4.5, and the properties of intersection of sets. $\Box$

5 Conclusion and related work

In this paper, we have considered how to introduce least fixpoint and conjunction of μ-calculus[1, 7, 15] into LOTOS. In order to express the least fixpoint in a Labelled Transition System, we have proposed an extended LTS called μLTS, and have defined a language μLOTOS based on the μLTS. Then, the $\wedge$-method has been presented for producing a conjunction specification. In general, the conjunction specification S is not executable, because it may contain Disjunctions, but an executable process can be produced from S by **CP** in Proposition 4.3.

As a related work on flexible specifications, Larsen presented Modal Specifications to express loose specifications by required transitions $\longrightarrow_{\Box}$ and allowed transitions $\longrightarrow_{\Diamond}$ in [8] and a language called *modal CCS* based on the transitions in [9, 10]. The difference between modal CCS and μLOTOS is explained by the following specifications S_1 in modal CCS and S_2 in μLOTOS.

$$S_1 := a_{\Diamond}; S_1 \,[\,]\, b_{\Box}; \mathbf{stop}, \qquad S_2 := \triangleleft a; S_2 \vee b; \mathbf{stop},$$

where $a_{\Diamond}$ represents an allowed action and $b_{\Box}$ represents a required action (LOTOS syntax is used also for S_1). The following process P_1 satisfies both S_1 and S_2, while the process P_2 satisfies only S_1, because the action a must not be infinitely performed in S_2, and the process P_3 satisfies only S_2, because the action b can not be postponed in S_1.

$$P_1 := b; \mathbf{stop}, \quad P_2 := a; P_2 \,[\,]\, b; \mathbf{stop}, \quad P_3 := a; b; \mathbf{stop}$$

The basic idea of the μLTS arose from the notion of divergence [16] (p.148 in [12]) which can avoid infinite loop by internal actions in the notion of divergence. An unstable state in μLTS is intuitively considered as a state which can perform internal actions. But internal actions are needed for expressing dynamic behavior such as timeout, and they should not be used for controlling resolution of Disjunction operators. Therefore, we have introduced an un-stabilizer $\lhd$.

For integration or refinement of specifications, a number of approaches were proposed, for example [2] [13] [10]. Brinksma[2] proposed a refined parallel operator for multiple labels. This operator is used to implement conjunction of LOTOS specifications. Steen et al.[13] proposed a conjunction operator $\otimes$ and a join operator $\bowtie$ in order to yield a common reduction and a common extension, respectively, in LOTOS. Larsen et al.[10] defined a conjunction operator $\wedge$ for loose specifications in modal CCS. However, these approaches do not consider the non-determinism of Disjunction operators $\vee$. Therefore, they can not be directly applied to μLOTOS.

For logical requirements, synthesis algorithms of processes were proposed in [7] and [11]. Kimura et al.[7] presented a synthesis algorithm for recursive processes by subcalculus of μ-calculus, but the subcalculus does not contain the disjunction $\vee$. Manna et al.[11] presented an algorithm for synthesizing a graph from requirements described in Propositional Temporal Logic (PTL). In PTL, eventualities can be expressed by an operator $\diamond$, but the synthesized graph from PTLs does *not* always represent *all* the common processes of them.

References

1. R.Barbuti : Selective mu-calculus: New Modal Operators for Proving Properties on Reduced Transition Systems, Formal Description Techniques and Protocol Specification, Testing and Verification, FORTE X/PSTV XVII, pp.519-534, 1997.
2. E.Brinksma : Constraint-oriented specification in a constructive formal description technique, LNCS 430, Springer-Verlag, pp.130-152, 1989.
3. P.J.B.Glabbeek and W.P.Weijland: Branching Time and Abstraction in Bisimulation Semantics, *Journal of the ACM*, Vol.43, No.3, pp.555-600, 1996.
4. C.A.R.Hoare :*Communicating Sequential Processes*, Prentice-Hall, 1985.
5. Y.Isobe, H.Nakada, Y.Sato, and K.Ohmaki : Stepwise Synthesis of Multi-Specifications using Static Choice Actions (in Japanese). Foundation of Software Engineering IV (FOSE'97), Lecture Notes 19, Kindaikagaku-sha, pp.12-19, 1997.
6. P.C.Kanellakis and S.A.Smolka: CCS Expressions, Finite State Processes, and Three Problems of Equivalence, *Information and Computation*, Vol.86, pp.43-68, 1990.
7. S.Kimura, A.Togashi and N.Shiratori : Synthesis Algorithm for Recursive Processes by μ-calculus, Algorithmic Learning Theory, LNCS 872,Springer-Verlag, pp.379-394, 1994.
8. K.G.Larsen : Modal Specifications, Automatic Verification Methods for Finite State Systems, LNCS 407, Springer-Verlag, pp.232-246, 1989.
9. K.G.Larsen : The expressive Power of Implicit Specifications, Automata, Languages and Programming, LNCS 510, Springer-Verlag, pp.204-216, 1991.
10. K.G.Larsen, B.Steffen, and C.Weise : A Constraint Oriented Proof Methodology Based on Modal Transition Systems, Tools and Algorithms for the Construction and Analysis of Systems, LNCS 1019, Springer-Verlag, pp.17-40, 1995.

11. Z.Manna and P.Wolper : Synthesis of Communicating Processes from Temporal Logic Specifications, *ACM Trans. on Programming Languages and Systems*, Vol.6, No.1, pp.67-93, 1984.
12. R.Milner : *Communication and Concurrency*, Prentice-Hall, 1989.
13. M.W.A.Steen, H.Bowman, and J.Derrick : Composition of LOTOS specification, Protocol Specification, Testing and Verification, XV, pp.73-88, 1995
14. M.W.A.Steen, H.Bowman, J.Derrick, and E.A.Boiten, : Disjunction of LOTOS specification, Formal Description Techniques and Protocol Specification, Testing and Verification, FORTE X/PSTV XVII, pp.177-192, 1997.
15. Colin Stirling : An Introduction to Modal and Temporal Logics for CCS, Concurrency: Theory, Language, and Architecture, LNCS 491, Springer-Verlag, pp.2-20, 1989.
16. D.J.Walker : Bisimulations and Divergence, Proc. of third annual IEEE symposium on Logic in Computer Science, pp.186 - 192, 1988.
17. ISO 8807: Information Processing Systems–Open System Interconnection–LOTOS–A formal description technique based on the temporal ordering of observational behavior, 1989.

A Appendix

Table 1. The notations used in this paper

Notation	Meaning
$\mathcal{N}$	the set of names $a, b, \cdots$
Act	the set of actions $\alpha, \beta, \cdots$, $(Act = \mathcal{N} \cup \{i\})$
Ψ	the set of state operators $\psi, \phi, \cdots$, $(\Psi = \{\circ, \triangleleft\})$
$\mathcal{K}$	the set of specification constants (Constants) $A, B, \cdots$
$\mathcal{X}$	the set of specification variables (Variables) $X, Y, \cdots$
$\mathcal{M}$	the set of specification expressions $M, N, \cdots$
$\mathcal{M}_0$	the set of specification expressions such that if $M_0 \mapsto M_0'$ then $M_0 \equiv M_0'$
$\mathcal{M}_\nu, \mathcal{M}_\mu$	subsets of $\mathcal{M}$, Definition 3.3 and 3.4
$\mathcal{S}$	the set of specifications $S, T, U, \cdots$
$\mathcal{S}_0$	the set of specifications such that if $S_0 \mapsto S_0'$ then $S_0 \equiv S_0'$
$\mathcal{P}$	the set of processes $P, Q, \cdots$
STD	the set of specifications in standard form, Definition 4.6
$Pre(STD)$	the set of specifications in pre-standard form
$Var(M)$	the set of Variables in M
$Proc(S)$	the set of all the processes to satisfy S, $(Proc(S) = \{P : P \models S\})$
$Dri(S_0)$	the set of derivations of S_0 $(Dri(S_0) = \{S' : \exists\alpha, S_0 \xrightarrow{\alpha} S'\})$
$\models$	satisfaction
$\sim$	a relation for checking consistency
$\simeq$	the relation $\{(S,T) : Proc(S) = Proc(T)\}$
$\cong$	full consistency, Definition 4.9
$:=$	definition of specification constants
$\equiv$	syntactic identity
$\theta(\mathcal{R})$	a relation to define $\models$, Definition 3.1
$\Theta(\mathcal{R})$	a relation to define $\sim$, Definition 4.2
St	a state function (if $S_0 \in \bigcirc$ then $St(S_0) = \circ$, otherwise $St(S_0) = \triangleleft$
$\mathbf{ST}(S)$	a specification in standard form, transformed from S