Failure-Divergence Refinement of Compensating Communicating Processes*

Zhenbang Chen[1], Zhiming Liu[2] and Ji Wang[1]

[1] National Laboratory for Parallel and Distributed Processing, Changsha, China
[2] International Institute for Software Technology, The United Nations University, Macao

Abstract. Compensating CSP (cCSP) extends CSP for specification and verification of long running transactions. The original cCSP is a modest extension to a subset of CSP that does not consider non-deterministic choice, synchronized composition, and recursion. There are a few further extensions. However, it remains a challenge to develop a fixed-point theory of process refinement in cCSP. This paper provides a complete solution to this problem and develops a theory of cCSP, corresponding to the theory of CSP, so that the verification techniques and their tools, such as FDR, can be extended for compensating processes.

1 Introduction

Service-oriented architecture (SOA) is a critical enabling technology for programming business processes composed of disparate services available on the web. These processes are required to have the "transactional characteristic": if a failure occurs in the execution, the changes made before the failure must be *undone* or *compensated*. The transactional property of business processes is different from the ACID properties of *atomic transactions* (or *short-life transactions*). A general business process is usually a long running transaction (LRT) [13] that takes a substantial amount of time to complete, involving interactions across many systems and possibly requiring human interventions [14]. The mechanism to ensure ACID by the holding of locks and tight coordination of the participating systems cannot be applied to LRTs. Furthermore, it is not required for such a business process to undo the entire change committed in case of an occurrence of a failure. Instead, a weakened or partial recovery is often required by the application. For example, when a flight booking is canceled, a cancelation fee is charged and only part of the payment can be recovered. For this reason, models of LRTs, such as Sagas [12] and BizTalk [15], allow programmer-specified undoing and recovery actions that are called *compensation*.

Compensation supports flexible treatment of different exceptional scenarios, but the flexibility makes the handling of failures complicated and ad-hoc. To help programmers master the complexity, Web Service languages, such as WS-BPEL and XLANG, provide mechanisms for exception handling. A common design principle of these languages is a combination of exception handling and failure recovery. In some situations, an exception is raised when a LRT is aborted and caught by a programmed handler.

* Supported in parts by projects NSFC 60725206, National 973 project 2011CB302603, NSFC 60970031, NSFC 61073022, and the MSTDF project GAVES.

In some other cases, actions of a LRT are programmed with the corresponding compensatory actions so that when a failure occurs, the recovery actions of the committed actions are activated and executed in their reverse order. This is the model of the backward recovery in Sagas [12] implemented in BizTalk.

The need for better understanding of complex LRTs and their mission-critical applications motivate the research on formal theories of compensation programming, *e.g.* [1, 4, 3, 11]. They differ mostly in the features that they support. For examples, non-deterministic choice and limited recursion are supported by the process calculus in [11], but not by cCSP in [6]. However, little discussion is given by the designers of the languages about the decisions they made. This indicates an insufficient understanding on what common features of LRTs should and can be formalized.

Contribution We extend the version of cCSP in [6], with non-deterministic choices, synchronization among parallel processes, and recursion (cf. Section 2). The main contribution is a semantic theory of *failures* and *divergences* of LRTs (cf. Sections 3&4). 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 programming of LRTs (cf. Section 5). It is a well-known challenge to establish, or even to show the existence of, such a fixed-point theory for a language like CSP with internal-choice and synchronization [18]. Literature, *e.g.* [8], also shows that if the internal choice is added to CCS, 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 abstract mechanisms for exception handling and compensation behavior. The technical details in Section 3&4 and the proofs of the theorems and laws in the technical report [10] show the inventive thinking needed. Because the application potential of cCSP, the extension and its well established failure-divergence semantic theory are important. 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, trace semantics, stable failures semantics [9] of cCSP. It completes the semantic theory of cCSP for specification of LRTs and can be used to underpin the extension to FDR of CSP [17] and the cCSP theorem proving tool [16] for verification of LRTs.

Related work cCSP in [6] is an extension to CSP [17] for LRTs. The recovery mechanism in cCSP is the same as the backward recovery in Sagas [12]. There are two types of processes in cCSP, the *standard processes* and *compensable processes*. The standard processes are only a subset of CSP processes, but with additional processes for exception handling and transaction block. A compensable process specifies the behavior of the recovery when an exception occurs. Non-deterministic choices and synchronization are not allowed in cCSP and thus it only has a trace semantics [4] and an operational semantics [7]. The consistency between these two semantics is studied in [16].

Our early work in [9] extends cCSP with the operators of non-deterministic choice, synchronized parallel composition, hiding and renaming, and defines a stable failures semantics. But that semantic model does not allows us to establish a fixed-point theory for recursion with a meaningful CPO. Thus, a new semantic domain has to be defined,

instead of simple extension of the stable failures semantic domain with a divergence set, has to be designed so as to define refinement and calculate fixed-points.

The main reason why we develop a semantic theory of LRTs by extending the original cCSP is because that cCSP shares most of the common features of other formal models [2, 11]. Also, CSP-like process calculi are used to give formal semantics of protocol description languages and orchestration languages, *e.g.* [5].

2 Syntax of the Extended cCSP

Assume a *finite* set Σ of names representing the *normal events* that the cCSP processes can perform, called the *alphabet* of processes. The syntax of the extended cCSP is defined in Fig. 1, where $a{\in}\Sigma$ represents an event, $X{\subseteq}\Sigma$ is a finite subset, and $R{\subseteq}\Sigma{\times}\Sigma$ is a *renaming relation*. cCSP defines two kinds of processes, the *standard processes* ranged over by P, and the *compensable processes* ranged over by PP.

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

Fig. 1: *The standard processes add four additional kinds of processes to CSP:* $\mathbf{throw}$ *throws an exception and interrupts the execution of the process,* $\mathbf{yield}$ *either terminates successfully or yields to an interruption from environment to interrupt the execution,* $P \rhd Q$ *behaves like* P *and it executes* Q *if there is an exception thrown by* P*, the transaction block* $[PP]$ *represents a long-running transaction specified by the compensable process* PP*.*

The standard processes extend those of the classical CSP processes with exception handling, interruption and transaction block specified by a compensable process. A compensable process is constructed from *compensation pairs* of the form $P \div Q$, in which the execution of the process Q can compensate the effect of the execution of P. P is called the *forward* (*sub-*) *process* and Q the *compensation* (*sub-*) *process*. The internal and external choices in the compensable processes are made according to the forward sub-processes. PP and QQ in $PP \underset{X}{\parallel} QQ$ synchronize on the events in X occurring in the behaviors of both forward and compensation sub-processes of PP and QQ. It is written $PP \| QQ$ when X is empty. $PP \boxtimes QQ$ is the speculative choice between two compensable processes, in which PP and QQ run in parallel until one of them succeeds, and after that the other is compensated. Process $\mathbf{skipp}$ immediately terminates successfully without the need to be compensated, and $\mathbf{throww}$ throws an exception and $\mathbf{yieldd}$ either terminates successfully or yields to an interruption.

3 Failure-Divergence Semantics of Standard Processes

3.1 Basic notations

Let A^* denote the set of finite sequences of the elements in a set A of symbols. In particular, Σ^* is the set of *interaction traces* of the cCSP processes. Let $\Omega{=}\{\checkmark, !, ?\}$ be

disjoint with Σ. Events in Ω are called *terminals* and they indicate different terminating scenarios: "$\checkmark$" represents that the execution terminates successfully, "!" indicates that the execution terminates with an occurrence of an exception, and "?" represents that the execution terminates by yielding to an interruption from environment. The traces of cCSP processes are thus formed from symbols in $\Gamma{=}\Sigma{\cup}\Omega$. Let $s{\cdot}t$ denote the *concatenation* of traces s and t, and $T_1{\cdot}T_2$ the set of concatenated traces of the trace sets T_1 and T_2. In particular, for a non-empty subset A of Ω, let $\Sigma_A^\star{=}\Sigma^*{\cdot}A$ denote the set of traces terminated with events in A, and let $\Sigma_A^\circledast{=}\Sigma^* \cup \Sigma_A^\star$. Thus, $\Sigma_{\{\checkmark\}}^\star$ is the set of *successfully terminated traces*. We use $\Sigma^\star$ and $\Sigma^\circledast$ as the shorthands of $\Sigma_\Omega^\star$ and $\Sigma_\Omega^\circledast$.

Processes need to follow different rules to synchronize on different terminals. We order the three terminals such that $! \prec ? \prec \checkmark$ and define $\omega_1 \| \omega_2 = \omega_1$ if $\omega_1 \preceq \omega_2$, and $\omega_1 \| \omega_2 = \omega_2 \| \omega_1$. Therefore, the synchronization of any terminal with an exception will result in an exception, and composition terminates successfully iff both parties do.

For two traces $s, t \in \Sigma^\circledast$ and a subset $X \subseteq \Sigma$, the set of synchronized traces $s \underset{X}{\|} t$, $s \| t$ when $X = \{\}$, is defined in the same way as in CSP [17] when $s, t \in \Sigma^*$, otherwise terminals of s and t synchronize in the following two patterns, where $s_1, t_1 \in \Sigma^*$.

$$s_1{\cdot}\langle\omega\rangle \underset{X}{\|} t_1 = \{\}\,,\ \ s_1{\cdot}\langle\omega_1\rangle \underset{X}{\|} t{\cdot}\langle\omega_2\rangle = \{u{\cdot}\langle\omega_1\|\omega_2\rangle \mid u \in s_1 \underset{X}{\|} t_1\}$$

3.2 Semantics of standard processes

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* and $\mathcal{D}(P) \subseteq \Sigma^\circledast$ the *divergence set*. The sets of *traces* and *terminated traces* of P are defined from the failures $\mathcal{F}(P)$ below.

$$\mathit{traces}(P) \ \widehat{=}\ \{s \mid (s, \{\}) \in \mathcal{F}(P)\},\ \ \mathit{trace}_t(P) \ \widehat{=}\ \mathit{traces}(P) \cap \Sigma^\star$$

We require that the FD semantics of P satisfies the axioms of the FD semantics of the classical CSP processes given in [17], for example, the divergence set $\mathcal{D}(P)$ is suffix closed and the trace set $\mathit{traces}(P)$ is prefix closed. However, the axioms about terminated traces need to be modified as follows.

$$s{\cdot}\langle\omega\rangle \in \mathit{traces}(P) \Rightarrow (s, \Gamma \setminus \{\omega\}) \in \mathcal{F}(P),\ \mathit{where}\ \omega \in \Omega \tag{1}$$

$$s \in \mathcal{D}(P) \cap \Sigma^* \wedge t \in \Sigma^\circledast \Rightarrow s{\cdot}t \in \mathcal{D}(P) \tag{2}$$

$$s{\cdot}\langle\omega\rangle \in \mathcal{D}(P) \Rightarrow s \in \mathcal{D}(P),\ \mathit{where}\ \omega \in \Omega \tag{3}$$

In what follows we define the *failure function* $\mathcal{F} : \mathcal{P} \to \mathbb{P}(\Sigma^\circledast \times \mathbb{P}(\Gamma))$ and the *divergence function* $\mathcal{D} : \mathcal{P} \to \mathbb{P}(\Sigma^\circledast)$, where $\mathcal{P}$ denotes the set of all standard processes.

Atomic and basic processes The semantics of the processes a, $\mathbf{skip}$ and $\mathbf{stop}$ are the same as their semantics in CSP. The divergence sets of processes $\mathbf{throw}$ and $\mathbf{yield}$ are both empty, and their failure sets are defined below.

$$\begin{array}{rcl}\mathcal{F}(\mathbf{throw}) = & \{(\langle\rangle, X) \mid X \subseteq \Gamma \wedge ! \notin X\} \cup \{(\langle !\rangle, X) \mid X \subseteq \Gamma\} \\ \mathcal{F}(\mathbf{yield}) = & \{(\langle\rangle, X) \mid X \subseteq \Gamma \wedge ? \notin X\} \cup \{(\langle ?\rangle, X) \mid X \subseteq \Gamma\} \\ & \cup\, \{(\langle\rangle, X) \mid X \subseteq \Gamma \wedge \checkmark \notin X\} \cup \{(\langle\checkmark\rangle, X) \mid X \subseteq \Gamma\}\end{array}$$

We use $\mathbf{div}$ to represent the process diverging immediately, *i.e.* $\langle\rangle \in \mathcal{D}(\mathbf{div})$.

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 $\langle\rangle$, at which $P \Box Q$ can refuse an event only if both P and Q can refuse it. Also, care should be taken about the terminals "?" and "!" when defining the failures to ensure the axiom (1).

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

Sequential composition The sequential composition here is also different from the classic CSP [17] because of the terminals "!" and "?".

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

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

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

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 classical definition in CSP of the synchronized failure set:

$$(s, Y) \underset{X}{\parallel} (t, Z) = \{(u, Y \cup Z) \mid Y \setminus (X \cup \Omega) = Z \setminus (X \cup \Omega) \wedge u \in s \underset{X}{\parallel} t\} \tag{4}$$

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 Definition (4) applies. For example, let $\Sigma = \{a, b\}$, P be the process a and Q the process $b; \mathbf{throw}$. As $(\langle\rangle, \{b, \checkmark, !, ?\})$ is a failure of P and $(\langle b\rangle, \{b, \checkmark, ?\})$ a failure of Q, $P \| Q$ has the failure $(\langle b\rangle, \{b, \checkmark, !, ?\})$. This case is reflected in the upper case in the definition Equation 5.
2. If both P and Q can terminate, the synchronized terminal, represented by Θ in the definition Equation 5, 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 $(\langle a\rangle, \{a, !, ?\})$ is a failure of P and $(\langle a\rangle, \{a, \checkmark, ?\})$ 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 $(\langle a\rangle, \{a, \checkmark, ?\})$ of $P \underset{\{a\}}{\parallel} Q$. If Definition (4) were applied, the refusal set would be $(\langle a\rangle, \{a, \checkmark, ?, !\})$ and $P \underset{\{a\}}{\parallel} Q$ would deadlock after executing $\langle a\rangle$.

The synchronized failure set of two failures is thus defined as

$$(s,Y) \underset{X}{\parallel} (t,Z) = \begin{cases} \{(u, Y \cup Z) \mid Y \setminus (X \cup \Omega) = Z \setminus (X \cup \Omega) \wedge u \in s \underset{X}{\parallel} t\} \\ \qquad\qquad \textbf{if } (s, Y \cup \Omega) \in \mathcal{F}(P) \vee (t, Z \cup \Omega) \in \mathcal{F}(Q) \\ \{(u, (Y \cup Z) \setminus \Theta(\omega_1, \omega_2)) \mid Y \setminus (X \cup \Omega) = Z \setminus (X \cup \Omega) \wedge \\ \qquad\qquad u \in s \underset{X}{\parallel} t\} \quad \textbf{otherwise} \end{cases} \tag{5}$$

Two variables ω_1 and ω_2 are used in Equation (5). 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

$$\forall (s, Y_1) \in \mathcal{F}(P) \bullet Y \subseteq Y_1 \Rightarrow (\omega_1 \in \Omega \wedge \omega_1 \notin Y_1) \tag{6}$$

The value of ω_1 is not defined, represented 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 that synchronizes ω_1 and ω_2 is defined as follows.

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

For example, consider P as the process $\mathbf{skip} \sqcap \mathbf{throw}$. P has the failures $(\langle\rangle, \{\checkmark, ?\})$ and $(\langle\rangle, \{?, !\})$. There is no ω_1 satisfying the Equation (6) for the failure $(\langle\rangle, \{?\})$ of P.

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

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

Consider $a \underset{\{a\}}{\parallel} (a; \mathbf{throw})$ as an example, and $\Sigma = \{a\}$. Its divergence set is $\{\}$, and its failure set is $\{(\langle\rangle, X) \mid X \subseteq \Omega\} \cup \{(\langle a\rangle, X) \mid X \subseteq \{a, \checkmark, ?\}\} \cup \{(\langle a, !\rangle, X) \mid X \subseteq \Gamma\}$. 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.

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

The exception handling is *associative* and *distributive* over internal choices to both left and right sides of $\rhd$. The hiding and renaming operators are not affected by the new terminals, and their definitions remain the same as those in the classical CSP.

4 Failure-Divergence Semantics of Compensable Processes

The semantics $[\![PP]\!]$ of a compensable process *PP* consists of its forward behavior and compensation behavior. It is thus defined as a tuple (F, D, F^c, D^c) of four sets. (F, D) are the *forward failures* and *forward divergences* (or forward FD sets), and (F^c, D^c) the *compensation FD sets* of PP, where $F^c \subseteq \Sigma^\star \times \Sigma^\circledast \times \mathbb{P}(\Gamma)$ and $D^c \subseteq \Sigma^\star \times \Sigma^\circledast$ are called the *compensation failures* and *compensation divergences* of PP, respectively. The forward FD sets satisfy the axioms of the semantics of the standard processes given in Section 3.2. A compensation failure (s, s_1, X) and a compensation divergence (s, s_1) record a failure and a divergence of the compensation behavior for the forward execution trace s, respectively. We define the set of *the forward terminated traces* $\mathit{trace}_f(PP)=\{s \mid (s, s_1, X) \in F^c\}$ in F^c (also denoted by $\mathit{trace}_f(F^c)$), and the set $\mathit{trace}_n(PP) = \mathit{trace}_f(PP)\backslash D$ of the *non-divergent* forward terminated traces. The compensation behavior (F^c, D^c) is required to satisfy the following axioms.

$$\mathit{trace}_f(F^c) = \{s \mid (s, s_1) \in D^c\}, \quad \mathit{trace}_f(F^c) \subseteq \Sigma^\star \cap \{s \mid (s, \{\}) \in F\} \tag{7}$$

For an s in $\mathit{trace}_f(F^c)$, let $(F^c, D^c)\lfloor s = (\{(s_1, X) \mid (s, s_1, X) \in F^c\}, \{s_1 \mid (s, s_1) \in D^c\})$, which is a FD pair and required to satisfy the axioms of standard processes. 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 are now to define the semantic function $[\![\cdot]\!]$ on the set $\mathcal{PP}$ of all the compensable processes in terms of four semantics functions $(\mathcal{F}_f, \mathcal{D}_f, \mathcal{F}_c, \mathcal{D}_c)$:

- 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 $\mathcal{F}_c : \mathcal{PP} \to \mathbb{P}(\Sigma^\star \times \Sigma^\circledast \times \mathbb{P}(\Gamma))$, and
- the *compensation divergence* (DC) function $\mathcal{D}_c : \mathcal{PP} \to \mathbb{P}(\Sigma^\star \times \Sigma^\circledast)$.

Compensation pair $P \div Q$ If the forward behavior specified by P terminates successfully, the recovery behavior specified by Q is recorded so that it can be executed to compensate the effect of P when triggered by an exception later. Otherwise, Q will not be executed. In the semantics $P \div Q$, the successfully terminated forward behavior defined by the traces in $\mathit{trace}_t(P) \cap \Sigma^\star_{\{\surd\}}$ is to be compensated by the execution of Q, and the non-successful terminated traces in $\Sigma^\star_{\{!,?\}}$ by "nothing", *i.e.* $\mathbf{skip}$, respectively.

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

The forward sub-processes of $\mathbf{skipp}$, $\mathbf{throww}$ and $\mathbf{yieldd}$ are $\mathbf{skip}$, $\mathbf{throw}$ and $\mathbf{yield}$, respectively. Their compensation sub-processes are all $\mathbf{skip}$. 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 $\mathbf{stopp}$ to denote any of these $\mathbf{stop} \div P$ whose forward behavior is $\mathbf{stop}$.

Transaction block A transaction block $[PP]$ is a standard process, and its semantics is derived from the semantics of the compensable process PP in the block.

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

The compensation of PP is executed to recover from an exception occurred 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 terminal, (b). the failures in the FF set that terminate with the exception terminal 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.

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

For example, $(a \div b_1 \sqcap a \div b_2) = (a \div (b_1 \sqcap b_2))$, whose FC set is $\{\langle b, \checkmark\rangle\} \times (\mathcal{F}(b_1) \cup \mathcal{F}(b_2))$, and the DC set is $\{\langle a, \checkmark\rangle\} \times (\mathcal{D}(b_1) \cup \mathcal{D}(b_2))$, *i.e.* $\{\}$. Internal choice is *idempotent*, *commutative* and *associative*.

External choice The external choice is also made during the forward behavior, but by the environment.

$$\begin{array}{l}\mathcal{D}_f(PP \,\Box\, QQ) = \mathcal{D}(fp(PP) \,\Box\, fp(QQ)),\ \mathcal{F}_f(PP \,\Box\, QQ) = \mathcal{F}(fp(PP) \,\Box\, fp(QQ)) \\ \mathcal{F}_c(PP \,\Box\, QQ) = \mathcal{F}_c(PP) \cup \mathcal{F}_c(QQ), \quad \mathcal{D}_c(PP \,\Box\, QQ) = \mathcal{D}_c(PP) \cup \mathcal{D}_c(QQ)\end{array}$$

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 $\mathcal{F}_f(\mathbf{stopp} \sqcap a \div b) = \mathcal{F}(a) \cup \mathcal{F}(\mathbf{stop})$. $\Box$ is *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, just like the model of Sagas [12]. 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 $\mathit{trace}_n(PP) \times \mathit{trace}_f(QQ)$. The compensation behavior of $PP;QQ$ is defined by the two cases below.

1. The forward execution of PP terminates successfully and the compensation behaviors of PP and QQ will be sequentially composed in the reverse order.

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

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

$$\mathcal{D}_c^2 = \{(s, s_c) \mid (s, s_c) \in \mathcal{D}_c(PP) \wedge (s \neq t{\cdot}\langle\checkmark\rangle \vee s \notin \mathit{trace}_n(PP))\}$$
$$\mathcal{F}_c^2 = \{(s, s_c, X_c) \mid (s, s_c, X_c) \in \mathcal{F}_c(PP) \wedge (s \neq t{\cdot}\langle\checkmark\rangle \vee s \notin \mathit{trace}_n(PP))\}$$

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

$$\mathcal{D}_f(PP \underset{X}{\parallel} QQ) = \mathcal{D}(fp(PP) \underset{X}{\parallel} fp(QQ)), \quad \mathcal{F}_f(PP \underset{X}{\parallel} QQ)=\mathcal{F}(fp(PP) \underset{X}{\parallel} fp(QQ))$$
$$\begin{aligned}\mathcal{D}_c(PP \underset{X}{\parallel} QQ) = \{(s, s_c) \mid {} & \exists s_1 \in \mathit{trace}_f(PP), s_2 \in \mathit{trace}_f(QQ)\bullet \\ & s \in (s_1 \underset{X}{\parallel} s_2) \wedge s_c \in \mathcal{D}(\mathit{cp}(PP, s_1) \underset{X}{\parallel} \mathit{cp}(QQ, s_2))\}\end{aligned}$$
$$\begin{aligned}\mathcal{F}_c(PP \underset{X}{\parallel} QQ) = \{(s, s_c, X) \mid {} & \exists s_1 \in \mathit{trace}_f(PP), s_2 \in \mathit{trace}_f(QQ)\bullet \\ & s \in (s_1 \underset{X}{\parallel} s_2) \wedge (s_c, X) \in \mathcal{F}(\mathit{cp}(PP, s_1) \underset{X}{\parallel} \mathit{cp}(QQ, s_2))\}\end{aligned}$$

Consider two examples. First, the equation $[(a{\div}b_1 \underset{\{a\}}{\parallel} b{\div}b_2);\mathbf{throww}] = a;b_1 \parallel b_2$ 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 a deadlock in the forward behavior.

Speculative Choice In a speculative choice $PP \boxtimes QQ$, 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 the compensation behaviors of PP and QQ will run in parallel to recover. Let $\mathbb{T}$ be the set $\mathit{trace}_f(PP) \times \mathit{trace}_f(QQ)$, $\mathcal{D}_f^1$ (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, where $\omega \in \Omega$:

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

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

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:

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

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

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

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.

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

   where $T_1 = (t_1 \| t_2) \cdot trace_t(cp(QQ, t_2 \cdot \langle \omega \rangle))$. Notice that the overall compensation of PP by $PP \boxtimes QQ$ can only start 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, where $T_2 = (t_1 \| t_2) \cdot trace_t(cp(PP, t_1 \cdot \langle \omega \rangle))$.

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

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 $\omega_1, \omega_2 \in \Omega \setminus \{\checkmark\}$.

$$\mathcal{D}_c^3 = \{(s, s_c) | \exists (t_1 \cdot \langle \omega_1 \rangle, t_2 \cdot \langle \omega_2 \rangle) \in \mathbb{T} \bullet s \in (t_1 \cdot \langle \omega_1 \rangle \| t_2 \cdot \langle \omega_2 \rangle) \wedge s_c \in \mathcal{D}(cp(PP, t_1 \cdot \langle \omega_1 \rangle) \| cp(QQ, t_2 \cdot \langle \omega_2 \rangle))\}$$

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

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

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.

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

Both hiding and renaming are distributive among internal choice. Not like renaming, hiding is not distributive among external choice. For example, the compensable process $((a;a_1) \div b \Box (a;a_2) \div b) \setminus \{a\}$ equals $a_1 \div b \sqcap a_2 \div b$.

5 Refinement Theory and Recursion Semantics

We define an order $\sqsubseteq$ on the semantic domain of the standard processes, and a partial order $\sqsubseteq_c$ on that of of the compensable processes. Each of the domains with the respective order forms a CPO, and their corresponding process operators are monotonic and continuous. The two orders are linked by the transaction block constructor $[PP]$. These form the theoretical foundation for the fixed-point theories of recursive standard and compensable processes, and for the refinement calculus of cCSP. We refer the reader to the technical report [10] for the proofs of theorems and laws.

5.1 Refinement of standard processes

The order $(F_1, D_1) \sqsubseteq (F_2, D_2)$ holds for the two FD pairs if $F_1 \supseteq F_2$ and $D_1 \supseteq D_2$. The *FD refinement* of a standard process P_1 by a standard process P_2 is defined as

$$P_1 \sqsubseteq P_2 \mathrel{\widehat{=}} \mathcal{F}(P_1) \supseteq \mathcal{F}(P_2) \wedge \mathcal{D}(P_1) \supseteq \mathcal{D}(P_2) \tag{8}$$

It means the refinement P_2 is neither more likely to refuse an interaction from the environment nor more likely to diverge than P_1.

Theorem 1. *The semantic domain of standard processes is a CPO under the refinement order $\sqsubseteq$, and $\mathbf{div}$ is the bottom (least) element. And 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 Theorem 1, 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 $a^0 = \langle\rangle$ and $a^{i+1} = \langle a\rangle{\cdot}a^i$, and the divergences $\mathcal{D}(\mu\, p.\ (a; p)) = \{\}$.

5.2 Refinement order of compensable processes

Given two tuples $PP_i = (F_i, D_i, F_i^c, D_i^c)$, $i \in \{1, 2\}$, of the semantic domain of compensable processes, we define the order

$$(F_1,D_1,F_1^c,D_1^c) \sqsubseteq_c (F_2,D_2,F_2^c,D_2^c) \mathrel{\widehat{=}} F_1 \supseteq F_2 \wedge D_1 \supseteq D_2 \wedge F_1^c \supseteq F_2^c \wedge D_1^c \supseteq D_2^c \tag{9}$$

A compensable process PP_2 is a *FD-refinement* of PP_1, also denoted by $PP_1 \sqsubseteq_c PP_2$, if their semantics are related by the order $\sqsubseteq_c$.

Theorem 2. *The semantic domain of compensable processes is a CPO w.r.t. $\sqsubseteq_c$ and $\mathbf{div} \div \mathbf{div}$ is the bottom element. And the operators of compensable processes are continuous w.r.t. $\sqsubseteq_c$.*

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.*

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

We thus can reduce refinement of compensable processes to that of standard processes.

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

5.3 Laws of long running transactions

The semantic theory provides the basis for proving fundamental laws of programming of long running transactions. Figure 2 gives some basic laws.

Units and zeros

$\mathbf{skip};P = P, \quad P;\mathbf{skip} = P, \quad P \triangleright \mathbf{throw} = P, \quad \mathbf{throw} \triangleright P = P$
$\mathbf{skipp};PP = PP, \;\; PP;\mathbf{skipp} = PP, \;\; \mathbf{throw};P = \mathbf{throw}, \;\; \mathbf{throww};PP = \mathbf{throww}$
$\mathbf{skip} \triangleright P{=}\mathbf{skip}, \quad \mathbf{yield} \triangleright P{=}\mathbf{yield}, \quad \mathbf{stop} \triangleright P{=}\mathbf{stop}$

Basic terminal processes

$\mathbf{skip}\Box\mathbf{yield}{=}\mathbf{yield}, \;\; \mathbf{skip}\Box\mathbf{throw}{=}\mathbf{skip}\sqcap\mathbf{throw}, \;\; \mathbf{yield}\Box\mathbf{throw} = \mathbf{yield}\sqcap\mathbf{throw}$
$\mathbf{yield} \underset{X}{\parallel} \mathbf{skip} = \mathbf{yield}, \quad \mathbf{throw} \underset{X}{\parallel} \mathbf{skip} = \mathbf{throw}, \quad \mathbf{throw} \underset{X}{\parallel} \mathbf{yield} = \mathbf{throw}$

Distribution laws

$[PP \sqcap QQ]{=}[PP] \sqcap [QQ], \;\; [PP \Box QQ]{=}[PP]\Box[QQ], \;\; P \div (Q \sqcap R){=}P \div Q \sqcap P \div R$
$(P \sqcap Q) \div R{=}(P \div R) \sqcap (Q \div R), \;\; (P \div Q)\backslash X{=}(P\backslash X \div Q\backslash X), \;\; [PP\backslash X]{=}[PP]\backslash X$

Fig. 2: Basic laws of long running transactions

Compensation The Saga nature of the backward recovery is reflected by the two laws below, where P, P_1 and P_2 are assumed not to terminate with an exception terminal.

$$[P \div Q;\mathbf{throww}] = P;Q, \quad [P_1 \div Q_1;P_2 \div Q_2;\mathbf{throww}] = P_1;P_2;Q_2;Q_1$$

Furthermore, the parallel composition enjoys the following laws.

$$[(P \div Q) \| \mathbf{throww}] = P;Q, \quad P_1 \div Q_1 \underset{X}{\|} P_2 \div Q_2 = (P_1 \underset{X}{\|} P_2) \div (Q_1 \underset{X}{\|} Q_2)$$

where P, Q, P_i, Q_i, $i \in \{1, 2\}$, do not diverge and terminate successfully, and with the same assumption, the following two laws hold.

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

For P_1, P_2, Q_1, Q_2 that do not diverge and terminate successfully, the speculative choice non-deterministically chooses one side to compensate if both succeed, which is

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

Interruption In a composition of standard processes, the interruption of one process by the other does not have priority over other events. That is, if P does not non-divergently terminate in an yield terminal, *i.e.* $\mathit{trace}_t(P) \cap \Sigma^{\star}_{\{?\}} = \{\}$, we have

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

A compensable process can be interrupted by $\mathbf{yieldd}$ to yield to an interruption from the environment, but a compensable process will not be interrupted if no $\mathbf{yieldd}$ is used (cf. the laws of **Compensation**). We thus have the following two laws, in which all the standard processes will not diverge and are assumed to terminate successfully.

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

6 Case study

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 that the Agency has to repeatedly request for the car rental service from the center at the destination before 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 carried successfully, *e.g.* canceling the air tickets, room reservations or car rentals, and the Agency sends a letter to the traveler for an apology. The alphabet Σ of the processes is

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

The processes Agency, Air1, Air2, Car, Hotel and Bank are as follows.

$$\begin{aligned}\texttt{Agency} = {} & (\texttt{reqTravel} \div \texttt{letter});\texttt{Res};\\ & (\texttt{confirm};(\texttt{agree} \,\Box\, (\texttt{disAgree};\texttt{throw}))) \div \texttt{skip};\\ & (\texttt{checkCredit};(\texttt{valid} \,\Box\, (\texttt{inValid};\texttt{throw}))) \div \texttt{skip};\\ & (\texttt{payment} \div \texttt{refund});(\texttt{pValid} \,\Box\, (\texttt{pInValid};\texttt{throw})) \div \texttt{skip};\\ & \texttt{result} \div \texttt{skip}\\ \texttt{Res} = {} & (\texttt{reqHotel};(\texttt{okRoom}\Box(\texttt{noRoom};\texttt{throw})))\div\texttt{cancelHotel}\|\\ & ((\texttt{yieldd};(\texttt{bookAir1};(\texttt{okAir1}\Box(\texttt{noAir1};\texttt{throw})))\div\texttt{cancelAir1})\boxtimes\\ & (\texttt{yieldd};(\texttt{bookAir2};(\texttt{okAir2}\Box(\texttt{noAir2};\texttt{throw})))\div\texttt{cancelAir2}))\|\\ & \mu\, pp.(\texttt{reqCar}\div\texttt{skip};((\texttt{noCar}\div\texttt{skip});pp)\Box(\texttt{hasCar}\div\texttt{cancelCar})))\\ \texttt{Hotel} = {} & \texttt{reqHotel}\div\texttt{skip};(\texttt{okRoom}\div\texttt{cancelHotel}\sqcap(\texttt{noRoom}\div\texttt{skip};\texttt{throww}))\\ \texttt{Air1} = {} & \texttt{bookAir1}\div\texttt{skip};(\texttt{okAir1}\div\texttt{cancelAir1}\sqcap(\texttt{noAir1}\div\texttt{skip};\texttt{throww}))\\ \texttt{Air2} = {} & \texttt{bookAir2}\div\texttt{skip};(\texttt{okAir2}\div\texttt{cancelAir2}\sqcap(\texttt{noAir2}\div\texttt{skip};\texttt{throww}))\\ \texttt{Car} = {} & \mu\, pp.(\texttt{reqCar} \div \texttt{skip};((\texttt{noCar}\div\texttt{skip});pp)\sqcap(\texttt{hasCar}\div\texttt{cancelCar})))\\ \texttt{Bank} = {} & (\texttt{checkCredit};(\texttt{valid} \sqcap (\texttt{inValid};\texttt{throw}))) \div \texttt{skip}\,;\\ & (\texttt{payment}\div\texttt{refund});(\texttt{pValid}\sqcap(\texttt{pInValid};\texttt{throw}))\div\texttt{skip}\end{aligned}$$

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

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

Use of laws GBP is a complex process. We thus hide some events in the forward behavior to get an abstract view, denoted by $\texttt{ABP} \mathrel{\widehat{=}} \texttt{GBP} \setminus X$, where

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

By the laws, we can transform the process ABP to the equivalent process ABP1 below.

$$
\begin{array}{ll}
\texttt{ABP1} & = \texttt{reqTravel} \div \texttt{letter};\texttt{Cancel};\texttt{PayRefund} \\
\texttt{Cancel} & = \texttt{CH} \parallel \texttt{CA} \parallel \texttt{CC} \\
\texttt{CH} & = (\texttt{throw} \sqcap \texttt{skip}) \div \texttt{cancelHotel} \\
\texttt{CA} & = \texttt{CA1} \boxtimes \texttt{CA2} \\
\texttt{CA1} & = \texttt{yieldd};((\texttt{throw} \sqcap \texttt{skip}) \div \texttt{cancelAir1}) \\
\texttt{CA2} & = \texttt{yieldd};((\texttt{throw} \sqcap \texttt{skip}) \div \texttt{cancelAir2}) \\
\texttt{CC} & = \mu\, pp.\ ((\texttt{noCar} \div \texttt{skip};pp) \sqcap (\texttt{skip} \div \texttt{cancelCar})) \\
\texttt{PayRefund} & = (\texttt{throww} \sqcap \texttt{skipp});\texttt{payment} \div \texttt{refund};(\texttt{throww} \sqcap \texttt{skipp})
\end{array}
$$

Analysis The formal theory, including the formal semantics and laws, can be used for rigorous analysis of progress GBP. For example, we can show the following results.

- GBP does not deadlock, may diverge if the car rental service cannot provide a car for the Agency. In that case events reqCar and noCar will be performed for an infinite number of times and they are in set X_4 of synchronized events.
- There are *five* different cases when an exception is raised: 1). no ticket from the airlines, 2). no room in the hotel, 3). the traveler refuses the offer, 4). credit card checking fails, and 5). authorization of payment fails. In any case, there are different cases for recovery because of the parallel composition in the reservation process.
- When there is no divergence but an exception, the compensation in GBP is

  $$
  \begin{array}{l}
  \texttt{refund} \sqcap \texttt{skip}; \\
  (\texttt{cancelAir1} \sqcap \texttt{cancelAir2} \sqcap \texttt{skip}) \parallel (\texttt{cancelHotel} \sqcap \texttt{skip}) \parallel \texttt{cancelCar}; \\
  \texttt{letter}
  \end{array}
  $$

  Therefore, event letter is the last action to be performed in the compensation when an exception occurs. When an exception is raised after the air ticket booking, *e.g.* when the traveler sends her disagreement, either cancelAir1 or cancelAir2 may be performed. The ticket booking can also be interrupted by an exception outside the airlines, Air1 and Air2, *e.g.* when no room is available in the hotel.

7 Conclusion

The full theory of CSP [17] is extended for specification and verification of LRTs. The extended theory of compensating CSP supports non-deterministic choice, parallel composition with synchronization and recursion. It allows us to handle problems of deadlock, livelock and nested LRTs. Its FD semantic theory also supports LRT program design by refinement, and transformations of specifications through algebraic laws.

The theory contributes to improving fundamental understanding of LRTs, and to underpinning the development of valid tool support to design and verification of LRTs.

From the way that the theory is developed, it is feasible to extend FDR to support our extended theory of cCSP. In addition, automated reasoning about LRTs can be developed based on our theory by following the ideas in the prototype theorem prover in [16]. Tool development along these two directions is part of our future research agenda.

Acknowledgement. We thank the referees for their valuable comments; our colleagues C. Bertolini, L. Chen, A.P. Ravn and H. Wang for the discussions and comments. The 2nd author acknowledges the support of NSFC 61073022 to his visit to Miaomiao Zhang at Tongji University in November 2010 when part of the research was done.

References

1. L. Bocchi, C. Laneve, and G. Zavattaro. A calculus for long-running transactions. In *Proc. FMOODS 2003, Lecture Notes In Computer Science 2884*, pages 124–138. Springer, 2003.
2. R. Bruni, M. J. Butler, C. Ferreira, C. A. R. Hoare, H. C. Melgratti, and U. Montanari. Comparing two approaches to compensable flow composition. In *Proc. CONCUR 2005, Lecture Notes in Computer Science 3653*, pages 383–397. Springer, 2005.
3. R. Bruni, H. C. Melgratti, and U. Montanari. Theoretical foundations for compensations in flow composition languages. In *Proc. POPL 2005*, pages 209–220. ACM Press, 2005.
4. M. J. Butler and C. Ferreira. An operational semantics for StAC, a language for modelling long-running business transactions. In *Proc. COORDINATION 2004, Lecture Notes in Computer Science 2949*, pages 87–104. Springer, 2004.
5. M. J. Butler, C. Ferreira, and M. Y. Ng. Precise modelling of compensating business transactions and its application to BPEL. *J. UCS*, 11(5):712–743, 2005.
6. M. J. Butler, C. A. R. Hoare, and C. Ferreira. A trace semantics for long-running transactions. In *25 Years Communicating Sequential Processes, Lecture Notes in Computer Science 3525*, pages 133–150. Springer, 2004.
7. M. J. Butler and S. Ripon. Executable semantics for compensating CSP. In *Proc. WS-FM 2005, Lecture Notes in Computer Science 3670*, pages 243–256. Springer, 2005.
8. G. Castagna, N. Gesbert, and L. Padovani. A theory of contracts for web services. In G. C. Necula and P. Wadler, editors, *POPL*, pages 261–272. ACM, 2008.
9. Z. Chen and Z. Liu. An Extended cCSP with Stable Failures Semantics. In *Proc. ICTAC, Lecture Notes in Computer Science 6255*, pages 121–136. Springer, 2010.
10. Z. Chen, Z. Liu, and J. Wang. A theory of failure-divergence refinement for long running transactions. Technical Report 447, UNU-IIST, 2011. `http://www.iist.unu.edu/www/docs/techreports/reports/report447.pdf`.
11. J. Fischer and R. Majumdar. Ensuring consistency in long running transactions. In *Proc. ASE 2007*, pages 54–63. ACM, 2007.
12. H. Garcia-Molina and K. Salem. SAGAS. In *Proc. SIGMOD 1987*, pages 249–259. ACM Press, 1987.
13. J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.
14. M. C. Little. Transactions and web services. *Commun. ACM*, 46(10):49–54, 2003.
15. Microsoft. Biztalk server. http://www.microsoft.com/biztalk/default.asp.
16. S. Ripon and M. J. Butler. PVS embedding of cCSP semantic models and their relationship. *Electr. Notes Theor. Comput. Sci.*, 250(2):103–118, 2009.
17. A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997.
18. A. W. Roscoe. The three platonic models of divergence-strict CSP. In *Proc. ICTAC, volume 5160 of Lecture Notes in Computer Science*, pages 23–49, 2008.