Failure-Divergence Semantics and Refinement of Long Running Transactions

Zhenbang Chen [a], Zhiming Liu [b], Ji Wang [a]

[a] *National Laboratory for Parallel and Distributed Processing, National University of Defense Technology, Changsha, China*

[b] *UNU-IIST, P.O. Box 3058, Macao SAR, China*

Abstract

Compensating CSP (cCSP) models long-running transactions. It can be used to specify service orchestrations written in programming languages like WS-BPEL. However, the original cCSP does not allow to model internal (non-deterministic) choice, synchronized parallel composition, hiding or recursion. In this paper, we introduce these operators and define for the extended language a failure-divergence (FD) semantics to allow reasoning about non-determinism, deadlock and livelock. Furthermore, we develop a refinement calculus that allows us to compare the level of non-determinism between long running transactions, and transform specifications for design and analysis.

Key words: Compensation, deadlock, non-determinism, failure-divergence model, livelock, refinement, recursion

1 Introduction

Long-Running Transactions (LRTs) have attracted research attention, because they are important in Service-Oriented Computing (SOC) [25]. A long-running transaction (LRT) in SOC lasts for "a long period of time", and involves interactions between different organizations. The *ACID properties* (Atomicity, Consistency, Isolation, and Durability) of *atomic transactions* are too restrictive on such transactions. Isolation, for instance, is often unnecessary and unrealistic for LRTs [25]. Instead, a LRT employs *compensation mechanisms* to recover from failures to ensure a relaxed atomicity and consistency as dictated by the application.

⋆ This is a combination and extension of the work published at ICTAC 2010 [12] and FM 2011 [13].

Industrial service composition languages, such as WS-BPEL [1] and XLANG [37], are designed and implemented for programming LRTs in service orchestrations. For the specification and verification of LRTs, some formalisms are proposed, such as StAC [7], SAGAs calculi [6] and cCSP [9]. These formalisms are used to define formal semantics for industrial languages and provide a foundation for understanding LRTs so that techniques and tool support for verification and analysis can be developed on a sound basis.

The starting point of our work is cCSP [9] of Butler *et al.* It extends Communicating Sequential Processes (CSP) [34] with the mechanisms for interruption and recovery from exceptions. The recovery mechanism in cCSP is the same as that of the backward recovery proposed in Sagas [15]. There are two types of processes in cCSP, *standard processes* and *compensable processes*. The standard processes are a subset of CSP processes extended with exception handling and transaction block (see Section 2). A compensable process specifies the recovery when an exception occurs. A trace semantics is given in [7] and an operational semantics in [10]. The consistency between these two semantic models is studied in [33]. However, without non-deterministic (internal) choice, synchronized parallel composition and hiding, cCSP does not provide enough support for compositional verification and design of LRTs by refinement and decomposition. Abstraction (via hiding) is a main source of non-determinism; and synchronization and non-determinism can cause deadlocks when composing processes. These, plus recursion for the specification of repetitive behavior, are actually the general technical challenges in defining a denotational theory of concurrent and distributed modeling notations.

In this paper, we extend the cCSP in [9] with *non-deterministic choices, synchronization among parallel processes, hiding* and *recursion*. The main contribution is a semantic theory of *failures* and *divergences* of LRTs. The theory includes a *complete partial order* (CPO) of the failure-divergences of processes that allows the calculation of a unique fixed-point of any recursive compensable process. The CPO also characterizes the laws of programming LRTs (*cf.* Section 4) as its partial order agrees with the *refinement relation* between cCSP processes. It is a well-known challenge to establish, or even to show the existence of a fixed-point theory for a language like CSP with internal-choice and synchronization [35]. Also, the literature, *e.g.* [11], shows that if internal choice is added to CCS [31], it is difficult to define a partial order to characterize the notion of refinement. This problem is even harder for the extended cCSP, due to the mechanisms for exception handling and compensation. The technical details in Sections 3 and 4 and the proofs of the theorems and laws show the complexity of the matter. Because of the application potential of cCSP, the extension with a well established failure-divergence semantic theory are particularly important. Theoretically, similar to the unification role that the failure-divergence semantics of CSP plays, the failure-divergence semantic theory integrates as its sub-theories the operational semantics, the

trace semantics, and the stable failures semantics [12] of cCSP. It completes the semantic theory of cCSP for specification of LRTs and can be used to underpin the extension to the model checker FDR of CSP [34] and the cCSP theorem proving tool [33] for the verification of LRTs. In addition, as a by product of our work, a few laws claimed in [7] actually do not hold under the original trace semantics (see Section 2.4). We fix these problems in the extended cCSP under our failure-divergence semantics.

This paper extends the colloquium paper [12]. In particular, the stable failure semantics in [12] is extended to a failure-divergence semantics for the extended cCSP and two refinement relations for standard and compensable processes, respectively. The refinement relation on compensable processes also makes the failure-divergence domain form a fixed-point theory for compensable processes. Additionally, a new case study is presented to demonstrate recursion and speculative choice. This paper also extends the symposium paper [13] with a coherent study of algebraic laws and the consistency relation to the trace semantics and operational semantics of cCSP. The proofs of the theorems, the major algebraic laws and the case study are also presented for the first time here. These together justify the correctness of the failure-divergence semantics proposed in this paper.

The rest of this paper is organized as follows. Section 2 introduces the syntax and the semantics of the original cCSP. In Section 3, the syntax of the original cCSP is extended, and the failure-divergence semantics is given. Section 4 presents the refinement theory and the algebraic laws for both standard processes and compensable processes. Section 5 uses a case study to demonstrate the use of the calculus in analysis of deadlock, divergence and compensation behavior. Section 6 reviews related work and draws conclusions. We provide the proofs of some laws in the appendices.

2 Original Compensating CSP

We recall the syntax of the original cCSP in [9], where P and PP represent a standard process and a compensable process, respectively.

$$\begin{aligned} P &::= a \mid P; P \mid P \Box P \mid P \| P \mid P \rhd P \mid [PP] \mid \mathbf{skip} \mid \mathbf{throw} \mid \mathbf{yield} \\ PP &::= P \div P \mid PP; PP \mid PP \Box PP \mid PP \| PP \mid PP \boxtimes PP \mid \mathbf{skipp} \mid \\ &\quad \mathbf{throww} \mid \mathbf{yieldd} \end{aligned}$$

Process a denotes the process that terminates successfully after performing the event a. There are three operators on both standard processes and compensable processes: sequential composition ($;$), choice ($\Box$) and parallel composition ($\|$). Process **skip** immediately terminates successfully; **throw** indicates the

occurrence of an exception and the process is interrupted; while **yield** can terminate successfully or yield to an exception from the environment resulting in an interruption. $P \rhd Q$ behaves like P unless an exception is thrown, then it behaves as Q. For example, the process $(a; \mathbf{throw}; c) \rhd b$ will perform the event a first, and then perform b because an exception occurs after a (caused by **throw**). $[PP]$ is a transaction block specifying a long-running transaction, in which a compensable process PP is defined to specify the transaction. For a long-running transaction, if there occurs an exception, the compensations of the already successfully terminated actions will be executed in a specific order for recovery.

A compensable process is constructed from *compensation pairs* of the form $P \div Q$, where the execution of Q will compensate the effects of executing P. Corresponding to the basic standard processes **skip**, **throw** and **yield**, we use **skipp**, **throww** and **yieldd** to denote the basic compensable processes. Compensable process **skipp** immediately terminates successfully without further need of compensation; **throww** throws an exception and **yieldd** either terminates successfully or yields to an exception. For example, for a transaction block process $[a \div b \ ; \ \mathbf{throww}]$, the compensation pair $a \div b$ is first executed, *i.e.* a is executed and b is recorded, then **throww** is executed to cause an exception, so b will be executed to compensate the effects of the already successfully executed process a. $PP \boxtimes QQ$ is the speculative choice between two compensable processes, in which the two processes will run in parallel first until one of them succeeds, then the other one will be compensated. We use $\mathcal{P}$ and $\mathcal{PP}$ to denote the set of standard processes and the set of compensable processes, respectively.

2.1 Basic notations

At any moment of time, a process performs either a *terminal event* or a *normal interaction event*. There are three terminal events $\Omega = \{\checkmark, !, ?\}$. Event $\checkmark$ is called the *success event* and indicates that the process terminates successfully. Event ! is called the *exception event* and indicates that the trace terminates with an exception. And event ? is called the *yield event* and indicates that the execution terminates by yielding to an exception from the environment.

Let Σ be the set of all the *normal events* that all processes can perform, called the *alphabet*, Σ^* the set of the finite traces over Σ, and Σ^ω the set of the inifinite traces over Σ. We denote a trace by a sequence of events, and the empty trace by ε. We define

- $\Gamma = \Sigma \cup \Omega$ to represent all the possible events,
- $s{\cdot}t$ to represent the *concatenation* of the traces s and t,

- $T_1 \cdot T_2 = \{s_1 \cdot s_2 \mid s_1 \in T_1 \wedge s_2 \in T_2\}$ for the concatenation of two sets of traces,
- $\Sigma_O^\bigstar = \{s \cdot \omega \mid s \in \Sigma^* \wedge \omega \in O\}$ and $\Sigma_O^\circledast = \Sigma^* \cup \Sigma_O^\bigstar$, where $O \subseteq \Omega$,

We use $\Sigma^\bigstar$ and $\Sigma^\circledast$ as shorthands for $\Sigma_\Omega^\bigstar$ and $\Sigma_\Omega^\circledast$. A trace in $\Sigma^\bigstar$ is called a *terminated trace* and a trace in $\Sigma_{\{\checkmark\}}^\bigstar$ a *successfully terminated trace*. For a normal event a, we use, depending on its context, a to represent the event, the process that performs a and then successfully terminates, and the trace of the single event a. The notations used in this paper are listed in Appendix D.

Synchronization. Processes need to follow a number of rules to synchronize on different terminal events. We order the three terminals such that $! \prec ? \prec \checkmark$ and define $\omega_1 \& \omega_2 = \omega_1$ if $\omega_1 \prec \omega_2$ or $\omega_1 = \omega_2$. Therefore, the synchronization of any terminal with an exception will result in an exception, and a composition terminates successfully iff both parties do. The synchronization of two traces in $\Sigma^\circledast$ on an event set $X \subseteq \Sigma$ is defined as follows, where $s, t \in \Sigma^*$, $x, x' \in X$, $y, y' \in \Sigma \setminus X$, and $\omega, \omega_1, \omega_2 \in \Omega$.

$$
\begin{aligned}
& s \underset{X}{\parallel} t = t \underset{X}{\parallel} s \ , \quad \varepsilon \underset{X}{\parallel} \varepsilon = \{\varepsilon\} \ , \quad \varepsilon \underset{X}{\parallel} x = \{\} \ , \ \varepsilon \underset{X}{\parallel} y = \{y\} \\
& x \cdot s \underset{X}{\parallel} y \cdot t = \{y \cdot u \mid u \in x \cdot s \underset{X}{\parallel} t\} \\
& x \cdot s \underset{X}{\parallel} x \cdot t = \{x \cdot u \mid u \in s \underset{X}{\parallel} t\} \\
& x \cdot s \underset{X}{\parallel} x' \cdot t = \{\} \quad if \ x \neq x' \\
& y \cdot s \underset{X}{\parallel} y' \cdot t = \{y \cdot u \mid u \in s \underset{X}{\parallel} y' \cdot t\} \cup \{y' \cdot u \mid u \in y \cdot s \underset{X}{\parallel} t\} \\
& s \cdot \omega \underset{X}{\parallel} t = \{\} \ , \quad t \underset{X}{\parallel} s \cdot \omega = \{\} \\
& s \cdot \omega_1 \underset{X}{\parallel} t \cdot \omega_2 = \{u \cdot \omega_1 \& \omega_2 \mid u \in s \underset{X}{\parallel} t\}
\end{aligned}
$$

We use $\parallel$ as shorthand for $\underset{\{\}}{\parallel}$, that is the parallel composition *without* synchronizing on any normal event.

2.2 Trace semantics

The failure-divergence semantics builds on the trace semantics. Therefore we introduce the simpler trace semantics here, and later justify the failure-divergence semantics by its consistency with the trace semantics and the operational semantics introduced in the next subsection.

The *trace semantic function* $\mathcal{T} : \mathcal{P} \to \mathbb{P}(\Sigma^\bigstar)$ assigns each process P a set $\mathcal{T}(P)$ of terminated traces. Figure 1 shows the definition of $\mathcal{T}$, where p and q are terminated traces.

Atomic process For all $a \in \Sigma$, $\mathcal{T}(a) = \{a\checkmark\}$

Sequential composition

$$p\ ;\ q = \begin{cases} p_1 \cdot q & p = p_1 \cdot \checkmark \\ p & p = p_1 \cdot \omega \wedge \omega \neq \checkmark \end{cases}, \mathcal{T}(P\ ;\ Q) = \{p\ ;\ q \mid p \in \mathcal{T}(P) \wedge q \in \mathcal{T}(Q)\}$$

Choice $\mathcal{T}(P \Box Q) = \mathcal{T}(P) \cup \mathcal{T}(Q)$

Parallel composition $\mathcal{T}(P \parallel Q) = \{r \mid r \in (p \parallel q) \wedge p \in \mathcal{T}(P) \wedge q \in \mathcal{T}(Q)\}$

Exception handling

$$p \rhd q = \begin{cases} p_1 \cdot q & p = p_1 \cdot ! \\ p & p = p_1 \cdot \omega \wedge \omega \neq ! \end{cases}, \mathcal{T}(P \rhd Q) = \{p \rhd q \mid p \in \mathcal{T}(P) \wedge q \in \mathcal{T}(Q)\}$$

Basic processes
$\mathcal{T}(\mathbf{skip}) = \{\checkmark\}$, $\mathcal{T}(\mathbf{throw}) = \{!\}$, $\mathcal{T}(\mathbf{yield}) = \{?, \checkmark\}$

Fig. 1. The trace semantics of standard processes

In contrast to the CSP convention [34], the trace set of a standard process in cCSP is not prefix closed. The processes in a parallel composition synchronize only on the terminal events, while performing other events in an interleaving manner. An exception occurs in the composition if any sub-process throws an exception, and the composition terminates successfully only if both sub-processes do. The semantics of **throw** means an exception happens. The semantics of $a \parallel \mathbf{throw}$ is $\{a!\}$ and it means a global exception is thrown after a is performed. The semantics of **yield** is $\{\checkmark, ?\}$, meaning that the process either terminates successfully or is interrupted. The semantics of $(a; \mathbf{yield}; b) \parallel \mathbf{throw}$ is $\{a!, ab!\}$ and shows that the process $a; \mathbf{yield}; b$ can be interrupted or continue to perform b after executing a. Notice that the semantics of $(a; b) \parallel \mathbf{throw}$ is $\{ab!\}$. Therefore, in cCSP, **yield** can be used to declare the places where a process can be interrupted. Even when there is no external exception, a process may terminate when reaching **yield**, *e.g.* $\mathcal{T}(a; \mathbf{yield}; b) = \{a?, ab\checkmark\}$.

A compensable process is defined by a set of pairs of traces, called the *forward trace* and *compensation trace*, respectively. The semantics of the sequential composition conforms to the semantics of the classical Sagas [15], and compensation actions are executed in the reverse order of their corresponding forward actions. For example, the forward behavior of $a_1 \div b_1\ ;\ a_2 \div b_2$ will perform a_1 followed by a_2, but the compensation behavior will perform b_1 after b_2 if an exception occurs later. Figure 2 defines the *trace semantic function* $\mathcal{T}_c : \mathcal{PP} \to \mathbb{P}(\Sigma^\star \times \Sigma^\star)$ of compensable processes.

In general a compensable process PP may yield to an interruption from the environment before it performs any event. Thus, the compensation trace set

Compensation pair

$$p \div q = \begin{cases} (p, q) & p = p_1 \cdot \checkmark \\ (p, \checkmark) & p = p_1 \cdot \omega \wedge \omega \neq \checkmark \end{cases}$$

$$\mathcal{T}_c(P \div Q) = \{p \div q \mid p \in \mathcal{T}(P) \wedge q \in \mathcal{T}(Q)\} \cup \{(?, \checkmark)\}$$

Compensable sequential composition

$$(p, p') \ ; \ (q, q') = \begin{cases} (p_1 \cdot q, q' \ ; \ p') & p = p_1 \cdot \checkmark \\ (p, p') & p = p_1 \cdot \omega \wedge \omega \neq \checkmark \end{cases}$$

$$\mathcal{T}_c(PP \ ; \ QQ) = \{(p, p') \ ; \ (q, q') \mid (p, p') \in \mathcal{T}_c(PP) \wedge (q, q') \in \mathcal{T}_c(QQ)\}$$

Compensable choice $\mathcal{T}_c(PP \Box QQ) = \mathcal{T}_c(PP) \cup \mathcal{T}_c(QQ)$

Compensable parallel composition

$$(p, p') \parallel (q, q') = \{(r, r') \mid r \in (p \parallel q) \wedge r' \in (p' \parallel q')\}$$

$$\mathcal{T}_c(PP \parallel QQ) = \{rr \mid rr \in (pp \parallel qq) \wedge pp \in \mathcal{T}_c(PP) \wedge qq \in \mathcal{T}_c(QQ)\}$$

Compensable speculative composition

$$(p \cdot \omega_1, p') \boxtimes (q \cdot \omega_2, q') = \begin{cases} \{(r \ ; \ q', p') \mid r \in (p \parallel q)\} & \omega_1 = \checkmark \wedge \omega_2 \neq \checkmark \\ \{(r \ ; \ p', q') \mid r \in (p \parallel q)\} & \omega_1 \neq \checkmark \wedge \omega_2 = \checkmark \\ \{(r \ ; \ q', p'), (r \ ; \ p', q') \mid r \in (p \parallel q)\} & \omega_1 = \checkmark \wedge \omega_2 = \checkmark \\ \{(r \ ; \ r', \checkmark) \mid r \in (p \parallel q) \wedge r' \in (p' \parallel q')\} & \omega_1 \neq \checkmark \wedge \omega_2 \neq \checkmark \end{cases}$$

$$\mathcal{T}_c(PP \boxtimes QQ) = \{rr \mid rr \in (pp \boxtimes qq) \wedge pp \in \mathcal{T}_c(PP) \wedge qq \in \mathcal{T}_c(QQ)\}$$

Basic compensable processes

skipp = **skip** $\div$ **skip**, **throww** = **throw** $\div$ **skip**, **yieldd** = **yield** $\div$ **skip**

Fig. 2. The semantics of compensable process

of $P \div Q$ contains the trace pair $(?, \checkmark)$ (see Figure 2). The semantics of a transaction block $[PP]$ is defined below.

$$\mathcal{T}([PP]) = \{p \cdot p' \mid (p \cdot !, p') \in \mathcal{T}_c(PP)\} \cup \{p \cdot \checkmark \mid (p \cdot \checkmark, p') \in \mathcal{T}_c(PP)\}$$

It says that after an exception occurs, the compensation trace will be executed to recover from the failure. Otherwise, the compensation trace is not executed.

2.3 Operational semantics

An operational semantics probably provides the most intuitive understanding of a process behavior, and it is also used to justify a denotational semantics, such as the trace semantics and the failure-divergence semantics. The operational semantics of cCSP [10] is defined in the same way as the operational

semantics of CSP [34]. In the following, we focus on the transition rules that are closely related to exception handling, interruption and compensation. In particular, we have the following rules for exception and interruption.

A standard process P, if it is not the null process 0, will perform an event to evolve to another process P'. Except for the case when P only performs a terminal event and P' is the null process 0, P' can continue to evolve.

$$\mathbf{throw} \xrightarrow{!} 0 \qquad \frac{P \xrightarrow{\alpha} P'}{P \triangleright Q \xrightarrow{\alpha} P' \triangleright Q}\ (\alpha \in \Sigma) \qquad \frac{P \xrightarrow{!} 0 \wedge Q \xrightarrow{\alpha} Q'}{P \triangleright Q \xrightarrow{\alpha} Q'}\ (\alpha \in \Gamma)$$

A compensable process PP also evolves by performing events, but to a standard process for compensation. The transition rules of compensation pairs are:

$$\frac{P \xrightarrow{\alpha} P'}{P \div Q \xrightarrow{\alpha} P' \div Q}\ (\alpha \in \Sigma) \qquad \frac{P \xrightarrow{\checkmark} 0}{P \div Q \xrightarrow{\checkmark} Q} \qquad \frac{P \xrightarrow{\omega} 0}{P \div Q \xrightarrow{\omega} \mathbf{skip}}\ (\omega \in \{?, !\})$$

During the evolving steps, the compensation processes of the subprocesses of PP are recorded for possible later compensation of PP. This is characterized by the following transition rules of sequential composition.

$$\frac{PP \xrightarrow{\alpha} PP'}{PP\ ;\ QQ \xrightarrow{\alpha} PP'\ ;\ QQ}\ (\alpha \in \Sigma) \qquad \frac{PP \xrightarrow{\omega} P}{PP\ ;\ QQ \xrightarrow{\omega} P}\ (\omega \in \Omega \wedge \omega \neq \checkmark)$$

$$\frac{PP \xrightarrow{\checkmark} P \wedge QQ \xrightarrow{\alpha} QQ'}{PP\ ;\ QQ \xrightarrow{\alpha} \langle QQ', P\rangle}\ (\alpha \in \Sigma) \qquad \frac{PP \xrightarrow{\checkmark} P \wedge QQ \xrightarrow{\omega} Q}{PP\ ;\ QQ \xrightarrow{\omega} Q\ ;\ P}\ (\omega \in \Omega)$$

$$\frac{QQ \xrightarrow{\alpha} QQ'}{\langle QQ, P\rangle \xrightarrow{\alpha} \langle QQ', P\rangle}\ (\alpha \in \Sigma) \qquad \frac{QQ \xrightarrow{\omega} Q}{\langle QQ, P\rangle \xrightarrow{\omega} Q\ ;\ P}\ (\omega \in \Omega)$$

Thus, PP evolves to its compensation process P which is recorded in $(\langle QQ', P\rangle)$ when QQ evolves. After QQ evolves to its compensation process Q, the compensation process for $PP\ ;\ QQ$ is the process $Q\ ;\ P$.

A transaction block defines when and how compensation is initiated. It happens after the compensable process in the transaction block evolves to a standard process; then the last performed terminal event determines whether the compensation process is executed. The following transition rules define the compensation mechanism.

$$\frac{PP \xrightarrow{\alpha} PP'}{[PP] \xrightarrow{\alpha} [PP']}\ (\alpha \in \Sigma) \qquad \frac{PP \xrightarrow{\checkmark} P}{[PP] \xrightarrow{\checkmark} P} \qquad \frac{PP \xrightarrow{!} P \wedge P \xrightarrow{\alpha} P'}{[PP] \xrightarrow{\alpha} P'}\ (\alpha \in \Gamma)$$

The following sequence of transitions shows how the above transition rules

together define the behavior of a transaction block.

$$[a_1 \div b_1 \ ; \ \mathbf{throww}] \xrightarrow{a_1} [\mathbf{skip} \div b_1 \ ; \ \mathbf{throww}] \xrightarrow{b_1} \mathbf{skip} \xrightarrow{\checkmark} 0$$

2.4 Discussion

When we study the relation between the failure-divergence semantics and the trace semantics, we find the following three algebraic laws that are claimed to hold in [9] are actually not valid with respect to the trace semantics.

(1) "PP;**skipp**=PP". Consider PP to be $a \div b$. According to the definition in Figure 2, the semantics of the process $a \div b$ is $\{(?, \checkmark), (a\checkmark, b\checkmark)\}$, but the semantics of $a \div b$; **skipp** is $\{(?, \checkmark), (a?, b\checkmark), (a\checkmark, b\checkmark)\}$.

(2) "$[P \div Q] = P$ provided P does not terminate with the yield terminal event ?". This equation does not hold if we let P be **throw** and Q be **skip**. $[\mathbf{throw} \div \mathbf{skip}] = \mathbf{skip}$ according to the semantics, and obviously it is not equal to **throw**.

(3) "$[P \div Q \ ; \ \mathbf{throww}] = P \ ; \ Q$ provided P does not terminate with the yield terminal event ?". This does not hold if we take P to be **throw** and Q to be **skip**. Then we have

$$[\mathbf{throw} \div \mathbf{skip} \ ; \ \mathbf{throww}] = \mathbf{skip} \neq (\mathbf{throw} \ ; \ \mathbf{skip}) = \mathbf{throw}$$

The first law should be valid, and it holds indeed under the operational semantics. This shows that the trace semantics needs to be adjusted. The problem is the extra trace pair $(?, \checkmark)$ added to a compensation pair (*cf.* Figure 2). The study of consistency between the trace semantics and the operational semantics in [32] fixed this problem by removing the extra trace pair $(?, \checkmark)$ from the compensation pair. In Section 3.3, we will show that the adjusted trace semantics can be derived from the failure-divergence semantics. Furthermore, the last two laws should not hold. This is shown by the counter-example processes above where the transaction block removes the exception terminal event ! of the forward trace.

3 Extended cCSP and its Failure-Divergence Semantics

Let Σ be a *finite* set of normal events, denoting the alphabet of the cCSP processes. The syntax of the extended cCSP is defined in Figure 3, where $a \in \Sigma$, $X \subseteq \Sigma$ is an event set and $\rho \subseteq \Sigma \times \Sigma$ is a renaming relation.

$P \sqcap Q$ and $P \Box Q$ represent internal and external choices, respectively. In the generalized parallel composition $P \underset{X}{\parallel} Q$, processes P and Q synchronize on the

$$
\begin{aligned}
P \quad ::= \quad & a \mid P; P \mid P \sqcap P \mid P \Box P \mid P \underset{X}{\parallel} P \mid P \setminus X \mid P[\![\rho]\!] \mid P \rhd P \mid [PP] \mid \\
& \mathbf{skip} \mid \mathbf{throw} \mid \mathbf{yield} \mid \mathbf{stop} \mid \mu\ p.\ F(p) \\
PP \quad ::= \quad & P \div P \mid PP; PP \mid PP \sqcap PP \mid PP \Box PP \mid PP \underset{X}{\parallel} PP \mid PP \boxtimes PP \mid \\
& PP \setminus X \mid PP[\![\rho]\!] \mid \mathbf{skipp} \mid \mathbf{throww} \mid \mathbf{yieldd} \mid \mu\ pp.\ FF(pp)
\end{aligned}
$$

Fig. 3. The syntax of the extended cCSP

events in X, as well as on the terminal events in Ω. When X is empty, the generalized parallel composition is written as $P \parallel Q$ and it agrees with the parallel composition of the original cCSP presented in Section 2. $P \setminus X$ is the process that internalizes the events (the environment cannot observe these events) in X during the execution of P, and $P[\![\rho]\!]$ the process obtained from P by renaming its events according to the renaming relation ρ. The process $\mu\ p.\ F(p)$ is a standard recursive process. Because unbounded nested long-running transactions are not realistic in practice, we do not allow a recursion variable appearing in a transaction block, *e.g.* $\mu\ p.\ [p \div a]$ is not allowed.

We extend the compensable processes with the same operators. The internal and external choices are made during the execution of the forward behaviors of the sub-processes. Both the forward and the compensation behaviors of the two sub-processes PP and QQ in $PP \underset{X}{\parallel} QQ$ synchronize on the events in X. $PP \setminus X$ hides the events in X appearing in the forward and compensation behaviors of PP, $PP[\![\rho]\!]$ renames the events in the forward and compensation behaviors of PP according to the renaming relation ρ, and $\mu\ pp.\ FF(pp)$ is a recursive compensable process.

3.1 Semantics of standard processes

We define the FD semantics of a standard process with the technique by which the failure set and the divergence set of a classical CSP process are defined in [34]. However, there are technical details specific to the treatment of the additional exception and yield terminal events and synchronization on terminal events.

The FD semantics $[\![P]\!]$ of a process P is a pair $(\mathcal{F}(P), \mathcal{D}(P))$, where

- $\mathcal{F}(P) \subseteq \Sigma^{\circledast} \times \mathbb{P}(\Gamma)$ is the *failure set*. A failure (s, X) in $\mathcal{F}(P)$ is a pair of a trace s and a refusal set X. It says that the process P refuses to perform any event in X after executing s.
- $\mathcal{D}(P) \subseteq \Sigma^{\circledast}$ is the *divergence set*. A divergence s in $\mathcal{D}(P)$ is an execution trace of P after which P enters a *chaos* state in which any event can be executed or refused.

The sets of *traces* and *terminated traces* of P are defined from the failures $\mathcal{F}(P)$ below.

$$traces_\perp(P) \mathrel{\widehat{=}} \{s \mid (s,\{\}) \in \mathcal{F}(P)\},\ trace_t(P) \mathrel{\widehat{=}} traces_\perp(P) \cap \Sigma^\bigstar$$

We use $trace_n(P) \mathrel{\widehat{=}} traces_\perp(P) \setminus \mathcal{D}(P)$ to denote the set of non-divergent traces of P. We require that the FD semantics of a standard process P satisfies the following axioms.

$$traces_\perp(P) \text{ is non-empty and prefix closed} \tag{1}$$
$$(s,X) \in \mathcal{F}(P) \wedge Y \subseteq X \Rightarrow (s,Y) \in \mathcal{F}(P) \tag{2}$$
$$(s,X) \in \mathcal{F}(P) \wedge \forall a \in Y \bullet s{\cdot}a \notin traces_\perp(P) \Rightarrow (s, X \cup Y) \in \mathcal{F}(P) \tag{3}$$
$$s{\cdot}\omega \in traces_\perp(P) \Rightarrow (s, \Gamma \setminus \{\omega\}) \in \mathcal{F}(P),\ \text{where } \omega \in \Omega \tag{4}$$
$$s \in \mathcal{D}(P) \cap \Sigma^* \wedge t \in \Sigma^{\circledast} \Rightarrow s{\cdot}t \in \mathcal{D}(P) \tag{5}$$
$$s \in D \Rightarrow (s,X) \in \mathcal{F}(P),\ \text{where } X \subseteq \Gamma \tag{6}$$
$$s{\cdot}\omega \in \mathcal{D}(P) \Rightarrow s \in \mathcal{D}(P),\ \text{where } \omega \in \Omega \tag{7}$$

The set $traces_\perp(P)$ is prefix closed (Axiom 1) because we need to model the refusal behavior of a process after it performs each event, and $\mathcal{D}(P)$ is suffix closed (Axiom 6) because a process can perform any event after entering a divergent state. These axioms are basically those of the FD semantics of classical CSP given in [34]. Axioms 4 and 7 show the differences of the semantics from the FD semantics of classical CSP. The difference is due to the additional terminal events.

Two standard processes are equivalent if and only if their FD semantic models are equal. We define the *failure function* $\mathcal{F} : \mathcal{P} \to \mathbb{P}(\Sigma^{\circledast} \times \mathbb{P}(\Gamma))$ and the *divergence function* $\mathcal{D} : \mathcal{P} \to \mathbb{P}(\Sigma^{\circledast})$ for the set $\mathcal{P}$ of all standard processes.

Atomic and basic processes The divergence sets of the atomic and basic processes a, **skip**, **stop**, **throw** and **yield** are all empty, and their failure sets are defined below.

$$\begin{aligned}
\mathcal{F}(a) &= \{(\varepsilon,X) \mid X \subseteq \Gamma \wedge a \notin X\} \cup \{(a,X) \mid X \subseteq \Gamma \wedge \checkmark \notin X\} \\
&\quad \cup \{(a\checkmark,X) \mid X \subseteq \Gamma\} \\
\mathcal{F}(\mathbf{skip}) &= \{(\varepsilon,X) \mid X \subseteq \Gamma \wedge \checkmark \notin X\} \cup \{(\checkmark,X) \mid X \subseteq \Gamma\} \\
\mathcal{F}(\mathbf{stop}) &= \{(\varepsilon,X) \mid X \subseteq \Gamma\} \\
\mathcal{F}(\mathbf{throw}) &= \{(\varepsilon,X) \mid X \subseteq \Gamma \wedge\ !\notin X\} \cup \{(!,X) \mid X \subseteq \Gamma\} \\
\mathcal{F}(\mathbf{yield}) &= \{(\varepsilon,X) \mid X \subseteq \Gamma \wedge\ ?\notin X\} \cup \{(?,X) \mid X \subseteq \Gamma\} \\
&\quad \cup \{(\varepsilon,X) \mid X \subseteq \Gamma \wedge \checkmark \notin X\} \cup \{(\checkmark,X) \mid X \subseteq \Gamma\}
\end{aligned}$$

In what follows, we use **div** to represent the process diverging immediately, *i.e.* $\varepsilon \in \mathcal{D}(\mathbf{div})$, and **interp** to represent the process whose failure set and diver-

gence set are $\{(\varepsilon, X) \mid X \subseteq \Gamma \wedge\ ? \notin X\} \cup \{(?, X) \mid X \subseteq \Gamma\}$ and $\{\}$, respectively.

Choices The semantics of the internal choice is the same as defined in CSP, but note that $\mathbf{yield} \sqcap \mathbf{skip} = \mathbf{yield}$ holds. External choice is different from internal choice on the empty trace ε, where $P \Box Q$ can refuse an event only if both P and Q can refuse it. Also, care should be taken about the terminal events "?" and "!" when defining the failures to ensure the axiom (4).

$$
\begin{aligned}
\mathcal{D}(P \sqcap Q) &= \quad \mathcal{D}(P) \cup \mathcal{D}(Q) \qquad \mathcal{F}(P \sqcap Q) = \mathcal{F}(P) \cup \mathcal{F}(Q) \\
\mathcal{D}(P \Box Q) &= \quad \mathcal{D}(P) \cup \mathcal{D}(Q) \\
\mathcal{F}(P \Box Q) &= \quad \{(\varepsilon, X) \mid (\varepsilon, X) \in \mathcal{F}(P) \cap \mathcal{F}(Q)\} \\
&\quad \cup \{(s, X) \mid (s, X) \in \mathcal{F}(P) \cup \mathcal{F}(Q) \wedge s \neq \varepsilon\} \\
&\quad \cup \{(\varepsilon, X) \mid X \subseteq \Gamma \setminus \{\omega\} \wedge \omega \in \mathit{traces}_{\perp}(P) \cup \mathit{traces}_{\perp}(Q) \wedge \omega \in \Omega\} \\
&\quad \cup \{(s, X) \mid s \in \mathcal{D}(P \Box Q) \wedge X \subseteq \Gamma\}
\end{aligned}
$$

Sequential composition Sequential composition is different from the classic CSP [34] because of the terminal events "!" and "?".

$$
\begin{aligned}
\mathcal{D}(P\ ;\ Q) &= \quad \mathcal{D}(P) \cup \{s{\cdot}t \mid s{\cdot}\checkmark \in \mathit{traces}_{\perp}(P) \wedge t \in \mathcal{D}(Q)\} \\
\mathcal{F}(P\ ;\ Q) &= \quad \{(s, X) \mid s \in \Sigma^{\circledast}_{\{?,!\}} \wedge (s, X \cup \{\checkmark\}) \in \mathcal{F}(P)\} \\
&\quad \cup \{(s{\cdot}t, X) \mid s{\cdot}\checkmark \in \mathit{traces}_{\perp}(P) \wedge (t, X) \in \mathcal{F}(Q)\} \\
&\quad \cup \{(s, X) \mid s \in \mathcal{D}(P\ ;\ Q) \wedge X \subseteq \Gamma\}
\end{aligned}
$$

Parallel composition We first define the divergence set of $P \underset{X}{\parallel} Q$, and then its failure set. The composition diverges if either P or Q diverges, which is

$$
\begin{aligned}
\mathcal{D}(P \underset{X}{\parallel} Q) = \{u{\cdot}v \mid v \in \Sigma^{\circledast}, \exists s \in \mathit{traces}_{\perp}(P), t \in \mathit{traces}_{\perp}(Q) \bullet \\
u \in (s \underset{X}{\parallel} t) \cap \Sigma^{*} \wedge (s \in \mathcal{D}(P) \vee t \in \mathcal{D}(Q))\}
\end{aligned} \tag{8}
$$

To define the failure set of the composition, we understand that $P \underset{X}{\parallel} Q$ can refuse an event in $X \cup \Omega$ if either P or Q can, and it can refuse an event outside $X \cup \Omega$ only if both P and Q can refuse it. For a failure (s, Y) of P and a failure (t, Z) of Q, recall the classical definition in CSP of the synchronized failure set:

$$
(s, Y) \underset{X}{\parallel} (t, Z) = \{(u, Y \cup Z) \mid Y \setminus (X \cup \Omega) = Z \setminus (X \cup \Omega) \wedge u \in s \underset{X}{\parallel} t\} \tag{9}
$$

We need to adjust this definition for cCSP to take into account the following two different cases of synchronization on terminals.

(1) If P or Q cannot perform a terminal after executing s or t, the composition cannot terminate. In this case Equation (9) applies. For exam-

ple, let $\Sigma = \{a, b\}$, P be the process a and Q the process $b; \mathbf{throw}$. As $(\varepsilon, \{b, \checkmark, !, ?\})$ is a failure of P and $(b, \{b, \checkmark, ?\})$ a failure of Q, $P \| Q$ has the failure $(b, \{b, \checkmark, !, ?\})$. This is reflected in the first case of Equation (10).

(2) If both P and Q can terminate, the synchronized terminal, represented by Θ in Equation (10), should be excluded from the refusal set. For example, let $\Sigma = \{a\}$, P be the process a and Q the process $a; \mathbf{throw}$. As $(a, \{a, !, ?\})$ is a failure of P and $(a, \{a, \checkmark, ?\})$ a failure of Q, P can perform $\checkmark$ and Q can perform ! to terminate, respectively. Their synchronization result is !, which does not appear in the refusal set $(a, \{a, \checkmark, ?\})$ of $P \underset{\{a\}}{\|} Q$. If Equation (9) is applied, the refusal set would be $(a, \{a, \checkmark, ?, !\})$, which indicates that $P \underset{\{a\}}{\|} Q$ would deadlock after executing a.

The synchronized failure set of two failures is therefore defined as

$$(s, Y) \underset{X}{\|} (t, Z) = \begin{cases} \{(u, Y \cup Z) \mid Y \setminus (X \cup \Omega) = Z \setminus (X \cup \Omega) \wedge u \in s \underset{X}{\|} t\} \\ \qquad \mathbf{if}\ (s, Y \cup \Omega) \in \mathcal{F}(P) \vee (t, Z \cup \Omega) \in \mathcal{F}(Q) \\ \{(u, (Y \cup Z) \setminus \Theta(\omega_1, \omega_2)) \mid Y \setminus (X \cup \Omega) = Z \setminus (X \cup \Omega) \wedge \\ \qquad u \in s \underset{X}{\|} t\} \quad \mathbf{otherwise} \end{cases} \tag{10}$$

Notice two variables ω_1 and ω_2 used in Equation (10). For the failure (s, Y) of P, ω_1 is the terminal event that P *must* perform after s, which is when the following condition holds

$$\forall (s, Y_1) \in \mathcal{F}(P) \bullet Y \subseteq Y_1 \Rightarrow (\omega_1 \in \Omega \wedge \omega_1 \notin Y_1) \tag{11}$$

The value of ω_1 is not defined, denoted by $\perp$, if there is no terminal event that P must perform after s. The value of ω_2 is determined in the same way for the failure (t, Z) of Q. The function Θ synchronizes ω_1 and ω_2 as follows.

$$\Theta(\omega_1, \omega_2) = \Theta(\omega_2, \omega_1) = \begin{cases} \{\omega_1 \& \omega_2\} & \omega_1 \in \Omega \wedge \omega_2 \in \Omega \\ \{\omega_1\} & \omega_1 \in \Omega \wedge \omega_2 = \perp \\ \{\} & \omega_1 = \perp \wedge \omega_2 = \perp \end{cases}$$

Let P be the process $\mathbf{skip} \sqcap \mathbf{throw}$ for example. P has the failures $(\varepsilon, \{\checkmark, ?\})$ and $(\varepsilon, \{?, !\})$. There is no ω_1 satisfying Equation (11) for the failure $(\varepsilon, \{?\})$ of P.

Now the failure set of $P \underset{X}{\parallel} Q$ is

$$\begin{aligned}\mathcal{F}(P \underset{X}{\parallel} Q) = \quad & \{(u, E) \mid \exists (s, Y) \in \mathcal{F}(P), (t, Z) \in \mathcal{F}(Q) \bullet (u, E) \in (s, Y) \underset{X}{\parallel} (t, Z)\} \\ & \cup \{(u, Y) \mid u \in \mathcal{D}(P \underset{X}{\parallel} Q) \wedge Y \subseteq \Gamma\}\end{aligned}$$

Consider $a \underset{\{a\}}{\parallel} (a; \mathbf{throw})$ and let $\Sigma = \{a\}$. Its divergence set is $\{\}$, and its refusal set is $\{(\varepsilon, X) | X \subseteq \Omega\} \cup \{(a, X) | X \subseteq \{a, \checkmark, ?\}\} \cup \{(a!, X) \mid X \subseteq \Gamma\}$.

Parallel composition is *commutative*, *associative* and *distributive* over internal choice.

Exception handling $P \rhd Q$ behaves similarly to $P; Q$, but Q starts to execute only after an exception is thrown in P.

$$\begin{aligned}\mathcal{D}(P \rhd Q) = \quad & \mathcal{D}(P) \cup \{s \cdot t \mid s \cdot ! \in \mathit{traces}_\perp(P) \wedge t \in \mathcal{D}(Q)\} \\ \mathcal{F}(P \rhd Q) = \quad & \{(s, X) \mid s \in \Sigma^{\circledast}_{\{\checkmark, ?\}} \wedge (s, X \cup \{!\}) \in \mathcal{F}(P)\} \\ & \cup \{(s \cdot t, X) \mid s \cdot ! \in \mathit{traces}_\perp(P) \wedge (t, X) \in \mathcal{F}(Q)\} \\ & \cup \{(s, X) \mid s \in \mathcal{D}(P \rhd Q) \wedge X \subseteq \Gamma\}\end{aligned}$$

Exception handling is *associative* and *distributive* over internal choices to both left and right sides of $\rhd$.

Hiding $P \setminus X$ behaves like P except that the events in X are hidden from interacting with the environment. The semantics is defined as follows, in which $X \subseteq \Sigma$, $s \setminus X$ denote the trace after removing all occurrences of each event in X from s, and $t < s$ says that the trace t is not equal to the trace s but a proper prefix of s.

$$\begin{aligned}\mathcal{D}(P \setminus X) = \quad & \{(s \setminus X) \cdot u \mid s \in \mathcal{D}(P) \wedge u \in \Sigma^{\circledast}\} \\ & \cup \{(s \setminus X) \cdot u \mid s \in \Sigma^{\omega} \wedge (s \setminus X) \text{ is finite } \wedge \forall t < s \bullet t \in \mathit{traces}_\perp(P) \\ & \qquad \wedge u \in \Sigma^{\circledast}\} \\ \mathcal{F}(P \setminus X) = \quad & \{(s \setminus X, Y) \mid (s, Y \cup X) \in \mathcal{F}(P)\} \\ & \cup \{(s, Y) \mid s \in \mathcal{D}(P \setminus X) \wedge Y \subseteq \Gamma\}\end{aligned}$$

The hiding operator is *distributive* over internal choice but *not* external choice. For example, the process $((a; b) \,\Box\, (a; c)) \setminus \{a\}$ is equal to $b \sqcap c$.

Renaming The behavior of $P[\![\rho]\!]$ is the behavior after renaming the events in the behavior of P with the renaming relation ρ. Due to the nature of a relation, for a renaming relation ρ and an event a, there may exist *multiple* events satisfying ρ with a. In the following definitions, the renaming relation

ρ is extended by mapping any terminal event to itself, and also by applying it to traces. In addition, the inverse relation of ρ on an event set is defined as usual $\rho^{-1}(X) = \{a \mid \exists a' \bullet \rho(a, a') \wedge a' \in X\}$.

$$\begin{aligned}
\mathcal{D}(P[\![\rho]\!]) = \quad & \{s'{\cdot}u \mid \exists s \in \mathcal{D}(P) \cap \Sigma^* \bullet s \ \rho \ s' \wedge u \in \Sigma^{\circledast}\} \\
\mathcal{F}(P[\![\rho]\!]) = \quad & \{(s', X) \mid \exists s \bullet s \ \rho \ s' \wedge (s, \rho^{-1}(X)) \in \mathcal{F}(P)\} \\
& \cup \{(s, X) \mid s \in \mathcal{D}(P[\![\rho]\!]) \wedge X \subseteq \Gamma\}
\end{aligned}$$

The renaming operator is *distributive* over both internal and external choices.

3.2 Semantics of compensable processes

The semantics, $[\![PP]\!]$, of a compensable process PP consists of its forward behavior and compensation behavior. Thus, it is defined as a tuple (F, D, F^c, D^c) of four sets. (F, D) are the *forward failures* and *forward divergences* and both together are called the *forward FD sets*. $F^c \subseteq \Sigma^{\bigstar} \times \Sigma^{\circledast} \times \mathbb{P}(\Gamma)$ and $D^c \subseteq \Sigma^{\bigstar} \times \Sigma^{\circledast}$ are called the *compensation failures* and *compensation divergences* of PP, respectively. The pair (F^c, D^c) is called the *compensation FD sets* of PP.

The forward FD sets satisfy the axioms of the semantics of the standard processes given in Section 3.1. A compensation failure (s, s_1, X) and a compensation divergence (s, s_1) record a standard failure and a divergence of the compensation behavior for the forward execution trace s, respectively. We define the set of *the forward terminated traces* $trace_f(PP)$=$\{s \mid (s, s_1, X) \in F^c\}$ in F^c (also denoted by $trace_f(F^c)$), and the set $trace_n(PP) = trace_f(PP) \backslash D$ of the *non-divergent* forward terminated traces. The compensation behavior (F^c, D^c) of PP is required to satisfy the following axioms.

$$trace_f(F^c) = \{s \mid (s, s_1) \in D^c\}, \quad trace_f(F^c) \subseteq \Sigma^{\bigstar} \cap \{s \mid (s, \{\}) \in F\} \tag{12}$$

For an s in $trace_f(F^c)$, let $(F^c, D^c)\lfloor s$=$(\{(s_1, X) \mid (s, s_1, X) \in F^c\}, \{s_1 \mid (s, s_1) \in D^c\})$. It is a FD pair and required to satisfy the axioms of standard processes defined in Section 3.1. For the semantics (F, D, F^c, D^c) of a PP, let $fp(PP)$ denote the *forward process behavior* (F, D), and $cp(PP, s)$ the *compensation behavior* $(F^c, D^c)\lfloor s$ for s. We will overload the semantic functions $\mathcal{F}$ and $\mathcal{D}$ and the process operators of standard processes and apply them to $fp(PP)$ and $cp(PP, s)$. For example, $\mathcal{F}(fp(PP)) = F$ and $\mathcal{D}(fp(PP)) = D$.

We define equivalence between two compensable processes as equality between their semantic models. We are now able to define the semantic function $[\![\cdot]\!]$ on the set $\mathcal{PP}$ of all the compensable processes in terms of four semantic functions $(\mathcal{F}_f, \mathcal{D}_f, \mathcal{F}_c, \mathcal{D}_c)$ by induction on the syntactic structure of a process PP.

- the *forward failure* (FF) function $\mathcal{F}_f : \mathcal{PP} \to \mathbb{P}(\Sigma^{\circledast} \times \mathbb{P}(\Gamma))$,
- the *forward divergence* (DF) function $\mathcal{D}_f : \mathcal{PP} \to \mathbb{P}(\Sigma^{\circledast})$,
- the *compensation failure* (FC) function $\mathcal{F}_c : \mathcal{PP} \to \mathbb{P}(\Sigma^{\bigstar} \times \Sigma^{\circledast} \times \mathbb{P}(\Gamma))$, and
- the *compensation divergence* (DC) function $\mathcal{D}_c : \mathcal{PP} \to \mathbb{P}(\Sigma^{\bigstar} \times \Sigma^{\circledast})$.

Compensation pair $P \div Q$. If the forward behavior specified by P terminates successfully, the compensation behavior specified by Q is recorded so that it can be executed to compensate the effect of P if triggered by a later exception. The successfully terminated forward behavior of $P \div Q$ defined by the traces in $\mathit{trace}_t(P) \cap \Sigma^{\bigstar}_{\{\checkmark\}}$ is to be compensated by the execution of Q, and the non-successful terminated traces in $\Sigma^{\bigstar}_{\{!,?\}}$ by "nothing", *i.e.* **skip**.

$$
\begin{aligned}
\mathcal{F}_f(P \div Q) &= \mathcal{F}(P), \quad \mathcal{D}_f(P \div Q) = \mathcal{D}(P) \\
\mathcal{F}_c(P \div Q) &= ((\mathit{trace}_t(P) \cap \Sigma^{\bigstar}_{\{\checkmark\}}) \times \mathcal{F}(Q)) \cup ((\mathit{trace}_t(P) \cap \Sigma^{\bigstar}_{\{!,?\}}) \times \mathcal{F}(\mathbf{skip})) \\
\mathcal{D}_c(P \div Q) &= ((\mathit{trace}_t(P) \cap \Sigma^{\bigstar}_{\{\checkmark\}}) \times \mathcal{D}(Q)) \cup ((\mathit{trace}_t(P) \cap \Sigma^{\bigstar}_{\{!,?\}}) \times \mathcal{D}(\mathbf{skip}))
\end{aligned}
$$

The forward sub-processes of **skipp**, **throww** and **yieldd** are **skip**, **throw** and **yield**, respectively. Their compensation sub-processes are all **skip**. Because $\mathit{trace}_t(\mathbf{stop})$ is empty, $\mathcal{F}_c(\mathbf{stop} \div P)$ and $\mathcal{D}_c(\mathbf{stop} \div P)$ are both empty for any P, we use **stopp** to denote any $\mathbf{stop} \div P$ whose forward behavior is **stop**. In the same way, **interpp** is used to denote the processes whose forward processes are all **interp**.

Transaction block. A transaction block $[PP]$ is a standard process. Its semantics is derived from the semantics of the compensable process PP.

$$
\begin{aligned}
\mathcal{D}([PP]) = \quad & \mathcal{D}_f(PP) \cup \{s_1 \cdot s_2 \mid (s, s_2) \in \mathcal{D}_c(PP) \wedge s = s_1 \cdot !\} \\
\mathcal{F}([PP]) = \quad & \{(s, X) \mid s \in \Sigma^{\circledast}_{\{\checkmark,?\}} \wedge (s, X \cup \{!\}) \in \mathcal{F}_f(PP)\} \\
& \cup \{(s_1 \cdot s_2, X) \mid (s, s_2, X) \in \mathcal{F}_c(PP) \wedge s = s_1 \cdot !\} \\
& \cup \{(s, X) \mid s \in \mathcal{D}([PP]) \wedge X \subseteq \Gamma\}
\end{aligned}
$$

The compensation of PP is executed to recover from an exception occuring in the forward behavior. The divergences of $[PP]$ contain the DF and DC sets of PP. The failures $\mathcal{F}([PP])$ contain (a) the failures in the FF set that do not terminate with the exception event, (b) the failures in the FF set that terminate with the exception event extended with their corresponding compensation failures, and (c) the failures caused by the divergences. In general $[P \div Q] = P \rhd \mathbf{skip}$ holds and in particular, $[\mathbf{throw} \div P] = \mathbf{skip}$ and $[\mathbf{stopp}] = \mathbf{stop}$.

Internal choice. The semantics of internal choice $PP \sqcap QQ$ is as follows.

$$\mathcal{D}_f(PP \sqcap QQ) = \mathcal{D}_f(PP) \cup \mathcal{D}_f(QQ),\ \mathcal{F}_f(PP \sqcap QQ) = \mathcal{F}_f(PP) \cup \mathcal{F}_f(QQ)$$
$$\mathcal{F}_c(PP \sqcap QQ) = \mathcal{F}_c(PP) \cup \mathcal{F}_c(QQ),\ \ \mathcal{D}_c(PP \sqcap QQ) = \mathcal{D}_c(PP) \cup \mathcal{D}_c(QQ)$$

For example, $(a \div b_1 \sqcap a \div b_2) = (a \div (b_1 \sqcap b_2))$. Its FC set is $\{a\checkmark\} \times (\mathcal{F}(b_1) \cup \mathcal{F}(b_2))$, and its DC set is $\{a\checkmark\} \times (\mathcal{D}(b_1) \cup \mathcal{D}(b_2))$ that equals $\{\}$. Assuming $\Sigma = \{a, b_1, b_2\}$, the compensation failures with the largest refusal sets in the FC set are

$(a\checkmark, \varepsilon, \{a, b_1, !, \checkmark, ?\})$ – b_2 is not refused next,

$(a\checkmark, \varepsilon, \{a, b_2, !, \checkmark, ?\})$ – b_1 is not refused next,

$(a\checkmark, b_1, \{a, b_1, b_2, !, ?\})$ – only $\checkmark$ is not refused next,

$(a\checkmark, b_2, \{a, b_1, b_2, !, ?\})$ – only $\checkmark$ is not refused next,

$(a\checkmark, b_1\checkmark, \{a, b_1, b_2, !, \checkmark, ?\})$ – all events will be refused next,

$(a\checkmark, b_2\checkmark, \{a, b_1, b_2, !, \checkmark, ?\})$ – all events will be refused next.

In summary, the execution of a can be compensated either by b_1 or by b_2.

Internal choice is *idempotent*, *commutative* and *associative*.

External choice. External choice is taken during the forward behavior, but by the environment.

$$\mathcal{D}_f(PP \Box QQ) = \mathcal{D}(fp(PP) \Box fp(QQ)),\ \mathcal{F}_f(PP \Box QQ) = \mathcal{F}(fp(PP) \Box fp(QQ))$$
$$\mathcal{F}_c(PP \Box QQ) = \mathcal{F}_c(PP) \cup \mathcal{F}_c(QQ), \qquad \mathcal{D}_c(PP \Box QQ) = \mathcal{D}_c(PP) \cup \mathcal{D}_c(QQ)$$

For example, $\mathcal{D}_c(\mathbf{stopp} \Box a \div b) = \mathcal{D}_c(\mathbf{stopp} \sqcap a \div b) = \{\}$, and the equality still holds if $\mathcal{D}_c$ is replaced by $\mathcal{F}_c$. For the corresponding FF sets however, $\mathcal{F}_f(\mathbf{stopp} \Box a \div b) = \mathcal{F}(a)$ but $\mathcal{F}_f(\mathbf{stopp} \sqcap a \div b) = \mathcal{F}(a) \cup \mathcal{F}(\mathbf{stop})$.

External choice is also *idempotent*, *commutative* and *associative*.

Sequential composition. In a sequential composition $PP;QQ$, the forward behaviors of PP and QQ are composed first, and the corresponding compensation behaviors $cp(PP, s_1)$ and $cp(QQ, s_2)$ are composed in the reverse direction. The forward behavior of $PP;QQ$ is the sequential composition of the forward behaviors of PP and QQ.

$$\mathcal{D}_f(PP;QQ) = \mathcal{D}(fp(PP);fp(QQ)), \quad \mathcal{F}_f(PP;QQ) = \mathcal{F}(fp(PP);fp(QQ))$$

Let $\mathbb{T}_n$ be the set $trace_n(PP) \times trace_f(QQ)$. The compensation behavior of $PP;QQ$ is defined by the two cases below.

(1) The forward execution of PP terminates successfully and the compensa-

tion behaviors of PP and QQ will be sequentially composed in the reverse order.

$$\mathcal{D}_c^1 = \{(s{\cdot}t, s_c) \mid \exists (s{\cdot}\checkmark, t) \in \mathbb{T}_n \bullet s_c \in \mathcal{D}(cp(QQ, t); cp(PP, s{\cdot}\checkmark))\}$$

$$\mathcal{F}_c^1 = \{(s{\cdot}t, s_c, X_c) \mid \exists (s{\cdot}\checkmark, t) \in \mathbb{T}_n \bullet (s_c, X_c) \in \mathcal{F}(cp(QQ, t); cp(PP, s{\cdot}\checkmark))\}$$

(2) PP fails or diverges in the forward behavior, the process cannot reach QQ, and only the compensation behavior of PP is recorded.

$$\mathcal{D}_c^2 = \{(s, s_c) \mid (s, s_c) \in \mathcal{D}_c(PP) \wedge (s \neq t{\cdot}\checkmark \vee s \notin trace_n(PP))\}$$

$$\mathcal{F}_c^2 = \{(s, s_c, X_c) \mid (s, s_c, X_c) \in \mathcal{F}_c(PP) \wedge (s \neq t{\cdot}\checkmark \vee s \notin trace_n(PP))\}$$

The DC and FC sets of $PP; QQ$ are $\mathcal{D}_c(PP; QQ) = \mathcal{D}_c^1 \cup \mathcal{D}_c^2$ and $\mathcal{F}_c(PP; QQ) = \mathcal{F}_c^1 \cup \mathcal{F}_c^2$.

Sequential composition is *associative* and *distributive* over internal choice.

Parallel composition. In $PP \underset{X}{\parallel} QQ$, the forward behaviors of PP and QQ synchronize on X, so do their compensation behaviors.

$$\mathcal{D}_f(PP \underset{X}{\parallel} QQ) = \mathcal{D}(fp(PP) \underset{X}{\parallel} fp(QQ))$$
$$\mathcal{F}_f(PP \underset{X}{\parallel} QQ) = \mathcal{F}(fp(PP) \underset{X}{\parallel} fp(QQ))$$

$$\mathcal{D}_c(PP \underset{X}{\parallel} QQ) = \{(s, s_c) \mid \exists s_1 \in trace_f(PP), s_2 \in trace_f(QQ) \bullet s \in (s_1 \underset{X}{\parallel} s_2) \wedge s_c \in \mathcal{D}(cp(PP, s_1) \underset{X}{\parallel} cp(QQ, s_2))\}$$
$$\mathcal{F}_c(PP \underset{X}{\parallel} QQ) = \{(s, s_c, X) \mid \exists s_1 \in trace_f(PP), s_2 \in trace_f(QQ) \bullet s \in (s_1 \underset{X}{\parallel} s_2) \wedge (s_c, X) \in \mathcal{F}(cp(PP, s_1) \underset{X}{\parallel} cp(QQ, s_2))\}$$

Consider two examples. First, the equation $[(a \div b_1 \underset{\{a\}}{\parallel} a \div b_2); \mathbf{throww}] = a; b_1 \parallel b_2$ shows the synchronization between the forward behaviors. Then, $a_1 \div b_1 \underset{\{a_1, a_2\}}{\parallel} a_2 \div b_2 = \mathbf{stopp}$ demonstrates how deadlock occurs in the forward behavior.

Parallel composition is *commutative*, *associative*, and *distributive* over internal choice.

Speculative choice. In $PP \boxtimes QQ$, the forward behaviors of PP and QQ will run in parallel first without synchronization. If one succeeds, the compensation of the other will be invoked. The forward execution of $PP \boxtimes QQ$ fails if both parties fail, and in that case the compensation behaviors of PP and QQ run in parallel for recovery. Let $\mathbb{T}$ be the set $trace_f(PP) \times trace_f(QQ)$, $\mathcal{D}_f^1$ (or $\mathcal{D}_f^2$) be the divergences in the case when the first party (or the second party, resp.) succeeds and the second party (or the first party, resp.) has to recover. We

define

$$\mathcal{D}_f^1 = \{s \mid \exists(t_1{\cdot}\checkmark, t_2{\cdot}\omega) \in \mathbb{T} \bullet s \in (t_1 \| t_2){\cdot}\mathcal{D}(cp(QQ, t_2{\cdot}\omega))\}$$
$$\mathcal{D}_f^2 = \{s \mid \exists(t_1{\cdot}\omega, t_2{\cdot}\checkmark) \in \mathbb{T} \bullet s \in (t_1 \| t_2){\cdot}\mathcal{D}(cp(PP, t_1{\cdot}\omega))\}$$

where $\omega \in \Omega$. The DF set of $PP \boxtimes QQ$ is thus defined as $\mathcal{D}(PP_f \parallel QQ_f) \cup \mathcal{D}_f^1 \cup \mathcal{D}_f^2$.

Similarly, we define $\mathcal{F}_f^1$ (or $\mathcal{F}_f^2$) to be the failures when the second (or the first) party succeeds and the first (or the second) party has to recover:

$$\mathcal{F}_f^1 = \{(s{\cdot}t, X) \mid \exists(t_1{\cdot}\checkmark, t_2{\cdot}\omega) \in \mathbb{T} \bullet s \in (t_1 \| t_2) \wedge (t, X) \in \mathcal{F}(cp(QQ, t_2{\cdot}\omega))\}$$
$$\mathcal{F}_f^2 = \{(s{\cdot}t, X) \mid \exists(t_1{\cdot}\omega, t_2{\cdot}\checkmark) \in \mathbb{T} \bullet s \in (t_1 \| t_2) \wedge (t, X) \in \mathcal{F}(cp(PP, t_1{\cdot}\omega))\}$$

The FF set of $PP \boxtimes QQ$ is thus the union of the following five sets, where $\omega_1, \omega_2 \in \Omega \backslash \{\checkmark\}$.

$$\begin{aligned}\mathcal{F}_f(PP \boxtimes QQ) = {} & \{(s, X) | s \in \Sigma^* \wedge (s, X \cup \Omega) \in \mathcal{F}(PP_f \parallel QQ_f)\} \cup \mathcal{F}_f^1 \cup \mathcal{F}_f^2 \\ & \cup \{(s, X) \mid \exists(t_1{\cdot}\omega_1, t_2{\cdot}\omega_2) \in \mathbb{T} \bullet s \in (t_1 \| t_2) \wedge X \subseteq \Gamma \backslash \{\omega_1 \& \omega_2\}\} \\ & \cup \{(s{\cdot}\omega_1 \& \omega_2, X) \mid \exists(t_1{\cdot}\omega_1, t_2{\cdot}\omega_2) \in \mathbb{T} \bullet s \in (t_1 \| t_2) \wedge X \subseteq \Gamma\}\end{aligned}$$

The first set includes the failures of the interleaving forward execution of PP and QQ, and the last two sets handle the synchronization of the terminals if both parties fail.

There are the following three cases when compensation of $PP \boxtimes QQ$ diverges.

(1) PP succeeds in the forward parallel execution of PP and QQ, and the compensation to the effect of PP diverges.

$$\mathcal{D}_c^1 = \{(s, s_c) \mid \exists(t_1{\cdot}\checkmark, t_2{\cdot}\omega) \in \mathbb{T} \bullet s \in T_1 \wedge s_c \in \mathcal{D}(cp(PP, t_1{\cdot}\checkmark))\}$$

where $T_1 = (t_1 \parallel t_2){\cdot}trace_t(cp(QQ, t_2{\cdot}\omega))$. Notice that the overall compensation of PP by $PP \boxtimes QQ$ starts after the effect of QQ is compensated.

(2) Symmetrically, QQ succeeds in the forward parallel execution of PP and QQ, and the compensation to the effect of QQ diverges.

$$\mathcal{D}_c^2 = \{(s, s_c) \mid \exists(t_1{\cdot}\omega, t_2{\cdot}\checkmark) \in \mathbb{T} \bullet s \in T_2 \wedge s_c \in \mathcal{D}(cp(QQ, t_2{\cdot}\checkmark))\}$$

where $T_2 = (t_1 \| t_2){\cdot}trace_t(cp(PP, t_1{\cdot}\omega))$.

(3) Both parties of the parallel forward execution of PP and QQ fail to terminate successfully, and the parallel compensation of the parties diverges, where $\omega_1, \omega_2 \in \Omega \backslash \{\checkmark\}$.

$$\begin{aligned}\mathcal{D}_c^3 = \{(s, s_c) \mid {} & \exists(t_1{\cdot}\omega_1, t_2{\cdot}\omega_2) \in \mathbb{T} \bullet s \in (t_1{\cdot}\omega_1 \| t_2{\cdot}\omega_2) \wedge \\ & s_c \in \mathcal{D}(cp(PP, t_1{\cdot}\omega_1) \parallel cp(QQ, t_2{\cdot}\omega_2))\}\end{aligned}$$

Therefore, the DC set of $PP \boxtimes QQ$ is defined as $\mathcal{D}_c(PP \boxtimes QQ) = \mathcal{D}_c^1 \cup \mathcal{D}_c^2 \cup \mathcal{D}_c^3$.

Similarly, $\mathcal{F}_c(PP \boxtimes QQ)$ is $\mathcal{F}_c^1 \cup \mathcal{F}_c^2 \cup \mathcal{F}_c^3$, where

$$\begin{aligned}
\mathcal{F}_c^1 &= \{(s, s_c, X) | \exists (t_1 \cdot \checkmark, t_2 \cdot \omega) \in \mathbb{T} \bullet s \in T_1 \wedge (s_c, X) \in \mathcal{F}(cp(PP, t_1 \cdot \checkmark))\} \\
\mathcal{F}_c^2 &= \{(s, s_c, X) \mid \exists (t_1 \cdot \omega, t_2 \cdot \checkmark) \in \mathbb{T} \bullet s \in T_2 \wedge (s_c, X) \in \mathcal{F}(cp(QQ, t_2 \cdot \checkmark))\} \\
\mathcal{F}_c^3 &= \{(s, s_c, X) \mid \exists (t_1 \cdot \omega_1, t_2 \cdot \omega_2) \in \mathbb{T} \bullet s \in (t_1 \cdot \omega_1 \| t_2 \cdot \omega_2) \wedge \\
&\qquad (s_c, X) \in \mathcal{F}(cp(PP, t_1 \cdot \omega_1) \parallel cp(QQ, t_2 \cdot \omega_2))\}
\end{aligned}$$

Consider the process $a_1 \div b_1 \boxtimes (a_2 \div b_2; \mathbf{throww})$ for example. Its forward divergence and failure sets are $\mathcal{D}((a_1 \parallel a_2); b_2)$ (equals $\{\}$) and $\mathcal{F}((a_1 \parallel a_2); b_2)$, and compensation divergence and failure sets are $\{\}$ and $\{a_1 a_2 b_2 \checkmark, a_2 a_1 b_2 \checkmark\} \times \mathcal{F}(b_1)$. This means the right side process $(a_2 \div b_2; \mathbf{throww})$ fails and has been compensated, and b_1 can be used to compensate the whole process.

Speculative choice is *commutative* and *distributive* over internal choice.

This definition of speculative choice differs from that of the original cCSP in treating the case when both sub-processes of the choice fail. We treat combination of the failures as a global failure, but the original cCSP handles the failure by immediately running the compensation when both processes fail. This prevents propagation of the failure (*cf.* the last case of the speculative choice in Figure 2). To demonstrate this difference, consider the process

$$[\ ((a_1 \div b_1; \mathbf{throww}) \boxtimes (a_2 \div b_2; \mathbf{throww})) \parallel (a_3 \div b_3)\ ]$$

Under the semantics of the original cCSP (*cf.* Figure 1&2), the event b_3 will never be performed. However, under our FD semantics, the process is equal to $(a_1 \| a_2 \| a_3); (b_1 \| b_2 \| b_3)$. Therefore, our semantics allows a speculative choice to propagate an exception caused by the failure of both sides.

Hiding and renaming. Hiding and renaming are defined in the standard way on the forward behavior and the compensation behavior, respectively.

$$\begin{aligned}
\mathcal{D}_f(PP \backslash X) &= \mathcal{D}(fp(PP) \backslash X), \quad \mathcal{F}_f(PP \backslash X) = \mathcal{F}(fp(PP) \backslash X) \\
\mathcal{D}_c(PP \backslash X) &= \{(s, s_c) | \exists s_1 \in trace_f(PP) \bullet s = s_1 \backslash X \wedge s_c \in \mathcal{D}(cp(PP, s_1) \backslash X)\} \\
\mathcal{F}_c(PP \backslash X) &= \{(s, s_c, X) | \exists s_1 \in trace_f(PP) \bullet s = s_1 \backslash X \wedge (s_c, X) \in \mathcal{F}(cp(PP, s_1) \backslash X)\}
\end{aligned}$$

Similarly, the semantics of renaming is as follows.

$$\begin{aligned}
\mathcal{D}_f(PP[\![\rho]\!]) &= \mathcal{D}(fp(PP)[\![\rho]\!]), \quad \mathcal{F}_f(PP[\![\rho]\!]) = \mathcal{F}(fp(PP)[\![\rho]\!]) \\
\mathcal{D}_c(PP[\![\rho]\!]) &= \{(s, s_c) \mid \exists s_1 \in trace_f(PP) \bullet s_1 \ \rho \ s \wedge s_c \in \mathcal{D}(cp(PP, s_1)[\![\rho]\!])\} \\
\mathcal{F}_c(PP[\![\rho]\!]) &= \{(s, s_c, X) \mid \exists s_1 \in trace_f(PP) \bullet s_1 \ \rho \ s \wedge (s_c, X) \in \mathcal{F}(cp(PP, s_1)[\![\rho]\!])\}
\end{aligned}$$

Both hiding and renaming are distributive over internal choice. Unlike renam-

ing, hiding is not distributive over external choice. For example, the compensable process $((a;a_1)\div b\Box(a;a_2)\div b)\setminus\{a\}$ equals $a_1\div b\sqcap a_2\div b$.

3.3 Relation to the trace semantics and operational semantics

A trace semantics can be derived from the FD semantics. For a standard process P, define the $\mathcal{T}(P)$ of P as the non-divergent terminated traces of P

$$\mathcal{T}(P) = \mathit{trace}_n(P) = (\{s \mid (s,\{\}) \in \mathcal{F}(P)\} \cap \Sigma^\star) \setminus \mathcal{D}(P)$$

For a compensable process PP, the set $\mathcal{T}_c(PP)$ is derived as follows.

$$\mathcal{T}_c(PP) = \{(s_t, s_c) \mid s_t \in \mathit{trace}_n(PP) \wedge s_c \in \mathit{trace}_n(\mathit{cp}(PP, s_t))\}$$

It is easy to check that the trace semantics derived from the FD semantics is equal to the original trace semantics summarized in Section 2.2, but with overcoming the problem that invalidates the law discussed in Section 2.4. Therefore, it is also consistent with the operational semantics in [32].

Furthermore, [32] extends cCSP with a general parallel composition operator and defines its operational semantics using the notion of "*partial behavior*". For this, an extra terminal symbol $\perp$ is introduced to define a partial trace that terminates with $\perp$. Such a partial trace represents that the process only executes a part of its behavior due to a deadlock. We can derive the partial traces semantics as follows.

$$\begin{aligned}
\mathcal{T}(P) &= \mathit{trace}_n(P) \cup \{s{\cdot}\perp \mid (s,\Gamma) \in \mathcal{F}(P) \wedge s \in \Sigma^* \wedge s \notin \mathcal{D}(P)\} \\
\mathcal{T}_c(PP) &= \{(s_t, s_c) \mid s_t \in \mathit{trace}_n(PP) \wedge s_c \in \mathit{trace}_n(\mathit{cp}(PP, s_t))\} \\
&\cup\ \{(s{\cdot}\perp, \perp) \mid (s,\Gamma) \in \mathcal{F}_f(PP) \wedge s \in \Sigma^* \wedge s \notin \mathcal{D}_f(PP)\}
\end{aligned}$$

This shows that the FD semantics is consistent with the extended operational semantics defined in [32].

4 Refinement and Algebraic Laws

We study the theory of *refinement relations* between processes. In general a process, either standard or compensable, refines another one if the former does not exhibit more failures or divergences, *i.e.* neither more likely to deadlock nor more likely to diverge, than the latter. This also means a refined process is *less non-deterministic*. The theory shows that the refinement relations between standard processes and between compensable processes form a CPO on the two domains of the processes, respectively. This allows definition

and calculation of the least fixed points of standard recursive processes and compensable recursive processes (*cf.* Section 4.1). We will study the algebraic laws of cCSP processes (*cf.* Sections 4.2&4.3).They characterize the principle of LRT programming and provide a justification for the correctness of the failure divergence semantics of cCSP. For the purpose of a compact presentation, we provide the proofs of the main theorems and the important laws in appendices.

4.1 Refinement and semantics of recursion

We define a partial order $\sqsubseteq$ on the FD sets of standard processes, and a partial order $\sqsubseteq_c$ on the forward and compensation FD sets of compensable processes. The two orders are linked by lifting the order $\sqsubseteq_c$ between compensable processes to the order $\sqsubseteq$ between standard processes through the transaction block constructor $[PP]$.

Definition 1 *The* FD refinement *of a standard process P_1 by a standard process P_2 is defined as*

$$P_1 \sqsubseteq P_2 \mathrel{\widehat{=}} \mathcal{F}(P_1) \supseteq \mathcal{F}(P_2) \wedge \mathcal{D}(P_1) \supseteq \mathcal{D}(P_2) \tag{13}$$

It means that the refinement P_2 is neither more likely to refuse an interaction from the environment nor more likely to diverge than P_1. Therefore, the refining process P_2 is *more deterministic* than the refined process P_1, and two equivalent processes refine each other, *i.e.* $P_1 = P_2$ iff $P_1 \sqsubseteq P_2 \wedge P_2 \sqsubseteq P_1$.

Theorem 1 *The semantic domain of standard processes is a complete partial order (CPO) under the refinement order $\sqsubseteq$, and* **div** *is the bottom element.*

The proof of this theorem is given in Appendix A.

Definition 2 *Given two compensable processes PP_i with their semantic models (F_i, D_i, F_i^c, D_i^c), for $i \in \{1, 2\}$, PP_2 is an* FD refinement *of PP_1 if*

$$PP_1 \sqsubseteq_c PP_2 \mathrel{\widehat{=}} F_1 \supseteq F_2 \wedge D_1 \supseteq D_2 \wedge F_1^c \supseteq F_2^c \wedge D_1^c \supseteq D_2^c \tag{14}$$

Theorem 2 *The semantic domain of compensable processes is a CPO w.r.t. $\sqsubseteq_c$ and* **div** $\div$ **div** *is the bottom element.*

The proof of the theorem is given in Appendix A as well. The two refinement relations are linked by the following theorem.

Theorem 3 *The two refinement relations $\sqsubseteq_c$ and $\sqsubseteq$ are consistently related.*

(1) If $PP_1 \sqsubseteq_c PP_2$ then $[PP_1] \sqsubseteq [PP_2]$.

(2) Refinement of compensable processes can be constructed from the refinement of forward or compensation processes.

$$Q_1 \sqsubseteq Q_2 \Rightarrow P \div Q_1 \sqsubseteq_c P \div Q_2, \quad P_1 \sqsubseteq P_2 \Rightarrow P_1 \div Q \sqsubseteq_c P_2 \div Q$$

The following two theorems establish the foundation for the calculation of the fixed point of a recursive process.

Theorem 4 *The operators of standard processes are continuous w.r.t.* $\sqsubseteq$.

Recursive standard processes If $\mu\ p.F(p)$ is a constructive standard process, its semantics is the *least fixed point* of the semantic function $[\![F]\!]$ of F. The semantics can be calculated, according to Theorems 1&4, as $\bigsqcup\{F^n(\mathbf{div}) \mid n \in \mathbb{N}\}$, where $F^0(\mathbf{div}) = \mathbf{div}$ and $F^{(n+1)}(\mathbf{div}) = F(F^n(\mathbf{div}))$, and $\bigsqcup S$ represents the least upper bound of the set S. For example, assume Σ is $\{a\}$, the failures $\mathcal{F}(\mu\ p.\ (a;p)) = \{(a^i, X) \mid i \in \mathbb{N} \wedge X \subseteq \Omega\}$, where $a^0 = \varepsilon$ and $a^{i+1} = a{\cdot}a^i$, and the divergences $\mathcal{D}(\mu\ p.\ (a;p)) = \{\}$.

According to the semantic definitions, the failure-divergence semantics of the process $(\mu\ p.\ (a;p)) \setminus \{a\}$ is equal to that of **div**. Obviously, the semantics of $(\mu\ p.\ (a;p)) \setminus \{a\}$ is different from that of the deadlocked process $a \underset{\{a,b\}}{\parallel} b$ (equal to **stop**). However, with respect to the traditional CSP trace semantics [34], the semantics of these two processes are both equal to $\{\varepsilon\}$. Therefore, the FD semantics supports investigation of deadlocks and livelocks better.

Theorem 5 *The operators of compensable processes are continuous w.r.t.* $\sqsubseteq_c$.

Recursive compensable processes Theorems 2&5 ensure the existence of the least fixed point of a recursive compensable process $\mu\ pp.FF(pp)$. It is calculated as $\bigsqcup\{FF^n(\mathbf{div} \div \mathbf{div}) \mid n \in \mathbb{N}\}$. Consider $\mu\ FF.(a \div b\ ;\ pp)$ for example. Its forward semantics is equal to the semantics of $\mu p.(a\ ;\ p)$, and both the FC and DC sets are empty.

4.2 Laws of standard processes

Algebraic laws are important for understanding fundamental principles of programming, and for justifying a denotational semantics. We thus present key algebraic laws of cCSP; proofs of the non-trivial laws are given in Appendix B. The first group of laws are those of classic CSP that remain valid in the extended cCSP. The purpose of presenting these laws is to justify the FD se-

mantics, and they are also used in some proofs.

$$P \sqcap P = P \qquad \text{(s-}\sqcap\text{-idem)}$$
$$P \sqcap Q = Q \sqcap P \qquad \text{(s-}\sqcap\text{-sym)}$$
$$P \sqcap (Q \sqcap R) = (P \sqcap Q) \sqcap R \qquad \text{(s-}\sqcap\text{-assoc)}$$
$$P \sqcap Q \sqsubseteq P \qquad \text{(s-}\sqsubseteq\text{-}\sqcap\text{)}$$
$$P \Box P = P \qquad \text{(s-}\Box\text{-idem)}$$
$$P \Box Q = Q \Box P \qquad \text{(s-}\Box\text{-sym)}$$
$$\mathbf{stop} \Box P = P \qquad \text{(s-}\Box\text{-unit)}$$
$$P \Box (Q \sqcap R) = (P \Box Q) \sqcap (P \Box R) \qquad \text{(s-}\Box\text{-dist)}$$
$$P \Box (Q \Box R) = (P \Box Q) \Box R \qquad \text{(s-}\Box\text{-assoc)}$$
$$\mathbf{skip} \ ; \ P = P \qquad \text{(s-;-unit-l)}$$
$$P \ ; \ \mathbf{skip} = P \qquad \text{(s-;-unit-r)}$$
$$(Q_1 \sqcap Q_2); P = (Q_1; P) \sqcap (Q_2; P) \qquad \text{(s-;-dist-l)}$$
$$P; (Q_1 \sqcap Q_2) = (P; Q_1) \sqcap (P; Q_2) \qquad \text{(s-;-dist-r)}$$
$$P; (Q; R) = (P; Q); R \qquad \text{(s-;-assoc)}$$
$$P \underset{X}{\parallel} Q = Q \underset{X}{\parallel} P \qquad \text{(s-}\parallel\text{-sym)}$$
$$P \underset{X}{\parallel} (Q \underset{X}{\parallel} R) = (P \underset{X}{\parallel} Q) \underset{X}{\parallel} R \qquad \text{(s-}\parallel\text{-assoc)}$$
$$P \underset{X}{\parallel} (Q_1 \sqcap Q_2) = (P \underset{X}{\parallel} Q_1) \sqcap (P \underset{X}{\parallel} Q_2) \qquad \text{(s-}\parallel\text{-dist)}$$
$$(P \sqcap Q) \backslash X = (P \backslash X) \sqcap (Q \backslash X) \qquad \text{(s-}\backslash\text{-dist)}$$
$$(P \backslash X_1) \backslash X_2 = P \backslash (X_1 \cup X_2) \qquad \text{(s-}\backslash\text{-}\cup\text{)}$$
$$(P \backslash X_1) \backslash X_2 = (P \backslash X_2) \backslash X_1 \qquad \text{(s-}\backslash\text{-sym)}$$
$$P \backslash \{\} = P \qquad \text{(s-}\backslash\text{-unit)}$$
$$(a; P) \backslash X = \begin{cases} P \backslash X, a \in X \\ a; (P \backslash X), a \notin X \end{cases} \qquad \text{(s-}\backslash\text{-step)}$$
$$(P \ ; \ Q) \backslash X = (P \backslash X) \ ; \ (Q \backslash X) \qquad \text{(s-}\backslash\text{-;)}$$
$$(P \parallel Q) \backslash X = (P \backslash X) \parallel (Q \backslash X) \qquad \text{(s-}\backslash\text{-}\parallel\text{)}$$
$$(P \sqcap Q)[\![\rho]\!] = (P[\![\rho]\!]) \sqcap (Q[\![\rho]\!]) \qquad \text{(s-}R\text{-dist)}$$
$$(P \Box Q)[\![\rho]\!] = (P[\![\rho]\!]) \Box (Q[\![\rho]\!]) \qquad \text{(s-}R\text{-}\Box\text{)}$$
$$(P[\![\rho_1]\!])[\![\rho_2]\!] = P[\![\rho_1 \circ \rho_2]\!] \qquad \text{(s-}R\text{-}\circ\text{)}$$

The rest of this subsection gives the special laws that characterize the behavior of exception handling and interruptions. First of all, internal and external choice are indistinguishable on the basic terminal processes.

$$\mathbf{skip} \Box \mathbf{yield} = \mathbf{yield} \qquad \text{(s-}\Box\text{-terminal-1)}$$
$$\mathbf{skip} \Box \mathbf{throw} = \mathbf{skip} \sqcap \mathbf{throw} \qquad \text{(s-}\Box\text{-terminal-2)}$$
$$\mathbf{yield} \Box \mathbf{throw} = \mathbf{yield} \sqcap \mathbf{throw} \qquad \text{(s-}\Box\text{-terminal-3)}$$

Law s-;-zero-1 below ensures the *exception-stop* semantics that is adopted in many languages.

$$\mathbf{throw}\ ;\ P = \mathbf{throw} \qquad \text{(s-;-zero-1)}$$
$$\mathbf{interp}\ ;\ P = \mathbf{interp} \qquad \text{(s-;-zero-2)}$$

In addition, there are following laws for exception handling.

$$\mathbf{throw} \rhd P = P \qquad \text{(s-}\rhd\text{-unit-l)}$$
$$P \rhd \mathbf{throw} = P \qquad \text{(s-}\rhd\text{-unit-r)}$$
$$\mathbf{skip} \rhd P = \mathbf{skip} \qquad \text{(s-}\rhd\text{-zero-1)}$$
$$\mathbf{yield} \rhd P = \mathbf{yield} \qquad \text{(s-}\rhd\text{-zero-2)}$$
$$\mathbf{stop} \rhd P = \mathbf{stop} \qquad \text{(s-}\rhd\text{-zero-3)}$$
$$P \rhd (Q \sqcap R) = (P \rhd Q) \sqcap (P \rhd R) \qquad \text{(s-}\rhd\text{-dist-r)}$$
$$(P \sqcap Q) \rhd R = (P \rhd R) \sqcap (Q \rhd R) \qquad \text{(s-}\rhd\text{-dist-l)}$$
$$P \rhd (Q \rhd R) = (P \rhd Q) \rhd R \qquad \text{(s-}\rhd\text{-assoc)}$$

throw is the unit of exception handling in standard processes. Synchronization of terminal events is characterized by the following laws.

$$\mathbf{skip} \underset{X}{\parallel} \mathbf{interp} = \mathbf{interp} \qquad \text{(s-}\parallel\text{-syn-1)}$$
$$\mathbf{throw} \underset{X}{\parallel} \mathbf{skip} = \mathbf{throw} \qquad \text{(s-}\parallel\text{-syn-2)}$$
$$\mathbf{throw} \underset{X}{\parallel} \mathbf{interp} = \mathbf{throw} \qquad \text{(s-}\parallel\text{-syn-3)}$$

4.2.1 Special laws of parallel composition

If P does not terminate with a yield terminal event, *i.e.* $\forall s{\in}trace_n(P){\bullet}s{\notin}\Sigma^{\star}_{\{?\}}$, parallel composition without synchronization $\parallel$ enjoys the following laws.

$$\mathbf{throw} \parallel P = P\ ;\ \mathbf{throw} \qquad \text{(s-}\parallel\text{-throw)}$$
$$\mathbf{throw} \parallel (\mathbf{yield}\ ;\ P) = \mathbf{throw} \sqcap (P\ ;\ \mathbf{throw}) \qquad \text{(s-}\parallel\text{-throw-yield)}$$

The second law says that a process can be interrupted by an exception from the environment, but the interruption does not have priority over other events. Parallel composition enjoys the following unit and step laws.

$$P \parallel \mathbf{skip} = P \qquad \text{(s-}\parallel\text{-unit)}$$
$$(a\ ;\ P) \underset{X}{\parallel} (a\ ;\ Q) = a\ ;\ (P \underset{X}{\parallel} Q) \quad if\ a \in X \qquad \text{(s-}\parallel\text{-step)}$$

We use $\alpha(P)$ to denote the set of all the non-terminal events appearing in the standard process P. The following laws hold for parallel composition.

$$(Q \parallel P) \underset{X}{\parallel} R = Q \parallel (P \underset{X}{\parallel} R) \quad if\ \alpha(Q) \cap X = \{\} \qquad \text{(s-}\parallel\text{-assoc-1)}$$

If P and Q are constructed only from basic processes and sequential composition, $\alpha(P) \cup \alpha(Q) \subseteq X$, $\alpha(P) \neq \{\}$ and $\alpha(P) \cap \alpha(Q) = \{\}$,

$$P \underset{X}{\parallel} P = P \qquad \text{(s-}\parallel\text{-idem)}$$

$$(P \Box Q) \underset{X}{\parallel} P = P \qquad \text{(s-}\parallel\text{-}\Box\text{)}$$

The laws below transform a parallel process to a sequential process for understanding and analyzing its behavior. Let $\alpha^i(P)$ denote the set of non-terminal starting events of P, *i.e.* $\alpha^i(P) = \{e \mid \exists s \bullet e{\cdot}s \in \mathit{traces}_\perp(P) \wedge e \notin \Omega\}$. If P terminates successfully, $\alpha(P) \cap X{=}\{\}$ and $\alpha^i(R) \subseteq X$, the following law holds.

$$(P \ ; \ Q) \underset{X}{\parallel} R = P \ ; \ (Q \underset{X}{\parallel} R) \qquad \text{(s-}\parallel\text{-;-head)}$$

If R terminates successfully, $\alpha(P) \cap X = \{\}$, $\alpha(R) \subseteq (\alpha(Q) \cap X)$, and there is no deadlock in $Q \underset{X}{\parallel} R$, we have

$$(Q \ ; \ P) \underset{X}{\parallel} R = (Q \underset{X}{\parallel} R) \ ; \ P \qquad \text{(s-}\parallel\text{-;-tail)}$$

If $\mathit{traces}_\perp(P_1) = \mathit{traces}_\perp(P_2)$, $\alpha(P_1) \subseteq X$ and no deadlock in $P_1 \underset{X}{\parallel} P_2$, then the following law holds.

$$(P_1 \ ; \ Q_1) \underset{X}{\parallel} (P_2 \ ; \ Q_2) = (P_1 \underset{X}{\parallel} P_2) \ ; \ (Q_1 \underset{X}{\parallel} Q_2) \qquad \text{(s-}\parallel\text{-step-1)}$$

4.3 Laws of long running transactions

We present the laws in groups according to the operators and the nature of the laws. The laws together can be used for design and analysis of compensable processes. The simple laws are important for understanding and proving more subtle laws. Proofs of complex laws are given in Appendix C.

4.3.1 **Refinement and lifting laws**

The following laws (the last three are from Theorem 3) relate refinement between standard processes and refinement between compensable processes.

$$PP \sqcap QQ \sqsubseteq_c PP \qquad \text{(c-}\sqsubseteq_c\text{-}\sqcap\text{)}$$

$$P_1 \sqsubseteq P_2 \Rightarrow P_1 \div Q \sqsubseteq_c P_2 \div Q \qquad \text{(c-link-}\sqsubseteq_c\text{-}\sqsubseteq\text{-f)}$$

$$Q_1 \sqsubseteq Q_2 \Rightarrow P \div Q_1 \sqsubseteq_c P \div Q_2 \qquad \text{(c-link-}\sqsubseteq_c\text{-}\sqsubseteq\text{-c)}$$

$$PP_1 \sqsubseteq_c PP_2 \Rightarrow [PP_1] \sqsubseteq [PP_2] \qquad \text{(c-lift-}\sqsubseteq_c\text{)}$$

The laws below lift an operator of compensable processes to an operator of standard processes.

$$[P \div Q] = P \rhd \mathbf{skip} \qquad \text{(c-lift-}\div\text{)}$$
$$[PP \sqcap QQ] = [PP] \sqcap [QQ] \qquad \text{(c-lift-}\sqcap\text{)}$$
$$[PP \Box QQ] = [PP] \Box [QQ] \qquad \text{(c-lift-}\Box\text{)}$$
$$[PP \setminus X] = [PP] \setminus X \qquad \text{(c-lift-}\setminus\text{)}$$
$$[PP[\![\rho]\!]] = [PP][\![\rho]\!] \qquad \text{(c-lift-}[\![R]\!]\text{)}$$

From the law c-lift-$\div$, we can instantiate processes P and Q to get some interesting equations, such as $[\mathbf{skip} \div P] = \mathbf{skip}$, $[\mathbf{throw} \div P] = \mathbf{skip}$ and $[\mathbf{yield} \div P] = \mathbf{yield}$. From the laws c-lift-$\sqcap$ and c-lift-$\Box$, we see that a transaction block process of a choice between compensable processes equals a choice between the transaction block processes of the compensable processes.

4.3.2 Basic algebraic laws

We have the following laws of idempotency, symmetry, associativity and distributivity for the operators on compensable processes.

$$(P \sqcap Q) \div R = P \div R \sqcap Q \div R \qquad \text{(c-}\div\text{-dist-l)}$$
$$P \div (Q \sqcap R) = P \div Q \sqcap P \div R \qquad \text{(c-}\div\text{-dist-r)}$$
$$PP \sqcap PP = PP \qquad \text{(c-}\sqcap\text{-idem)}$$
$$PP \sqcap QQ = QQ \sqcap PP \qquad \text{(c-}\sqcap\text{-sym)}$$
$$PP \sqcap (QQ \sqcap RR) = (PP \sqcap QQ) \sqcap RR \qquad \text{(c-}\sqcap\text{-assoc)}$$
$$PP \Box PP = PP \qquad \text{(c-}\Box\text{-idem)}$$
$$PP \Box QQ = QQ \Box PP \qquad \text{(c-}\Box\text{-sym)}$$
$$PP \Box (QQ \Box RR) = (PP \Box QQ) \Box RR \qquad \text{(c-}\Box\text{-assoc)}$$
$$PP \Box (QQ \sqcap RR) = (PP \Box QQ) \sqcap (PP \Box RR) \qquad \text{(c-}\Box\text{-dist)}$$
$$PP \boxtimes QQ = QQ \boxtimes PP \qquad \text{(c-}\boxtimes\text{-sym)}$$
$$PP \boxtimes (QQ \sqcap RR) = (PP \boxtimes QQ) \sqcap (PP \boxtimes RR) \qquad \text{(c-}\boxtimes\text{-dist)}$$
$$PP \mathbin{;} (QQ \mathbin{;} RR) = (PP \mathbin{;} QQ) \mathbin{;} RR \qquad \text{(c-;-assoc)}$$
$$(PP \sqcap QQ) \mathbin{;} RR = (PP \mathbin{;} RR) \sqcap (QQ \mathbin{;} RR) \qquad \text{(c-;-dist-l)}$$
$$PP \mathbin{;} (QQ \sqcap RR) = (PP \mathbin{;} QQ) \sqcap (PP \mathbin{;} RR) \qquad \text{(c-;-dist-r)}$$
$$PP \underset{X}{\parallel} QQ = QQ \underset{X}{\parallel} PP \qquad \text{(c-}\parallel\text{-sym)}$$
$$PP \underset{X}{\parallel} (QQ \underset{X}{\parallel} RR) = (PP \underset{X}{\parallel} QQ) \underset{X}{\parallel} RR \qquad \text{(c-}\parallel\text{-assoc)}$$
$$PP \underset{X}{\parallel} (QQ \sqcap RR) = (PP \underset{X}{\parallel} QQ) \sqcap (PP \underset{X}{\parallel} RR) \qquad \text{(c-}\parallel\text{-dist)}$$
$$(PP \setminus X) \setminus Y = (PP \setminus Y) \setminus X \qquad \text{(c-}\setminus\text{-sym)}$$
$$(PP \sqcap QQ) \setminus X = (PP \setminus X) \sqcap (QQ \setminus X) \qquad \text{(c-}\setminus\text{-dist)}$$
$$(P \div Q) \setminus X = (P \setminus X) \div (Q \setminus X) \qquad \text{(c-}\setminus\text{-pair-dist)}$$
$$(PP \mathbin{;} QQ) \setminus X = (PP \setminus X) \mathbin{;} (QQ \setminus X) \qquad \text{(c-}\setminus\text{-;-dist)}$$
$$(PP \parallel QQ) \setminus X = (PP \setminus X) \parallel (QQ \setminus X) \qquad \text{(c-}\setminus\text{-}\parallel\text{-dist)}$$

$$(PP \sqcap QQ)[\![\rho]\!] = PP[\![\rho]\!] \sqcap QQ[\![\rho]\!] \qquad \text{(c-}[\![R]\!]\text{-dist)}$$
$$(PP \Box QQ)[\![\rho]\!] = PP[\![\rho]\!] \Box QQ[\![\rho]\!] \qquad \text{(c-}[\![R]\!]\text{-}\Box\text{-dist)}$$
$$(P \div Q)[\![\rho]\!] = (R[\![\rho]\!]) \div (Q[\![\rho]\!]) \qquad \text{(c-}[\![R]\!]\text{-pair-dist)}$$

The first two laws also say that non-determinism in standard processes can cause non-determinism in compensable processes.

Unit and Zero. The below laws define the units and zeros of some operators.

$$\mathbf{stopp} \Box PP = PP \qquad \text{(c-}\Box\text{-unit)}$$
$$\mathbf{skipp} \ ; \ PP = PP \qquad \text{(c-;-unit-l)}$$
$$PP \ ; \ \mathbf{skipp} = PP \qquad \text{(c-;-unit-r)}$$
$$PP \setminus \{\} = PP \qquad \text{(c-}\setminus\text{-unit)}$$
$$\mathbf{throww} \ ; \ PP = \mathbf{throww} \qquad \text{(c-;-zero-1)}$$
$$\mathbf{interpp} \ ; \ PP = \mathbf{interpp} \qquad \text{(c-;-zero-2)}$$

If all standard processes terminate successfully, the following laws hold.

$$P \div Q \boxtimes \mathbf{throww} = P \div Q \qquad \text{(c-}\boxtimes\text{-unit-1)}$$
$$P \div Q \boxtimes \mathbf{interpp} = P \div Q \qquad \text{(c-}\boxtimes\text{-unit-2)}$$

4.3.3 Laws of compensation and interruption

If the standard processes P, P_1 and P_2 do not terminate with an exception event, then the following laws hold.

$$[P \div Q \ ; \ \mathbf{throww}] = P \ ; \ Q \qquad \text{(c-Saga-1)}$$
$$[P_1 \div Q_1 \ ; \ P_2 \div Q_2 \ ; \ \mathbf{throww}] = P_1 \ ; \ P_2 \ ; \ Q_2 \ ; \ Q_1 \qquad \text{(c-Saga-2)}$$

The second law reflects the nature of the backward recovery in a long-running transaction. If an exception occurs, the compensation actions are executed in the reverse order of their corresponding forward actions.

If all the standard processes terminate successfully, then the following laws hold.

$$[(P \div Q) \parallel \mathbf{throww}] = P \ ; \ Q \qquad \text{(c-Saga-}\|\text{-1)}$$
$$[(P \div Q \ ; \ \mathbf{interpp}) \parallel \mathbf{throww}] = P \ ; \ Q \qquad \text{(c-Saga-}\|\text{-2)}$$
$$[(P_1 \div Q_1 \boxtimes P_2 \div Q_2); \mathbf{throww}] = (P_1 \| P_2); ((Q_1; Q_2) \sqcap (Q_2; Q_1)) \qquad \text{(c-Saga-}\boxtimes\text{)}$$
$$P_1 \div Q_1 \ ; \ P_2 \div Q_2 = (P_1 \ ; \ P_2) \div (Q_2 \ ; \ Q_1) \qquad \text{(c-;-pair)}$$
$$(P_1 \div Q_1) \underset{X}{\parallel} (P_2 \div Q_2) = (P_1 \underset{X}{\parallel} P_2) \div (Q_1 \underset{X}{\parallel} Q_2) \qquad \text{(c-}\|\text{-pair)}$$

The law c-Saga-$\boxtimes$ states that a speculative choice where both components succeed non-deterministically selects one to compensate. Based on the above

laws, we get the following interruption laws.

$$[(P_1 \div Q_1 \underset{X}{\parallel} P_2 \div Q_2)\ ;\ \mathbf{throww}] = (P_1 \underset{X}{\parallel} P_2)\ ;\ (Q_1 \underset{X}{\parallel} Q_2) \qquad \text{(c-;-}\|\text{)}$$

$$[(P_1 \div Q_1\ ;\ P_2 \div Q_2) \parallel \mathbf{throww}] = P_1\ ;\ P_2\ ;\ Q_2\ ;\ Q_1 \qquad \text{(c-}\|\text{-;)}$$

$$[(\mathbf{yieldd}\ ;\ P_1 \div Q_1\ ;\ \mathbf{yieldd}\ ;\ P_2 \div Q_2) \parallel \mathbf{throww}] = \mathbf{skip} \sqcap (P_1\ ;\ Q_1) \sqcap (P_1\ ;\ P_2\ ;\ Q_2\ ;\ Q_1) \qquad \text{(c-}\|\text{-inter-1)}$$

$$[(\mathbf{yieldd}\ ;\ P_1 \div Q_1) \parallel (\mathbf{yieldd}\ ;\ P_2 \div Q_2) \parallel \mathbf{throww}] = \mathbf{skip} \sqcap (P_1\ ;\ Q_1) \sqcap (P_2\ ;\ Q_2) \sqcap ((P_1 \parallel P_2)\ ;\ (Q_1 \parallel Q_2)) \qquad \text{(c-}\|\text{-inter-2)}$$

The last two laws say that a compensable process in a parallel composition can be interrupted by **yieldd**, meaning that the sub compensable process yields to an exception from the environment. The law c-‖-; states that a compensable process will not be interrupted if no **yieldd** is used. This is a difference from the original cCSP, where a compensable process can implicitly yield to an exception from the environment (*cf.* Section 2.2). We believe that it is more reasonable to let the designer to specify where a compensable process can yield to an exception from the environment.

4.3.4 Laws of parallel composition of LRTs

The laws below reflect the synchronization policies of LRTs.

$$\mathbf{interpp} \underset{X}{\parallel} \mathbf{throww} = \mathbf{throww} \qquad \text{(c-}\|\text{-syn-1)}$$

$$\mathbf{skipp} \underset{X}{\parallel} \mathbf{throww} = \mathbf{throww} \qquad \text{(c-}\|\text{-syn-2)}$$

$$\mathbf{skipp} \underset{X}{\parallel} \mathbf{interpp} = \mathbf{interpp} \qquad \text{(c-}\|\text{-syn-3)}$$

The following expansion laws are used to transform parallel compositions into sequential compositions under specific assumptions, and they will be used later in our case study.

The parallel composition of compensation pairs satisfies the following law.

$$(P_1 \div \mathbf{skip}) \underset{X}{\parallel} (P_2 \div \mathbf{skip}) = (P_1 \underset{X}{\parallel} P_2) \div \mathbf{skip} \qquad \text{(c-}\|\text{-pair-f)}$$

As in the standard processes, we use $\alpha(PP)$ and $\alpha^i(PP) = \alpha^i(fp(PP))$ to respectively denote the sets of all the non-terminal events appearing and first appearing in the compensable process PP. Let $\alpha^c(PP)$ be the set of the events appearing in the compensation behavior of PP. Similar to the law s-‖-assoc-1 of standard processes, we have the following law for compensable processes.

$$(QQ \parallel PP) \underset{X}{\parallel} RR = QQ \parallel (PP \underset{X}{\parallel} RR) \quad if\ \alpha(QQ) \cap X = \{\} \qquad \text{(c-}\|\text{-assoc-1)}$$

Assume $(\alpha(P) \cup \alpha(Q)) \cap X = \{\}$, $\alpha^i(QQ) \subseteq X$, $\alpha^c(QQ) \subseteq (\alpha^c(PP) \cap X)$, P terminates successfully, the compensation behavior of QQ terminates successfully, and there is no deadlock in $PP \underset{X}{\parallel} QQ$. Then laws s-‖-;-head and s-‖-;-tail

of standard processes imply the following law.

$$(P \div Q \ ; \ PP) \underset{X}{\parallel} QQ = P \div Q \ ; \ (PP \underset{X}{\parallel} QQ) \qquad \text{(c-}\parallel\text{-;-head)}$$

If Q terminates successfully, $(\alpha(P) \cup \alpha(Q)) \cap X = \{\}$, $\alpha^c(QQ) \subseteq X$, $fp(QQ)$ terminates successfully, $\alpha^i(QQ) \subseteq (\alpha^i(PP) \cap X)$ and there is no deadlock in $PP \underset{X}{\parallel} QQ$, then the following law holds.

$$(PP \ ; \ P \div Q) \underset{X}{\parallel} QQ = (PP \underset{X}{\parallel} QQ) \ ; \ P \div Q \qquad \text{(c-}\parallel\text{-;-tail)}$$

5 Case study

In this section, we use a business process of a travel agency to demonstrate the synchronized parallel composition, recursion, speculative choice and hiding operators in the extended cCSP. The business behavior of each party in the business process is compensable, and will be specified as a compensable process. We define the processes in the extended cCSP, and apply the calculus to show that the system does not deadlock. We will also show how the calculus is used to extract the compensation behavior of the system, and how a divergence can occur.

5.1 Specification

An Online Travel Agency provides Web Services for booking air tickets, reserving hotel rooms and renting cars. It interacts with business partners including Travelers, Airlines, Hotels, Car Rental Centers and Banks. The business processes are described below.

A traveler makes a request to the Agency for arranging a travel. After receiving the request, the Agency processes it and then starts the air ticket booking, hotel reservation and car rental processes in parallel. Assume the Agency interacts with two airlines to get air tickets. If tickets are available from both airlines, the Agency non-deterministically chooses one. However, it is often the case that the Agency has to send requests to the car rental service provided by the center at the destination repeatedly, until a car is available. If all the three processes succeed the Agency sends the booking information to the traveler and waits for her confirmation. After receiving the confirmation, the Agency makes a request to the bank, according to the information given by the traveler, for the payment service. If this is successful, the Agency sends the completed booking to the traveler, including the reservation details. If an

exception occurs in any of the above steps, the whole business process will be recovered by compensating the steps that have been successful, *e.g.* canceling the air tickets, room reservations or car rentals, and the Agency sends a letter of apology to the traveler. The alphabet Σ of the processes is

$$\begin{aligned}\Sigma = \{&\texttt{reqTravel, letter, reqHotel, okRoom, noRoom,}\\ &\texttt{cancelHotel, bookAir1, okAir1, noAir1, cancelAir1,}\\ &\texttt{bookAir2, okAir2, noAir2, cancelAir2, reqCar, noCar,}\\ &\texttt{hasCar, cancelCar, sendConfirm, agree, disAgree, checkCredit,}\\ &\texttt{valid, inValid, payment, refund, pValid, pInValid, result}\}\end{aligned}$$

We specify the processes Agency, Air1, Air2, Car, Hotel and Bank as follows.

Travel Agency. After receiving a request, the Travel Agency will first reserve the hotel room, book the flight ticket and arrange a car for the client (Res). Then, if the client agrees to what the Agency reserves (Cfm), the Agency will complete the payment (Pay) to finalize this order.

$$\begin{aligned}\textsc{Agency} &= (\texttt{reqTravel} \div \texttt{letter})\ ;\ \textsc{Res}\ ;\ \textsc{Cfm}\ ;\ \textsc{Pay}\ ;\ \texttt{result} \div \texttt{skip}\\ \textsc{Res} &= (\texttt{reqHotel;(okRoom}\Box\texttt{(noRoom;throw)))}\div\texttt{cancelHotel}\|\\ &\quad ((\texttt{bookAir1;(okAir1}\Box\texttt{(noAir1;throw)))}\div\texttt{cancelAir1}\boxtimes\\ &\quad (\texttt{bookAir2;(okAir2}\Box\texttt{(noAir2;throw)))}\div\texttt{cancelAir2)}\|\\ &\quad \mu\ pp.(\texttt{reqCar}\div\texttt{skip;(((noCar}\div\texttt{skip);}pp)\Box\texttt{(hasCar}\div\texttt{cancelCar)))}\\ \textsc{Cfm} &= (\texttt{sendConfirm}\ ;\ (\texttt{agree}\ \Box\ (\texttt{disAgree;throw})))\div\texttt{skip}\\ \textsc{Pay} &= (\texttt{checkCredit}\ ;\ (\texttt{valid}\ \Box\ (\texttt{inValid;throw})))\div\texttt{skip;}\\ &\quad (\texttt{payment}\div\texttt{refund});(\texttt{pValid}\ \Box\ (\texttt{pInValid;throw}))\div\texttt{skip;}\end{aligned}$$

Hotel. The Hotel receives the requests from clients and determines whether there is a room for each client.

$$\textsc{Hotel} = \texttt{reqHotel}\div\texttt{skip;(okRoom}\div\texttt{cancelHotel}\sqcap\texttt{(noRoom}\div\texttt{skip;throww))}$$

Airline. Both airline companies implement the same procedure for providing a ticket services, *i.e.* receiving requests and then determining whether there are tickets available.

$$\begin{aligned}\textsc{Air1} &= \texttt{bookAir1}\div\texttt{skip;(okAir1}\div\texttt{cancelAir1}\sqcap\texttt{(noAir1}\div\texttt{skip;throww))}\\ \textsc{Air2} &= \texttt{bookAir2}\div\texttt{skip;(okAir2}\div\texttt{cancelAir2}\sqcap\texttt{(noAir2}\div\texttt{skip;throww))}\end{aligned}$$

These two processes could be specified by one parameterized process, and our extended cCSP can be extended to support parameterized processes. It is trivial to do so for finite parameter domains.

Car. Any client can request a car multiple times until the client gets a car.

$$\textsc{Car} = \mu\, pp.(\texttt{reqCar} \div \texttt{skip};((\texttt{noCar} \div \texttt{skip});pp) \sqcap (\texttt{hasCar} \div \texttt{cancelCar})))$$

Bank. The Bank will first validate the credit card, and then do the payment.

$$\begin{aligned}\textsc{Bank} = {} & (\texttt{checkCredit};(\texttt{valid} \sqcap (\texttt{inValid};\texttt{throw}))) \div \texttt{skip}\ ; \\ & (\texttt{payment} \div \texttt{refund});(\texttt{pValid} \sqcap (\texttt{pInValid};\texttt{throw})) \div \texttt{skip}\end{aligned}$$

The global business process (GBP) is the transaction block of the synchronized parallel composition of the above six processes.

$$\begin{aligned}\text{GBP} &= [((((\textsc{Agency} \underset{X_1}{\parallel} \textsc{Hotel}) \underset{X_2}{\parallel} \textsc{Air1}) \underset{X_3}{\parallel} \textsc{Air2}) \underset{X_4}{\parallel} \textsc{Car}) \underset{X_5}{\parallel} \textsc{Bank})], \\ X_1 &= \{\texttt{reqHotel}, \texttt{okRoom}, \texttt{noRoom}, \texttt{cancelHotel}\} \\ X_2 &= \{\texttt{bookAir1}, \texttt{okAir1}, \texttt{noAir1}, \texttt{cancelAir1}\} \\ X_3 &= \{\texttt{bookAir2}, \texttt{okAir2}, \texttt{noAir2}, \texttt{cancelAir2}\} \\ X_4 &= \{\texttt{reqCar}, \texttt{noCar}, \texttt{hasCar}, \texttt{cancelCar}\} \\ X_5 &= \{\texttt{checkCredit},\texttt{valid},\texttt{inValid},\texttt{payment},\texttt{refund},\texttt{pValid},\texttt{pInValid}\}\end{aligned}$$

In principle, the process GBP of the online travel agency service can also be composed with other standard processes for constructing a value-added service, *e.g.* the service is used by an office automation workflow of a company.

5.2 Deadlock analysis

We apply the algebraic laws and the formal semantics to transform the process GBP step by step to a compensable process without synchronizations. From such a process we can easily see if it deadlocks or not. The transformation is mainly the application of the (*expansion*) laws for synchronization.

We first transform $\textsc{Agency} \underset{X_1}{\parallel} \textsc{Hotel}$ to an equivalent process without synchronization. For doing this, we consider the process $\textsc{Res} \underset{X_1}{\parallel} \textsc{Hotel}$. We use $\textsc{Rh}$ (Hotel), $\textsc{Ra}$ (Airline) and $\textsc{Rc}$ (Car) to represent the three sub-processes in

Res, *i.e.* Res = Rh ‖ Ra ‖ Rc. We transform Res $\underset{X_1}{\|}$ Hotel as follows.

$$
\begin{aligned}
&\textsc{Res} \underset{X_1}{\|} \textsc{Hotel} \\
=&(\textsc{Rh} \| \textsc{Ra} \| \textsc{Rc}) \underset{X_1}{\|} \textsc{Hotel} && \text{(c-}\|\text{-sym, c-}\|\text{-assoc-1)} \\
=&(\textsc{Ra} \| \textsc{Rc}) \| (\textsc{Rh} \underset{X_1}{\|} \textsc{Hotel})
\end{aligned}
$$

The process Hotel is transformed as follows.

$$
\begin{aligned}
&\textsc{Hotel} \\
=&\texttt{reqHotel} \div \texttt{skip} \ ; \ (\texttt{okRoom} \div \texttt{cancelHotel} \sqcap (\texttt{noRoom} \div \texttt{skip}; \texttt{throww})) \\
& && (\text{def}^{\text{n}} \text{ of } ; \text{ and } \div) \\
=&\texttt{reqHotel} \div \texttt{skip} \ ; \ (\texttt{okRoom} \div \texttt{cancelHotel} \sqcap (\texttt{noRoom}; \texttt{throw}) \div \texttt{skip}) \\
& && (\text{def}^{\text{n}} \text{ of } \sqcap \text{ and } \div) \\
=&\texttt{reqHotel} \div \texttt{skip} \ ; \ (\texttt{okRoom} \sqcap (\texttt{noRoom}; \texttt{throw})) \div \texttt{cancelHotel} \\
& && (\text{def}^{\text{n}} \text{ of } ; \text{ and } \div) \\
=&(\texttt{reqHotel} \ ; \ (\texttt{okRoom} \sqcap (\texttt{noRoom}; \texttt{throw}))) \div \texttt{cancelHotel}
\end{aligned}
$$

By using this result, Rh $\underset{X_1}{\|}$ Hotel is expanded as follows.

$$
\begin{aligned}
&\textsc{Rh} \underset{X_1}{\|} \textsc{Hotel} \\
=&(\texttt{reqHotel} \ ; \ (\texttt{okRoom} \Box (\texttt{noRoom} \ ; \ \textbf{throw}))) \div \texttt{cancelHotel} \underset{X_1}{\|} \\
& (\texttt{reqHotel} \ ; \ (\texttt{okRoom} \sqcap (\texttt{noRoom} \ ; \ \textbf{throw}))) \div \texttt{cancelHotel} && \text{(c-}\|\text{-pair-f)} \\
=&((\texttt{reqHotel} \ ; \ (\texttt{okRoom} \Box (\texttt{noRoom} \ ; \ \textbf{throw}))) \underset{X_1}{\|} \\
& (\texttt{reqHotel} \ ; \ (\texttt{okRoom} \sqcap (\texttt{noRoom} \ ; \ \textbf{throw})))) \div \texttt{cancelHotel} && \text{(s-}\|\text{-step)} \\
=&(\texttt{reqHotel} \ ; \ (\texttt{okRoom} \Box (\texttt{noRoom} \ ; \ \textbf{throw})) \underset{X_1}{\|} \\
& \qquad (\texttt{okRoom} \sqcap (\texttt{noRoom} \ ; \ \textbf{throw}))) \div \texttt{cancelHotel} && \text{(s-}\|\text{-dist and s-}\|\text{-}\Box\text{)} \\
=&(\texttt{reqHotel} \ ; \ (\texttt{okRoom} \sqcap (\texttt{noRoom} \ ; \ \textbf{throw}))) \div \texttt{cancelHotel} \\
=&\textsc{Hotel}
\end{aligned}
$$

From this result, we obtain the following equation.

$$\textsc{Rh} \underset{X_1}{\|} \textsc{Hotel} = \textsc{Hotel} \tag{15}$$

Hence, we can rewrite the process Res $\underset{X_1}{\|}$ Hotel to (Ra ‖ Rc) ‖ Hotel. Apparently, there is no deadlock in Hotel, *i.e.* no deadlock in Res $\underset{X_1}{\|}$ Hotel. By

using this result, we transform the process $\textsc{Agency} \underset{X_1}{\parallel} \textsc{Hotel}$ as follows.

$$
\begin{aligned}
& \textsc{Agency} \underset{X_1}{\parallel} \textsc{Hotel} && (\text{def}^{\text{n}} \text{ of } \textsc{Agency}) \\
=& ((\texttt{reqTravel} \div \texttt{letter}) \,;\, \textsc{Res} \,;\, \textsc{Cfm} \,;\, \textsc{Pay} \,;\, (\texttt{result} \div \texttt{skip})) \underset{X_1}{\parallel} \textsc{Hotel} \\
& && (\text{c-}\|\text{-;-head}) \\
=& (\texttt{reqTravel} \div \texttt{letter}) \,; \\
& (\textsc{Res} \,;\, \textsc{Cfm} \,;\, \textsc{Pay} \,;\, (\texttt{result} \div \texttt{skip})) \underset{X_1}{\parallel} \textsc{Hotel} && (\text{c-}\|\text{-;-tail}) \\
=& (\texttt{reqTravel} \div \texttt{letter}) \,;\, (\textsc{Res} \underset{X_1}{\parallel} \textsc{Hotel}) \,;\, \textsc{Cfm} \,;\, \textsc{Pay} \,;\, (\texttt{result} \div \texttt{skip})
\end{aligned}
$$

As in the above transformation process, we can transform the compensable process $(((\textsc{Agency} \underset{X_1}{\parallel} \textsc{Hotel}) \underset{X_2}{\parallel} \textsc{Air1}) \underset{X_3}{\parallel} \textsc{Air2}) \underset{X_4}{\parallel} \textsc{Car}$ to the following process.

$$
\begin{aligned}
& (\texttt{reqTravel} \div \texttt{letter}) \,;\, ((((\textsc{Res} \underset{X_1}{\parallel} \textsc{Hotel}) \underset{X_2}{\parallel} \textsc{Air1}) \underset{X_3}{\parallel} \textsc{Air2}) \underset{X_4}{\parallel} \textsc{Car}) \,; \\
& \textsc{Cfm} \,;\, \textsc{Pay} \,;\, (\texttt{result} \div \texttt{skip})
\end{aligned}
$$

During the transformation, we need to make use of the following two equations with a proof similar to $\textsc{Rh} \underset{X_1}{\parallel} \textsc{Hotel} = \textsc{Hotel}$.

$$
(\textsc{Ra} \underset{X_2}{\parallel} \textsc{Air1}) \underset{X_3}{\parallel} \textsc{Air2} = \textsc{Air1} \boxtimes \textsc{Air2} \tag{16}
$$

$$
\textsc{Rc} \underset{X_4}{\parallel} \textsc{Car} = \textsc{Car} \tag{17}
$$

In addition, we can transform $(((\textsc{Res} \underset{X_1}{\parallel} \textsc{Hotel}) \underset{X_2}{\parallel} \textsc{Air1}) \underset{X_3}{\parallel} \textsc{Air2}) \underset{X_4}{\parallel} \textsc{Car}$ (denoted by $\textsc{Rhaac}$) to a process without synchronization, *i.e.* no deadlock. The transformation is as follows.

$$
\begin{aligned}
& (((\textsc{Res} \underset{X_1}{\parallel} \textsc{Hotel}) \underset{X_2}{\parallel} \textsc{Air1}) \underset{X_3}{\parallel} \textsc{Air2}) \underset{X_4}{\parallel} \textsc{Car} && (\text{def}^{\text{n}} \text{ of } \textsc{Res}) \\
=& ((((\textsc{Ra} \parallel \textsc{Rc} \parallel \textsc{Rh}) \underset{X_1}{\parallel} \textsc{Hotel}) \underset{X_2}{\parallel} \textsc{Air1}) \underset{X_3}{\parallel} \textsc{Air2}) \underset{X_4}{\parallel} \textsc{Car} && (\text{c-}\|\text{-assoc-1}) \\
=& ((((\textsc{Ra} \parallel \textsc{Rc}) \parallel (\textsc{Rh} \underset{X_1}{\parallel} \textsc{Hotel})) \underset{X_2}{\parallel} \textsc{Air1}) \underset{X_3}{\parallel} \textsc{Air2}) \underset{X_4}{\parallel} \textsc{Car} && (\text{equation (15)}) \\
=& ((((\textsc{Ra} \parallel \textsc{Rc}) \parallel \textsc{Hotel}) \underset{X_2}{\parallel} \textsc{Air1}) \underset{X_3}{\parallel} \textsc{Air2}) \underset{X_4}{\parallel} \textsc{Car} && (\text{c-}\|\text{-assoc, c-}\|\text{-assoc-1}) \\
=& ((\textsc{Rc} \parallel \textsc{Hotel}) \parallel ((\textsc{Ra} \underset{X_2}{\parallel} \textsc{Air1}) \underset{X_3}{\parallel} \textsc{Air2})) \underset{X_4}{\parallel} \textsc{Car} && (\text{equation (16)}) \\
=& ((\textsc{Rc} \parallel \textsc{Hotel}) \parallel (\textsc{Air1} \boxtimes \textsc{Air2})) \underset{X_4}{\parallel} \textsc{Car} && (\text{c-}\|\text{-assoc, c-}\|\text{-assoc-1}) \\
=& (\textsc{Hotel} \parallel (\textsc{Air1} \boxtimes \textsc{Air2})) \parallel (\textsc{Rc} \underset{X_4}{\parallel} \textsc{Car}) && (\text{equation (17)}) \\
=& \textsc{Hotel} \parallel (\textsc{Air1} \boxtimes \textsc{Air2}) \parallel \textsc{Car}
\end{aligned}
$$

Then, in the same way, the compensable process in the process GBP can be transformed to the following process, denoted by CGBP.

$$(\texttt{reqTravel} \div \texttt{letter})\ ;\ ((((\textsc{Res} \underset{X_1}{\|} \textsc{Hotel}) \underset{X_2}{\|} \textsc{Air1}) \underset{X_3}{\|} \textsc{Air2}) \underset{X_4}{\|} \textsc{Car})\ ;$$
$$\textsc{Cfm}\ ;\ (\textsc{Pay} \underset{X_5}{\|} \textsc{Bank})\ ;\ (\texttt{result} \div \texttt{skip}) \tag{18}$$

In the transformation, we use the following equation to indicate no deadlock in $\textsc{Pay} \underset{X_5}{\|} \textsc{Bank}$ with a proof similar to $\textsc{Rh} \underset{X_1}{\|} \textsc{Hotel} = \textsc{Hotel}$.

$$\textsc{Pay} \underset{X_5}{\|} \textsc{Bank} = \textsc{Bank} \tag{19}$$

Now, we can transform the global process GBP

$$\begin{aligned}
&\textsc{GBP} && \text{(Process 18)}\\
=&[(\texttt{reqTravel} \div \texttt{letter})\ ;\ ((((\textsc{Res} \underset{X_1}{\|} \textsc{Hotel}) \underset{X_2}{\|} \textsc{Air1}) \underset{X_3}{\|} \textsc{Air2}) \underset{X_4}{\|} \textsc{Car})\ ;\\
&\quad \textsc{Cfm}; (\textsc{Pay} \underset{X_5}{\|} \textsc{Bank}); (\texttt{result} \div \texttt{skip})] && \text{(result of Rhaac \& equation (19))}\\
=&[(\texttt{reqTravel} \div \texttt{letter})\ ;\ (\textsc{Hotel} \parallel (\textsc{Air1} \boxtimes \textsc{Air2}) \parallel \textsc{Car})\ ;\\
&\quad \textsc{Cfm}\ ;\ \textsc{Bank}\ ;\ (\texttt{result} \div \texttt{skip})]
\end{aligned}$$

Let TGBP be the final transformed process. There is no synchronization in this process, and thus no deadlock.

According to the transformations above, the forward process of Hotel is `reqHotel` ; (`okRoom`$\sqcap$(`noRoom`;`throw`)) (*cf.* Page 31). From the semantic definitions, the forward behavior of the process Agency must perform `reqTravel` at first, we thus can represent the forward process of Agency as `reqTravel` ; P. If we add the event `reqTravel` to the synchronization set X_1, consider the process $\textsc{Agency} \underset{Y}{\|} \textsc{Hotel}$, where $Y = X_1 \cup \{\texttt{reqTravel}\}$. Under the definition of parallel composition, unless both traces are empty traces, the synchronization trace set of a trace from the forward process of Agency and a trace from the forward process of Hotel is empty. Therefore, we only take into account the failures with the empty trace of both forward processes. The set of failures with the empty trace of the forward process of Agency is $\{(\varepsilon, X) \mid X \subseteq \Gamma \setminus \{\texttt{reqTravel}\}\}$, and that of Hotel is $\{(\varepsilon, X) \mid X \subseteq \Gamma \setminus \{\texttt{reqHotel}\}\}$. With respect to the definition of parallel composition, the divergence set of the parallel composition of forward processes is $\{\}$, and the failure set is $\{(\varepsilon, X) \mid X \subseteq \Gamma\}$. Thus, the semantics of the process $\textsc{Agency} \underset{Y}{\|} \textsc{Hotel}$ is.

$$(\{(\varepsilon, X) \mid X \subseteq \Gamma\}, \{\}, \{\}, \{\})$$

That is, a deadlock happens at the beginning, and no terminated trace results from the forward behavior. According to the semantics, this will cause a global deadlock. The reason is that, besides Agency, no other process can execute

the event `reqTravel` at the beginning. However, this is the expected deadlock in an open system that is waiting for activation.

For large applications, the deadlock analysis of an application requires analyzing all the synchronizations in the application. It thus requires tool support.

5.3 Extracting compensation behavior

It is often the case that the main (or typical) course of actions in a LRT is not so difficult to understand, but the execution of the compensation actions is not easy to figure out, because compensation actions are scattered among different points of executions when exceptions occur. In this subsection, we show how to use the hiding operator and the algebraic laws to extract the compensation behavior from the system GBP.

The idea is to hide a set X of actions in the typical course of the process from the specification. In our example, we analyze ABP $\widehat{=}$ GBP $\setminus X$, where X is the set below

$$X = (X_1 \cup X_2 \cup X_3 \cup X_4 \cup X_5 \cup Y) \setminus Z, \quad \text{where}$$
$$Y = \{\texttt{sendConfirm}, \texttt{agree}, \texttt{disAgree}, \texttt{result}, \texttt{noCar}\}$$
$$Z = \{\texttt{payment}, \texttt{refund}, \texttt{cancelHotel}, \texttt{cancelAir1}, \texttt{cancelAir2}, \texttt{cancelCar}\}$$

We take the synchronization free version TGBP of the system that we obtained in the previous subsection. Thus we have

$$\text{ABP} = \text{TGBP}\backslash X \tag{20}$$

Therefore, we have the following algebraic transformations.

$$\begin{aligned}
&\text{ABP} && \text{(equation (20), TGBP and c-lift-}\backslash\text{)}\\
=&[((\texttt{reqTravel} \div \texttt{letter})\ ;\ (\textsc{Hotel} \parallel (\textsc{Air1} \boxtimes \textsc{Air2}) \parallel \textsc{Car})\ ;\\
&\ \textsc{Cfm}\ ;\ \textsc{Bank}\ ;\ (\texttt{result} \div \texttt{skip}))\backslash Y] \\
& && \text{(Hidng Laws, Unit laws, Dist laws and c-}\div\text{-dist-l)}\\
=&[(\texttt{reqTravel} \div \texttt{letter})\ ; && \text{(ABP-Result)}\\
&\ ((\texttt{skip} \div \texttt{cancelHotel} \sqcap \texttt{throww}) \parallel\\
&\ \ ((\texttt{skip} \div \texttt{cancelAir1} \sqcap \texttt{throww}) \boxtimes (\texttt{skip} \div \texttt{cancelAir2} \sqcap \texttt{throww})) \parallel\\
&\ \ (\mu\, pp.(((\texttt{noCar} \div \texttt{skip}); pp) \ \sqcap\ (\texttt{skip} \div \texttt{cancelCar}))))\ ;\\
&\ \texttt{throww} \sqcap \texttt{skipp}\ ;\ \texttt{payment} \div \texttt{refund}\ ;\ \texttt{throww} \sqcap \texttt{skipp}\ ]
\end{aligned}$$

It shows that we can rewrite ABP as follows.

$$\begin{aligned}
\text{ABP} &= [\ \texttt{reqTravel} \div \texttt{letter};\textsc{Cancel};\textsc{PayRefund}\ ]\\
\textsc{Cancel} &= \text{CH} \parallel \text{CA} \parallel \text{CC}\\
\text{CH} &= (\texttt{throw} \sqcap \texttt{skip}) \div \texttt{cancelHotel}\\
\text{CA} &= \text{CA1} \boxtimes \text{CA2}\\
\text{CA1} &= (\texttt{throw} \sqcap \texttt{skip}) \div \texttt{cancelAir1}\\
\text{CA2} &= (\texttt{throw} \sqcap \texttt{skip}) \div \texttt{cancelAir2}\\
\text{CC} &= \mu\ pp.\ ((\texttt{noCar} \div \texttt{skip};pp) \sqcap (\texttt{skip} \div \texttt{cancelCar}))\\
\textsc{PayRefund} &= (\texttt{throww} \sqcap \texttt{skipp});\texttt{payment} \div \texttt{refund};(\texttt{throww} \sqcap \texttt{skipp})
\end{aligned}$$

From the rewritten process ABP, we observe that there are *five* different cases when an exception is raised: 1) no room in the hotel, 2) no ticket from the airlines, 3) the traveler refuses the offer, 4) credit card checking fails, and 5) authorization of payment fails. In each case, there is a different example of recovery because of the parallel composition (Cancel) in the reservation process.

In addition, according to the formal semantics, event `letter` is the last action to be performed in the compensation when an exception occurs. For the process ABP, if the car rental is successful, and there is an exception caused by the failure of payment authorization, the compensation is represented by the process CBP below.

$$\text{CBP}=\texttt{refund};((\texttt{cancelAir1}\sqcap\texttt{cancelAir2})\parallel\texttt{cancelHotel}\parallel\texttt{cancelCar});\texttt{letter}$$

The transformation to get the compensation is as follows.

$$\begin{aligned}
&\text{ABP} \qquad\qquad (\text{ABP-Result and Assump})\\
=&[(\texttt{reqTravel} \div \texttt{letter})\ ;\\
&((\texttt{skip} \div \texttt{cancelHotel}) \parallel ((\texttt{skip} \div \texttt{cancelAir1}) \boxtimes (\texttt{skip} \div \texttt{cancelAir2})) \parallel\\
&(\mu\ pp.(((\texttt{noCar}\div\texttt{skip});pp)\ \sqcap\ (\texttt{skip}\div\texttt{cancelCar}))))\ ;\\
&\texttt{payment}\div\texttt{refund}\ ;\ \texttt{throww}\ ] \qquad (\text{def}^{\text{n}}\text{ of } \boxtimes \text{ and Recursion})\\
=&[(\texttt{reqTravel} \div \texttt{letter})\ ;\\
&((\texttt{skip} \div \texttt{cancelHotel}) \parallel\\
&((\texttt{cancelAir1} \div \texttt{cancelAir2}) \sqcap (\texttt{cancelAir1} \div \texttt{cancelAir2})) \parallel\\
&((\mu\ p.((\texttt{noCar};p)\ \sqcap\ \texttt{skip}))\div\texttt{cancelCar}))\ ;\\
&\texttt{payment}\div\texttt{refund}\ ;\ \texttt{throww}\ ]\\
&\qquad (\text{c-;-dist-l, c-;-dist-r, c-lift-}\sqcap\text{, c-}\parallel\text{-pair, and c-;-pair})\\
=&[(\texttt{reqTravel}\ ;\ (\texttt{cancelAir1}\parallel(\mu\ p.((\texttt{noCar};p)\sqcap\texttt{skip})))\ ;\ \texttt{payment}) \div\\
&(\texttt{refund}\ ;\ (\texttt{cancelAir2}\parallel\texttt{cancelHotel}\parallel\texttt{cancelCar})\ ;\ \texttt{letter})\ ;\ \texttt{throww}\ ]\sqcap\\
&[(\texttt{reqTravel}\ ;\ (\texttt{cancelAir2}\parallel(\mu\ p.((\texttt{noCar};p)\sqcap\texttt{skip})))\ ;\ \texttt{payment}) \div\\
&(\texttt{refund}\ ;\ (\texttt{cancelAir1}\parallel\texttt{cancelHotel}\parallel\texttt{cancelCar})\ ;\ \texttt{letter})\ ;\ \texttt{throww}\ ]
\end{aligned}$$

By using the laws c-Saga-1, s-;-dist-r, s-;-dist-l and s-||-dist, we can extract the compensation behavior to be the standard process CBP. This standard process says: during the recovery process, either `cancelAir1` or `cancelAir2` will be executed for the cancellation of the airline ticket. It matches the speculative choice when booking airline tickets.

5.4 Divergence analysis

For the abstract process ABP in the last subsection, if we hide the event `noCar` from the process, *i.e.* ABP $\setminus \{\texttt{noCar}\}$, the subprocess CC will be as follows.

$$\mu\ pp.\ ((\texttt{skip} \div \texttt{skip};pp) \sqcap (\texttt{skip} \div \texttt{cancelCar}))$$

The resulting process is equal to $\mu\ pp.\ (pp \sqcap (\texttt{skip} \div \texttt{cancelCar}))$. Thus, the compensable process diverges immediately according to the recursion semantics, *i.e.* $\mathbf{div} \div \mathbf{div}$. In the result, a divergence will happen in the process ABP $\setminus \{\texttt{noCar}\}$.

6 Related Work and Conclusions

Before drawing our conclusions, we review a number of related formalisms and discuss their differences from the work presented here.

6.1 Related work

Most LRT formalisms are based on extensions to process algebras.

Extensions to CSP. StAC [7,8] supports synchronized parallel composition, internal and external choices, hiding, programmable compensation and indexed compensation tasks. However, StAC does not support compositional reasoning. Finding a denotational semantics for StAC is still an open problem. To support compositional reasoning, Butler *et al.* propose cCSP [9] that is a subset of StAC, and give it a trace semantics. An operational semantics of cCSP is presented in [10] by Butler and Ripon. The same authors propose in [33] a method for relating these two semantics and show their consistency. Ripon [32] extends cCSP with synchronized parallel composition with a trace semantics.

Extensions to π-calculus. To support mobile computing in LRTs, there are extensions to asynchronous π-calculus. These extensions include πt [2], c-join

[5], webπ [22] and dcπ [38]. They mainly differ in the features they support. Nested LRTs can be modeled in πt, c-join and dcπ, but not in webπ. However, webπ provides facilities to model timed LRTs, and it contains an untimed sub-calculus webπ_∞ [30]. Furthermore, πt, c-join and webπ only support static compensation, *i.e.* the compensation actions of the forward actions are fixed before the execution, but dcπ allows compensation actions to be dynamically composed to forward actions during the execution.

Automata-based formalisms. There is comparatively much less work in automata-based extensions. The reason is probably that an automaton-based formalism for LRTs requires complex hierarchical structures to be expressive enough, but the behavior of such an automaton would be complicated for verification. Nevertheless, in [23], Lanotte *et al.* proposes a model of communicating hierarchical timed automata for sequential and parallel compositions of nested LRTs. In [14], a transactional service interface model is proposed to characterize the interfaces of services that contain LRTs.

Specially built formalisms. There are also formalisms specially defined for LRTs, *e.g.* SAGAs calculi [6], COWS [24] and SOCK [16]. SAGAs calculi define parallel composition and exception handling of LRTs, and allow nested LRTs. The calculi have a big-step semantics and describe different interruption policies, including both centralized and distributed policies. Like cCSP, SAGAs calculi use backward recovery if a failure happens. A dynamic version of SAGAs calculi is proposed in [21], in which the compensation behavior of a component of parallel composition is recorded during the execution of the forward behavior. For example, $a_1; a_2; b_1; b_2$ is an allowed execution of $a_1 \div b_1 \mid a_2 \div b_2$ in SAGAs calculi but not in the dynamic SAGAs calculus. The dynamic SAGAs calculus allows $a_1 \div b_1 \mid a_2 \div b_2$ to have $a_2; a_1; b_1; b_2$ and $a_1; a_2; b_2; b_1$ only when an exception occurs. Most of the calculi developed for Service-Oriented Computing (SOC), such as COWS and SOCK, consider compensation as a basic programming element for SOC. In addition to the above event-based formalisms, He Jifeng proposes a relational semantics of LRTs [17] in the framework of Unifying Theories of Programming [18], to model data functionality and state-based synchronization.

The relation among different formalisms. In [4], Bruni *et al.* compare SAGAs calculi and cCSP in details, and show that a subset of one of them can be encoded as a subset of the other. They also compare the policies of the interruption and compensation in these two formalisms, and prove that the revised SAGAs calculus in [6] is more expressive than cCSP. In [3], the expressive power of interruption and try-catch in CCS processes is studied. There, the complexity is given for determining existential termination or universal termination under different assumptions. In particular, checking the existential termination of recursive CCS processes with interruption is shown to be undecidable. Also, different compensation-based recovery mechanisms

are studied in [20], and it is proved that static recovery is as expressive as parallel recovery, but general dynamic recovery is strictly more expressive than static recovery.

In this paper, we contribute a denotational semantics and a theory of refinement for LRTs. We have extended the original cCSP with hiding, internal choice for non-determinism, synchronized parallel composition for deadlock and recursion for livelock to improve the expressiveness. Compared with the revised SAGAs calculus, the extended cCSP is more expressive in the modeling of basic concurrent features, such as deadlock and livelock, but the revised SAGAs calculus is more expressive in the interruption policies of parallel composition of LRTs.

6.2 Conclusions and future work

LRTs are used in many applications in Service-Oriented Computing to ensure consistency. It is important that LRTs can be formally specified and verified with tool support. The extension of CSP to cCSP is an useful attempt in this direction. However, the original cCSP cannot specify internal choice, hiding, synchronization and recursion. Therefore, it does not provide strong means of abstraction and lacks support for design by refinement. In this paper, the full theory of CSP [34] is extended for specification and verification of LRTs. It allows us to handle the problems of deadlock, livelock and nested LRTs. Its FD semantics supports LRT program design by refinement, and transformations of specifications through algebraic laws. The consistency between the FD semantics with the trace semantics and the operational semantics of cCSP are also discussed. Together with the validity of the algebraic laws and the refinement relation reducing non-determinism, justify the FD semantics.

The theory improves the fundamental understanding of LRTs, and underpins the development of valid tool support to design and verify LRTs. The future work mainly lies in two aspects. Based on the failure-divergence theory of LRTs, we will extend current verification tools for CSP, such as the model checkers FDR [34], PAT [36] and the theorem prover CSP-Prover [19], to support verification of LRTs in the extended cCSP. We are also interested in building an integrated formal technique of component-based modeling, design and analysis of LRTs [29] on top of the failure-divergence semantic foundation.

cCSP provides a notation for an abstract specification of checkpointing and recovery in service-oriented programming paradigm. In addition for abstract specification and verification of LRTs, it can be used in the design of high-level programming languages. This is in contrast to the lower-level formal specification and verification of checkpointing and recovery mechanisms and

algorithms, such as those described in [26,27,28]. It is interesting to study the relation between our FD semantics and a lower level model as an implementation.

Acknowledgements

We would like to thank the anonymous referees for their very constructive and detailed comments that helped us a lot to improve our paper. This work is supported by the projects National 973 project 2011CB302603, NFSC-61103013, NSFC-61120106006, NSFC-90818024, National 863 project 2011AA010106, the Macau Science and Technology Development grants GAVES and SAFEHR, and the Specialized Research Fund for the Doctoral Program of Higher Education 20114307120015. We would also like to thank the following people for the discussions and comments: Cristiano Bertolini, Chris George, Anders P. Ravn, Martin Schäf, Jiri Srba, Saleem Vighio, and Hao Wang.

References

[1] A. Alves, A. Arkin, S. Askary, B. Bloch, F. Curbera, Y. Goland, N. Kartha, Sterling, D. König, V. Mehta, S. Thatte, D. van der Rijn, P. Yendluri, A. Yiu, Web Services Business Process Execution Language Version 2.0, OASIS Committee Draft (May 2006).

[2] L. Bocchi, C. Laneve, G. Zavattaro, A Calculus for Long-Running Transactions, in: E. Najm, U. Nestmann, P. Stevens (eds.), FMOODS, vol. 2884 of Lecture Notes in Computer Science, Springer, 2003.

[3] M. Bravetti, G. Zavattaro, On the expressive power of process interruption and compensation, Mathematical Structures in Computer Science 19 (3) (2009) 565–599.

[4] R. Bruni, M. J. Butler, C. Ferreira, C. A. R. Hoare, H. C. Melgratti, U. Montanari, Comparing Two Approaches to Compensable Flow Composition, in: M. Abadi, L. de Alfaro (eds.), CONCUR, vol. 3653 of Lecture Notes in Computer Science, Springer, 2005.

[5] R. Bruni, H. C. Melgratti, U. Montanari, Nested Commits for Mobile Calculi: Extending Join, in: J.-J. Lévy, E. W. Mayr, J. C. Mitchell (eds.), IFIP TCS, Kluwer, 2004.

[6] R. Bruni, H. C. Melgratti, U. Montanari, Theoretical foundations for compensations in flow composition languages, in: J. Palsberg, M. Abadi (eds.), POPL, ACM, 2005.

[7] M. J. Butler, C. Ferreira, An Operational Semantics for StAC, a Language for Modelling Long-Running Business Transactions, in: R. D. Nicola, G. L. Ferrari, G. Meredith (eds.), COORDINATION, vol. 2949 of Lecture Notes in Computer Science, Springer, 2004.

[8] M. J. Butler, C. Ferreira, M. Y. Ng, Precise Modelling of Compensating Business Transactions and its Application to BPEL, J. UCS 11 (5) (2005) 712–743.

[9] M. J. Butler, C. A. R. Hoare, C. Ferreira, A Trace Semantics for Long-Running Transactions, in: A. E. Abdallah, C. B. Jones, J. W. Sanders (eds.), 25 Years Communicating Sequential Processes, vol. 3525 of Lecture Notes in Computer Science, Springer, 2004.

[10] M. J. Butler, S. Ripon, Executable Semantics for Compensating CSP, in: M. Bravetti, L. Kloul, G. Zavattaro (eds.), EPEW/WS-FM, vol. 3670 of Lecture Notes in Computer Science, Springer, 2005.

[11] G. Castagna, N. Gesbert, L. Padovani, A theory of contracts for web services, in: G. C. Necula, P. Wadler (eds.), POPL, ACM, 2008.

[12] Z. Chen, Z. Liu, An Extended cCSP with Stable Failures Semantics, in: A. Cavalcanti, D. Déharbe, M.-C. Gaudel, J. Woodcock (eds.), ICTAC, vol. 6255 of Lecture Notes in Computer Science, Springer, 2010.

[13] Z. Chen, Z. Liu, J. Wang, Failure-divergence refinement of compensating communicating processes, in: M. Butler, W. Schulte (eds.), FM, vol. 6664 of Lecture Notes in Computer Science, Springer, 2011.

[14] Z. Chen, J. Wang, W. Dong, Z. Qi, Towards Formal Interfaces for Web Services with Transactions, in: E. Damiani, K. Yétongnon, R. Chbeir, A. Dipanda (eds.), SITIS, vol. 4879 of Lecture Notes in Computer Science, Springer, 2006.

[15] H. Garcia-Molina, K. Salem, SAGAS, in: U. Dayal, I. L. Traiger (eds.), SIGMOD Conference, ACM Press, 1987.

[16] C. Guidi, R. Lucchi, R. Gorrieri, N. Busi, G. Zavattaro, SOCK: A Calculus for Service Oriented Computing, in: A. Dan, W. Lamersdorf (eds.), ICSOC, vol. 4294 of Lecture Notes in Computer Science, Springer, 2006.

[17] J. He, UTP Semantics for Web Services, in: J. Davies, J. Gibbons (eds.), IFM, vol. 4591 of Lecture Notes in Computer Science, Springer, 2007.

[18] C. Hoare, H. Jifeng, Unifying Theories of Programming, Prentice Hall, 1998.

[19] Y. Isobe, M. Roggenbach, A Generic Theorem Prover of CSP Refinement, in: N. Halbwachs, L. D. Zuck (eds.), TACAS, vol. 3440 of Lecture Notes in Computer Science, Springer, 2005.

[20] I. Lanese, C. Vaz, C. Ferreira, On the Expressive Power of Primitives for Compensation Handling, in: A. D. Gordon (ed.), ESOP, vol. 6012 of Lecture Notes in Computer Science, Springer, 2010.

[21] I. Lanese, G. Zavattaro, Programming Sagas in SOCK, in: D. V. Hung, P. Krishnan (eds.), SEFM, IEEE Computer Society, 2009.

[22] C. Laneve, G. Zavattaro, Foundations of Web Transactions, in: V. Sassone (ed.), FoSSaCS, vol. 3441 of Lecture Notes in Computer Science, Springer, 2005.

[23] R. Lanotte, A. Maggiolo-Schettini, P. Milazzo, A. Troina, Modeling Long-Running Transactions with Communicating Hierarchical Timed Automata, in: R. Gorrieri, H. Wehrheim (eds.), FMOODS, vol. 4037 of Lecture Notes in Computer Science, Springer, 2006.

[24] A. Lapadula, R. Pugliese, F. Tiezzi, A Calculus for Orchestration of Web Services, in: R. D. Nicola (ed.), ESOP, vol. 4421 of Lecture Notes in Computer Science, Springer, 2007.

[25] M. C. Little, Transactions and Web services, Commun. ACM 46 (10) (2003) 49–54.

[26] Z. Liu, M. Joseph, Transformation of programs for fault-tolerance, Formal Asp. Comput. 4 (5) (1992) 442–469.

[27] Z. Liu, M. Joseph, Specification and verification of recovery in asynchronous communicating systems, in: J. Vytopil (ed.), Formal techniques in real-time and fault-tolerant systems, Kluwer Academic Publishers, 1993, pp. 137–165.

[28] Z. Liu, M. Joseph, Specification and verification of fault-tolerance, timing, and scheduling, ACM Trans. Program. Lang. Syst. 21 (1) (1999) 46–89.

[29] Z. Liu, C. Morisset, V. Stolz, rCOS: Theory and Tool for Component-Based Model Driven Development, in: F. Arbab, M. Sirjani (eds.), FSEN, vol. 5961 of Lecture Notes in Computer Science, Springer, 2009.

[30] M. Mazzara, I. Lanese, Towards a Unifying Theory for Web Services Composition, in: M. Bravetti, M. Núñez, G. Zavattaro (eds.), WS-FM, vol. 4184 of Lecture Notes in Computer Science, Springer, 2006.

[31] R. Milner, A Calculus of Communicating Systems, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

[32] S. Ripon, Extending and Relating Semantic Models of Compensating CSP, Ph.D. thesis, University of Southampton (2008).

[33] S. Ripon, M. J. Butler, PVS embedding of cCSP semantic models and their relationship, Electr. Notes Theor. Comput. Sci. 250 (2) (2009) 103–118.

[34] A. W. Roscoe, The Theory and Practice of Concurrency, Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997.

[35] A. W. Roscoe, The three platonic models of divergence-strict CSP, in: Proc. ICTAC, volume 5160 of Lecture Notes in Computer Science, 2008.

[36] J. Sun, Y. Liu, J. S. Dong, Model Checking CSP Revisited: Introducing a Process Analysis Toolkit, in: T. Margaria, B. Steffen (eds.), ISoLA, vol. 17 of Communications in Computer and Information Science, Springer, 2008.

[37] S. Thatte, XLANG Web Services for Business Process Design (2001).

[38] C. Vaz, C. Ferreira, A. Ravara, Dynamic Recovering of Long Running Transactions, in: C. Kaklamanis, F. Nielson (eds.), TGC, vol. 5474 of Lecture Notes in Computer Science, Springer, 2008.

Appendix A - Proof of theorems

Proof of Theorem 1. The extra two terminals do not affect the properties of the FD semantic domain. Using the finiteness of the alphabet Σ, the CPO Theorem 8.3.1 in [34] for the FD semantic model of classical CSP extends to the standard processes here. □

Lemma 1 *The semantic domain of compensable processes is a partial order w.r.t.* $\sqsubseteq_c$.

Proof. This follows from the fact that the superset relation ($\supseteq$) is a partial order. □

Proof of Theorem 2 From Lemma 1, we get that the semantic domain is a partial order under $\sqsubseteq_c$. It is straightforward to check that the least element is $\mathbf{div} \div \mathbf{div}$. Now, we need to prove that each *directed set* of semantic models has a least upper bound. For any directed semantic model set S, we define the following semantic model PP_g:

$$(F_g, D_g, F_g^c, D_g^c) =_{df} (\bigcap\{F \mid (F, D, F^c, D^c) \in S\}, \bigcap\{D \mid (F, D, F^c, D^c) \in S\},$$
$$\bigcap\{F^c \mid (F, D, F^c, D^c) \in S\}, \bigcap\{D^c \mid (F, D, F^c, D^c) \in S\})$$

PP_g is the least upper bound for S. From the definition of refinement (*cf.* Equation 14), it is apparent that PP_g refines any element in S. So PP_g is a upper bound for S. For each upper bound $PP_b = (F_b, D_b, F_b^c, D_b^c)$ of S, because each element in S is refined by PP_b, we get the following relations:

$$\bigcap\{F \mid (F, D, F^c, D^c) \in S\} \supseteq F_b \wedge \bigcap\{D \mid (F, D, F^c, D^c) \in S\} \supseteq D_b \wedge$$
$$\bigcap\{F^c \mid (F, D, F^c, D^c) \in S\} \supseteq F_b^c \wedge \bigcap\{D^c \mid (F, D, F^c, D^c) \in S\} \supseteq D_b^c$$

So we have $PP_g \sqsubseteq_c PP_b$ with respect to the refinement definition. Therefore, PP_g is the least upper bound. Based on the Theorem 8.3.1 in [34] and the axioms for compensable behavior, we can proof that PP_g satisfies all the axioms (*cf.* Section 3.2) of the semantic model of compensable processes. □

Using the semantic definitions of transaction block, compensation pair and refinement, Theorem 3 can be proved based on set theory as follows.

Law c-lift-$\sqsubseteq_c$: $PP_1 \sqsubseteq_c PP_2 \Rightarrow [PP_1] \sqsubseteq [PP_2]$

Proof.

$$
\begin{aligned}
&\mathcal{D}([PP_2]) && (\text{def}^{\text{n}} \text{ of } [PP])\\
=&\mathcal{D}_f(PP_2) \cup \{s_1{\cdot}s_2 \mid (s, s_2) \in \mathcal{D}_c(PP_2) \wedge s = s_1{\cdot}\langle !\rangle\} && (\text{Assump and Set theory})\\
=&\mathcal{D}_f(PP_1) \cup \{s_1{\cdot}s_2 \mid (s, s_2) \in \mathcal{D}_c(PP_1) \wedge s = s_1{\cdot}\langle !\rangle\} && (\text{def}^{\text{n}} \text{ of } [PP])\\
=&\mathcal{D}([PP_1])
\end{aligned}
$$

Similarly, we can prove $\mathcal{F}([PP_2]) \subseteq \mathcal{F}([PP_1])$. Hence the law holds. □

Law c-link-$\sqsubseteq_c$-$\sqsubseteq$-f: $P_1 \sqsubseteq P_2 \Rightarrow P_1 \div Q \sqsubseteq_c P_2 \div Q$

Proof. By the definition of $P_1 \sqsubseteq P_2$, *i.e.* $\mathcal{D}(P_2) \subseteq \mathcal{D}(P_1) \wedge \mathcal{F}(P_2) \subseteq \mathcal{F}(P_1)$, recall the semantic definition of the compensation pair, and we have

$$
\begin{aligned}
\mathcal{D}_f(P_2 \div Q) =& \mathcal{D}(P_2) \subseteq \mathcal{D}(P_1) = \mathcal{D}_f(P_1 \div Q)\\
\mathcal{D}_c(P_2 \div Q) =& ((trace_t(P_2) \cap \Sigma^{\star}_{\{\checkmark\}}) \times \mathcal{D}(Q)) \cup ((trace_t(P_2) \cap \Sigma^{\star}_{\{!,?\}}) \times \mathcal{D}(\mathbf{skip}))\\
\subseteq& ((trace_t(P_1) \cap \Sigma^{\star}_{\{\checkmark\}}) \times \mathcal{D}(Q)) \cup ((trace_t(P_1) \cap \Sigma^{\star}_{\{!,?\}}) \times \mathcal{D}(\mathbf{skip}))\\
=& \mathcal{D}_c(P_1 \div Q)
\end{aligned}
$$

The subset relations $\mathcal{F}_f(P_2 \div Q) \subseteq \mathcal{F}_f(P_1 \div Q)$ and $\mathcal{F}_c(P_2 \div Q) \subseteq \mathcal{F}_c(P_1 \div Q)$ can be proved in the same way. Thus the law holds. □

Similarly, we can prove the validity of $Q_1 \sqsubseteq Q_2 \Rightarrow P \div Q_1 \sqsubseteq_c P \div Q_2$. Totally, Theorem 3 is proved.

Proof of Theorem 4 Based on the continuity Lemmas 8.3.4 and 8.3.5 in [34], Theorem 3 and assumptions, because we restrict the using of a standard recursion variable inside a transaction block, the proof of this theorem is just different from that of classical CSP in [34] on the exception handling operator ($\triangleright$). From the definition of exception handling, we can see that it differs from sequential composition in the condition for running the second process. Because sequential composition is continuous, it is straightforward to prove that exception handling is also continuous by using the same steps (*cf.* Lemmas 8.3.3 and 8.3.4 in [34]). □

Proof of Theorem 5 We need to use Theorem 4, the semantics definition of each operator in compensable processes, the refinement definition and the finiteness of the alphabet to prove this theorem. Notice that only finite nondeterminism is allowed in the extended cCSP according to the syntax in Figure 3. The idea is the same as that in [34]. We first prove the continuity of the operators except hiding by the finiteness of Σ, then use the finiteness of nondeterminism to prove the continuity of hiding.

- For the continuity of operators other than hiding, we can first prove the full distributivity (*cf.* Lemma 8.3.3 in [34]) of each operator over internal choice using the semantic definition and the refinement definition, and then use

finiteness of the alphabet Σ to induce the continuity of each operator (*cf.* Lemma 8.3.4 in [34]).

- For the continuity of the hiding operator, the steps are basically the same as those of Lemma 8.3.5 in [34], in which König's lemma and the finiteness of non-determinism are used.

□

Appendix B - Proof of the laws in standard processes

Law s-□-terminal-2: $\mathbf{skip}\Box\mathbf{throw} = \mathbf{skip} \sqcap \mathbf{throw}$

Proof.

$$
\begin{aligned}
&\mathcal{D}(\mathbf{skip}\Box\mathbf{throw}) = \mathcal{D}(\mathbf{skip})\cup\mathcal{D}(\mathbf{throw}) && (\text{def}^{\text{n}} \text{ of } \sqcap, \Box)\\
=&\mathcal{D}(\mathbf{skip}\sqcap\mathbf{throw})
\end{aligned}
$$

$$
\begin{aligned}
&\mathcal{F}(\mathbf{skip}\Box\mathbf{throw}) && (\text{def}^{\text{n}} \text{ of } \Box)\\
=&\{(\varepsilon, X)|X\subseteq\Gamma\setminus\{\checkmark,!\}\}\cup\{(\checkmark, X)|X\subseteq\Gamma\}\cup\\
&\{(!, X)|X\subseteq\Gamma\}\cup\{(\varepsilon, X)|X\subseteq\Gamma\setminus\{\checkmark\}\}\cup\{(\varepsilon, X)|X\subseteq\Gamma\setminus\{!\}\} && (\text{Set Theory})\\
=&\{(\varepsilon, X)|X\subseteq\Gamma\setminus\{\checkmark\}\}\cup\{(\checkmark, X)|X\subseteq\Gamma\}\cup\\
&\{(\varepsilon, X)|X\subseteq\Gamma\setminus\{!\}\}\cup\{(!, X)|X\subseteq\Gamma\} && (\text{def}^{\text{n}} \text{ of } \mathbf{skip})\\
=&\mathcal{F}(\mathbf{skip})\cup\{(\varepsilon, X)|X\subseteq\Gamma\setminus\{!\}\}\cup\{(!, X)|X\subseteq\Gamma\} && (\text{def}^{\text{n}} \text{ of } \mathbf{throw})\\
=&\mathcal{F}(\mathbf{skip})\cup\mathcal{F}(\mathbf{throw}) && (\text{def}^{\text{n}} \text{ of } \sqcap)\\
=&\mathcal{F}(\mathbf{skip}\sqcap\mathbf{throw})
\end{aligned}
$$

Hence, $\mathbf{skip}\Box\mathbf{throw} = \mathbf{skip}\sqcap\mathbf{throw}$ holds. □

We can prove the law s-□-terminal-3 and the equation $\mathbf{skip}\Box\mathbf{interp} = \mathbf{skip}\sqcap\mathbf{interp}$ in a similar way. The latter implies the law s-□-terminal-1.

Law s-‖-syn-1: $\mathbf{skip} \underset{X}{\parallel} \mathbf{interp} = \mathbf{interp}$

Proof. From the semantic definitions of the basic processes and Equation 8, we have $\mathcal{D}(\mathbf{skip} \underset{X}{\parallel} \mathbf{interp}) = \{\} = \mathcal{D}(\mathbf{interp})$. The failure sets of **skip** and **interp** are both unions of two parts, with the non-terminated and terminated traces, respectively.

$$
\begin{aligned}
\mathcal{F}(\mathbf{skip}) &= \{(\varepsilon, X) \mid X \subseteq \Gamma \wedge \checkmark \notin X\} \cup \{(\checkmark, X) \mid X \subseteq \Gamma\}\\
\mathcal{F}(\mathbf{interp}) &= \{(\varepsilon, X) \mid X \subseteq \Gamma \wedge ? \notin X\} \cup \{(?, X) \mid X \subseteq \Gamma\}
\end{aligned}
$$

Consider the synchronization of an element $(s, Y) \in \{(\varepsilon, X) \mid X \subseteq \Gamma \wedge \checkmark \notin X\}$ and an element $(t, Z) \in \{(\varepsilon, X) \mid X \subseteq \Gamma \wedge ? \notin X\}$ from the first parts. We obtain the failures in $\{(\varepsilon, X) \mid X \subseteq \Gamma \wedge ? \notin X\}$ from the definition of $(s, Y) \underset{X}{\parallel} (t, Z)$ by Equation 9, because $\Theta(\checkmark, ?) = ?$. Similarly, the synchronization of the elements

of the second parts of **skip** and **interp** gives the failures $\{(?, X) \mid X \subseteq \Gamma\}$. Finally, the synchronization of the failures with terminated traces of one of the two processes with the failures with non-terminated traces of the other results in the empty set. Hence,

$$\begin{aligned}\mathcal{F}(\mathbf{skip} \underset{X}{\parallel} \mathbf{interp}) &= \{(\varepsilon, X) \mid X \subseteq \Gamma \wedge ? \notin X\} \cup \{(?, X) \mid X \subseteq \Gamma\} \\ &= \mathcal{F}(\mathbf{interp})\end{aligned}$$

Therefore the law holds. □

In the same way, we can prove the following two laws: s-‖-syn-2 and s-‖-syn-3.

$$\mathbf{throw} \underset{X}{\parallel} \mathbf{skip} = \mathbf{throw} \quad \mathbf{throw} \underset{X}{\parallel} \mathbf{interp} = \mathbf{throw}$$

Law s-‖-throw: If P does not terminate with the yield terminal event, *i.e.* $\forall s \in \mathit{trace}_n(P) \bullet s \notin \Sigma^{\bigstar}_{\{?\}}$, then

$$\mathbf{throw} \parallel P = P \ ; \ \mathbf{throw}$$

Proof. We prove the equivalence between divergence sets as follows, where we use the equation $\varepsilon \parallel t = \{t\}$ if t is a non-terminated trace.

$$\begin{aligned}
&\mathcal{D}(\mathbf{throw} \parallel P) && (\text{def}^{\text{n}} \text{ of } \underset{X}{\parallel}) \\
=&\{u{\cdot}v \mid v \in \Sigma^{\circledast}, \exists s \in \mathit{traces}_{\perp}(P), t \in \mathit{traces}_{\perp}(\mathbf{throw}) \bullet \\
&\ u \in (s \parallel t) \cap \Sigma^* \wedge (s \in \mathcal{D}(P) \vee t \in \mathcal{D}(\mathbf{throw}))\} && (\text{def}^{\text{n}} \text{ of } \mathbf{throw}) \\
=&\{u{\cdot}v \mid v \in \Sigma^{\circledast}, \exists s \in \mathit{traces}_{\perp}(P) \bullet u \in (s \parallel \varepsilon) \cap \Sigma^* \wedge s \in \mathcal{D}(P)\} && (\text{def}^{\text{n}} \text{ of } \underset{X}{\parallel}) \\
=&\{u{\cdot}v \mid v \in \Sigma^{\circledast} \wedge u \in \mathcal{D}(P) \cap \Sigma^*\} && (\mathsf{Axiom}) \\
=&\mathcal{D}(P) && (\mathcal{D}(\mathbf{throw}) = \{\}) \\
=&\mathcal{D}(P) \cup \{s{\cdot}t \mid s{\cdot}\checkmark \in \mathit{traces}_{\perp}(P) \wedge t \in \mathcal{D}(\mathbf{throw})\} && (\text{def}^{\text{n}} \text{ of } ;) \\
=&\mathcal{D}(P \ ; \ \mathbf{throw})
\end{aligned}$$

For the equivalence between failure sets, we divide the failure sets of **throw** and P into the non-terminated and terminated traces respectively.

$$\begin{aligned}
&\mathcal{F}(\mathbf{throw}) = \{(\varepsilon, X) \mid X \subseteq \Gamma \wedge ! \notin X\} \cup \{(!, X) \mid X \subseteq \Gamma\} \\
&\mathcal{F}(P) = \{(s, X) \mid (s, X) \in \mathcal{F}(P) \wedge s \in \Sigma^*\} \cup \{(s, X) \mid (s, X) \in \mathcal{F}(P) \wedge s \in \Sigma^{\bigstar}\}
\end{aligned}$$

Similar to the above proof of the law s-‖-syn-1, we get the following failure set of the synchronization.

$$\begin{aligned}
&\{(s, X) \mid (s, X \cup \Omega) \in \mathcal{F}(P) \wedge s \in \Sigma^*\} \cup \\
&\{(s, (X \cup \Omega) \setminus \{!\}) \mid (s, X) \in \mathcal{F}(P) \wedge (s, X \cup \Omega) \notin \mathcal{F}(P) \wedge s \in \Sigma^*\} \cup \\
&\{(s_1{\cdot}!, X) \mid (s_1{\cdot}\omega, X) \in \mathcal{F}(P) \wedge \omega \in \Omega \wedge X \subseteq \Gamma\}
\end{aligned}$$

This set is the result of $\{(u,E)|\exists(s,Y)\in\mathcal{F}(\mathbf{throw}),(t,Z)\in\mathcal{F}(P)\bullet(u,E)\in(s,Y)\|(t,Z)\}$, and is equal to the following set according to the assumption and set theory.

$$\{(s,X) \mid s\in\Sigma^{\circledast}_{\{!\}}\wedge(s,X\cup\{\checkmark\})\in\mathcal{F}(P)\}\cup$$
$$\{(s{\cdot}t,X) \mid s{\cdot}\checkmark\in traces_{\perp}(P)\wedge(t,X)\in\mathcal{F}(\mathbf{throw})\}$$

From the failure definitions of parallel and sequential compositions, this equivalence, and the already proved equivalence relation between $\mathcal{D}(P\parallel\mathbf{throw})$ and $\mathcal{D}(P\ ;\ \mathbf{throw})$, we draw the conclusion: $\mathcal{F}(\mathbf{throw}\parallel P)=\mathcal{F}(P\ ;\ \mathbf{throw})$. □

Law s-‖-throw-yield: If P does not terminate with the yield terminal event, *i.e.* $\forall s\in trace_n(P)\bullet s\notin\Sigma^{\bigstar}_{\{?\}}$, then

$$\mathbf{throw}\parallel(\mathbf{yield}\ ;\ P)=\mathbf{throw}\sqcap(P\ ;\ \mathbf{throw})$$

Proof.

$$\begin{array}{ll}
\mathbf{throw}\parallel(\mathbf{yield}\ ;\ P) & (\text{def}^{\text{n}}\text{ of }\mathbf{yield})\\
=\mathbf{throw}\parallel((\mathbf{skip}\sqcap\mathbf{interp})\ ;\ P) & (\text{s-;-dist-l})\\
=\mathbf{throw}\parallel((\mathbf{skip}\ ;\ P)\sqcap(\mathbf{interp}\ ;\ P)) & (\text{s-;-unit-l and s-;-zero-2})\\
=\mathbf{throw}\parallel(P\sqcap\mathbf{interp}) & (\text{s-}\|\text{-dist})\\
=(\mathbf{throw}\parallel P)\sqcap(\mathbf{throw}\parallel\mathbf{interp}) & (\text{s-}\|\text{-throw and s-}\|\text{-syn-3})\\
=(P\ ;\ \mathbf{throw})\sqcap\mathbf{throw} & (\text{s-}\sqcap\text{-sym})\\
=\mathbf{throw}\sqcap(P\ ;\ \mathbf{throw}) &
\end{array}$$

□

Law s-‖-□: If P and Q are constructed from basic processes using the sequential operator, $\alpha(P)\cup\alpha(Q)\subseteq X$, $\alpha(P)\cap\alpha(Q)=\{\}$ and $\alpha(Q)\neq\{\}$, then

$$(P\Box Q)\underset{X}{\parallel}P=P$$

Proof. From the assumptions, we know that P and Q do not diverge, *i.e.* $\mathcal{D}(P)=\{\}=\mathcal{D}(Q)$, and $\{u\mid\exists s{\in}traces_{\perp}(Q),t{\in}traces_{\perp}(P)\bullet u{\in}s\underset{X}{\parallel}t\}{=}\{\varepsilon\}$. Therefore, it is easy to prove $\mathcal{D}((P\Box Q)\underset{X}{\parallel}P)$ is equal to $\mathcal{D}(P)$, *i.e.* $\{\}$. For the equality between failure sets, we divide the failures of $P\Box Q$ into two sets, with the empty trace and with non-empty traces.

$$\{(\varepsilon,X)\mid(\varepsilon,X)\in\mathcal{F}(P)\cap\mathcal{F}(Q)\}\cup$$
$$\{(\varepsilon,X)\mid X\subseteq\Gamma\setminus\{\omega\}\wedge\omega\in traces_{\perp}(P)\cup traces_{\perp}(Q)\wedge\omega\in\Omega\}\cup$$
$$\{(s,X)\mid(s,X)\in\mathcal{F}(P)\cup\mathcal{F}(Q)\wedge s\neq\varepsilon\}$$

From the assumptions, Q dose not terminate initially. We discuss P in two cases:

- If P terminates initially, then the failure set of $P\Box Q$ is equal to

$$\mathcal{F}(P)\cup\{(s,X)\mid(s,X)\in\mathcal{F}(Q)\wedge s\neq\varepsilon\}$$

For any element (s, X) in the second part, because (s, X) is a failure of Q and $s \neq \varepsilon$ and the assumption $\alpha(P) \cap \alpha(Q) = \{\}$, the failure set of its synchronization with any element (t, Y) in $\mathcal{F}(P)$, *i.e.* $(s, X) \underset{X}{\parallel} (t, Y)$, is empty. Therefore, the failures of $P \Box Q$ that can produce synchronization failures are only from P, *i.e.* $(P \Box Q) \underset{X}{\parallel} P = P \underset{X}{\parallel} P$. The law s-‖-idem gives $(P \Box Q) \underset{X}{\parallel} P = P$.

- If P does not terminate initially, the failure set of $P \Box Q$ is

$$\{(\varepsilon, X) \mid (\varepsilon, X) \in \mathcal{F}(P) \cap \mathcal{F}(Q)\} \cup \{(s, X) \mid (s, X) \in \mathcal{F}(P) \wedge s \neq \varepsilon\} \cup \{(s, X) \mid (s, X) \in \mathcal{F}(Q) \wedge s \neq \varepsilon\}$$

As in the first case, the third part of the above set does not contribute to the failure set of synchronizations. Thus we consider the first two parts only. For the first set $\{(\varepsilon, X) \mid (\varepsilon, X) \in \mathcal{F}(P) \cap \mathcal{F}(Q)\}$, according to the semantics definition, its synchronization with the failures of empty trace in $\mathcal{F}(P)$, *i.e.* $\{(\varepsilon, X) \mid (\varepsilon, X) \in \mathcal{F}(P)\}$, is still $\{(\varepsilon, X) \mid (\varepsilon, X) \in \mathcal{F}(P)\}$; its synchronization set with the failures of non-empty trace in $\mathcal{F}(P)$ is empty. For the second set $\{(s, X) \mid (s, X) \in \mathcal{F}(P) \wedge s \neq \varepsilon\}$, its synchronization with the failures of empty trace in $\mathcal{F}(P)$ is empty; its synchronization set with the failures of non-empty trace in $\mathcal{F}(P)$, is itself, *i.e.* $\{(s, X) \mid (s, X) \in \mathcal{F}(P) \wedge s \neq \varepsilon\}$. Therefore, $\mathcal{F}((P \Box Q) \underset{X}{\parallel} P)$ is $\{(\varepsilon, X) \mid (\varepsilon, X) \in \mathcal{F}(P)\} \cup \{(s, X) \mid (s, X) \in \mathcal{F}(P) \wedge s \neq \varepsilon\}$, *i.e.* $\mathcal{F}(P)$.

Thus the law is proved. □

Law s-‖-;-head: If P terminates successfully, $\alpha(P) \cap X = \{\}$ and $\alpha^i(R) \subseteq X$, then

$$(P \ ; \ Q) \underset{X}{\parallel} R = P \ ; \ (Q \underset{X}{\parallel} R)$$

Proof. Let $a \in X$ and $\omega \in \Omega$. If there is no event in X appearing in s, we have

$$(s \cdot t) \underset{X}{\parallel} a \cdot r = \{s \cdot u \mid u \in (t \underset{X}{\parallel} a \cdot r)\} \qquad \text{(assit-1)}$$
$$(s \cdot t) \underset{X}{\parallel} \omega = \{s \cdot u \mid u \in (t \underset{X}{\parallel} \omega)\}$$

From this, we prove the equivalence between the divergence sets.

$$\begin{aligned}
& \mathcal{D}((P \ ; \ Q) \underset{X}{\parallel} R) && (\text{def}^{\text{n}} \text{ of } \underset{X}{\parallel}) \\
=& \{u \cdot v \mid v \in \Sigma^{\circledast}, \exists s \in \mathit{traces}_{\perp}(P \ ; \ Q), t \in \mathit{traces}_{\perp}(Q) \bullet \\
& u \in (s \underset{X}{\parallel} t) \cap \Sigma^* \wedge (s \in \mathcal{D}(P \ ; \ Q) \vee t \in \mathcal{D}(R))\} && (\text{def}^{\text{n}} \text{ of } ; \text{ and Set theory}) \\
=& \{u \cdot v \mid v \in \Sigma^{\circledast}, \exists s \in \mathit{traces}_{\perp}(P) \cap \Sigma^*, t \in \mathit{traces}_{\perp}(Q) \bullet \\
& u \in (s \underset{X}{\parallel} t) \cap \Sigma^* \wedge (s \in \mathcal{D}(P \ ; \ Q) \vee t \in \mathcal{D}(R))\} \cup
\end{aligned}$$

$$\{u{\cdot}v \mid v \in \Sigma^{\circledast}, \exists s{\cdot}\checkmark \in traces_{\perp}(P), t \in traces_{\perp}(Q), r \in traces_{\perp}(R)\bullet$$
$$u \in ((s{\cdot}t) \underset{X}{\parallel} r) \cap \Sigma^* \wedge (s{\cdot}t \in \mathcal{D}(P\ ;\ Q) \vee r \in \mathcal{D}(R))\}$$

(Assump, Trace Sync and assit-1)

$$= \mathcal{D}(P) \cup$$
$$\{s{\cdot}u{\cdot}v \mid v \in \Sigma^{\circledast}, \exists s{\cdot}\checkmark \in traces_{\perp}(P), t \in traces_{\perp}(Q), r \in traces_{\perp}(R)\bullet$$
$$u \in (t \underset{X}{\parallel} r) \cap \Sigma^* \wedge (t \in \mathcal{D}(Q) \vee r \in \mathcal{D}(R))\}$$

(def$^{\text{n}}$ of $\underset{X}{\parallel}$ and Set theory)

$$= \mathcal{D}(P) \cup \{s{\cdot}t \mid s{\cdot}\checkmark \in traces_{\perp}(P) \wedge t \in \mathcal{D}(Q \underset{X}{\parallel} R)\}$$
$$= \mathcal{D}(P\ ;\ (Q \underset{X}{\parallel} R))$$

By the assumptions and the equations in assit-1, we can prove in a similar way the equality between the failure sets of the processes on the two sides of the law. □

Law s-‖-;-tail: If R terminates successfully, $\alpha(P) \cap X = \{\}$, $\alpha(R) \subseteq (\alpha(Q) \cap X)$, and there is no deadlock in $Q \underset{X}{\parallel} R$, then

$$(Q\ ;\ P) \underset{X}{\parallel} R = (Q \underset{X}{\parallel} R)\ ;\ P$$

Proof. The equivalence between the divergence sets can be proved as in the preceding proof of the law s-‖-;-head using the following result on traces, where all the non-terminal events appearing in r are contained in X, and there is no event in X appearing in t.

$$s{\cdot}t \underset{X}{\parallel} r = \{u{\cdot}t \mid u \in (s \underset{X}{\parallel} r)\} \qquad \text{(assit-2)}$$

For the failure sets, we prove the inclusion $\mathcal{F}((Q\ ;\ P) \underset{X}{\parallel} R) \subseteq \mathcal{F}((Q \underset{X}{\parallel} R)\ ;\ P)$. Inclusion in the opposite direction is proved in the same manner. The proof is as follows, where we do not consider divergence, and the failures are those with maximal refusal sets.

For a failure (s, X) in the failure set of $(Q\ ;\ P) \underset{X}{\parallel} R$, there must exist a failures (t, Y) of $Q\ ;\ P$ and a failure (r, Z) of R, such that $(s, X) \in (t, Y) \underset{X}{\parallel} (r, Z)$. Because (t, Y) is a failure of $Q\ ;\ P$, there are two cases: 1) $(t, Y) \in \mathcal{F}(Q)$; 2) $(t, Y) = (t_1{\cdot}t_2, Y)$, where $t_1{\cdot}\checkmark \in traces_{\perp}(Q)$ and $(t_2, Y) \in P$.

- For the first case, we have $(t, Y) \underset{X}{\parallel} (r, Z) \subseteq \mathcal{F}(Q \underset{X}{\parallel} R)$, and it is easy to prove: $(t, Y) \underset{X}{\parallel} (r, Z) \subseteq \mathcal{F}((Q \underset{X}{\parallel} R)\ ;\ P)$. Therefore, (s, X) belongs to the failure set of $(Q \underset{X}{\parallel} R)\ ;\ P$.

- For the second case, $(t, Y) = (t_1{\cdot}t_2, Y)$, where $t_1{\cdot}\checkmark \in traces_\perp(Q)$ and $(t_2, Y) \in P$, according to the assumptions: $\alpha(R) \subseteq (\alpha(Q) \cap X)$, R terminates successfully, there is no deadlock in $Q \underset{X}{\parallel} R$ and the equation assit-2, we have: $s \in \{u_1{\cdot}t_2 \mid u_1 \in (t_1 \underset{X}{\parallel} r)\}$ and $X = Y$. From the semantic definitions of parallel and sequential compositions, we can easily prove $(s, X) \in \mathcal{F}((Q \underset{X}{\parallel} R); P)$.

□

Appendix C - Proof of the laws in compensable processes

Law c-lift-÷: $[P \div Q] = P \rhd \mathbf{skip}$

Proof.

$$
\begin{array}{ll}
\quad \mathcal{D}([P \div Q]) & (\text{def}^{\text{n}} \text{ of } [PP]) \\
= \mathcal{D}_f(P \div Q) \cup \{s_1{\cdot}s_2 \mid (s, s_2) \in \mathcal{D}_c(P \div Q) \wedge s = s_1{\cdot}!\} & (\text{def}^{\text{n}} \text{ of } P \div Q) \\
= \mathcal{D}(P) \cup \{s_1{\cdot}s_2 \mid s_1{\cdot}! \in trace_t(P) \wedge s_2 \in \mathcal{D}(\mathbf{skip})\} & (\text{def}^{\text{n}} \text{ of } P \rhd Q) \\
= \mathcal{D}(P \rhd \mathbf{skip}) &
\end{array}
$$

$$
\begin{array}{ll}
\quad \mathcal{F}([P \div Q]) & (\text{def}^{\text{n}} \text{ of } [PP]) \\
= \{(s, X) \mid s \in \Sigma^{\circledast}_{\{\checkmark,?\}} \wedge (s, X \cup \{!\}) \in \mathcal{F}_f(P \div Q)\} \cup & \\
\quad \{(s_1{\cdot}s_2, X) \mid (s, s_2, X) \in \mathcal{F}_c(P \div Q) \wedge s = s_1{\cdot}!\} \cup & \\
\quad \{(s, X) \mid s \in \mathcal{D}([P \div Q]) \wedge X \subseteq \Gamma\} & (\text{def}^{\text{n}} \text{ of } P \div Q) \\
= \{(s, X) \mid s \in \Sigma^{\circledast}_{\{\checkmark,?\}} \wedge (s, X \cup \{!\}) \in \mathcal{F}(P)\} \cup & \\
\quad \{(s{\cdot}t, X) \mid s{\cdot}! \in traces_\perp(P) \wedge (t, X) \in \mathcal{F}(\mathbf{skip})\} \cup & \\
\quad \{(s, X) \mid s \in \mathcal{D}(P \rhd \mathbf{skip}) \wedge X \subseteq \Gamma\} & (\text{def}^{\text{n}} \text{ of } P \rhd Q) \\
= \mathcal{F}(P \rhd \mathbf{skip}) &
\end{array}
$$

This proves the law. □

Law c-lift-⊓: $[PP \sqcap QQ] = [PP] \sqcap [QQ]$

Proof.

$$
\begin{array}{ll}
\quad \mathcal{D}([PP \sqcap QQ]) & (\text{def}^{\text{n}} \text{ of } [PP]) \\
= \mathcal{D}(PP \sqcap QQ) \cup & \\
\quad \{s_1{\cdot}s_2 \mid (s, s_2) \in \mathcal{D}_c(PP \sqcap QQ) \wedge s = s_1{\cdot}!\} & (\text{def}^{\text{n}} \text{ of } \sqcap \text{ and Set theory}) \\
= (\mathcal{D}(PP) \cup \{s_1{\cdot}s_2 \mid (s, s_2) \in \mathcal{D}_c(PP) \wedge s = s_1{\cdot}!\}) \cup & \\
\quad (\mathcal{D}(QQ) \cup \{s_1{\cdot}s_2 \mid (s, s_2) \in \mathcal{D}_c(QQ) \wedge s = s_1{\cdot}!\}) & (\text{def}^{\text{n}} \text{ of } [PP]) \\
= \mathcal{D}([PP]) \cup \mathcal{D}([QQ]) & (\text{def}^{\text{n}} \text{ of } \sqcap) \\
= \mathcal{D}([PP] \sqcap [QQ]) &
\end{array}
$$

The equivalence between the failures sets can be proved similarly. □

Law c-lift-□: $[PP \Box QQ] = [PP] \Box [QQ]$

Proof. The proof of equivalence between the divergence sets is similar to that of the above law c-lift-⊓. Next we proof equivalence between the refusal sets.

$$
\begin{aligned}
&\mathcal{F}([PP \Box QQ]) && (\text{def}^{\text{n}} \text{ of } [PP]) \\
=&\{(s, X) \mid s \in \Sigma^{\circledast}_{\{\checkmark,?\}} \wedge (s, X \cup \{!\}) \in \mathcal{F}_f(PP \Box QQ)\} \cup \\
&\{(s_1 \cdot s_2, X) \mid (s, s_2, X) \in \mathcal{F}_c(PP \Box QQ) \wedge s = s_1 \cdot !\} \cup \\
&\{(s,X) \mid s \in \mathcal{D}([PP \Box QQ]) \wedge X \subseteq \Gamma\} && (\text{def}^{\text{n}} \text{ of } PP \Box QQ) \\
\\
=&\{(s, X) \mid s \in \Sigma^{\circledast}_{\{\checkmark,?\}} \wedge (s, X \cup \{!\}) \in \mathcal{F}(fp(PP) \Box fp(QQ))\} \cup \\
&\{(s_1 \cdot s_2, X) \mid (s, s_2, X) \in \mathcal{F}_c(PP) \wedge s = s_1 \cdot !\} \cup \\
&\{(s_1 \cdot s_2, X) \mid (s, s_2, X) \in \mathcal{F}_c(QQ) \wedge s = s_1 \cdot !\} \cup \\
&\{(s,X) \mid s \in \mathcal{D}([PP \Box QQ]) \wedge X \subseteq \Gamma\} && (\text{def}^{\text{n}} \text{ of } P \Box Q) \\
=&\{(\varepsilon, X) \mid (\varepsilon, X \cup \{!\}) \in \mathcal{F}(fp(PP)) \cap \mathcal{F}(fp(QQ))\} \cup && (\text{Set } a_1) \\
&\{(s, X) \mid (s, X \cup \{!\}) \in \mathcal{F}(fp(PP)) \cup \mathcal{F}(fp(QQ)) \wedge s \neq \varepsilon\} \cup && (\text{Set } a_2) \\
&\{(\varepsilon, X) \mid X \cup \{!\} \subseteq \Sigma^{\Omega} \setminus \{\omega\} \wedge \omega \in traces_{\perp}(fp(PP)) \cup traces_{\perp}(fp(QQ)) \wedge \omega \in \Omega\} \cup \\
& && (\text{Set } a_3) \\
&\{(s_1 \cdot s_2, X) \mid (s, s_2, X) \in \mathcal{F}_c(PP) \wedge s = s_1 \cdot !\} \cup && (\text{Set } a_4) \\
&\{(s_1 \cdot s_2, X) \mid (s, s_2, X) \in \mathcal{F}_c(QQ) \wedge s = s_1 \cdot !\} \cup && (\text{Set } a_5) \\
&\{(s,X) \mid s \in \mathcal{D}([PP \Box QQ]) \wedge X \subseteq \Gamma\} \\
\\
&\mathcal{F}([PP] \Box [QQ]) && (\text{def}^{\text{n}} \text{ of } \Box) \\
=&\{(\varepsilon, X) \mid (\varepsilon, X) \in \mathcal{F}([PP]) \cap \mathcal{F}([QQ])\} \cup && (\text{Set } b_1) \\
&\{(s, X) \mid (s, X) \in \mathcal{F}([PP]) \cup \mathcal{F}([QQ]) \wedge s \neq \varepsilon\} \cup && (\text{Set } b_2) \\
&\{(\varepsilon, X) \mid X \subseteq \Sigma^{\Omega} \setminus \{\omega\} \wedge \omega \in traces_{\perp}([PP]) \cup traces_{\perp}([QQ]) \wedge \omega \in \Omega\} && (\text{Set } b_3) \\
&\{(s,X) \mid s \in \mathcal{D}([PP] \Box [QQ]) \wedge X \subseteq \Gamma\}
\end{aligned}
$$

Next, we prove $a_1 \cup a_2 \cup a_3 \cup a_4 \cup a_5 = b_1 \cup b_2 \cup b_3$.

- For any element $(\varepsilon, X) \in a_1$, by the definition of $[PP]$, (ε, X) is also in the set b_1. For the same reason $a_2 \subseteq b_2$ and $a_3 \subseteq b_3$. For any element $(s, X) \in a_4 \cup a_5$, if $s = \varepsilon$, $(s, X) \in b_1 \cup b_3$. Otherwise, $(s, X) \in b_2$. Hence $a_1 \cup a_2 \cup a_3 \cup a_4 \cup a_5 \subseteq b_1 \cup b_2 \cup b_3$.
- Similarly, for any element $(\varepsilon, X) \in b_1$, we have $(\varepsilon, X) \in a_1 \cup a_4 \cup a_5$ by the definition of $[PP]$. We can also prove $b_2 \subseteq a_2 \cup a_4 \cup a_5$ and $b_3 \subseteq a_3 \cup a_4 \cup a_5$. Hence $b_1 \cup b_2 \cup b_3 \subseteq a_1 \cup a_2 \cup a_3 \cup a_4 \cup a_5$ holds.

Finally, $\mathcal{F}([PP \Box QQ]) = \mathcal{F}([PP] \Box [QQ])$ holds, completing the proof. □

Law c-Saga-1: If P does not terminate with the exception terminal event, then

$$[P \div Q \ ; \ \mathbf{throww}] = P \ ; \ Q$$

Proof. We prove the law for the three cases when the assumption of the law holds.

- Consider the case when P does not terminate, *i.e.* P diverges. Hence, we can have $[P \div Q\ ;\ \mathbf{throww}] = [P \div Q] = P \rhd \mathbf{skip}$ by the semantics of sequential composition and the law c-lift-$\div$. Because P will not terminate, $P \rhd \mathbf{skip} = P = P\ ;\ Q$. This proves $[P \div Q\ ;\ \mathbf{throww}] = P\ ;\ Q$.
- Consider the case when P terminates with the yield terminal event only, such as **interp**. From the semantics of sequential composition of standard processes, $P\ ;\ Q = P$, $P \rhd Q = P$ and $[P \div Q\ ;\ \mathbf{throww}] = [P \div Q]$ hold. These imply $[P \div Q] = P \rhd \mathbf{skip} = P = P\ ;\ Q$.
- Finally, consider the case when P terminates successfully.

$$
\begin{array}{ll}
\quad \mathcal{D}([P \div Q\ ;\ \mathbf{throww}]) & (\text{def}^{\text{n}} \text{ of } [PP]) \\
= \mathcal{D}_f(P \div Q\ ;\ \mathbf{throww}) \cup & \\
\quad \{s_1 \cdot s_2 \mid (s, s_2) \in \mathcal{D}_c(P \div Q; \mathbf{throww}) \wedge s = s_1 \cdot !\} & (\text{def}^{\text{n}} \text{ of } PP\ ;\ QQ) \\
= \mathcal{D}(P\ ;\ \mathbf{throw}) \cup \{t \cdot s_2 \mid \exists s \in \mathit{trace}_t(P\ ;\ \mathbf{throw}), s_2 \in \mathcal{D}(\mathbf{skip}\ ;\ Q) \bullet s = t \cdot !\} & \\
& (\text{def}^{\text{n}} \text{ of } P; Q \text{ and s-;-unit-l}) \\
= \mathcal{D}(P) \cup \{t \cdot s_2 | t \cdot \checkmark \in \mathit{trace}_t(P) \wedge s_2 \in \mathcal{D}(Q)\} & (\text{def}^{\text{n}} \text{ of } P; Q) \\
= \mathcal{D}(P\ ;\ Q) &
\end{array}
$$

Equality between the failure sets is proved similarly.

□

Law c-Saga-2: If the standard processes P_1 and P_2 do not terminate with the exception terminal event, then

$$[P_1 \div Q_1\ ;\ P_2 \div Q_2\ ;\ \mathbf{throww}] = P_1\ ;\ P_2\ ;\ Q_2\ ;\ Q_1$$

Proof. The law is proved in the same way as the above law c-Saga-1 by discussing the different cases of P_1 and P_2. □

Law c-;-pair: If all the standard processes terminate successfully, then

$$P_1 \div Q_1\ ;\ P_2 \div Q_2 = (P_1\ ;\ P_2) \div (Q_2\ ;\ Q_1)$$

Proof.

$$
\begin{array}{ll}
\quad \mathcal{D}_f(P_1 \div Q_1\ ;\ P_2 \div Q_2) & (\text{def}^{\text{n}} \text{ of } ; \text{ and def}^{\text{n}} \text{ of } P \div Q) \\
= \mathcal{D}(P_1\ ;\ P_2) & (\text{def}^{\text{n}} \text{ of } P \div Q) \\
= \mathcal{D}_f((P_1\ ;\ P_2) \div (Q_2\ ;\ Q_1)) &
\end{array}
$$

$$
\begin{array}{ll}
\quad \mathcal{D}_c(P_1 \div Q_1\ ;\ P_2 \div Q_2) & (\text{def}^{\text{n}} \text{ of } ; \text{ and def}^{\text{n}} \text{ of } P \div Q) \\
= \{(s, s_c) \mid \exists s_1 \in \mathit{trace}_t(P_1 \div Q_1), s_2 \in \mathit{trace}_t(P_2 \div Q_2) \bullet &
\end{array}
$$

$$(s_1 = t\cdot\checkmark \wedge s_1 \in trace_n(P_1) \wedge s = t\cdot s_2 \wedge s_c \in \mathcal{D}(Q_2 \ ; \ Q_1)) \vee$$
$$((s_1 \neq t\cdot\checkmark \vee s_1 \notin trace_n(P_1)) \wedge s = s_1 \wedge s_c \in \mathcal{D}(cp(PP, s_1))) \ \}$$
(def^n of $P \div Q$ and Assump)

$= \{(s, s_c) \mid s_1 \in trace_t(P_1 \ ; \ P_2) \cap \Sigma^{\star}_{\{\checkmark\}} \wedge s_c \in \mathcal{D}(Q_2 \ ; \ Q_1)\}$ (Set theory)

$= (trace_t(P_1 \ ; \ P_2) \cap \Sigma^{\star}_{\{\checkmark\}}) \times \mathcal{D}(Q_2 \ ; \ Q_1)$ (def^n of $P \div Q$ and Assump)

$= \mathcal{D}_c((P_1 \ ; \ P_2) \div (Q_2 \ ; \ Q_1))$

The equivalence between the forward failure sets and the equivalence between the compensation failure sets can be proved similarly. □

Law c-$\|$-pair: If all the standard processes terminate successfully, then

$$(P_1 \div Q_1) \underset{X}{\|} (P_2 \div Q_2) = (P_1 \underset{X}{\|} P_2) \div (Q_1 \underset{X}{\|} Q_2)$$

Proof.

$\mathcal{D}_f((P_1 \div Q_1) \underset{X}{\|} (P_2 \div Q_2))$ (def^n of $\underset{X}{\|}$ and def^n of $P \div Q$)

$= \mathcal{D}(P_1 \underset{X}{\|} P_2)$ (def^n of $P \div Q$)

$= \mathcal{D}_f((P_1 \underset{X}{\|} P_2) \div (Q_1 \underset{X}{\|} Q_2))$

$\mathcal{D}_c(P_1 \div Q_1 \ \underset{X}{\|} \ P_2 \div Q_2)$ (def^n of $\underset{X}{\|}$ and def^n of $P \div Q$)

$= \{(s, s_c) \mid \exists s_1 \in trace_f(P_1 \div Q_1), s_2 \in trace_f(P_2 \div Q_2) \bullet$
$s \in (s_1 \underset{X}{\|} s_2) \wedge s_c \in \mathcal{D}(Q_1 \underset{X}{\|} Q_2)\}$ (def^n of $P \div Q$, def^n of $P_1 \underset{X}{\|} P_2$ and Assump)

$= \{(s, s_c) \mid s_1 \in trace_t(P_1 \underset{X}{\|} P_2) \wedge s_c \in \mathcal{D}(Q_1 \underset{X}{\|} Q_2)\}$ (Set theory)

$= trace_t(P_1 \underset{X}{\|} P_2) \times \mathcal{D}(Q_1 \underset{X}{\|} Q_2)$ (def^n of $P \div Q$ and Assump)

$= \mathcal{D}_c((P_1 \underset{X}{\|} P_2) \div (Q_1 \underset{X}{\|} Q_2))$

Equality between the forward failure sets and between the compensation failure sets can be proved similarly. □

Law c-Saga-$\|$-1: If all the standard processes terminate successfully, then

$$[(P \div Q) \parallel \mathbf{throww}] = P \ ; \ Q$$

Proof.

$\mathcal{D}([(P \div Q) \parallel \mathbf{throww}])$ (def^n of $[PP]$)

$= \mathcal{D}_f((P \div Q) \parallel \mathbf{throw} \div \mathbf{skip}) \cup$
$\{s_1 \cdot s_2 \mid (s, s_2) \in \mathcal{D}_c((P \div Q) \parallel \mathbf{throw} \div \mathbf{skip}) \wedge s = s_1 \cdot !\}$
(def^n of $P \div Q$ and def^n of $\underset{X}{\|}$)

$$
\begin{aligned}
&= \mathcal{D}(P \parallel \mathbf{throw}) \cup \{s{\cdot}t \mid s{\cdot}! \in \mathit{trace}_t(P \parallel \mathbf{throw}) \wedge t \in \mathcal{D}(Q \parallel \mathbf{skip})\} \\
&\qquad\qquad \text{(Assump, s-}\parallel\text{-throw and s-}\parallel\text{-unit)} \\
&= \mathcal{D}(P \mathbin{;} \mathbf{throw}) \cup \{s{\cdot}t \mid s{\cdot}! \in \mathit{trace}_t(P \mathbin{;} \mathbf{throw}) \wedge t \in \mathcal{D}(Q)\} \\
&\qquad\qquad \text{(Assump and def}^{\text{n}}\text{ of } P \mathbin{;} Q) \\
&= \mathcal{D}(P) \cup \{s{\cdot}t \mid s{\cdot}\checkmark \in \mathit{traces}_\perp(P) \wedge t \in \mathcal{D}(Q)\} \qquad \text{(def}^{\text{n}}\text{ of } P \mathbin{;} Q) \\
&= \mathcal{D}(P \mathbin{;} Q)
\end{aligned}
$$

Equality between the refusal sets can be proved similarly. □

Law c-Saga-$\parallel$-2: If all the standard processes terminate successfully, then

$$[(P \div Q \mathbin{;} \mathbf{interpp}) \parallel \mathbf{throww}] = P \mathbin{;} Q$$

Proof. For P and Q that both terminate successfully, we have $Q \parallel \mathbf{skip} = Q$ and $(P;\mathbf{interp}) \parallel \mathbf{throw} = P;\mathbf{throw}$. Then, by case analysis similar to the previous proof for c-Saga-$\parallel$-1, according to the semantics, we can prove the validity of the law. □

Law c-$\parallel$-;: If all the standard processes terminate successfully, then

$$[(P_1 \div Q_1 \mathbin{;} P_2 \div Q_2) \parallel \mathbf{throww}] = P_1 \mathbin{;} P_2 \mathbin{;} Q_2 \mathbin{;} Q_1$$

Proof.

$$
\begin{aligned}
&[(P_1 \div Q_1 \mathbin{;} P_2 \div Q_2) \parallel \mathbf{throww}] && \text{(Assump and c-;-pair)} \\
&= [(P_1 \mathbin{;} P_2) \div (Q_2 \mathbin{;} Q_1) \parallel \mathbf{throww}] && \text{(Assump and c-Saga-}\parallel\text{-1)} \\
&= P_1 \mathbin{;} P_2 \mathbin{;} Q_2 \mathbin{;} Q_1
\end{aligned}
$$

□

Law c-$\parallel$-inter-2: If all the standard processes terminate successfully, then

$$
\begin{aligned}
&[(\mathbf{yieldd} \mathbin{;} P_1 \div Q_1) \parallel (\mathbf{yieldd} \mathbin{;} P_2 \div Q_2) \parallel \mathbf{throww}] = \\
&\quad \mathbf{skip} \sqcap (P_1 \mathbin{;} Q_1) \sqcap (P_2 \mathbin{;} Q_2) \sqcap ((P_1 \parallel P_2) \mathbin{;} (Q_1 \parallel Q_2))
\end{aligned}
$$

Proof.

$$
\begin{aligned}
&[(\mathbf{yieldd} \mathbin{;} P_1 \div Q_1) \parallel (\mathbf{yieldd} \mathbin{;} P_2 \div Q_2) \parallel \mathbf{throww}] \\
&\qquad\qquad (\mathbf{yieldd}{=}\mathbf{skipp}{\sqcap}\mathbf{interpp}\text{, c-;-dist-l, c-;-zero-2 and c-;-unit-l)} \\
&= [((P_1 \div Q_1) \sqcap \mathbf{interpp}) \parallel ((P_2 \div Q_2) \sqcap \mathbf{interpp}) \parallel \mathbf{throww}] \\
&\qquad\qquad \text{(c-}\parallel\text{-dist and c-lift-}\sqcap\text{)} \\
&= [(P_1 \div Q_1 \parallel P_2 \div Q_2) \parallel \mathbf{throww}] \sqcap [P_1 \div Q_1 \parallel \mathbf{interpp} \parallel \mathbf{throww}] \sqcap \\
&\quad [P_2 \div Q_2 \parallel \mathbf{interpp} \parallel \mathbf{throww}] \sqcap [\mathbf{interpp} \parallel \mathbf{interpp} \parallel \mathbf{throww}] \\
&\qquad\qquad \text{(c-}\parallel\text{-pair, c-}\parallel\text{-syn-1, c-}\parallel\text{-assoc, c-lift-}\div\text{ and c-Saga-}\parallel\text{-1)} \\
&= ((P_1 \parallel P_2) \mathbin{;} (Q_1 \parallel Q_2)) \sqcap (P_1 \mathbin{;} Q_1) \sqcap (P_2 \mathbin{;} Q_2) \sqcap (\mathbf{throw} \rhd \mathbf{skip}) \\
&\qquad\qquad \text{(s-}\rhd\text{-unit-l, s-}\sqcap\text{-assoc and s-}\sqcap\text{-sym)} \\
&= \mathbf{skip} \sqcap (P_1 \mathbin{;} Q_1) \sqcap (P_2 \mathbin{;} Q_2) \sqcap ((P_1 \parallel P_2) \mathbin{;} (Q_1 \parallel Q_2))
\end{aligned}
$$

□

Law c-‖-assoc-1: $(QQ \parallel PP) \underset{X}{\parallel} RR = QQ \parallel (PP \underset{X}{\parallel} RR) \quad if\ \alpha(QQ) \cap X = \{\}$

Proof. The equivalence between the forward behaviors is proved by using the law s-‖-assoc-1 of standard processes, *i.e.*

$$(Q \parallel P) \underset{X}{\parallel} R = Q \parallel (P \underset{X}{\parallel} R) \quad if\ \alpha(Q) \cap X = \{\}$$

and the following equation on traces, where there is no event in X appearing in the trace s, $r \underset{X}{\parallel} S$ represents the set $\{u \mid u \in (r \underset{X}{\parallel} s) \wedge s \in S\}$ and $r \underset{X}{\parallel} S = S \underset{X}{\parallel} r$:

$$(s \parallel t) \underset{X}{\parallel} r = s \parallel (t \underset{X}{\parallel} r) \qquad \text{(assis-2)}$$

We prove equality of the compensation divergence sets as follows.

$$\begin{aligned}
&\mathcal{D}_c((QQ \parallel PP) \underset{X}{\parallel} RR) && (\text{def}^{\text{n}}\ \text{of}\ \underset{X}{\parallel})\\
=&\{(s, s_c) \mid \exists s_1 \in trace_f(QQ \parallel PP), s_2 \in trace_f(RR) \bullet \\
&\ s \in (s_1 \underset{X}{\parallel} s_2) \wedge s_c \in \mathcal{D}(cp((QQ \parallel PP), s_1) \underset{X}{\parallel} cp(RR, s_2))\} && (\text{def}^{\text{n}}\ \text{of}\ \underset{X}{\parallel})\\
=&\{(s, s_c) \mid \exists s_1 \in trace_f(QQ), s_3 \in trace_f(PP), s_2 \in trace_f(RR) \bullet \\
&\ s \in ((s_1 \parallel s_3) \underset{X}{\parallel} s_2) \wedge s_c \in \mathcal{D}((cp(QQ, s_1) \parallel cp(PP, s_3)) \underset{X}{\parallel} cp(RR, s_2))\} \\
& && (\text{assis-2 and s-‖-assoc-1})\\
=&\{(s, s_c) \mid \exists s_1 \in trace_f(QQ), s_3 \in trace_f(PP), s_2 \in trace_f(RR) \bullet \\
&\ s \in (s_1 \parallel (s_3 \underset{X}{\parallel} s_2)) \wedge s_c \in \mathcal{D}((cp(QQ, s_1) \parallel (cp(PP, s_3) \underset{X}{\parallel} cp(RR, s_2)))\} \\
& && (\text{def}^{\text{n}}\ \text{of}\ \underset{X}{\parallel})\\
=&\{(s, s_c) \mid \exists s_1 \in trace_f(QQ), s_2 \in trace_f(PP \underset{X}{\parallel} RR) \bullet \\
&\ s \in (s_1 \parallel s_2) \wedge s_c \in \mathcal{D}(cp(QQ, s_1) \parallel cp((PP \underset{X}{\parallel} RR), s_2))\} && (\text{def}^{\text{n}}\ \text{of}\ \underset{X}{\parallel})\\
=&\mathcal{D}_c(QQ \parallel (PP \underset{X}{\parallel} RR))
\end{aligned}$$

Similarly, equality between the compensation refusal sets can be proved. □

Law c-‖-;-head: If P terminates successfully, $(\alpha(P) \cup \alpha(Q)) \cap X = \{\}$, $\alpha^i(QQ) \subseteq X$, $\alpha^c(QQ) \subseteq (\alpha^c(PP) \cap X)$, the compensation behavior of QQ terminates successfully, and there is no deadlock in $PP \underset{X}{\parallel} QQ$, then

$$(P \div Q\ ;\ PP) \underset{X}{\parallel} QQ = P \div Q\ ;\ (PP \underset{X}{\parallel} QQ)$$

Proof. The law can be proved just as the above law c-‖-assoc-1 by the laws s-‖-;-head and s-‖-;-tail, which are

$$(P\ ;\ Q) \underset{X}{\parallel} R = P\ ;\ (Q \underset{X}{\parallel} R) \quad (Q\ ;\ P) \underset{X}{\parallel} R = (Q \underset{X}{\parallel} R)\ ;\ P$$

The law s-‖-;-head can be used in the proof of equality between the forward behaviors, and the law s-‖-;-tail is used for the compensation behaviors. □

Law c-‖-;-tail: If Q terminates successfully, $(\alpha(P) \cup \alpha(Q)) \cap X = \{\}$, $\alpha^c(QQ) \subseteq X$, $fp(QQ)$ terminates successfully, $\alpha^i(QQ) \subseteq (\alpha^i(PP) \cap X)$ and there is no deadlock in $PP \underset{X}{\parallel} QQ$, then

$$(PP \ ; \ P \div Q) \underset{X}{\parallel} QQ = (PP \underset{X}{\parallel} QQ) \ ; \ P \div Q$$

Proof. Like the above law c-‖-;-head, this law can be proved from the laws s-‖-;-head and s-‖-;-tail in standard processes. □

Law c-Saga-⊠: If all the standard processes terminate successfully, then

$$[(P_1 \div Q_1 \boxtimes P_2 \div Q_2) \ ; \ \mathbf{throww}] = (P_1 \parallel P_2); ((Q_1; Q_2) \sqcap (Q_2; Q_1))$$

Proof.

$$\begin{array}{ll}
& [(P_1 \div Q_1 \boxtimes P_2 \div Q_2) \ ; \ \mathbf{throww}] \qquad (\text{def}^{\text{n}} \text{ of } \boxtimes) \\
= & [(((P_1 \parallel P_2) \ ; \ Q_1) \div Q_2) \sqcap (((P_1 \parallel P_2) \ ; \ Q_2) \div Q_1) \ ; \ \mathbf{throww}] \\
& \qquad (\text{c-;-dist-l and c-lift-}\sqcap) \\
= & [(((P_1 \parallel P_2) \ ; \ Q_1) \div Q_2) \ ; \ \mathbf{throww}] \sqcap \\
& [(((P_1 \parallel P_2) \ ; \ Q_2) \div Q_1) \ ; \ \mathbf{throww}] \qquad (\text{c-Saga-1}) \\
= & (((P_1 \parallel P_2) \ ; \ Q_1) \ ; \ Q_2) \sqcap (((P_1 \parallel P_2) \ ; \ Q_2) \ ; \ Q_1) \qquad (\text{s-;-assoc and s-;-dist-r}) \\
= & (P_1 \parallel P_2); ((Q_1; Q_2) \sqcap (Q_2; Q_1))
\end{array}$$

□

Appendix D - Notation

The notations used in this paper are.

Σ	the alphabet of all processes
Ω	terminal event set $\{\checkmark, ?, !\}$
Γ	$\Sigma \cup \Omega$
ε	empty trace
$a_1 a_2 ... a_n$	a finite trace containing $a_1, ..., a_n$ in order
Σ^*	the set of finite traces over Σ
$s \cdot t$	concatenation of two traces
$T_1 \cdot T_2$	concatenation of two trace sets $\{s_1 \cdot s_2 \mid s_1 \in T_1 \wedge s_2 \in T_2\}$
$\Sigma_O^\star$	$\{s \cdot \omega \mid s \in \Sigma^* \wedge \omega \in O\}$ for an $O \subseteq \Omega$

$\Sigma^{\bigstar}$	$\Sigma_{\Omega}^{\bigstar}$
$\Sigma_{O}^{\circledast}$	$\Sigma^{*} \cup \Sigma_{O}^{\bigstar}$
$\Sigma^{\circledast}$	$\Sigma_{\Omega}^{\circledast}$
$\omega_1 \& \omega_2$	synchronization between two terminal events
$s \underset{X}{\parallel} t$	the trace set of two traces synchronizing on the event set X
$s \setminus X$	removing all members of the event set X from the trace s
$s_1 \ \rho \ s_2$	two traces satisfying a renaming relation: s_1 can be renamed to s_2 w.r.t. the renaming relation ρ
$\rho_1 \circ \rho_2$	relational composition of two renaming relations
(s, X)	a failure, a pair of a trace s and an event set X
$\alpha(P)$	set of non-terminal events in the process P
$\alpha^{i}(P)$	set of starting non-terminal events of P
$\mathcal{F}(P)$	failure set of a standard process
$\mathcal{D}(P)$	divergence set of a standard process
$traces_{\perp}(P)$	trace set of a standard process
$trace_{t}(P)$	terminated trace set of a standard process
$trace_{n}(P)$	non-divergent, terminated trace set of a standard process
$(s, Y) \underset{X}{\parallel} (t, Z)$	the failure set of two failures synchronizing on the event set X
$\mathcal{F}_{f}(PP)$	forward failure set of a compensable process
$\mathcal{D}_{f}(PP)$	forward divergence set of a compensable process
$\mathcal{F}_{c}(PP)$	compensation failure set of a compensable process
$\mathcal{D}_{c}(PP)$	compensation divergence set of a compensable process
$trace_{f}(PP)$	forward terminated trace set of a compensable process
$trace_{n}(PP)$	non-divergent, forward terminated trace set of a compensable process
$(F^{c}, D^{c}) \lfloor s$	$(\{(s_1, X) \mid (s, s_1, X) \in F^{c}\}, \{s_1 \mid (s, s_1) \in D^{c}\})$
$fp(PP)$	forward process behavior of a compensable process
$cp(PP, s)$	compensation process behavior of a compensable process for s
$P_1 \sqsubseteq P_2$	refinement between two standard processes: P_2 refines P_1
$PP_1 \sqsubseteq_{c} PP_2$	refinement between two compensable processes: PP_2 refines PP_1