Comparing two approaches to compensable flow composition

Roberto Bruni[1], Michael Butler[2], Carla Ferreira[3], Tony Hoare[4], Hernán Melgratti[1], and Ugo Montanari[1]

[1] Dipartimento di Informatica, Università di Pisa, Italy
[2] School of Electronics and Computer Science, University of Southamptom, UK
[3] Department of Computer Science, Technical University of Lisbon, Portugal
[4] Microsoft Research Cambridge, UK

Abstract. Web services composition is an emerging paradigm for the integration of long running business processes, attracting the interest of both Industry, in terms of XML-based standards for business description, and Academy, exploiting process description languages. The key challenging aspects to model are orchestration workflows, choreography of exchanged messages, fault handling, and transactional integrity with compensation mechanisms. Few recent proposals attempted to mitigate the explosion of XML-constructs in ad hoc standards by a careful selection of a small set of primitives related to the above aspects. This papers clarifies analogies and differences between two such recent process description languages: one based on interleaving trace semantics and the other on concurrent traces. We take advantage of their comparison to characterise and relate four different coordination policies for compensating parallel processes. Such policies differ on the way in which the abort of a process influences the execution of sibling processes, and whether compensation is distributed or centralised.

1 Introduction

Orchestration and choreography languages are tailored to the definition of web service composition. Typically, these languages provide, among others, programming primitives for the definition of business transactions, i.e., transactions that may require long periods of time to complete, also called *Long-Running Transactions* (LRTs). Moreover, they may be interactive and hence not able to be check-pointed. Consequently, LRTs cannot be based on locking (as usual for database transactions), but instead they rely on a weaker notion of atomicity based on *compensations* [8]. Compensations are activities programmed ad hoc to recover partial executions of transactional processes.

The existing babel of approaches developed along the years for orchestration and choreography building on WSDL [15] (WSCL [14], WSCI [13], BPML [3], WSFL [10], XLANG [16], BPEL4WS [2]) witnesses the need of languages for service integration with solid theoretical foundations. Several proposals have recently appeared in the literature focused on the formalisation of compensable processes using process calculi. They can be roughly divided into two types: (i) compensable flow composition [6,5,7] closer to the spirit of orchestration languages like BPEL4WS, where suitable process algebras are designed from the scratch to describe the possible flow of control among services; and

(ii) interaction based compensations [1,4,9,11], as suitable extensions of well-known name passing calculi, like the π-calculus and join-calculus, for describing transactional choreographies, where each service describes its possible interactions, and the actual composition takes place dynamically, i.e. when services interact.

In this paper we pursue the first approach, i.e., to study the abstract composition of services according to basic workflow shapes (sequential and parallel) and compensable transaction mechanisms (compensable activities, compensation scope, transaction scope, nesting). Nevertheless, we are not aimed at designing a new language but at comparing two main proposals, namely *compensating* CSP (cCSP) [7] and Sagas calculi [5]. Apart from stylistic differences (e.g., the trace semantics of cCSP and the big step SOS semantics of Sagas calculi), this comparison highlights the fact that such proposals account for different compensation policies when handling concurrent processes. First of all, we characterise such policies as the combination of two orthogonal strategies: (i) whether parallel flows are forced to interrupt their executions when a sibling process aborts; and (ii) whether compensation handling is centralised or distributed. The combination of such strategies gives rise to the following four policies:

1. *No interruption and centralised compensation.* All concurrent processes execute until completion, and only then they are compensated for if some abort.
2. *No interruption and distributed compensation.* All parallel flows execute until completion but, if needed, they compensate without waiting the completion of siblings.
3. *Coordinated interruption.* Parallel branches may be stopped when one flow aborts, but the activation of the compensation procedure is handled in a centralised way, i.e., all component flows have to be stopped, and only then the corresponding compensations are executed.
4. *Distributed interruption.* Flows, if needed, are interrupted and then their compensation procedures can be activated independently from the rest of the flows.

We show that all these policies can be defined by following either the cCSP approach or the Sagas style. Moreover, we note that the semantics of original cCSP corresponds to policy (3), while the two original semantics of Sagas Calculi, called Naïve and Revised, follow respectively policies (2) and (4). Finally, we compare the four alternatives (and hence the original semantics of both proposals) by relating the set of traces that each policy associates to a process. In particular, we show that these policies form a partial order of traces, where original cCSP and Naïve Sagas are more restrictive than Revised Sagas, but original cCSP is unrelated to Naïve Sagas.

Structure of the paper. We start by recalling in § 2 the syntax and semantics of cCSP and Sagas from [5,7]. Then, we outline the conceptual and stylistic similarities and differences between the two approaches in § 3. The more technical contribution starts in § 4, where we focus on the key aspects for the sequential case, by taking the corresponding fragments of the calculi associated with sequential processes, for which we prove the correspondence of both semantics by means of two straightforward encodings. The different policies implemented by the two approaches emerge in § 5, where we analyse the case of parallel processes in transactions. The formal comparison of compensation policies is summarised in § 6. Finally, in § 7 we draw some conclusions and discuss future work. Due to space limitation most proofs are omitted and some just sketched.

2 Background

In this section we summarise the basics of *Compensating* CSP (cCSP) proposed in [7] and of Sagas calculi from [5]. We focus on simplified versions by leaving out several features present in both proposals, like exception handling and nesting.

2.1 Compensating CSP

The set of cCSP processes is defined by the following grammar:

(STANDARD PROCESSES)

$$P,Q ::= A \mid P;Q \mid P|Q \mid SKIP \mid THROW \mid YIELD \mid [PP]$$

(COMPENSABLE PROCESSES)

$$PP,QQ ::= P \div Q \mid PP;QQ \mid PP|QQ \mid SKIPP \mid THROWW \mid YIELDD$$

A standard process is either a basic activity A from an alphabet Σ, the sequential composition $P;Q$ of processes, the parallel composition $P|Q$, the empty process $SKIP$, the raise of an interruption $THROW$, the yield to an interruption $YIELD$, or a transaction block $[PP]$. A basic compensable process is a compensation pair $P \div Q$ where P is an atomic process and Q is its compensation. Compensable processes can be composed either in sequence $PP;QQ$ or in parallel $PP|QQ$. The remaining processes are the compensable counterpart of the standard ones.

Figure 1 summarises the trace semantics of cCSP. A trace for a standard process is a string $s\langle\omega\rangle$, where $s \in \Sigma^*$ is said the *observable flow* and $\omega \in \Omega$ is the *final event*, with $\Omega = \{\checkmark, !, ?\}$, with $\Sigma \cap \Omega = \emptyset$ ($\checkmark$ stands for success, ! for fail, and ? for yield). The sequential composition $p;q$ concatenates the observable flows of p and q only when p terminates with success, otherwise it is p. The composition of two concurrent traces $p\langle\omega\rangle||q\langle\omega'\rangle$ corresponds to the set $p|||q$ of all possible interleavings of the observable flows, with final event $\omega\&\omega'$, where & is associative and commutative.

The definition for the traces of standard processes is straightforward. The most interesting one is that of a transaction block $[PP]$. Note that any trace of a compensable process PP is a pair $(p\langle\omega\rangle, p')$, where $p\langle\omega\rangle$ is the forward trace and p' is a compensation trace for p. Then, $[PP]$ selects all successful traces $p\langle\checkmark\rangle$ of PP, and the traces pp', corresponding to failed forward flows $p\langle!\rangle$ followed by their compensations p'.

When composing compensable traces in series, the forward trace corresponds to the sequential composition of the original forward traces, while the compensation trace starts by the second compensation followed by the first one. The parallel composition is defined as all possible interleavings of the forward and the backward flows, separately.

2.2 Sagas Calculi

We report here the two alternative semantics proposed in [5] for parallel Sagas, namely the *naïve* and *revised* versions. The main difference between the two semantics is that the latter allows the interruption of flow executions when a transaction fails. The set of parallel Sagas is given by the following grammar:

COMPOSITION OF STANDARD TRACES

$$\textbf{Sequential} \begin{cases} p\langle\checkmark\rangle; q = pq \\ p\langle\omega\rangle; q = p\langle\omega\rangle \text{ when } \omega \neq \checkmark \end{cases}$$

$$\textbf{Parallel} \qquad p\langle\omega\rangle||q\langle\omega'\rangle = \{r\langle\omega\&\omega'\rangle | r \in (p|||q)\}, \qquad \text{where} \quad \begin{array}{c|cccccc} \omega & ! & ! & ! & ? & ? & \checkmark \\ \omega' & ! & ? & \checkmark & ? & \checkmark & \checkmark \\ \hline \omega\&\omega' & ! & ! & ! & ? & ? & \checkmark \end{array}$$

$$\text{and} \begin{cases} p|||\langle\rangle = \{p\} \\ \langle\rangle|||q = \{q\} \\ \langle x\rangle p|||\langle y\rangle q = \{\langle x\rangle r | r \in (p|||\langle y\rangle q)\} \cup \{\langle y\rangle r | r \in (\langle x\rangle p|||q)\} \end{cases}$$

TRACES OF STANDARD PROCESSES

$$\begin{array}{rl} A =_{\mathsf{traces}} & \{\langle A, \checkmark\rangle\} \quad \text{for } A \in \Sigma \\ P;Q =_{\mathsf{traces}} & \{p;q | p \in P \wedge q \in Q\} \\ P|Q =_{\mathsf{traces}} & \{r | r \in (p||q) \wedge p \in P \wedge q \in Q\} \\ [PP] =_{\mathsf{traces}} & \{pp' | (p\langle !\rangle, p') \in PP\} \cup \{p\langle\checkmark\rangle | (p\langle\checkmark\rangle, p') \in PP\} \end{array} \qquad \begin{array}{rl} SKIP =_{\mathsf{traces}} & \{\langle\checkmark\rangle\} \\ THROW =_{\mathsf{traces}} & \{\langle !\rangle\} \\ YIELD =_{\mathsf{traces}} & \{\langle ?\rangle\} \end{array}$$

COMPOSITION OF COMPENSABLE TRACES

$$\textbf{Sequential} \quad \begin{cases} (p\langle\checkmark\rangle, p'); (q, q') = (pq, q'; p') \\ (p\langle\omega\rangle, p'); (q, q') = (p\langle\omega\rangle, p') \text{ when } \omega \neq \checkmark \end{cases}$$

$$\textbf{Parallel} \quad (p, p')||(q, q') = \{(r, r') | r \in (p||q) \wedge r' \in (p'||q')\}$$

$$\textbf{Compensation pair} \begin{cases} p\langle\checkmark\rangle \div q = (p\langle\checkmark\rangle, q) \\ p\langle\omega\rangle \div q = (p\langle\omega\rangle, \langle\checkmark\rangle) \text{ when } \omega \neq \checkmark \end{cases}$$

TRACES OF COMPENSABLE PROCESSES

$$\begin{array}{rl} P \div Q =_{\mathsf{traces}} & \{(\langle ?\rangle, \langle\checkmark\rangle)\} \cup \{p \div q | p \in P \wedge q \in Q\} \\ PP;QQ =_{\mathsf{traces}} & \{pp;qq | pp \in PP \wedge qq \in QQ\} \\ PP|QQ =_{\mathsf{traces}} & \{rr | rr \in (pp||qq) \wedge pp \in PP \wedge qq \in QQ\} \\ SKIPP =_{\mathsf{traces}} & SKIP \div SKIP \ =_{\mathsf{traces}} \ \{(\langle ?\rangle, \langle\checkmark\rangle), (\langle\checkmark\rangle, \langle\checkmark\rangle)\} \\ THROWW =_{\mathsf{traces}} & THROW \div SKIP \ =_{\mathsf{traces}} \ \{(\langle ?\rangle, \langle\checkmark\rangle), (\langle !\rangle, \langle\checkmark\rangle)\} \\ YIELDD =_{\mathsf{traces}} & YIELD \div SKIP \ =_{\mathsf{traces}} \ \{(\langle ?\rangle, \langle\checkmark\rangle)\} \end{array}$$

Fig. 1. Trace semantics of cCSP

$$\begin{array}{lrcl} (\text{STEP}) & X & ::= & 0 \mid A \mid A \div B \\ (\text{PROCESS}) & P & ::= & X \mid P;P \mid P|P \\ (\text{SAGA}) & S & ::= & \{[P]\} \end{array}$$

A saga S encloses a process P in a transaction scope. Each step in P corresponds either to an activity A or a compensated activity $A \div B$, where A is the activity of the normal flow and B its compensation. The term 0 represents the inert process, $P;P$ stands for the sequential composition of processes, and $P|P$ for the parallel composition.

To reduce the number of rules, the semantics of Sagas is defined up-to structural congruence over processes given by the following axioms:

$$\begin{array}{rclcrclcrcl} A \div 0 & \equiv & A & \quad & 0;P \equiv P;0 & \equiv & P & \quad & (P;Q);R & \equiv & P;(Q;R) \\ P|Q & \equiv & Q|P & & P|0 & \equiv & P & & P|(Q|R) & \equiv & (P|Q)|R \end{array}$$

Moreover, activities are assumed to be named differently. The set of possible results for the execution of a saga is $\mathcal{R} = \{\boxdot, \boxtimes, \boxast\}$, where $\boxdot$ stands for *commit*, $\boxtimes$ for (compensated) *abort*, and $\boxast$ for *abnormal termination* (when the compensation procedure fails). We let $\Box$ to range over $\mathcal{R}$. The execution of a sequential saga is described in terms of a context Γ, i.e., a partial function $\Gamma : \mathcal{A} \to \{\boxtimes, \boxdot\}$ that maps any activity to the result obtained with its execution. Activities can only commit or abort (they do not terminate abnormally). A particular function Γ is written $A_1 \mapsto \Box_1, \ldots, A_n \mapsto \Box_n$, where $A_i \neq A_j$ for all $i \neq j$. (Note that ',' stands for the disjoint union of partial functions).

The semantics of a saga S is given by a relation $\Gamma \vdash S \xrightarrow{\alpha} \Box$, which denotes that the execution of S produces $\Box$ when the atomic activities behave like Γ. The observation α describes the actual flow of control occurring when executing S under the context Γ. The flow α is a process whose activities have no compensations. The auxiliary relation $\Gamma \vdash \langle P, \beta \rangle \xrightarrow{\alpha} \langle \Box, \beta' \rangle$ describes the behaviour of a process P within a saga that already installed the compensation β (but β itself contains no compensation). When P is executed inside a saga, it can either commit, abort, or fail, but additionally, it can change the compensations to β', for instance by installing new activities.

Naïve semantics. The naïve semantics for a parallel saga is shown in Figure 2(a). Rule (S-ACT) stands for the successful execution of the compensated activity $A \div B$ that installs B in front of β. Rules (S-CMP) and (F-CMP) describe the execution of $A \div B$ when A fails. Both rules activate the compensation β (premises of the rules). In particular, (S-CMP) stands for the successful compensation, while rule (F-CMP) handles the failure of the compensation procedure. Rule (S-STEP) describes the behaviour of a process $P;Q$ when the step P commits. In such case, Q is executed with the compensation installed by P. Rule (A-STEP) handles the case in which $P;Q$ is stopped because P aborts or ends abnormally. Rule (SAGA) states that the execution of a saga $\{[P]\}$ runs P in a thread that initially has no compensations. The rules described above give the semantics for the sequential case, while the remaining rules define the naïve semantics of parallel composition. Rule (S-PAR) deals with the successful execution of both branches, while the remaining rules handle the cases in which at least one branch fails.

Revised semantics. The revised semantics avoids the unnecessary execution of activities in the forward flow when the saga fails. This is achieved by stopping the execution of the forward flow when some activity fails. For this reason, the execution of a process may also finish with: (i) $\overline{\boxtimes}$, i.e. the execution is forced to compensate and the compensation is successful, and (ii) $\overline{\boxast}$, i.e., the execution is forced to compensate and the compensation procedure fails. The associative and commutative operator $\wedge$ expresses the result obtained by combining the execution of two parallel branches (see Figure 3). Note that $\wedge$ is not defined when one operand is $\boxdot$ and the other is not. In fact, it is not possible for a branch to commit when the other aborts or fails: in $P|Q$ when P can commit but Q aborts, then P is forced to compensate.

For the revised semantics, all rules for the sequential case are as in Figure 2(b), but considering for rule (A-STEP) the side condition $\sigma \in \{\boxtimes, \boxast, \overline{\boxtimes}, \overline{\boxast}\}$), and for rule (SAGA) the side condition $\Box \in \{\boxdot, \boxtimes, \boxast\}$. In addition, rules in Figure 2(b) describe the behaviour of concurrent processes. Rule (FORCED-ABT) forces the activation of the

(ZERO)
$$\Gamma \vdash \langle 0, \beta\rangle \xrightarrow{0} \langle \boxdot, \beta\rangle$$

(S-ACT)
$$A \mapsto \boxdot, \Gamma \vdash \langle A \div B, \beta\rangle \xrightarrow{A} \langle \boxdot, B;\beta\rangle$$

(S-CMP)
$$\frac{\Gamma \vdash \langle \beta, 0\rangle \xrightarrow{\alpha} \langle \boxdot, 0\rangle}{A \mapsto \boxtimes, \Gamma \vdash \langle A \div B, \beta\rangle \xrightarrow{\alpha} \langle \boxtimes, 0\rangle}$$

(F-CMP)
$$\frac{\Gamma \vdash \langle \beta, 0\rangle \xrightarrow{\alpha} \langle \boxtimes, 0\rangle}{A \mapsto \boxtimes, \Gamma \vdash \langle A \div B, \beta\rangle \xrightarrow{\alpha} \langle \boxast, 0\rangle}$$

(S-STEP)
$$\frac{\Gamma \vdash \langle P, \beta\rangle \xrightarrow{\alpha} \langle \boxdot, \beta''\rangle \quad \Gamma \vdash \langle Q, \beta''\rangle \xrightarrow{\alpha'} \langle \Box, \beta'\rangle}{\Gamma \vdash \langle P;Q, \beta\rangle \xrightarrow{\alpha;\alpha'} \langle \Box, \beta'\rangle}$$

(A-STEP)
$$\frac{\Gamma \vdash \langle P, \beta\rangle \xrightarrow{\alpha} \langle \sigma, 0\rangle}{\Gamma \vdash \langle P;Q, \beta\rangle \xrightarrow{\alpha} \langle \sigma, 0\rangle} \quad \sigma \in \{\boxtimes, \boxast\}$$

(SAGA)
$$\frac{\Gamma \vdash \langle P, 0\rangle \xrightarrow{\alpha} \langle \Box, \beta\rangle}{\Gamma \vdash \{[P]\} \xrightarrow{\alpha} \Box}$$

(S-PAR)
$$\frac{\Gamma \vdash \langle P, 0\rangle \xrightarrow{\alpha} \langle \boxdot, \beta'\rangle \quad \Gamma \vdash \langle Q, 0\rangle \xrightarrow{\alpha'} \langle \boxdot, \beta''\rangle}{\Gamma \vdash \langle P|Q, \beta\rangle \xrightarrow{\alpha|\alpha'} \langle \boxdot, (\beta'|\beta'');\beta\rangle}$$

(F-PAR-NAÏVE-1)
$$\frac{\Gamma \vdash \langle P, 0\rangle \xrightarrow{\alpha} \langle \boxtimes, 0\rangle \quad \Gamma \vdash \langle Q, 0\rangle \xrightarrow{\alpha'} \langle \boxtimes, 0\rangle \quad \Gamma \vdash \langle \beta, 0\rangle \xrightarrow{\alpha''} \langle \Box_1, 0\rangle}{\Gamma \vdash \langle P|Q, \beta\rangle \xrightarrow{(\alpha|\alpha');\alpha''} \langle \Box_2, 0\rangle} \quad \Box_2 = \begin{cases} \boxtimes & \text{if } \Box_1 = \boxdot \\ \boxast & \text{otherwise} \end{cases}$$

(F-PAR-NAÏVE-2)
$$\frac{\Gamma \vdash \langle P, 0\rangle \xrightarrow{\alpha} \langle \boxast, 0\rangle \quad \Gamma \vdash \langle Q, 0\rangle \xrightarrow{\alpha'} \langle \boxdot, \beta'\rangle \quad \Gamma \vdash \langle \beta', 0\rangle \xrightarrow{\alpha''} \langle \Box, 0\rangle}{\Gamma \vdash \langle P|Q, \beta\rangle \xrightarrow{\alpha|(\alpha';\alpha'')} \langle \boxast, 0\rangle}$$

(F-PAR-NAÏVE-3)
$$\frac{\Gamma \vdash \langle P, 0\rangle \xrightarrow{\alpha} \langle \boxast, 0\rangle \quad \Gamma \vdash \langle Q, 0\rangle \xrightarrow{\alpha'} \langle \sigma, 0\rangle}{\Gamma \vdash \langle P|Q, \beta\rangle \xrightarrow{(\alpha|\alpha')} \langle \boxast, 0\rangle} \quad \textit{with}\ \sigma \in \{\boxtimes, \boxast\}$$

(F-PAR-NAÏVE-4A)
$$\frac{\Gamma \vdash \langle P, 0\rangle \xrightarrow{\alpha} \langle \boxdot, \beta'\rangle \quad \Gamma \vdash \langle Q, 0\rangle \xrightarrow{\alpha'} \langle \boxtimes, 0\rangle \quad \Gamma \vdash \langle \beta', 0\rangle \xrightarrow{\alpha''} \langle \boxtimes, 0\rangle}{\Gamma \vdash \langle P|Q, \beta\rangle \xrightarrow{(\alpha;\alpha'')|\alpha'} \langle \boxast, 0\rangle}$$

(F-PAR-NAÏVE-4B)
$$\frac{\Gamma \vdash \langle P, 0\rangle \xrightarrow{\alpha} \langle \boxdot, \beta'\rangle \quad \Gamma \vdash \langle Q, 0\rangle \xrightarrow{\alpha'} \langle \boxtimes, 0\rangle \quad \Gamma \vdash \langle \beta', 0\rangle \xrightarrow{\alpha''} \langle \boxdot, 0\rangle \quad \Gamma \vdash \langle \beta, 0\rangle \xrightarrow{\alpha'''} \langle \Box_1, 0\rangle}{\Gamma \vdash \langle P|Q, \beta\rangle \xrightarrow{((\alpha;\alpha'')|\alpha');\alpha'''} \langle \Box_2, 0\rangle} \quad \Box_2 = \begin{cases} \boxtimes & \text{if } \Box_1 = \boxdot \\ \boxast & \text{otherwise} \end{cases}$$

(a) Naïve semantics of parallel Sagas.

(FORCED-ABT)
$$\frac{\Gamma \vdash \langle \beta, 0\rangle \xrightarrow{\alpha} \langle \Box_1, 0\rangle}{\Gamma \vdash \langle P, \beta\rangle \xrightarrow{\alpha} \langle \Box_2, 0\rangle} \quad \Box_2 = \begin{cases} \overline{\boxtimes} & \text{if } \Box_1 = \boxdot \\ \boxast & \text{otherwise} \end{cases}$$

(S-PAR)
$$\frac{\Gamma \vdash \langle P, 0\rangle \xrightarrow{\alpha} \langle \boxdot, \beta'\rangle \quad \Gamma \vdash \langle Q, 0\rangle \xrightarrow{\alpha'} \langle \boxdot, \beta''\rangle}{\Gamma \vdash \langle P|Q, \beta\rangle \xrightarrow{\alpha|\alpha'} \langle \boxdot, (\beta'|\beta'');\beta\rangle}$$

(F-PAR)
$$\frac{\Gamma \vdash \langle P, 0\rangle \xrightarrow{\alpha} \langle \sigma_1, 0\rangle \quad \Gamma \vdash \langle Q, 0\rangle \xrightarrow{\alpha} \langle \sigma_2, 0\rangle}{\Gamma \vdash \langle P|Q, \beta\rangle \xrightarrow{\alpha|\alpha'} \langle \sigma_1 \wedge \sigma_2, 0\rangle} \quad \begin{cases} \sigma_1 \in \{\boxast, \overline{\boxast}\} \\ \sigma_2 \in \{\boxtimes, \boxast, \overline{\boxtimes}, \overline{\boxast}\} \end{cases}$$

(C-PAR)
$$\frac{\Gamma \vdash \langle P, 0\rangle \xrightarrow{\alpha} \langle \sigma_1, 0\rangle \quad \Gamma \vdash \langle Q, 0\rangle \xrightarrow{\alpha'} \langle \sigma_2, 0\rangle \quad \Gamma \vdash \langle \beta, 0\rangle \xrightarrow{\gamma} \langle \Box_1, 0\rangle}{\Gamma \vdash \langle P|Q, \beta\rangle \xrightarrow{(\alpha|\alpha');\gamma} \langle \sigma_1 \wedge \sigma_2 \wedge \Box_2, 0\rangle} \quad \sigma_1, \sigma_2 \in \{\boxtimes, \overline{\boxtimes}\} \text{ and } \Box_2 = \begin{cases} \overline{\boxtimes} & \text{if } \Box_1 = \boxdot \\ \overline{\boxast} & \text{otherwise} \end{cases}$$

(b) Revised semantics of parallel Sagas.

Fig. 2. Concurrent semantics of Sagas.

$\wedge$	$\boxdot$	$\boxtimes$	$\boxast$	$\overline{\boxtimes}$	$\overline{\boxast}$
$\boxdot$	$\boxdot$	—	—	—	—
$\boxtimes$	—	$\boxtimes$	$\boxast$	$\boxtimes$	$\boxast$
$\boxast$	—	$\boxast$	$\boxast$	$\boxast$	$\boxast$
$\overline{\boxtimes}$	—	$\boxtimes$	$\boxast$	$\overline{\boxtimes}$	$\overline{\boxast}$
$\overline{\boxast}$	—	$\boxast$	$\boxast$	$\overline{\boxast}$	$\overline{\boxast}$

Fig. 3. The operator $\wedge$.

compensation before executing P, which will produce a forced termination $\overline{\boxtimes}$ or $\overline{\boxast}$. Rule (S-PAR) is the same as in the naïve semantics, while the rollback of a branch is handled by (F-PAR) and (C-PAR). Rule (C-PAR) handles the case in which both P and Q are successfully compensated for, while (F-PAR) handles the failure of the compensation procedure.

An interesting aspect on the revised semantics is that rule (SAGA) requires P to end with $\boxdot$, $\boxtimes$ or $\boxast$, but not with forced termination. This implies that a saga aborts if and only if (at least) one activity aborts.

3 cCSP vs Sagas Calculi

In this section we try to enucleate the main conceptual differences between the two approaches and to give an informal account of the underlying different policies for business process design and execution.

Executions of activities. An activity A is always successful in cCSP. Instead, the execution of activities in Sagas depends on a particular execution context Γ, which allows to evaluate the semantics of a process according to different scenarios.

Aborted activities vs Programmable abort. In cCSP the special primitive *THROW* introduces programmable aborts. Instead, the abort of a saga is caused by the abort of an activity in the scenario Γ. Thus, the primitive *THROW* roughly corresponds to a Sagas activity that always fails in any Γ.

Yielding to interrupt. In cCSP the yielding to interrupt is explicitly programmed by using the special primitive *YIELD*. Instead, in Sagas the yielding to interrupt is wired in the semantics rules and cannot be programmed.

Failed compensations. Different from Sagas calculi, the abort of and the successful compensation of a transaction block in cCSP is silent to the parent process, i.e., there is no possibility to distinguish this case from the situation in which the forward flow complete successfully. Although not reported in § 2, Sagas calculi provide the primitive **try** S **or** P in [5], which allows to activates P when S aborts and it is compensated successfully.

Interleaving vs concurrent traces. The semantics of a cCSP process is given by listing all possible executions that differ on the interleaving of their concurrent activities. Instead, in the Sagas calculi computations are described up-to interleavings. Note that any label α in a reduction denotes a set of possible executions.

Compensation of parallel processes. As described in § 1, the most important distinction of both proposal is when defining the compensation for $P|Q$, since they use different compensation policies. This distinction is formalised in § 5

Nesting. The primitive $P \div Q$ of cCSP allow for the nesting of transactions. The Sagas counterpart is called nested Sagas and it is presented in [5], which provides two different kinds of compensations called *default compensations* and *programmed compensation*. The latter is equivalent to the cCSP primitive. The common fragment to cCSP and Sagas we shall discuss does not allow nesting, and therefore only compensable activities $A \div B$ will be considered.

Adequacy of the semantics. Although not described here, correctness of cCSP semantics is stated in terms of self-cancelling properties. That is, when assuming compensations to be perfect, it is shown that the execution of a transaction is equivalent to its forward flow or to *SKIP*. In Sagas, the meaning of the execution of a transaction is shown by suitable adequacy theorems, which are more precise but less intuitive and more complex to express than the self-cancelling properties.

In the rest of the paper we shall focus on the formal comparison of the sequential and parallel fragments of the two calculi, leaving to future work the treatment of the last two items from the above list (nesting and adequacy). The yielding modality and parallel compensations are discussed in detail in § 5, while all the remaining items are relevant also for the sequential fragment in § 4.

4 The sequential case

In this section we focus on the subset of sequential processes and we show that both semantics coincide by giving two encodings. Sequential cCSP is obtained by restricting the syntax of compensable processes as follow.

$$PP, QQ ::= \ A \div B \ \mid \ PP;QQ \ \mid \ SKIPP \ \mid \ THROWW \ \mid \ YIELDD$$

Note that instead of having $P \div Q$, we only allow basic activities to be compensated by basic activities. Sequential Sagas is obtained by forbidding the parallel composition of processes $P|P$. We denote by $\mathsf{cCSP}_{\mathsf{seq}}$ the set of sequential cCSP processes, and by $\mathsf{Sagas}_{\mathsf{seq}}$ the set of sequential Sagas processes.

4.1 Encoding $\mathsf{cCSP}_{\mathsf{seq}}$ into $\mathsf{Sagas}_{\mathsf{seq}}$

The main idea is that any process $PP \in \mathsf{cCSP}_{\mathsf{seq}}$ is associated with both a saga process $P \in \mathsf{Sagas}_{\mathsf{seq}}$ and a particular environment $\Gamma \in \nabla$ in which all activities of P commits (∇ stands for the set of all possible environments). Moreover, the *THROWW* primitive is represented by a fresh activity that aborts in Γ. The last subtlety is that all activities in P have to be named differently, for this reason the encoding assures activities in P to have different names. Formally, the encoding is given by the following function

$$[\![\_]\!] : \mathsf{cCSP}_{\mathsf{seq}} \to \mathsf{Sagas}_{\mathsf{seq}} \times \nabla$$

which is defined in terms of the auxiliary function (used to assure activity names to be different)

$$[\![\_]\!]_{\_} : \mathsf{cCSP}_{\mathsf{seq}} \times \mathbb{N}^* \to \mathsf{Sagas}_{\mathsf{seq}} \times \nabla$$

The encoding is defined by letting $[\![PP]\!] = [\![PP]\!]_0$, with:

$$[\![A \div B]\!]_\sigma = A_\sigma \div B_\sigma, \{A_\sigma \mapsto \boxdot, B_\sigma \mapsto \boxdot\}$$
$$[\![PP_1;PP_2]\!]_\sigma = P_1;P_2,\Gamma_1 \uplus \Gamma_2 \quad s.t.\ [\![PP_i]\!]_{\sigma.i} = P_i,\Gamma_i \quad \text{for } i = 1,2$$
$$[\![SKIPP]\!]_\sigma = [\![YIELDD]\!]_\sigma = 0,\emptyset \qquad [\![THROWW]\!]_\sigma = \mathsf{T}_\sigma, \{\mathsf{T}_\sigma \mapsto \boxtimes\}$$

Notation 1 *We let $\llcorner\alpha\lrcorner$ be obtained from α by removing all the subscripts σ from activities and by considering 0 as SKIP. Given a saga S, we let $\mathcal{A}(S) = \{A \mid A \text{ occurs in } S\}$ be the set of its activities and $|S|$ be its* forward flow*, which is obtained by replacing the pattern $A \div B$ by A everywhere in S (i.e., removing all compensations).*

Theorem 4.1. *Let $[\![PP]\!] = P,\Gamma$. If $\Gamma \vdash \{[P]\} \xrightarrow{\alpha} \Box$, then $\llcorner\alpha\lrcorner =_{\mathsf{traces}} [PP]$.*

Proof (Sketch). The proof is by induction on the structure of PP, showing that one of the following conditions holds (for any β and Γ' s.t. $\Gamma' \vdash \langle\beta,0\rangle \xrightarrow{\beta} \langle\boxdot,0\rangle$):

– $\Gamma,\Gamma' \vdash \langle P,\beta\rangle \xrightarrow{\alpha} \langle\boxdot,\beta';\beta\rangle$ and $PP = \{(p\langle\checkmark\rangle,p')|p\langle\checkmark\rangle \in \llcorner\alpha\lrcorner \wedge p' \in \llcorner\beta'\lrcorner\} \cup T$, where T is the set of all yielding traces $(q\langle ?\rangle, q'\langle\checkmark\rangle)$ s.t. q and q' have the same length and q is a prefix of a trace in $\llcorner\alpha\lrcorner$ and q' is prefix of a trace in $\llcorner\beta\lrcorner$.
– $\Gamma,\Gamma' \vdash \langle P,\beta\rangle \xrightarrow{\alpha;\alpha';\beta} \langle\boxtimes,0\rangle$ s.t. $\mathcal{A}(\alpha) \subseteq \mathcal{A}(|P|) \wedge \mathcal{A}(\alpha') \cap \mathcal{A}(|P|) = \emptyset$, and $PP = \{(p\langle !\rangle,p')|p\langle\checkmark\rangle \in \llcorner\alpha\lrcorner \wedge p' \in \llcorner\alpha'\lrcorner\} \cup T$, where T is defined as before. $\Box$

4.2 Encoding $\mathsf{Sagas_{seq}}$ into $\mathsf{cCSP_{seq}}$

Any process $P \in \mathsf{Sagas_{seq}}$ represents a set of processes $PP \in \mathsf{cCSP_{seq}}$, one for any possible environment $\Gamma \in \nabla$. Hence, the encoding is defined as follow:

$$[\![\_]\!]_\_ : \mathsf{Sagas_{seq}} \times \nabla \to \mathsf{cCSP_{seq}}$$

$$\begin{array}{rclrcl}
[\![0]\!]_\Gamma & = & SKIPP & [\![P;Q]\!]_\Gamma & = & [\![P]\!]_\Gamma;[\![Q]\!]_\Gamma \\
[\![A]\!]_{A\mapsto\boxdot,\Gamma} & = & A & [\![A]\!]_{A\mapsto\boxtimes,\Gamma} & = & THROWW \\
[\![A \div B]\!]_{A\mapsto\boxdot,B\mapsto\boxdot,\Gamma} & = & A \div B & [\![A \div B]\!]_{A\mapsto\boxtimes,\Gamma} & = & THROWW
\end{array}$$

Note that the encoding for a compensation pair is defined only when the compensation B is an activity that commits, because the fragment of cCSP we are considering does not allow *THROW* in compensation pairs. Hence, we shall account only for contexts Γ that never make a saga to terminate abnormally (by adequacy results in [12,5]).

Theorem 4.2. *Let Γ be an environment, $P \in \mathsf{Sagas_{seq}}$, and $[\![P]\!]_\Gamma = PP$. If $\Gamma \vdash \{[P]\} \xrightarrow{\alpha} \Box$, then $\llcorner\alpha\lrcorner =_{\mathsf{traces}} [PP]$.*

5 Alternative semantics for parallel compensations

In this section we formally characterise the four compensation policies mentioned in § 1.

Notation 2 *We write $\mathsf{cCSP_{par\mathit{i}}}$ and $\mathsf{Sagas_{par\mathit{i}}}$ to denote the cCSP and Sagas semantics when considering the strategy $i = 1,\ldots,4$, as enumerated in § 1.*

In all remaining sections assume the encoding functions extended as follow

$$\begin{aligned}[\![PP_1|PP_2]\!]_\sigma &= P_1|P_2, \Gamma_1 \uplus \Gamma_2 \quad s.t.\ [\![PP_i]\!]_{\sigma.i} = P_i, \Gamma_i \quad \text{for } i = 1,2\\ [\![P|Q]\!]_\Gamma &= [\![P]\!]_\Gamma | [\![Q]\!]_\Gamma\end{aligned}$$

5.1 No interruption and centralised compensation

The desired behaviour for a parallel transaction when assuming no interruption and centralised compensation can be illustrated with the following law for $\mathsf{cCSP}_{\mathsf{par1}}$:

$$[A \div A' \mid B \div B' \mid THROWW] =_{\mathsf{traces}} (A|B);(A'|B')$$

The forward flow $A|B$ is executed completely before the compensation $A'|B'$. Moreover, all activities in the forward flow are observed even though their execution could be avoided in a clever system (since the transaction will fail anyway).

Trace semantics. The trace semantics for this case is obtained by redefining the traces of compensation pairs and parallel composition. Since parallel branches do not yield to an interrupt, the definition for a compensation pair is simplified as follow:

$$A \div B \ =_{\mathsf{traces}}\ \{p \div q | p \in A \wedge q \in B\}\ =_{\mathsf{traces}}\ \{(\langle A, \checkmark\rangle, \langle B, \checkmark\rangle)\}$$

We remove from the original definition the possibility for a compensation pair to yield to an interrupt before executing the forward flow A. On the other hand, the traces for parallel composition $P|Q$ consider only the traces of P and Q that have finished either successfully or with a failure, but not those yielding to an interruption, i.e.,

$$p\langle\omega\rangle || q\langle\omega'\rangle = \{r\langle\omega \& \omega'\rangle | r \in (p|||q) \wedge \omega, \omega' \in \{\checkmark, !\}\}$$

Since we do not allow interruption, *YIELDD* has no effects and, hence, we let $YIELDD =_{\mathsf{traces}} SKIPP =_{\mathsf{traces}} \{(\langle\checkmark\rangle, \langle\checkmark\rangle)\}$. Moreover, $THROWW =_{\mathsf{traces}} \{(\langle !\rangle, \langle\checkmark\rangle)\}$.

SOS semantics. The SOS semantics for the case of no interruption and centralised compensation is in Figure 4. The main differences with the rules in Figure 2(a) is that the activation of the compensation procedure is left to the rule (SAGA) and not to (F-ACT). Note also that the result for $P|Q$ is given by & (not by $\wedge$ as in § 2.2), which is analogous to the trace semantics.

Correspondence. The following results assure the correspondence between the two semantics.

Theorem 5.1. *Let* $PP \in \mathsf{cCSP}_{\mathsf{par1}}$ *and* $[\![PP]\!] = P, \Gamma$*, with* $P \in \mathsf{Sagas}_{\mathsf{par1}}$*. If* $\Gamma \vdash \{[P]\} \xrightarrow{\alpha} \Box$*, then* $\llcorner\alpha\lrcorner =_{\mathsf{traces}} [PP]$.

Theorem 5.2. *Let* Γ *be an environment,* $P \in \mathsf{Sagas}_{\mathsf{par1}}$*, and* $[\![P]\!]_\Gamma = PP$*, with* $PP \in \mathsf{cCSP}_{\mathsf{par1}}$*. If* $\Gamma \vdash \{[P]\} \xrightarrow{\alpha} \Box$*, then* $\llcorner\alpha\lrcorner =_{\mathsf{traces}} [PP]$.

(ZERO)
$$\Gamma \vdash \langle 0, \beta \rangle \xrightarrow{0} \langle \boxdot, \beta \rangle$$

(F-ACT)
$$A \mapsto \boxtimes, \Gamma \vdash \langle A \div B, \beta \rangle \xrightarrow{0} \langle \boxtimes, \beta \rangle$$

(A-STEP)
$$\frac{\Gamma \vdash \langle P, \beta \rangle \xrightarrow{\alpha} \langle \boxtimes, \beta' \rangle}{\Gamma \vdash \langle P; Q, \beta \rangle \xrightarrow{\alpha} \langle \boxtimes, \beta' \rangle}$$

(CMT-SAGA)
$$\frac{\Gamma \vdash \langle P, 0 \rangle \xrightarrow{\alpha} \langle \boxdot, \beta \rangle}{\Gamma \vdash \{[P]\} \xrightarrow{\alpha} \boxdot}$$

(FAILED-SAGA)
$$\frac{\Gamma \vdash \langle P, 0 \rangle \xrightarrow{\alpha} \langle \boxtimes, \beta \rangle \quad \Gamma \vdash \langle \beta, 0 \rangle \xrightarrow{\beta'} \langle \boxtimes, 0 \rangle}{\Gamma \vdash \{[P]\} \xrightarrow{\alpha;\beta'} \boxast}$$

(S-ACT)
$$A \mapsto \boxdot, \Gamma \vdash \langle A \div B, \beta \rangle \xrightarrow{A} \langle \boxdot, B; \beta \rangle$$

(S-STEP)
$$\frac{\Gamma \vdash \langle P, \beta \rangle \xrightarrow{\alpha} \langle \boxdot, \beta'' \rangle \quad \Gamma \vdash \langle Q, \beta'' \rangle \xrightarrow{\alpha'} \langle \Box, \beta' \rangle}{\Gamma \vdash \langle P; Q, \beta \rangle \xrightarrow{\alpha;\alpha'} \langle \Box, \beta' \rangle}$$

(PAR)
$$\frac{\Gamma \vdash \langle P, 0 \rangle \xrightarrow{\alpha_1} \langle \Box_1, \beta_1 \rangle \quad \Gamma \vdash \langle Q, 0 \rangle \xrightarrow{\alpha_2} \langle \Box_2, \beta_2 \rangle}{\Gamma \vdash \langle P|Q, \beta \rangle \xrightarrow{\alpha_1|\alpha_2} \langle \Box_1 \& \Box_2, \beta_1 | \beta_2; \beta \rangle}$$
where $\boxdot \& \boxdot = \boxdot, \boxdot \& \boxtimes = \boxtimes, \textit{and}\ \boxtimes \& \boxtimes = \boxtimes$

(ABORTED-SAGA)
$$\frac{\Gamma \vdash \langle P, 0 \rangle \xrightarrow{\alpha} \langle \boxtimes, \beta \rangle \quad \Gamma \vdash \langle \beta, 0 \rangle \xrightarrow{\beta} \langle \boxdot, 0 \rangle}{\Gamma \vdash \{[P]\} \xrightarrow{\alpha;\beta} \boxtimes}$$

Fig. 4. SOS for no interruption and centralised compensation.

5.2 No interruption and distributed compensation

As aforementioned, a distributed procedure for compensating parallel branches may allow the execution of activities of the backward flow even when parts of the forward flow are still in execution. As an example, the following law should hold in $\mathsf{cCSP}_{\mathsf{par2}}$ (i.e., by assuming no interruption and distributed compensation):

$$[A \div A' \mid B \div B' \mid \mathit{THROWW}] =_{\mathsf{traces}} A; A' | B; B'$$

Note that the forward flows A and B are executed entirely, but parallel branches are independently compensated for. For example, A' can be executed even before B.

Trace semantics. As for the previous case, the traces of a compensation pair do not have yielding behaviours, and *SKIPP*, *YIELDD* and *THROWW* are defined analogously. Instead, the parallel composition of traces is as follow

$$\begin{aligned}(p\langle \checkmark \rangle, p') || (q\langle \checkmark \rangle, q') &= \{(r\langle \checkmark \rangle, r'\langle \checkmark \rangle) | r \in (p|||q) \wedge r'\langle \checkmark \rangle \in (p'||q')\} \\ &\quad \cup \{(r\langle ? \rangle, \langle \checkmark \rangle) | r\langle \checkmark \rangle \in (pp'||qq')\} \\ (p\langle \omega \rangle, p') || (q\langle \omega' \rangle, q') &= \{(r\langle \omega \& \omega' \rangle, \langle \checkmark \rangle) | r\langle \checkmark \rangle \in (pp'||qq')\} \quad \text{if} \quad \omega \& \omega' \in \{!, ?\}\end{aligned}$$

Note that the parallel composition of two successful traces contains all the interleavings of the forward flows compensated with the interleavings of the original compensations, and a set of yielding traces. Yielding traces stand for the behaviours of processes

$PP|QQ$ in case they are composed in parallel with a process that fails, for instance $PP|QQ|THROWW$. Finally, the parallel composition when at least one trace ends with ? or ! is defined as the interleavings of the original compensated flows.

SOS semantics. This case corresponds to the naïve semantics described in § 2.2.

Correspondence. Different from previous cases, for a saga $\{[P]\}$ and an environment Γ there can be several α_i s.t. $\Gamma \vdash \{[P]\} \xrightarrow{\alpha_i} \square$. For instance, consider $P = A_1 \div B_1|A_2 \div B_2|F_1$ and $\Gamma = A_1 \mapsto \boxdot, A_2 \mapsto \boxdot, B_1 \mapsto \boxdot, B_2 \mapsto \boxdot, F_1 \mapsto \boxtimes$. Then, it is easy to check that $\Gamma \vdash \{[P]\} \xrightarrow{\alpha_i} \boxtimes$ for $\alpha_1 = A_1;B_1|A_2;B_2$ and $\alpha_2 = (A_1|A_2);(B_1|B_2)$, depending on whether P is considered either as $(A_1 \div B_1|A_2 \div B_2)|F_1$ or as $A_1 \div B_1|(A_2 \div B_2|F_1)$. Nevertheless, note that the result $\square$ is always unique by results in [5].

We note $\Gamma \vdash \{[P]\} \xrightarrow{\kappa} \square$, where $\kappa = \{\alpha_i | \Gamma \vdash \{[P]\} \xrightarrow{\alpha_i} \square\}$ and let

$$\llcorner \kappa \lrcorner =_{\mathsf{traces}} \cup_{\alpha_i \in \kappa} \llcorner \alpha_i \lrcorner$$

Theorem 5.3. *Let $PP \in \mathsf{cCSP}_{\mathsf{par2}}$ and $[\![PP]\!] = P, \Gamma$, with $P \in \mathsf{Sagas}_{\mathsf{par2}}$. If $\Gamma \vdash \{[P]\} \xrightarrow{\kappa} \square$, then $\llcorner \kappa \lrcorner =_{\mathsf{traces}} [PP]$.*

Theorem 5.4. *Let Γ be an environment, $P \in \mathsf{Sagas}_{\mathsf{par2}}$, and $[\![P]\!]_\Gamma = PP$, with $PP \in \mathsf{cCSP}_{\mathsf{par2}}$. If $\Gamma \vdash \{[P]\} \xrightarrow{\kappa} \square$, then $\llcorner \kappa \lrcorner =_{\mathsf{traces}} [PP]$.*

5.3 Interruption and centralised compensation

When considering interruption, the main idea is to avoid the execution of steps by stopping the forward flow as soon as an activity fails. Nevertheless, in a distributed setting we cannot expect processes to be stopped immediately. The law we would like to prove when using this strategy is the following.

$$[A \div A' \mid B \div B' \mid THROWW] =_{\mathsf{traces}} SKIP \cup (A;A') \cup (B;B') \cup (A|B);(A'|B')$$

The first three terms show that parallel branches can be aborted even before starting their execution when one process fails (i.e., *THROWW*). Instead, the last term of the right hand side means that compensation is centralised.

Trace semantics. The case of interruption and centralised compensation corresponds to the original proposal of the trace semantics summarised in § 2.1.

SOS semantics. The SOS semantics for this strategy is obtained by adding forced termination to the rules corresponding to the policy of no interruption and centralised compensation (shown in Figure 4). In order to achieve that, rules in Figure 4 are extended with the additional rule

$$(\textsc{forced-abt}) \qquad \Gamma \vdash \langle P, \beta \rangle \xrightarrow{0} \langle \overline{\boxdot}, \beta \rangle$$

which introduces forced termination. In this case, it is enough to consider one result, which we note $\overline{\boxdot}$. Moreover we extend the definition of & used in rule (PAR), as follow $\overline{\boxdot} \& \overline{\boxdot} = \overline{\boxdot}$, $\overline{\boxdot} \& \boxdot = \overline{\boxdot}$, $\overline{\boxdot} \& \boxtimes = \boxtimes$. (Note that this definition makes the operator & isomorphic in both the trace and the SOS semantics).

Correspondence. As for the previous cases, we have the following correspondence results for $\mathsf{cCSP}_{\mathsf{par3}}$ and $\mathsf{Sagas}_{\mathsf{par3}}$

Theorem 5.5. *Let* $PP \in \mathsf{cCSP}_{\mathsf{par3}}$ *and* $[\![PP]\!] = P, \Gamma$, *with* $P \in \mathsf{Sagas}_{\mathsf{par3}}$. *If* $\Gamma \vdash \{[P]\} \xrightarrow{\kappa} \square$, *then* $\llcorner\kappa\lrcorner =_{\mathsf{traces}} [PP]$.

Theorem 5.6. *Let* Γ *be an environment,* $P \in \mathsf{Sagas}_{\mathsf{par3}}$, *and* $[\![P]\!]_\Gamma = PP$, *with* $PP \in \mathsf{cCSP}_{\mathsf{par3}}$. *If* $\Gamma \vdash \{[P]\} \xrightarrow{\kappa} \square$, *then* $\llcorner\kappa\lrcorner =_{\mathsf{traces}} [PP]$.

5.4 Interruption and distributed compensation

This policy can be illustrated by the following equality in $\mathsf{cCSP}_{\mathsf{par4}}$:

$$[A \div A'|B \div B'|THROWW] =_{\mathsf{traces}} SKIP \cup (A;A') \cup (B;B') \cup (A;A')|(B;B')$$

The difference with the policy reported in § 5.3 relies in the last term of the summation in the right hand side of the equality. In fact, the last term of the above equality shows that the compensation is handled in a distributed way. The remaining terms stand for the cases in which the forward flow is stopped before completion.

Trace semantics. The trace semantics for this policy is obtained from the original one (see Figure 1) by changing the definition for the parallel composition of traces as in § 5.2, i.e.,

$$\begin{aligned}(p\langle\checkmark\rangle, p')||(q\langle\checkmark\rangle, q') &= \{(r\langle\checkmark\rangle, r'\langle\checkmark\rangle)|r \in (p|||q) \wedge r'\langle\checkmark\rangle \in (p'||q')\}\\ &\quad \cup\{(r\langle ?\rangle, \langle\checkmark\rangle)|r\langle\checkmark\rangle \in (pp'||qq')\}\\ (p\langle\omega\rangle, p')||(q\langle\omega'\rangle, q') &= \{(r\langle\omega\&\omega'\rangle, \langle\checkmark\rangle)|r\langle\checkmark\rangle \in (pp'||qq')\} \quad \text{if} \quad \omega\&\omega' \in \{!, ?\}\end{aligned}$$

SOS semantics. This strategy corresponds to the original revised semantics of parallel Sagas (Figure 2(b)).

Correspondence. The following results state the correspondence between the trace and SOS semantics for this policy.

Theorem 5.7. *Let* $PP \in \mathsf{cCSP}_{\mathsf{par4}}$ *and* $[\![PP]\!] = P, \Gamma$, *with* $P \in \mathsf{Sagas}_{\mathsf{par4}}$. *If* $\Gamma \vdash \{[P]\} \xrightarrow{\kappa} \square$, *then* $\llcorner\kappa\lrcorner =_{\mathsf{traces}} [PP]$.

Theorem 5.8. *Let* Γ *be an environment,* $P \in \mathsf{Sagas}_{\mathsf{par3}}$, *and* $[\![P]\!]_\Gamma = PP$, *with* $PP \in \mathsf{cCSP}_{\mathsf{par3}}$. *If* $\Gamma \vdash \{[P]\} \xrightarrow{\kappa} \square$, *then* $\llcorner\kappa\lrcorner =_{\mathsf{traces}} [PP]$.

6 Relation of the proposed semantics

The four strategies presented in § 5 correspond to alternative implementations for the compensation mechanism. In this section, we analyse the relation among such policies. The following result states the relation among the traces of a transaction $[PP]$ accordingly to the four possible semantics for compensating parallel processes.

Theorem 6.1. *Let* $[PP]$ *be a parallel* cCSP *process, and let* $[PP]_{\mathsf{cCSP}_{\mathsf{par}i}}$ *denote the traces of* $[PP]$ *when considering the strategy* $i = 1,\ldots,4$. *Then, the four trace semantics satisfy the following diagram*

$$\begin{array}{ccccc} & [PP]_{\mathsf{cCSP}_{\mathsf{par1}}} & \xrightarrow{\subseteq} & [PP]_{\mathsf{cCSP}_{\mathsf{par2}}} & \text{Naïve Sagas} \\ & \big\downarrow{\scriptstyle\subseteq} & & \big\downarrow{\scriptstyle\subseteq} & \\ \text{Original cCSP} & [PP]_{\mathsf{cCSP}_{\mathsf{par3}}} & \xrightarrow{\subseteq} & [PP]_{\mathsf{cCSP}_{\mathsf{par4}}} & \text{Revised Sagas} \end{array}$$

Proof (Sketch). The proof for any inclusion follows by showing (by induction on the structure of PP) that any trace in $PP_{\mathsf{cCSP}_{\mathsf{par}i}}$ corresponds with a trace in $PP_{\mathsf{cCSP}_{\mathsf{par}j}}$. For instance, that

– $(p\langle\checkmark\rangle, p') \in PP_{\mathsf{cCSP}_{\mathsf{par1}}} \quad\Rightarrow\quad (p\langle\checkmark\rangle, p') \in PP_{\mathsf{cCSP}_{\mathsf{par2}}}$
– $(p\langle !\rangle, p'p'') \in PP_{\mathsf{cCSP}_{\mathsf{par1}}} \quad\Rightarrow\quad (pp'\langle !\rangle, p'') \in PP_{\mathsf{cCSP}_{\mathsf{par2}}}$. □

Note that the above diagram does not include $[PP]_{\mathsf{cCSP}_{\mathsf{par2}}} \subseteq [PP]_{\mathsf{cCSP}_{\mathsf{par3}}}$ nor $[PP]_{\mathsf{cCSP}_{\mathsf{par3}}} \subseteq [PP]_{\mathsf{cCSP}_{\mathsf{par2}}}$. In fact, it is easy to check that there are processes $[PP]$ for which none of them holds. For instance, consider $P = [A \div A'; B \div B' | C \div C' | THROWW]$. Note that $p = \langle A, B, B', A', C, C', \checkmark\rangle \in P_{\mathsf{cCSP}_{\mathsf{par2}}}$, but $p \notin P_{\mathsf{cCSP}_{\mathsf{par3}}}$, since compensations A' and B' take place before C. On the other hand, note that $q = \langle\checkmark\rangle \in P_{\mathsf{cCSP}_{\mathsf{par3}}}$, but $q \notin P_{\mathsf{cCSP}_{\mathsf{par2}}}$ since the forward flow is required to execute until termination.

The above result makes incomparable the semantics of original cCSP and naïve Sagas. On the other hand, it shows that the revised version of Sagas allows more traces than cCSP, and hence it is less restrictive on which are the acceptable executions of processes. Nevertheless, the distributed compensation mechanism of $\mathsf{cCSP}_{\mathsf{par4}}$ includes a "guessing mechanisms" that allows branches on the forward flow to compensate even before an activity aborts. For instance, $[A \div A'; THROWW | B \div B']$ has the trace $p = \langle B; B'; A; A'\rangle$. Since A is executed after B, p stands for an execution in which the branch $B \div B'$ starts its compensation before $THROWW$ is reached. Although this is an acceptable and valid execution of the above transaction, it is hard to imagine a plausible implementation of such a mechanism, which suggests that a more realistic policy relies in between cCSP and revised Sagas.

7 Final Remarks

We have compared two recent formal approaches to the modelling of compensable flow composition, that have been proposed independently in [5,7]. For the sequential case we have shown that the two frameworks essentially coincide by providing fully abstract encodings. For the parallel case we have observed that the two approaches followed different compensation policies, and that up to four different choices were possible for activating compensations in parallel branches. We have shown that each alternative can be formalised by adjusting the semantics of the two calculi. Finally we have related all different policies by showing that they form a partial order of trace models.

Our more ambitious research programme is to extend the comparison to deal with more advanced features, like nesting, joint transactions, message passing and action refinement. To this end, the research presented here has been valuable in deepening our understanding of the phenomenon of a compensable parallel transaction and the range of available design options.

Acknowledgements Research supported by the project HPRN-CT-2002-00275 SEGRAVIS. We thank Microsoft Research (Cambridge) for hosting two workshops at which the ideas behind the paper were initiated and discussed. We also thank the anonymous referees for their helpful comments.

References

1. L. Bocchi, C. Laneve, and G. Zavattaro. A calculus for long-running transactions. In E. Najm, U. Nestmann, and P. Stevens, editors, *Proceedings of FMOODS 2003, 6th IFIP International Conference on Formal Methods for Open-Object Based Distributed Systems*, volume 2884 of *Lect. Notes in Comput. Sci.*, pages 124–138. Springer Verlag, 2003.
2. BPEL Specification (v.1.1). `http://www.ibm.com/developerworks/library/ws-bpel`.
3. Business Process Modeling Language (BPML). `http://www.bpmi.org/BPML.htm`.
4. R. Bruni, H. Melgratti, and U. Montanari. Nested commits for mobile calculi: extending Join. In J.-J. Lévy, E. Mayr, and J. Mitchell, editors, *Proceedings of the 3rd IFIP-TCS 2004, 3rd IFIP Intl. Conference on Theoretical Computer Science*, pages 569–582. Kluwer Academic Publishers, 2004.
5. R. Bruni, H. Melgratti, and U. Montanari. Theoretical foundations for compensations in flow composition languages. In *Proceedings of POPL 2005, 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 209–220. ACM Press, 2005.
6. M. Butler and C. Ferreira. An operational semantics for StAC, a language for modelling long-running business transactions. In R. De Nicola, G. Ferrari, and G. Meredith, editors, *Proceedings of Coordination 2004*, volume 2949 of *Lect. Notes in Comput. Sci.*, pages 87–104. Springer Verlag, 2004.
7. M. Butler, T. Hoare, and C. Ferreira. A trace semantics for long-running transactions. In A. Abdallah, C.B. Jones, and J. Sanders, editors, *Proceedings of 25 Years of CSP*, volume 3525 of *Lect. Notes in Comput. Sci.*, pages 133–150. Springer Verlag, 2005.
8. H. Garcia-Molina and K. Salem. Sagas. In U. Dayal and I.L. Traiger, editors, *Proceedings of the ACM Special Interest Group on Management of Data Annual Conference*, pages 249–259. ACM Press, 1987.
9. C. Laneve and G. Zavattaro. Foundations of web transactions. In V. Sassone, editor, *Proceedings of FoSSaCS 2005, 8th International Conference on Foundations of Software Science and Computational Structures*, volume 3441 of *Lect. Notes in Comput. Sci.*, pages 282–298. Springer Verlag, 2005.
10. F. Leymann. WSFL Specification (v.1.0). `http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf`, May 2001.
11. M. Mazzara and R. Lucchi. A framework for generic error handling in business processes. In M. Bravetti and G. Zavattaro, editors, *Proceedings of WS-FM 2004, 1st International Workshop on Web Services and Formal Methods*, 2004. To appear as ENTCS.
12. H. Melgratti. *Models and Languages for Global Computing Transaction*. PhD thesis, Computer Science Department, University of Pisa, 2005. Submitted.
13. Web Service Choreography Interface (WSCI) 1.0. `http://www.w3.org/TR/wsci`.
14. Web Services Conversation Language (WSCL) 1.0. `http://www.w3.org/TR/wscl10/`.
15. Web Service Description Language (WSDL). `http://www.w3.org/TR/wsdl`.
16. Web Services for Business Process Design (XLANG). `http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm`.