Acta Informatica 33, 153–176 (1996)

Parallel composition of assumption-commitment specifications

A unifying approach for shared variable and distributed message passing concurrency

A. Cau[1]*, P. Collette[2]**

[1] Institut für Informatik und Praktische Mathematik II, Christian-Albrechts-Universität zu Kiel, D-24105 Kiel, Germany
[2] Département d'Ingénierie Informatique, Université Catholique de Louvain, B-1348 Louvain-la-Neuve, Belgium

Received: March 14, 1994 / November 28, 1994

Abstract. We unify the parallel composition rule of assumption-commitment specifications for respectively state-based and message-based concurrent processes. Without providing language-dependent definitions, we first assume that the model of a process can be given as a set of 'sequences' (e.g., traces, state sequences). Then we assume the existence of a merging operator that captures the compositionality of that model. On this basis, we formulate a semantic parallel composition rule for assumption-commitment specifications wherein the merging operator behaves as a parameter. Then, by providing suitable language-specific definitions for the model of a process and the merging operator, we transform the semantic rule into syntactic ones, both for the state-based and message-based approaches to concurrency.

1 Introduction

In the concurrent programming community, communication between processes is usually modeled in two ways. The first one uses shared variables as a mean for communication and the other one uses distributed message passing. Both approaches are well established and have their own advantages and disadvantages.

In both cases, certain compositional methods for the development of parallel or distributed systems are based on the assumption-commitment paradigm as this approach is called within the message-based concurrency community, also referred to as the rely-guarantee paradigm within the state-based concurrency community. Examples may be found in e.g. [2, 17, 18, 21, 22, 26, 29, 31]. Intuitively, an assumption-commitment specification of an open system (a process or a process network) asserts that the commitment of a system holds provided that the system operates in an environment that respects the assumption. In such state-based and message-based compositional methods, parallel rules have been devised to compose assumption-commitment specifications of parallel processes. These composition rules are usually hard to construct because of mutual dependency: each process belongs to the environment of the

* Partially Supported by ESPRIT Project 6021 (REACT).
** Supported by National Fund for Scientific Research (Belgium).

other ones and the commitment of a process thus influences the assumptions of the other ones.

Although this problem exists whatever communication model is adopted (state-based or message-based), the corresponding assumption-commitment methods evolved to different rules for parallel composition. In the state-based approach, a typical premise of the rule for deducing a specification of $P_1 \| P_2$ from the specifications of P_1 and P_2 is of the form $A \vee C_1 \Rightarrow A_2$ [17, 27, 29, 30], where A is the assumption of $P_1 \| P_2$, C_1 the commitment of P_1, and A_2 the assumption of P_2, i.e. the most prominent operator is *disjunction*. In the message-based approach, the corresponding premise is of the form $A \wedge C_1 \Rightarrow A_2$ [18, 21, 31, 32], where A, C_1 and A_2 are as before, i.e. the most prominent operator is *conjunction*. Essentially, disjunction in the state-based case comes from the use of predicates on state transitions: a transition of $P_1 \| P_2$ is either a transition of P_1 *or* a transition of P_2. Conjunction in the message-based case comes from the use of trace predicates: a joint communication of $P_1 \| P_2$ is both a communication of P_1 *and* a communication of P_2.

The purpose of this paper is to establish more explicit relations between two specific parallel rules for assumption-commitment specifications. To achieve this goal, we show that the parallel rules for state-based and message-based approaches are particular *instances* of the *same* semantic rule. This semantic rule, which is independent of the communication mechanism, takes its origin in [2, 4] and has been further investigated in [12]; however, the version proposed here is slightly different and more similar to the one in [3]. It is also more abstract in the sense that parallel composition is represented by a semantic merging operator $\otimes$ that can be instantiated in several ways. Actually, this operator reflects the compositionality of the computational model. The soundness proof of the semantic rule can be carried out without a concrete (language-dependent) definition for this operator.

Section 2 introduces the semantic basis of our approach, the semantic assumption-commitment specifications and the semantic parallel composition rule. Furthermore a check-list is given for deriving the syntactic rules. Section 3 gives the syntax and the operational readiness semantics for both state-based processes and message-based processes in order to be able to detect deadlocks and their absence. Section 4 shows that the parallel composition rule for state-based assumption-commitment specifications is an instance of the semantic parallel composition rule. The same is done in Sect. 5 for the parallel composition rule for message-based assumption-commitment specifications. Related work is further discussed in Sect. 6.

2 Semantic analysis

The proposed semantic model is quite general; only a few constraints are imposed. Indeed, we first introduce computations as labeled sequences but, intentionally, do not define the elements of these sequences, nor the labels. This abstraction makes the model independent of the kind of communication behavior. It can be instantiated for, e.g., message-based concurrency (sequences of messages) or state-based concurrency (sequences of states). Then, based upon sets of computations, semantic assumption-commitment specifications are introduced. Finally, the semantic rule for parallel composition is given.

Definition 1 (Computation) A computation is a non-empty sequence

$$\sigma = \chi_0 \xrightarrow{l_1} \chi_1 \xrightarrow{l_2} \chi_2 \xrightarrow{l_3} \cdots$$

where the χ_i's are the configurations and l_i's the labels of the computation. These configurations and labels will be made more concrete in the message-based and state-based cases. We use k to range over indexes:

- $l_k.\sigma$ denotes the k^{th} label of σ $(k > 0)$,
- $\chi_k.\sigma$ denotes the k^{th} configuration of σ $(k \geq 0)$,
- $\sigma|_k$ denotes the prefix of σ ending with $\chi_k.\sigma$,
- $|\sigma|$ denotes the length of σ, i.e. the index of the last configuration in σ if σ is finite, and ∞, otherwise.

We suppose that k is finite and does not exceed $|\sigma|$.

In this semantic analysis, we consider specifications as *sets* of computations. *Safety* sets, that play an important role in the rest of this paper, are closed under finite prefixes and under limits. Closure under limits means: whenever all finite prefixes of a computation are in a set, the complete computation is in the set. Conversely, closure under finite prefixes means: whenever a computation is in a set, all its finite prefixes are in the set.

Definition 2 (Closed Sets) Let S be a set of computations. S is *closed under finite prefixes* iff

$$\forall \sigma : \sigma \in S \Rightarrow (\forall k : \sigma|_k \in S)$$

S is *closed under limits* iff

$$\forall \sigma : \sigma \in S \Leftarrow (\forall k : \sigma|_k \in S)$$

S is a *safety set* iff S is closed under finite prefixes and S is closed under limits.

The sets **SA**, **SC** and **OC** for assumption-commitment specifications of a process P are introduced to specify the interaction between the process and its environment.

- **SA** is a safety set that characterizes those computations that satisfy the assumptions on the environment,
- **SC** is a safety set that characterizes those computations that satisfy safety commitments of the process, hence assumptions that can be made by other processes.
- **OC** is a set that characterizes those computations that satisfy other commitments of the process, especially liveness commitments.

We then use **C** to denote the pair (**SC**, **OC**). This notation indicates that the safety commitments, represented by **SC** at the semantic level, are clearly identified from other commitments. The rules in [2, 3] focus on the particular case where **SC** is the smallest safety set greater than **SC** $\cap$ **OC** but keeping this generality allows a direct mapping into the specifications of [17, 21] that we want to consider in this paper.

Example 1 We later consider a tuple $(pre, rely, guar, post)$ of predicates. Then, **SA**, **SC**, and **OC** are the sets of computations allowed by $(pre, rely)$, $(guar)$, and $(post)$ respectively.

Given these sets, assumption-commitment specifications can be interpreted in several ways (see [2, 3, 13] for a detailed discussion). The simplest interpretation is given by **SA** $\longrightarrow$ **C**: if the (complete) computation satisfies the assumptions, it must satisfy all the commitments. When only safety commitments are considered, a second interpretation is given by **SA** $\rhd$ **SC**: if the computation satisfies the assumptions up to step k, then it must satisfy the commitments up to step k. A third interpretation is given by **SA** @ **SC**:

the commitments hold initially and, if the computation satisfies the assumptions up to step $k-1$, then it must satisfy the commitments up to step k. The last considered interpretation is based on a separate treatment for the safety commitments; it is given by $\mathbf{SA}@\_{\to}\mathbf{C}$ which is a combination of $\mathbf{SA} \to \mathbf{C}$ and $\mathbf{SA} @ \mathbf{SC}$.

Definition 3 (Semantic specification) Let σ be a computation:

$$\begin{array}{lcl}
\sigma \in (\mathbf{SA} \to \mathbf{C}) & \overset{\text{def}}{=} & \sigma \in \mathbf{SA} \Rightarrow \sigma \in \mathbf{SC} \wedge \sigma \in \mathbf{OC} \\
\sigma \in (\mathbf{SA} \rhd \mathbf{SC}) & \overset{\text{def}}{=} & \forall k : \sigma|_k \in \mathbf{SA} \Rightarrow \sigma|_k \in \mathbf{SC} \\
\sigma \in (\mathbf{SA} @ \mathbf{SC}) & \overset{\text{def}}{=} & (\sigma|_0 \in \mathbf{SC}) \wedge (\forall k > 0 : \sigma|_{k-1} \in \mathbf{SA} \Rightarrow \sigma|_k \in \mathbf{SC}) \\
\sigma \in (\mathbf{SA}@\_{\to}\mathbf{C}) & \overset{\text{def}}{=} & \sigma \in (\mathbf{SA} \to \mathbf{C}) \wedge \sigma \in (\mathbf{SA} @ \mathbf{SC})
\end{array}$$

This paper aims at the unification of specific rules for assumption-commitment specifications of message-based and state-based processes. The former [18, 21, 31] are interpreted by $\mathbf{SA}@\_{\to}\mathbf{C}$ whereas the latter [17, 27, 30] are interpreted by $\mathbf{SA} \to \mathbf{C}$. Fortunately, as proved in Sect. 4, the latter can be equivalently interpreted by $\mathbf{SA}@\_{\to}\mathbf{C}$ which is thus the appropriate candidate for formulating the semantic rule.

Then, we denote by $\mathcal{M}(P) \subseteq \mathbf{SA}@\_{\to}\mathbf{C}$ that a process P is correct w.r.t. specification $\mathbf{SA}@\_{\to}\mathbf{C}$; $\mathcal{M}(P)$ (the model of P) is the set of computations of P; specific definitions for state-based and message-based processes are given later.

We also need to represent parallel composition at the semantic level. Keeping as much generality as possible, we consider that the computations of $P_1 \| P_2$ are given by $\mathcal{M}(P_1)\, {}_{\beta_1}\!\bigotimes_{\beta_2} \mathcal{M}(P_2)$ where β_1 and β_2 are the bases of P_1 and P_2 respectively, and the semantic operator $\bigotimes$ on *sets* of computations is defined in terms of the more basic operator $\otimes$ on computations.

Definition 4 (Conjoining) Let S_i $(i = 1, 2)$ be sets of computations:

$$S_1 \, {}_{\beta_1}\!\bigotimes_{\beta_2} S_2 \overset{\text{def}}{=} \{\sigma \mid \exists \sigma_1 \in S_1, \sigma_2 \in S_2 : {}_{\beta_1}\!\otimes_{\beta_2} (\sigma_1, \sigma_2, \sigma)\}$$

Specific definitions for $\otimes$ are deliberately omitted at this stage. Usually, $\otimes$ takes computations σ_1 of P_1, σ_2 of P_2 and merges them into a computation σ of $P_1 \| P_2$. In fact, the only requirements imposed upon this $\otimes$ operator are

(1) $${}_{\beta_1}\!\otimes_{\beta_2} (\sigma_1, \sigma_2, \sigma) \Rightarrow |\sigma_1| = |\sigma_2| = |\sigma|$$

(2) $${}_{\beta_1}\!\otimes_{\beta_2} (\sigma_1, \sigma_2, \sigma) \Rightarrow \forall k : {}_{\beta_1}\!\otimes_{\beta_2} (\sigma_1|_k, \sigma_2|_k, \sigma|_k)$$

The requirement for computations of parallel processes to be of equal length seems to be a strong requirement. For instance, the traces of message-based parallel processes are usually defined from their *projections* onto the channels of their subprocesses. However, traces of equal length can be obtained by shuffling arbitrary communications over other channels. Indeed, let $T(P)$ be the set of traces of P. Then $\mathcal{M}(P)$ can be defined as $\{t \mid t \downarrow ch(P) \in T(P)\}$ where t may mention any channel and $t \downarrow ch(P)$ is the projection onto the channels of P.

The parallel rule aims at proving the correctness of $P_1 \| P_2$ from the correctness of P_1, the correctness of P_2 and relations between the corresponding assumption-commitment specifications. Intuitively, the premises of the subsequent semantic rule (3) can be interpreted as follows:

(i) The assumptions on the environment of P_1 (resp. P_2) follow from the assumptions on the overall environment and the commitments of P_2 (resp. P_1). Indeed, P_2 (resp. P_1) is a part of the environment of P_1 (resp. P_2).

(*ii*) The safety commitments of $P_1 \| P_2$ follow from the safety commitments of P_1 and P_2

(*iii*) Under the assumptions on the overall environment, the other commitments of $P_1 \| P_2$ follow from the other commitments of P_1 and P_2.

Theorem 1 *Let* $[{}_{\beta_1}\!\otimes_{\beta_2} : P(\sigma_1, \sigma_2, \sigma)]$ *indicate that P is universally quantified over all the computations* $\sigma_1, \sigma_2, \sigma$ *such that* ${}_{\beta_1}\!\otimes_{\beta_2}(\sigma_1, \sigma_2, \sigma)$ *holds. Then, provided that* $\otimes$ *satisfies the requirements* (1) *and* (2) *above, the following rule is sound:*

$$(3)\quad \frac{\begin{array}{ll} \mathcal{M}(P_1) \subseteq (\mathbf{SA}_1 @_{\rightarrow} \mathbf{C}_1) & \\ \mathcal{M}(P_2) \subseteq (\mathbf{SA}_2 @_{\rightarrow} \mathbf{C}_2) & \\ (i) & [{}_{\beta_1}\!\otimes_{\beta_2} : \sigma \in \mathbf{SA} \wedge \sigma_1 \in \mathbf{SC}_1 \wedge \sigma_2 \in \mathbf{SC}_2 \Rightarrow \sigma_1 \in \mathbf{SA}_1 \wedge \sigma_2 \in \mathbf{SA}_2] \\ (ii) & [{}_{\beta_1}\!\otimes_{\beta_2} : \sigma_1 \in \mathbf{SC}_1 \wedge \sigma_2 \in \mathbf{SC}_2 \Rightarrow \sigma \in \mathbf{SC}] \\ (iii) & [{}_{\beta_1}\!\otimes_{\beta_2} : \sigma \in \mathbf{SA} \wedge \sigma_1 \in \mathbf{OC}_1 \wedge \sigma_2 \in \mathbf{OC}_2 \Rightarrow \sigma \in \mathbf{OC}] \end{array}}{\mathcal{M}(P_1)\, {}_{\beta_1}\!\bigotimes_{\beta_2} \mathcal{M}(P_2) \subseteq (\mathbf{SA} @_{\rightarrow} \mathbf{C})}$$

Although carried out in another framework, the proof of Theorem 1 (see appendix) is similar to other proofs in [2, 3, 4, 12]. A comparison with the rule of [3], also based on the interpretation $\mathbf{SA} @_{\rightarrow} \mathbf{C}$, will be given in Sect. 6.

The syntactic parallel rules for state-based and message-based concurrency are of the following form:

$$\frac{\begin{array}{l} P_1 \ \mathbf{sat}\ \mathsf{spec}_1 \\ P_2 \ \mathbf{sat}\ \mathsf{spec}_2 \\ \text{syntactic\_premises}(\mathsf{spec}_1, \mathsf{spec}_2, \mathsf{spec}) \end{array}}{P_1 \| P_2 \ \mathbf{sat}\ \mathsf{spec}}$$

Therefore, to show that these are *instances* of the semantic rule (3), one has to:

1. Define computations, i.e. define what their configurations and labels are, define models $\mathcal{M}(P)$ for P, and define the operator $\otimes$.
2. Check the compositionality of the model w.r.t. the parallel composition operator, i.e. $\mathcal{M}(P_1 \| P_2) = \mathcal{M}(P_1)\, {}_{\beta_1}\!\bigotimes_{\beta_2} \mathcal{M}(P_2)$, and check the requirements (1) and (2) on $\otimes$.
3. Give definitions of **SA**, **SC**, **OC** from spec and check $\models P$ **sat** spec $\equiv$ $\mathcal{M}(P) \subseteq (\mathbf{SA} @_{\rightarrow} \mathbf{C})$.
4. Prove that the semantic premises (*i*)-(*iii*) follow from the syntactic premises.

3 Syntax and semantics of processes

In this section, we give the syntax and the operational readiness semantics for both state-based and message-based processes in order to be able to detect deadlocks and their absence. This operational readiness semantics allows us to define computations, the model $\mathcal{M}(P)$ of a process and the $\otimes$ operator. Clearly, the amount of information recorded in a computation may vary from one definition to the other but there must be enough information for specifications to be given a semantics in terms of allowed computations. Since our concern is not to discuss the semantics of processes, we choose to keep the construction of a computation as simple as possible and thus record more information than strictly necessary.

A major characteristic of the proposed model is its compositionality. To achieve this, we adopt Aczel's view of parallel composition [5, 23] and incorporate environment steps in computations.

3.1 Syntax

Definition 5 (Basis) Let PV be the set of process variables and $Chan$ be the set of channel names. The basis of a process is a tuple (I, O, V, X) that consists of the sets of respectively input channels $I \subseteq Chan$, output channels $O \subseteq Chan$, *shared* variables $V \subseteq PV$, and *local* variables $X \subseteq PV$ of the process ($V \cap X = \emptyset$).

The sets I and O are not necessarily disjoint; $I \cap O$ is the set of internal channels of a process. In state-based concurrency, the sets I and O are empty. In message-based concurrency, the set V is empty. By convention, β, β_1, and β_2 denote the bases (I, O, V, X), (I_1, O_1, V_1, X_1), and (I_2, O_2, V_2, X_2). When parallel composition is considered, we further assume:

$$\begin{array}{rclcrcl} I_1 \cap I_2 &=& \emptyset, & & I &=& I_1 \cup I_2, \\ O_1 \cap O_2 &=& \emptyset, & & O &=& O_1 \cup O_2, \\ (V_1 \cup X_1) \cap X_2 &=& \emptyset, & & V &=& V_1 \cup V_2, \\ (V_2 \cup X_2) \cap X_1 &=& \emptyset, & & X &=& X_1 \cup X_2. \end{array}$$

Definition 6 (Process) Let v, x, C, D denote a variable in $V \cup X$, a variable in X, a channel in I, and a channel in O respectively. Let e be an expression and b a boolean condition. Then, the syntax of a process (state-based and message-based) of basis β is given by:

$$\begin{array}{llcl} \text{Program} & S & ::= & v := e \mid C?x \mid D!e \mid S_1 ; S_2 \mid \textbf{wait}\ b \\ & & & \mid \textbf{while}\ b\ \textbf{do}\ S_1\ \textbf{od} \mid \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi} \\ \text{Process} & P & ::= & S \mid P_1 \| P_2 \end{array}$$

where P_1 and P_2 are processes of bases β_1 and β_2 respectively.

In state-based concurrency, the statements $C?x$ and $D!e$ are ignored; then processes synchronize via **wait** b statements; for the sake of simplicity, the traditional **await** b **do** $\langle S \rangle$ construct has been replaced with the simple blocking statement **wait** b. In message-based concurrency, **wait** b statements are ignored, i.e. the processes synchronize through messages.

3.2 Operational readiness semantics

Definition 7 (State, configuration) A *configuration* is a tuple (P, s) where P is a process and s is a state. The special symbol Ξ denotes the empty process. A *state* is a mapping s from the set of process variables PV and freeze (logical) variables FV to the set of values Val. A state s assigns to each variable $y \in PV \cup FV$ a value $s(y)$. By extension, $s(b)$ and $s(e)$ denote the boolean value of condition b at state s and the value of expression e at state s respectively. The restriction of s to a set X of variables is denoted by $s \lceil X$. A *variant* of a state s with respect to a variable y and a value μ, denoted by $(s : y \mapsto \mu)$, is given by

$$(s : y \mapsto \mu)(z) = \begin{cases} \mu & , \text{if } z \equiv y \\ s(z) & , \text{if } z \not\equiv y, \end{cases}$$

where $\equiv$ denotes syntactic equality.

Table 1. Operational semantics

$\vdash (v := e, s) \xrightarrow{\mathbf{i}}_{\beta} (\Xi, (s : v \mapsto s(e)))$

$\vdash (C?x, s) \xrightarrow{C?w}_{\beta} (\Xi, (s : x \mapsto w))$ for any $w \in Val$

$\vdash (C!e, s) \xrightarrow{C!s(e)}_{\beta} (\Xi, s')$ with $s'\lceil X = s\lceil X$

$\vdash (\mathbf{wait}\ b, s) \xrightarrow{\mathbf{i}}_{\beta} (\Xi, s)$ if $s(b)$

$\vdash (\mathbf{while}\ b\ \mathbf{do}\ S\ \mathbf{od}, s) \xrightarrow{\mathbf{i}}_{\beta} (S; \mathbf{while}\ b\ \mathbf{do}\ S\ \mathbf{od}, s)$ if $s(b)$

$\vdash (\mathbf{while}\ b\ \mathbf{do}\ S\ \mathbf{od}, s) \xrightarrow{\mathbf{i}}_{\beta} (\Xi, s)$ if not $s(b)$

$\vdash (\mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}, s) \xrightarrow{\mathbf{i}}_{\beta} (S_1, s)$ if $s(b)$

$\vdash (\mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \mathbf{fi}, s) \xrightarrow{\mathbf{i}}_{\beta} (S_2, s)$ if not $s(b)$

$$\frac{(S_1, s) \xrightarrow{l}_{\beta} (S_1', s')}{(S_1; S_2, s) \xrightarrow{l}_{\beta} (S_1'; S_2, s')} \qquad \frac{(P_1, s) \xrightarrow{\mathbf{i}}_{\beta_1} (P_1', s')}{(P_1 \| P_2, s) \xrightarrow{\mathbf{i}}_{\beta} (P_1' \| P_2, s')} \qquad \frac{(P_2, s) \xrightarrow{\mathbf{i}}_{\beta_2} (P_2', s')}{(P_1 \| P_2, s) \xrightarrow{\mathbf{i}}_{\beta} (P_1 \| P_2', s')}$$

$$\frac{(P_1, s) \xrightarrow{C.w}_{\beta_1} (P_1', s')}{(P_1 \| P_2, s) \xrightarrow{C.w}_{\beta} (P_1' \| P_2, s')} \qquad \frac{(P_2, s) \xrightarrow{C.w}_{\beta_2} (P_2', s')}{(P_1 \| P_2, s) \xrightarrow{C.w}_{\beta} (P_1 \| P_2', s')}$$

$$\frac{\begin{array}{c}(P_1, s) \xrightarrow{C?w}_{\beta_1} (P_1', s') \\ (P_2, s) \xrightarrow{C!w}_{\beta_2} (P_2', s'),\ C \in (I_1 \cap O_2)\end{array}}{(P_1 \| P_2, s) \xrightarrow{C.w}_{\beta} (P_1' \| P_2', s')} \qquad \frac{\begin{array}{c}(P_1, s) \xrightarrow{C!w}_{\beta_1} (P_1', s') \\ (P_2, s) \xrightarrow{C?w}_{\beta_2} (P_2', s'),\ C \in (O_1 \cap I_2)\end{array}}{(P_1 \| P_2, s) \xrightarrow{C.w}_{\beta} (P_1' \| P_2', s')}$$

$$\frac{(P_1, s) \xrightarrow{C!w}_{\beta_1} (P_1', s'),\ C \in (O_1 \setminus I_2),\ s'\lceil X_2 = s\lceil X_2}{(P_1 \| P_2, s) \xrightarrow{C!w}_{\beta} (P_1' \| P_2, s')} \qquad \frac{(P_1, s) \xrightarrow{C?w}_{\beta_1} (P_1', s'),\ C \in (I_1 \setminus O_2)}{(P_1 \| P_2, s) \xrightarrow{C?w}_{\beta} (P_1' \| P_2, s')}$$

$$\frac{(P_2, s) \xrightarrow{C!w}_{\beta_2} (P_2', s'),\ C \in (O_2 \setminus I_1),\ s'\lceil X_1 = s\lceil X_1}{(P_1 \| P_2, s) \xrightarrow{C!w}_{\beta} (P_1 \| P_2', s')} \qquad \frac{(P_2, s) \xrightarrow{C?w}_{\beta_2} (P_2', s'),\ C \in (I_2 \setminus O_1)}{(P_1 \| P_2, s) \xrightarrow{C?w}_{\beta} (P_1 \| P_2', s')}$$

The *operational* semantics of processes (based on [15, 26]) is given in Table 1, by structural induction on the syntax of processes. For the sake of brevity, the distinction between $\Xi; S$, $\Xi \| S$, $S \| \Xi$, and S is omitted.

The label l in the transition $(P, s) \xrightarrow{l}_{\beta} (P', s')$ is either $\mathbf{i}$ to denote a computation step or is of the form $C?w$, $C!w$, $C.w$ for respectively input, output, and internal communication of value w. In state-based concurrency, this label is always $\mathbf{i}$. Notice that, following [18, 21], we distinguish parallel composition from network abstraction (hiding of internal communications); introducing network abstraction requires an additional rule that transforms internal communications (labels of the form $C.w$) into computation steps (label $\mathbf{i}$). In this paper, we focus on parallel composition only and refer to [21] for a treatment of network abstraction.

The ready set *Ready*(P, s) (based on [8]) is defined in Table 2. It records which actions are to be taken by process P at state s:

- $C? \in$ *Ready*(P, s): P is ready for an input communication over channel C,
- $C! \in$ *Ready*(P, s): P is ready for an output communication over channel C,
- $* \in$ *Ready*(P, s): P is ready for a computation (non-communicating) step.

Table 2. Ready set

$Ready(\Xi, s)$	=	$\{\}$	$Ready(S_1; S_2, s)$	=	$Ready(S_1, s)$
$Ready(v := e, s)$	=	$\{*\}$	$Ready(\textbf{wait } b, s)$	=	$\{* \mid s(b)\}$
$Ready(C?x, s)$	=	$\{C?\}$	$Ready(\textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2 \textbf{ fi}, s)$	=	$\{*\}$
$Ready(C!e, s)$	=	$\{C!\}$	$Ready(\textbf{while } b \textbf{ od } S \textbf{ od}, s)$	=	$\{*\}$
$Ready(P_1 \| P_2, s)$	=	$Ready(P_1, s) \cup Ready(P_2, s)$			

For example, $Ready(C?x \| (y := y + 1; D!y), s) = \{C?, *\}$. Observe that a process P whose sole channel is an internal channel C is deadlock-free at state s if and only if $* \in Ready(P, s)$ or $\{C!, C?\} \subseteq Ready(P, s)$, or P is terminated.

In message-based concurrency, compositional trace models of P can be obtained by linking successive transitions. In state-based concurrency, compositionality is achieved only if the model copes with *interferences* [17], i.e. modifications of the shared variables by the environment of P. As proposed by Aczel [5], it suffices to extend the set of transitions by allowing arbitrary environment transitions. These new transitions, that we label with **e**, are defined in Table 3; a similar construction can be found in [6, 23, 26, 27, 30].

Table 3. State-based concurrency: extended semantics

$$\vdash (P, s) \xrightarrow{\mathbf{e}}_{\beta} (P, s') \text{ where } s' \lceil X = s \lceil X$$

In order to meet the requirements (1) and (2), we follow [7] and extend this construction to message-based concurrency: successive transitions of P can be interleaved with arbitrary communications over channels not connected to P and computation steps of the environment not involving P. These new transitions are defined in Table 4.

Table 4. Message-based concurrency: extended semantics

$$\vdash (P, s) \xrightarrow{\mathbf{e}}_{\beta} (P, s') \text{ where } s' \lceil X = s \lceil X$$
$$\vdash (P, s) \xrightarrow{C!w}_{\beta} (P, s') \text{ where } C \notin (I \cup O) \text{ and } s' \lceil X = s \lceil X$$
$$\vdash (P, s) \xrightarrow{C?w}_{\beta} (P, s') \text{ where } C \notin (I \cup O) \text{ and } s' \lceil X = s \lceil X$$
$$\vdash (P, s) \xrightarrow{C.w}_{\beta} (P, s') \text{ where } C \notin (I \cup O) \text{ and } s' \lceil X = s \lceil X$$

Definition 8 (Computation and model) Let P be a process with basis β. Then $\mathcal{M}(P)$ is the set of (potential) computations of P. These are sequences

$$\sigma = (P_0, s_0) \xrightarrow{l_1} (P_1, s_1) \xrightarrow{l_2} \ldots$$

of consecutive transitions (defined in Tables 1, 3, and 4) such that P is the process appearing in the initial configuration, i.e. $P = P_0$, and for all freeze variables $v \in FV$ the value $s_0(v)$ is not changed, i.e. $s_k(v) = s_0(v)$ for all k. We also adopt the following notation:

– $P_k.\sigma$, $s_k.\sigma$: denoting the k^{th} process and k^{th} state of σ.

Example 2 In message-based concurrency, let

$$P_1 \overset{\text{def}}{=} A?x; x := x + 1; B!x \qquad \text{and} \qquad P_2 \overset{\text{def}}{=} B?y$$

A potential computation σ of the process $P_1 \| P_2$ is

$$\begin{array}{ll} & ((A?x; x := x+1; B!x) \| B?y, s) \\ \xrightarrow{A?0} & ((x := x+1; B!x) \| B?y, (s : x \mapsto 0)) \\ \xrightarrow{\mathbf{i}} & (B!x \| B?y, (s : x \mapsto 1)) \\ \xrightarrow{B.1} & (\Xi, (s : x \mapsto 1 : y \mapsto 1)) \end{array}$$

This potential computation of $P_1 \| P_2$ can be decomposed into potential computations

$$\begin{array}{ll|ll} & (A?x; x := x+1; B!x, s) & & (B?y, s) \\ \xrightarrow{A?0} & (x := x+1; B!x, (s : x \mapsto 0)) & \xrightarrow{A?0} & (B?y, (s : x \mapsto 0)) \\ \xrightarrow{\mathbf{i}} & (B!x, (s : x \mapsto 1)) & \xrightarrow{\mathbf{e}} & (B?y, (s : x \mapsto 1)) \\ \xrightarrow{B!1} & (\Xi, (s : x \mapsto 1 : y \mapsto 1)) & \xrightarrow{B?1} & (\Xi, (s : x \mapsto 1 : y \mapsto 1)) \end{array}$$

σ_1 of and P_1 and σ_2 of P_2 respectively. Observe that the second one includes a computation step of the environment (labeled with **e**) and a communication over channel A, although A is not a channel of P_2. These additional steps do not alter the evaluation of P_2 **sat** spec_2 if the specification spec_2 is over the channels and variables of P_2 only.

In the case of message-based concurrency, the traces of a process can be retrieved by appending the successive communications appearing in its potential computations.

Definition 9 (Trace) The trace of a computation σ, denoted by $tr(\sigma)$, is defined inductively:

$$\begin{array}{lcll} tr(\sigma|_0) & = & \epsilon & \\ tr(\sigma|_k) & = & tr(\sigma|_{k-1}) \frown Comm(l_k.\sigma) & (0 < k \leq |\sigma|) \end{array}$$

where $Comm(l_k)$ denotes the communication of label l_k:

$$\begin{array}{lcll} Comm(d) & = & \epsilon & \text{for } d \in \{\mathbf{i}, \mathbf{e}\} \\ Comm(d) & = & (C, w) & \text{for } d \in \{C?w, C!w, C.w\} \end{array}$$

3.3 Conjoining

In an interleaving approach to state-based concurrency, a step of $P_1 \| P_2$ is either a step of P_1 or a step of P_2. This motivates the definition of the merge operator $\otimes$, similar to previous definitions in [23, 26, 27, 30].

Definition 10 (State-based conjoining) Let σ, σ_1, σ_2 be potential computations:

$$\begin{array}{l} {}_{\beta_1}\!\otimes_{\beta_2} (\sigma_1, \sigma_2, \sigma) \\ \text{iff} \\ -|\sigma| = |\sigma_1| = |\sigma_2| \\ -\forall k : P_k.\sigma = P_k.\sigma_1 \| P_k.\sigma_2 \\ -\forall k : s_k.\sigma = s_k.\sigma_1 = s_k.\sigma_2 \\ -\forall k : \\ \qquad \quad l_k.\sigma_1 = \mathbf{i} \wedge l_k.\sigma_2 = \mathbf{e} \wedge l_k.\sigma = \mathbf{i} \\ \qquad \vee \; l_k.\sigma_1 = \mathbf{e} \wedge l_k.\sigma_2 = \mathbf{i} \wedge l_k.\sigma = \mathbf{i} \\ \qquad \vee \; l_k.\sigma_1 = \mathbf{e} \wedge l_k.\sigma_2 = \mathbf{e} \wedge l_k.\sigma = \mathbf{e} \end{array}$$

In the case of message-based concurrency, the definition is similar, except that joint communications are allowed. See Example 2 above for an illustration. Note that $tr(\sigma) = tr(\sigma_1) = tr(\sigma_2)$ follows from the definition of ${}_{\beta_1}\otimes_{\beta_2}(\sigma_1, \sigma_2, \sigma)$.

Definition 11 (Message-based conjoining) Let σ, σ_1, σ_2 be computations:

$$
\begin{array}{l}
{}_{\beta_1}\otimes_{\beta_2}(\sigma_1, \sigma_2, \sigma) \\
\text{iff} \\
-|\sigma| = |\sigma_1| = |\sigma_2| \\
-\forall k : P_k.\sigma = P_k.\sigma_1 \| P_k.\sigma_2 \\
-\forall k : s_k.\sigma = s_k.\sigma_1 = s_k.\sigma_2 \\
-\forall k :
\end{array}
$$

$$
\begin{array}{ll}
 & l_k.\sigma_1 = \mathbf{i} \wedge l_k.\sigma_2 = \mathbf{e} \wedge l_k.\sigma = \mathbf{i} \\
\vee & l_k.\sigma_1 = \mathbf{e} \wedge l_k.\sigma_2 = \mathbf{i} \wedge l_k.\sigma = \mathbf{i} \\
\vee & l_k.\sigma_1 = \mathbf{e} \wedge l_k.\sigma_2 = \mathbf{e} \wedge l_k.\sigma = \mathbf{e} \\
\vee & \exists C, w : C \in ((I_1 \cap O_1) \cup (I_2 \cap O_2)) \wedge l_k.\sigma = l_k.\sigma_1 = l_k.\sigma_2 = C.w \\
\vee & \exists C, w : C \in ((O_1 \cup O_2) \setminus (I_1 \cup I_2)) \wedge l_k.\sigma = l_k.\sigma_1 = l_k.\sigma_2 = C!w \\
\vee & \exists C, w : C \in ((I_1 \cup I_2) \setminus (O_1 \cup O_2)) \wedge l_k.\sigma = l_k.\sigma_1 = l_k.\sigma_2 = C?w \\
\vee & \exists C, w : C \in (I_1 \cap O_2) \wedge l_k.\sigma = C.w \wedge l_k.\sigma_1 = C?w \wedge l_k.\sigma_2 = C!w \\
\vee & \exists C, w : C \in (I_2 \cap O_1) \wedge l_k.\sigma = C.w \wedge l_k.\sigma_1 = C!w \wedge l_k.\sigma_2 = C?w \\
\vee & \exists C, w : C \notin (I_1 \cup I_2 \cup O_1 \cup O_2) \ \wedge l_k.\sigma \in \{C!w, C?w, C.w\} \\
 & \qquad\qquad\qquad\qquad\qquad\qquad \wedge l_k.\sigma = l_k.\sigma_1 = l_k.\sigma_2
\end{array}
$$

Clearly, the previous definitions of $\otimes$ satisfy the requirements (1) and (2) imposed in Sect. 2. The compositionality of the modeling function can be proved by case analysis on the labels.

Theorem 2 (Compositionality) $\mathcal{M}(P_1 \| P_2) = \mathcal{M}(P_1)\,{}_{\beta_1}\!\otimes_{\beta_2} \mathcal{M}(P_2)$

This completes points 1. and 2. of the check list at the end of Sect. 2.

4 State-based rule

We first recall the format of assumption-commitment specifications for state-based processes [17, 27, 30] and interpret correctness formulas in terms of sets of computations. We then derive the corresponding syntactic parallel rule from the semantic rule by considering the points 3. and 4. of our check-list.

4.1 State-based process specifications

First, the syntax and semantics of binary assertions is given. Binary assertions include primed variables and are thus evaluated on pairs of states. For instance, $(s, s') \models y' \geq y$ if and only if $s'(y)$ is greater than $s(y)$.

Definition 12 (Syntax and semantics of assertions) A *binary* assertion over B is a first-order formula whose free variables range over $V \cup X \cup V' \cup X'$ where V' (resp. X') is the set of all the variables y' such that $y \in V$ (resp. $y \in X$). The notation $(s, s') \models q$ indicates that the *binary* assertion q evaluates to true if s and s' interpret, respectively, the variables in $V \cup X$ and $V' \cup X'$. We will use $s \models q$ if no primed variable occurs free in q.

Definition 13 (State-based specification) An assumption-commitment specification of a state-based process P is a tuple $(pre, rely, wait, guar, post)$ where $rely$, $wait$, $guar$, $post$ are binary assertions over the basis of process P and no primed variable occurs in pre (i.e. pre is an unary assertion).

The informal interpretation is as follows: if the precondition pre holds initially and any state transition performed by the environment of P satisfies $rely$, then any state transition performed by P satisfies $guar$, and P gets blocked in a state that satisfies the condition $wait$ or terminates in a state that satisfies the postcondition $post$; thus, $wait = false$ means that the program must terminate. For stuttering transitions to be allowed, the predicates $rely$ and $guar$ are usually required to be reflexive.

Example 3 We develop a process P that computes the sum of $\{1, \ldots, N\}$ and adds this result to the variable z. A possible assumption-commitment specification of P is given by:

$$\begin{array}{ll} pre: & i = 0 \wedge j = 0 \\ rely: & z = z' \wedge i = i' \wedge j = j' \\ wait: & false \\ guar: & z' \geq z \\ post: & z' = z + \text{sum}\{1 \ldots N\} \end{array}$$

The variables i and j have been introduced to ease the decomposition of P into $P_1 \| P_2$. Using a counter i (resp. j) the program P_1 (resp. P_2) will compute the sum of the odd (resp. even) numbers of $\{1, \ldots, N\}$. The assumption-commitment specifications of P_1 and P_2 are:

$$\begin{array}{ll} pre_1: & i = 0 \\ rely_1: & z' \geq z \wedge i' = i \\ wait_1: & false \\ guar_1; & j' = j \wedge ((z' = z \wedge i' = i) \vee (z' = z + 2 * i + 1 \wedge i' = i + 1)) \wedge i \geq 0 \\ post_1: & i' = (N + 1) \text{ div } 2 \end{array}$$

$$\begin{array}{ll} pre_2: & j = 0 \\ rely_2: & z' \geq z \wedge j' = j \\ wait_2: & false \\ guar_2; & i' = i \wedge ((z' = z \wedge j' = j) \vee (z' = z + 2 * j + 2 \wedge j' = j + 1)) \wedge j \geq 0 \\ post_2: & j' = N \text{ div } 2 \end{array}$$

The correctness of a state-based process w.r.t. an assumption-commitment specification is now formally defined. The assertion pre refers to the initial state; the assertion $rely$ refers to state transitions labeled with **e**; the assertion $guar$ refers to state transitions labeled with **i**; the assertion $wait$ refers to the blocked states (w.r.t the initial state); the assertion $post$ refers to the terminated states (w.r.t. the initial state). Recall that the computations of a state-based process include transitions labeled in $\{\mathbf{i}, \mathbf{e}\}$ only; there are no communications.

Definition 14 (Correctness of state-based processes) Let P be a state-based process:

$$\models P \textbf{ sat } (pre, rely, wait, guar, post) \text{ iff } \mathcal{M}(P) \subseteq (\mathbf{SA} \rightarrow \mathbf{C})$$

where

$$
\begin{aligned}
\sigma \in \mathbf{SA} &\overset{\text{def}}{=} \sigma \in PRE \wedge \sigma \in RELY \\
\sigma \in \mathbf{SC} &\overset{\text{def}}{=} \sigma \in GUAR \\
\sigma \in \mathbf{OC} &\overset{\text{def}}{=} \sigma \in CONV \wedge \sigma \in WAIT \wedge \sigma \in POST
\end{aligned}
$$

and

$$
\begin{aligned}
\sigma \in PRE &\overset{\text{def}}{=} s_0.\sigma \models pre \\
\sigma \in RELY &\overset{\text{def}}{=} \forall k : l_k.\sigma = \mathbf{e} \Rightarrow (s_{k-1}.\sigma, s_k.\sigma) \models rely \\
\sigma \in GUAR &\overset{\text{def}}{=} \forall k : l_k.\sigma = \mathbf{i} \Rightarrow (s_{k-1}.\sigma, s_k.\sigma) \models guar \\
\sigma \in CONV &\overset{\text{def}}{=} \#\{k \mid l_k.\sigma = \mathbf{i}\} < \infty \\
\sigma \in WAIT &\overset{\text{def}}{=} \forall k : (* \notin Ready(P_k.\sigma, s_k.\sigma) \wedge P_k.\sigma \not\equiv \Xi) \\
&\qquad\qquad \Rightarrow (s_0.\sigma, s_k.\sigma) \models wait \\
\sigma \in POST &\overset{\text{def}}{=} \forall k : P_k.\sigma = \Xi \Rightarrow (s_0.\sigma, s_k.\sigma) \models post
\end{aligned}
$$

Due to the commitment *CONV* (converge), the process is required to get blocked in a state that satisfies *wait* or to terminate in a state that satisfies *post*. Consequently, if $wait \equiv false$, the process is required to terminate. Obviously, because no progress property is included in the definition of $\mathcal{M}(P)$, termination means divergence-freedom and deadlock-freedom. Indeed, the process might be continuously overtaken by its environment (other processes) and thus never reach its terminating state. Nevertheless, if all processes can be proved to be divergence-free and deadlock-free, then termination is ensured. Notice that including a progress property in the definition of $\mathcal{M}(P)$ would not alter the validity of the discussion (see [13, 27]): although proofs must be adapted, Theorems 2 and 3 still hold.

Theorem 3 asserts that the interpretations $\mathbf{SA} \rightarrow \mathbf{C}$ and $\mathbf{SA}@_{\rightarrow}\mathbf{C}$ are equivalent for the assumption-commitment specifications of state-based processes. This establishes point 3. of our check-list at the end of Sect. 2.

Theorem 3 *Let* **SA**,**SC** *and* **OC** *be as in* Definition 14. *Then,*

$$P \ \mathbf{sat}\ (pre, rely, wait, guar, post) \text{ iff } \mathcal{M}(P) \subseteq (\mathbf{SA}@_{\rightarrow}\mathbf{C})$$

Proof Since $(\mathbf{SA}@_{\rightarrow}\mathbf{C}) = (\mathbf{SA} \rightarrow \mathbf{C}) \cap (\mathbf{SA} \ @\ \mathbf{SC})$, it suffices to prove

$$\mathcal{M}(P) \subseteq (\mathbf{SA} \rightarrow \mathbf{C}) \Rightarrow \mathcal{M}(P) \subseteq (\mathbf{SA} \ @\ \mathbf{SC})$$

Clearly, $\sigma|_0 \in GUAR$ and thus $\sigma|_0 \in \mathbf{SC}$. Then, assume $\mathcal{M}(P) \subseteq (\mathbf{SA} \rightarrow \mathbf{C})$, and let $k > 0$:

$$
\begin{array}{ll}
& \sigma \in \mathcal{M}(P) \wedge \sigma|_{k-1} \in \mathbf{SA} \\
\Rightarrow & \quad \% \ \mathcal{M}(P) \text{ is prefix-closed} \\
& \sigma|_k \in \mathcal{M}(P) \wedge \sigma|_{k-1} \in \mathcal{M}(P) \wedge \sigma|_{k-1} \in \mathbf{SA} \\
\Rightarrow & \quad \% \ \mathcal{M}(P) \subseteq (\mathbf{SA} \rightarrow \mathbf{C}) \\
& \sigma|_k \in (\mathbf{SA} \rightarrow \mathbf{C}) \wedge \sigma|_{k-1} \in (\mathbf{SA} \rightarrow \mathbf{C}) \wedge \sigma|_{k-1} \in \mathbf{SA} \\
\Rightarrow & \quad \% \text{ Definition of } \mathbf{SA} \rightarrow \mathbf{C} \\
& \sigma|_k \in (\mathbf{SA} \rightarrow \mathbf{C}) \wedge \sigma|_{k-1} \in \mathbf{SA} \wedge \sigma|_{k-1} \in \mathbf{SC} \\
\Rightarrow & \quad \% \text{ State-based processes } : l_k.\sigma = \mathbf{i} \vee l_k.\sigma = \mathbf{e} \\
& \quad\quad \mathbf{SA} \text{ constrains the } \mathbf{e}\text{-labeled transitions only} \\
& \quad\quad \mathbf{SC} \text{ constrains the } \mathbf{i}\text{-labeled transitions only} \\
& \sigma|_k \in (\mathbf{SA} \rightarrow \mathbf{C}) \wedge (\sigma|_k \in \mathbf{SA} \vee \sigma|_k \in \mathbf{SC}) \\
\Rightarrow & \quad \% \text{ Definition of } \mathbf{SA} \rightarrow \mathbf{C} \\
& \sigma|_k \in \mathbf{SC}
\end{array}
$$

□

4.2 The state-based parallel composition rule

The syntactic parallel composition rule presented below could be further simplified by e.g. replacing $post$ with $post_1 \wedge post_2$. A proof system [27, 30] indeed includes adaptation rules such as weakening rules. Nevertheless, our formulation better highlights the relation with the semantic rule.

$$(4) \quad \frac{\begin{array}{ll} P_1 \ \textbf{sat} \ (pre_1, rely_1, wait_1, guar_1, post_1) & \\ P_2 \ \textbf{sat} \ (pre_2, rely_2, wait_2, guar_2, post_2) & \\ pre \Rightarrow pre_1 \wedge pre_2 & wait_1 \wedge post_2 \Rightarrow wait \\ rely \vee guar_1 \Rightarrow rely_2 & wait_2 \wedge post_1 \Rightarrow wait \\ rely \vee guar_2 \Rightarrow rely_1 & wait_1 \wedge wait_2 \Rightarrow wait \\ guar_1 \vee guar_2 \Rightarrow guar & post_1 \wedge post_2 \Rightarrow post \end{array}}{P_1 \| P_2 \ \textbf{sat} \ (pre, rely, wait, guar, post)}$$

The disjunction in for instance premise $guar_1 \vee guar_2 \Rightarrow guar$ can be explained as follows: a state transition from $P_1 \| P_2$ is either a state transition from P_1 *or* a state transition from P_2. This will be made more apparent when proving that the semantic premises of Rule (3) follow from the above ones.

Example 4 By combining Rule (4) with other adaptation rules, we may weaken the premise $post_1 \wedge post_2 \Rightarrow post$ into $dinv \wedge post_1 \wedge post_2 \Rightarrow post$. The additional binary assertion $dinv$ (dynamic invariant [17]) expresses a relation between the initial state and any further state in a computation; this must be checked against the assertions pre (initially), $rely$ (environment transition), $guar_1$ (transition of P_1), and $guar_2$ (transition of P_2). In case of Example 3, the decomposition can be proved correct by choosing:

$$dinv: \quad z' = z + \sum_{l=1}^{i'}(2*l-1) + \sum_{l=1}^{j'}(2*l)$$

Let for $i \in \{1, 2\}$ the properties PRE_i, $RELY_i$, ... be defined from pre_i, $rely_i$, ... as in Definition 14, and let

$$\begin{array}{rcl} \sigma \in \mathbf{SA} & \overset{\text{def}}{=} & \sigma \in PRE \wedge \sigma \in RELY \\ \sigma \in \mathbf{SC} & \overset{\text{def}}{=} & \sigma \in GUAR \\ \sigma \in \mathbf{OC} & \overset{\text{def}}{=} & \sigma \in WAIT \wedge \sigma \in POST \wedge \sigma \in CONV \\ \sigma_i \in \mathbf{SA}_i & \overset{\text{def}}{=} & \sigma_i \in PRE_i \wedge \sigma_i \in RELY_i \\ \sigma_i \in \mathbf{SC}_i & \overset{\text{def}}{=} & \sigma_i \in GUAR_i \\ \sigma_i \in \mathbf{OC}_i & \overset{\text{def}}{=} & \sigma_i \in WAIT_i \wedge \sigma_i \in POST_i \wedge \sigma_i \in CONV \end{array}$$

The rest of this section is devoted to point 4. of the check list, hence to the soundness of Rule (4). In all cases, the proof of $[\beta_1 \otimes \beta_2 : P(\sigma_1, \sigma_2, \sigma)]$ proceeds as follows: we assume $\beta_1 \otimes \beta_2 \, (\sigma_1, \sigma_2, \sigma)$ and prove $P(\sigma_1, \sigma_2, \sigma)$. Throughout that proof, we may thus assume $|\sigma| = |\sigma_1| = |\sigma_2|$ and $\beta_1 \otimes \beta_2 \, (\sigma_1|_k, \sigma_2|_k, \sigma|_k)$ for all k.

Theorem 4 *The semantic premises*

$$\begin{array}{ll} (i) & [\beta_1 \otimes \beta_2 : \sigma \in \mathbf{SA} \wedge \sigma_1 \in \mathbf{SC}_1 \wedge \sigma_2 \in \mathbf{SC}_2 \Rightarrow \sigma_1 \in \mathbf{SA}_1 \wedge \sigma_2 \in \mathbf{SA}_2] \\ (ii) & [\beta_1 \otimes \beta_2 : \sigma_1 \in \mathbf{SC}_1 \wedge \sigma_2 \in \mathbf{SC}_2 \Rightarrow \sigma \in \mathbf{SC}] \\ (iii) & [\beta_1 \otimes \beta_2 : \sigma \in \mathbf{SA} \wedge \sigma_1 \in \mathbf{OC}_1 \wedge \sigma_2 \in \mathbf{OC}_2 \Rightarrow \sigma \in \mathbf{OC}] \end{array}$$

of Rule (3) follow from the syntactic premises of Rule (4).

Proof We first consider semantic premise (i). Observe the introduction of disjunction in the second proof step.

$$\begin{array}{ll}
 & \sigma \in \mathbf{SA} \wedge \sigma_1 \in \mathbf{SC}_1 \wedge \sigma_2 \in \mathbf{SC}_2 \\
\equiv & \quad \text{\% Definition of } \mathbf{SA}, \mathbf{SC}_1, \mathbf{SC}_2, \mathit{PRE}, \mathit{RELY}, \mathit{GUAR}_1, \mathit{GUAR}_2 \\
 & s_0.\sigma \models \mathit{pre} \\
 & \forall k : l_k.\sigma = \mathbf{e} \Rightarrow (s_{k-1}.\sigma, s_k.\sigma) \models \mathit{rely} \\
 & \forall k : l_k.\sigma_1 = \mathbf{i} \Rightarrow (s_{k-1}.\sigma_1, s_k.\sigma_1) \models \mathit{guar}_1 \\
 & \forall k : l_k.\sigma_2 = \mathbf{i} \Rightarrow (s_{k-1}.\sigma_2, s_k.\sigma_2) \models \mathit{guar}_2 \\
\Rightarrow & \quad \text{\% Definition of } \otimes : \; s_{k-1}.\sigma = s_{k-1}.\sigma_1 = s_{k-1}.\sigma_2 \\
 & \qquad\qquad\qquad\qquad s_k.\sigma = s_k.\sigma_1 = s_k.\sigma_2 \\
 & \qquad\qquad\qquad\qquad l_k.\sigma_1 = \mathbf{e} \Rightarrow l_k.\sigma = \mathbf{e} \vee l_k.\sigma_2 = \mathbf{i} \\
 & \qquad\qquad\qquad\qquad l_k.\sigma_2 = \mathbf{e} \Rightarrow l_k.\sigma = \mathbf{e} \vee l_k.\sigma_1 = \mathbf{i} \\
 & s_0.\sigma_1 \models \mathit{pre} \wedge s_0.\sigma_2 \models \mathit{pre} \\
 & \forall k : l_k.\sigma_1 = \mathbf{e} \Rightarrow (s_{k-1}.\sigma_1, s_k.\sigma_1) \models (\mathit{rely} \vee \mathit{guar}_2) \\
 & \forall k : l_k.\sigma_2 = \mathbf{e} \Rightarrow (s_{k-1}.\sigma_2, s_k.\sigma_2) \models (\mathit{rely} \vee \mathit{guar}_1) \\
\Rightarrow & \quad \text{\% Premises } \mathit{rely} \vee \mathit{guar}_2 \Rightarrow \mathit{rely}_1,\; \mathit{rely} \vee \mathit{guar}_1 \Rightarrow \mathit{rely}_2, \\
 & \qquad \text{and } \mathit{pre} \Rightarrow \mathit{pre}_1 \wedge \mathit{pre}_2 \\
 & s_0.\sigma_1 \models \mathit{pre}_1 \wedge s_0.\sigma_2 \models \mathit{pre}_2 \\
 & \forall k : l_k.\sigma_1 = \mathbf{e} \Rightarrow (s_{k-1}.\sigma_1, s_k.\sigma_1) \models \mathit{rely}_1 \\
 & \forall k : l_k.\sigma_2 = \mathbf{e} \Rightarrow (s_{k-1}.\sigma_2, s_k.\sigma_2) \models \mathit{rely}_2 \\
\equiv & \quad \text{\% Definition of } \mathit{PRE}_1, \mathit{PRE}_2, \mathit{RELY}_1, \mathit{RELY}_2, \mathbf{SA}_1, \mathbf{SA}_2 \\
 & \sigma_1 \in \mathbf{SA}_1 \wedge \sigma_2 \in \mathbf{SA}_2
\end{array}$$

Next, we consider semantic premise (ii). Again, observe the introduction of disjunction in the second proof step.

$$\begin{array}{ll}
 & \sigma_1 \in \mathbf{SC}_1 \wedge \sigma_2 \in \mathbf{SC}_2 \\
\equiv & \quad \text{\% Definition of } \mathbf{SC}_1, \mathbf{SC}_2, \mathit{GUAR}_1, \mathit{GUAR}_2 \\
 & \forall k : l_k.\sigma_1 = \mathbf{i} \Rightarrow (s_{k-1}.\sigma_1, s_k.\sigma_1) \models \mathit{guar}_1 \\
 & \forall k : l_k.\sigma_2 = \mathbf{i} \Rightarrow (s_{k-1}.\sigma_2, s_k.\sigma_2) \models \mathit{guar}_2 \\
\Rightarrow & \quad \text{\% Definition of } \otimes : \; s_{k-1}.\sigma = s_{k-1}.\sigma_1 = s_{k-1}.\sigma_2 \\
 & \qquad\qquad\qquad\qquad s_k.\sigma = s_k.\sigma_1 = s_k.\sigma_2 \\
 & \qquad\qquad\qquad\qquad l_k.\sigma = \mathbf{i} \Rightarrow l_k.\sigma_1 = \mathbf{i} \vee l_k.\sigma_2 = \mathbf{i} \\
 & \qquad \text{Premise } \mathit{guar}_1 \vee \mathit{guar}_2 \Rightarrow \mathit{guar} \\
 & \forall k : l_k.\sigma = \mathbf{i} \Rightarrow (s_{k-1}.\sigma, s_k.\sigma) \models \mathit{guar} \\
\equiv & \quad \text{\% Definition of } \mathit{GUAR}, \mathbf{SC} \\
 & \sigma \in \mathbf{SC}
\end{array}$$

We postpone the proof of semantic premise (iii) and consider intermediate results. Firstly, by definition of $\otimes$, we have $l_k.\sigma = \mathbf{i} \Leftrightarrow l_k.\sigma_1 = \mathbf{i} \vee l_k.\sigma_2 = \mathbf{i}$ for any k. Therefore, we immediately deduce:

$$[_{\beta_1 \otimes \beta_2} : \sigma \in \mathit{CONV} \Leftrightarrow \sigma_1 \in \mathit{CONV} \wedge \sigma_2 \in \mathit{CONV}]$$

Then, we observe that if the process $P_1 \| P_2$ is waiting, either both processes are waiting, or one is terminated and the other is waiting; if the process $P_1 \| P_2$ has terminated, both processes have terminated. More formally, from the definition of $\otimes$ and the definition of ready sets, we deduce:

$$\begin{array}{ll} & * \notin \mathit{Ready}(P_k.\sigma, s_k.\sigma) \wedge P_k.\sigma \neq \Xi \Rightarrow \\ & \quad (P_k.\sigma_1 \neq \Xi \wedge P_k.\sigma_2 \neq \Xi \wedge \\ & \quad\ * \notin \mathit{Ready}(P_k.\sigma_1, s_k.\sigma_1) \wedge * \notin \mathit{Ready}(P_k.\sigma_2, s_k.\sigma_2)) \\ \vee & \quad (P_k.\sigma_1 = \Xi \wedge P_k.\sigma_2 \neq \Xi \wedge * \notin \mathit{Ready}(P_k.\sigma_2, s_k.\sigma_2)) \\ \vee & \quad (P_k.\sigma_1 \neq \Xi \wedge P_k.\sigma_2 = \Xi \wedge * \notin \mathit{Ready}(P_k.\sigma_1, s_k.\sigma_1)) \end{array}$$

and

$$P_k.\sigma = \Xi \Rightarrow P_k.\sigma_1 = \Xi \wedge P_k.\sigma_2 = \Xi$$

Using these preliminary results, semantic premise (*iii*) is now proved:

$$\begin{array}{ll} & \sigma \in \mathbf{SA} \wedge \sigma_1 \in \mathbf{OC}_1 \wedge \sigma_2 \in \mathbf{OC}_2 \\ \Rightarrow & \quad \%\ \text{Definition of } \mathbf{OC}_1, \mathbf{OC}_2, \mathit{WAIT}_1, \mathit{WAIT}_2, \mathit{POST}_1, \mathit{POST}_2 \\ & \sigma_1 \in \mathit{CONV} \wedge \sigma_2 \in \mathit{CONV} \\ & \forall k : * \notin \mathit{Ready}(P_k.\sigma_1, s_k.\sigma_1) \wedge P_k.\sigma_1 \neq \Xi \Rightarrow (s_0.\sigma_1, s_k.\sigma_1) \models \mathit{wait}_1 \\ & \forall k : * \notin \mathit{Ready}(P_k.\sigma_2, s_k.\sigma_2) \wedge P_k.\sigma_2 \neq \Xi \Rightarrow (s_0.\sigma_2, s_k.\sigma_2) \models \mathit{wait}_2 \\ & \forall k : P_k.\sigma_1 = \Xi \Rightarrow (s_0.\sigma_1, s_k.\sigma_1) \models \mathit{post}_1 \\ & \forall k : P_k.\sigma_2 = \Xi \Rightarrow (s_0.\sigma_2, s_k.\sigma_2) \models \mathit{post}_2 \\ \Rightarrow & \quad \%\ \text{Preliminary results} \\ & \quad\ \ \text{Definition of } \otimes : \ s_0.\sigma = s_0.\sigma_1 = s_0.\sigma_2 \\ & \qquad\qquad\qquad\qquad\quad s_k.\sigma = s_k.\sigma_1 = s_k.\sigma_2 \\ & \sigma \in \mathit{CONV} \\ & \forall k : * \notin \mathit{Ready}(P_k.\sigma, s_k.\sigma) \wedge P_k.\sigma \neq \Xi \Rightarrow \\ & \quad (s_0.\sigma, s_k.\sigma) \models ((\mathit{wait}_1 \wedge \mathit{wait}_2) \vee (\mathit{wait}_2 \wedge \mathit{post}_1) \vee (\mathit{wait}_1 \wedge \mathit{post}_2)) \\ & \forall k : P_k.\sigma = \Xi \Rightarrow (s_0.\sigma, s_k.\sigma) \models \mathit{post}_1 \wedge \mathit{post}_2 \\ \Rightarrow & \quad \%\ \text{Premises} \ \ \mathit{wait}_1 \wedge \mathit{wait}_2 \Rightarrow \mathit{wait} \qquad \mathit{wait}_2 \wedge \mathit{post}_1 \Rightarrow \mathit{wait} \\ & \qquad\qquad\quad\ \ \mathit{post}_1 \wedge \mathit{post}_2 \Rightarrow \mathit{post} \qquad \mathit{wait}_1 \wedge \mathit{post}_2 \Rightarrow \mathit{wait} \\ & \sigma \in \mathit{CONV} \\ & \forall k : * \notin \mathit{Ready}(P_k.\sigma, s_k.\sigma) \wedge P_k.\sigma \neq \Xi \Rightarrow (s_0.\sigma, s_k.\sigma) \models \mathit{wait} \\ & \forall k : P_k.\sigma = \Xi \Rightarrow (s_0.\sigma, s_k.\sigma) \models \mathit{post} \\ \equiv & \quad \%\ \text{Definition of } \mathit{WAIT}, \mathit{POST}, \mathbf{OC} \\ & \sigma \in \mathbf{OC} \end{array}$$

□

5 Message-based rule

We first recall the format of assumption-commitment specifications for message-based processes [21, 31, 32] and interpret correctness formulas in terms of sets of computations. We then derive the corresponding syntactic parallel rule from the semantic rule by considering the points 3. and 4. of our check-list.

5.1 Message-based process specifications

First, the syntax and semantics of assertions is given. No primed variables are allowed, but assertions may refer to freeze variables, to a trace variable, and to some enabledness flags.

Definition 15 (Syntax and semantics of assertions) A *unary* assertion over the basis $(I, O, \emptyset, X)$ is a first-order formula with free variables in the set $FV \cup X \cup \{h\} \cup Flags(P)$; h is the communication trace variable, and $Flags(P)$ contains the termination flag u, the enabledness flag $en(*)$, and the enabledness flags $en(C?)$, $en(D!)$ for each $C \in I$ and each $D \in O$. The trace variable h must appear under the scope of a projection onto channels in $I \cup O$.

The notation $(P, s, tr) \models q$ indicates that the unary assertion q evaluates to true if s interprets the freeze and process variables, h takes the value tr, and the boolean flags u, $en(*)$, $en(C?)$, $en(D!)$ take the value true if and only if $P = \Xi$, $* \in Ready(P, s)$, $C? \in Ready(P, s)$, and $D! \in Ready(P, s)$ respectively. We will use $(s, tr) \models q$ if no flag occurs free in q.

Definition 16 (Message-based specification) An assumption-commitment specification of a message-based process is a tuple $(pre, rely, guar, post)$ of assertions over the basis of that process, with the restriction that no flag occurs free in pre, $rely$, $post$, and no process variable occurs free in $rely$ and $guar$.

The informal interpretation is as follows: $guar$ holds initially and, if pre holds initially and $rely$ holds initially and after each communication, then $guar$ holds after each communication and $post$ holds when the program terminates. As discussed in [21], deadlock-freedom can be expressed within $guar$. For example, if the basis of P is $\{\{A, B\}, \{B, C\}, \emptyset, X\}$, the commitment for deadlock freedom is $u \vee en(*) \vee en(A?) \vee en(C!) \vee (en(B!) \wedge en(B?))$. Indeed, P is deadlock-free at state s if and only if it is terminated, ready for a computation step, waiting for an input communication, waiting for an output communication, or ready for an internal communication.

Example 5 Compositional verification [24]. Consider the process $P_1 \| P_2$ where P_1 and P_2 communicate via channels A and B. P_2 communicates with the external environment via channel C:

$$\begin{array}{ll} P_1 :: & A?x; x := x + 1; B!(x + 2) \\ P_2 :: & C?y; A!y; B?y; y := y + 2 \end{array}$$

Then, if P_2 receives the value m on channel C, $P_1 \| P_2$ terminates with $x = m + 1$ and $y = m + 5$. The specification of $P_1 \| P_2$ is:

$$\begin{array}{ll} pre: & h_A = \epsilon \wedge h_B = \epsilon \wedge h_C = \epsilon \\ rely: & h_C \sqsubseteq \langle m \rangle \\ guar: & true \\ post: & x = m + 1 \wedge y = m + 5 \end{array}$$

where h_X is the sequence of values transmitted along channel X in the trace h and $r \sqsubseteq s$ denotes that r is a prefix of s. This specification can be proved from the following specifications of P_1 and P_2 (m, n, p and q are freeze variables):

$$\begin{array}{ll||ll} pre_1: & h_A = \epsilon \wedge h_B = \epsilon & pre_2: & h_A = \epsilon \wedge h_B = \epsilon \wedge h_C = \epsilon \\ rely_1: & h_A \sqsubseteq \langle n \rangle & rely_2: & h_C \sqsubseteq \langle p \rangle \wedge h_B \sqsubseteq \langle q \rangle \\ guar_1: & h_B \sqsubseteq \langle n + 3 \rangle & guar_2: & h_A \sqsubseteq \langle p \rangle \\ post_1: & x = n + 1 & post_2: & y = q + 2 \end{array}$$

The correctness of a message-based process w.r.t. an assumption-commitment specification is now formally defined. Note that the universal quantification over all the possible assignments to the freeze variables is captured by the definition of $\mathcal{M}(P)$.

Definition 17 (Correctness of message-based processes) Let P be a message-based process:

$$\models P \textbf{ sat } (pre, rely, guar, post) \text{ iff } \mathcal{M}(P) \subseteq (\mathbf{SA}@_{\rightarrow}\mathbf{C})$$

where

$$\begin{array}{rcl} \sigma \in \mathbf{SA} & \overset{\text{def}}{=} & \sigma \in PRE \wedge \sigma \in RELY \\ \sigma \in \mathbf{SC} & \overset{\text{def}}{=} & \sigma \in GUAR \\ \sigma \in \mathbf{OC} & \overset{\text{def}}{=} & \sigma \in CONV \wedge \sigma \in POST \end{array}$$

and

$$\begin{array}{rcl} \sigma \in PRE & \overset{\text{def}}{=} & (s_0.\sigma, tr(\sigma|_0)) \models pre \\ \sigma \in RELY & \overset{\text{def}}{=} & \forall k : (s_k.\sigma, tr(\sigma|_k)) \models rely \\ \sigma \in GUAR & \overset{\text{def}}{=} & \forall k : (P_k.\sigma, s_k.\sigma, tr(\sigma|_k)) \models guar \\ \sigma \in DIV & \overset{\text{def}}{=} & \#\{k \mid l_k.\sigma = \mathbf{i}\} = \infty \wedge (\exists k : \forall j > k : tr(\sigma|_j) = tr(\sigma|_k)) \\ \sigma \in CONV & \overset{\text{def}}{=} & \neg(\sigma \in DIV) \\ \sigma \in POST & \overset{\text{def}}{=} & \forall k : P_k.\sigma = \Xi \Rightarrow (s_k.\sigma, tr(\sigma|_k) \models post \end{array}$$

In this case, point 3. of our check list is directly established by the definition.

5.2 *The message-based parallel composition rule*

If no flag occurs in $guar_i$, the syntactic rule for message-based processes is:

$$(5) \qquad \frac{\begin{array}{ll} P_1 \textbf{ sat } (pre_1, rely_1, guar_1, post_1) & \\ P_2 \textbf{ sat } (pre_2, rely_2, guar_2, post_2) & \\ pre \Rightarrow pre_1 \wedge pre_2 & \\ rely \wedge guar_1 \Rightarrow rely_2 & guar_1 \wedge guar_2 \Rightarrow guar \\ rely \wedge guar_2 \Rightarrow rely_1 & post_1 \wedge post_2 \Rightarrow post \end{array}}{P_1 \| P_2 \textbf{ sat } (pre, rely, guar, post)}$$

The conjunction in for instance premise $guar_1 \wedge guar_2 \Rightarrow guar$ can be explained as follows: a joint communication from $P_1\|P_2$ is both a communication of P_1 and a communication of P_2. This will be made more apparent when proving that the premises of Rule (3) follow from the ones above.

Example 6 See Example 5. Since m, n, p, q are freeze variables, the specifications of P_1 and P_2 can be rewritten by replacing n,p,q with m,m,$m + 3$ respectively. Then, the premises of Rule (5) can be checked easily.

If $guar_i$ includes flags, suitable renaming must be done [21]: the premise $guar_1 \wedge guar_2 \Rightarrow guar$ must be replaced by

$$Merge \wedge guar_1[u_1/u, en(*_1)/en(*)] \wedge guar_2[u_2/u, en(*_2)/en(*)] \Rightarrow guar$$

where $Merge = (u \Leftrightarrow u_1 \wedge u_2) \wedge (en(*) \Leftrightarrow en(*_1) \vee en(*_2))$. The first equivalence asserts that a parallel process has terminated if and only if all its sub-processes have terminated. The second one asserts that a computation step is enabled in a parallel process if and only if it is enabled in one of its sub-processes.

Let for $i \in \{1,2\}$ the properties PRE_i,$RELY_i$,... be defined from pre_i,$rely_i$,... as in Definition 17, and let

$$\begin{array}{rcl}
\sigma \in \mathbf{SA} & \overset{\text{def}}{=} & \sigma \in PRE \wedge \sigma \in RELY \\
\sigma \in \mathbf{SC} & \overset{\text{def}}{=} & \sigma \in GUAR \\
\sigma \in \mathbf{OC} & \overset{\text{def}}{=} & \sigma \in POST \wedge \sigma \in CONV \\
\sigma_i \in \mathbf{SA}_i & \overset{\text{def}}{=} & \sigma_i \in PRE_i \wedge \sigma_i \in RELY_i \\
\sigma_i \in \mathbf{SC}_i & \overset{\text{def}}{=} & \sigma_i \in GUAR_i \\
\sigma_i \in \mathbf{OC} & \overset{\text{def}}{=} & \sigma_i \in POST_i \wedge \sigma_i \in CONV
\end{array}$$

The rest of this section is devoted to point 4. of the check list, hence to the soundness of Rule (5).

Theorem 5 *The semantic premises*

$$\begin{array}{rl}
(i) & [\beta_1 \otimes \beta_2 : \sigma \in \mathbf{SA} \wedge \sigma_1 \in \mathbf{SC}_1 \wedge \sigma_2 \in \mathbf{SC}_2 \Rightarrow \sigma_1 \in \mathbf{SA}_1 \wedge \sigma_2 \in \mathbf{SA}_2] \\
(ii) & [\beta_1 \otimes \beta_2 : \sigma_1 \in \mathbf{SC}_1 \wedge \sigma_2 \in \mathbf{SC}_2 \Rightarrow \sigma \in \mathbf{SC}] \\
(iii) & [\beta_1 \otimes \beta_2 : \sigma \in \mathbf{SA} \wedge \sigma_1 \in \mathbf{OC}_1 \wedge \sigma_2 \in \mathbf{OC}_2 \Rightarrow \sigma \in \mathbf{OC}]
\end{array}$$

of Rule (3) follow from the syntactic premises of Rule (5).

Proof We first consider semantic premise (i). Observe the introduction of conjunction in the third proof step: a trace of σ is both a trace of σ_1 and a trace of σ_2.

$$\begin{array}{ll}
 & \sigma \in \mathbf{SA} \wedge \sigma_1 \in \mathbf{SC}_1 \wedge \sigma_2 \in \mathbf{SC}_2 \\
\equiv & \quad \text{\% Definition of } \mathbf{SA}, \mathbf{SC}_1, \mathbf{SC}_2, PRE, RELY, GUAR_1, GUAR_2 \\
 & (s_0.\sigma, tr(\sigma|_0)) \models pre \\
 & \forall k : (s_k.\sigma, tr(\sigma|_k)) \models rely \\
 & \forall k : (P_k.\sigma_1, s_k.\sigma_1, tr(\sigma_1|_k)) \models guar_1 \\
 & \forall k : (P_k.\sigma_2, s_k.\sigma_2, tr(\sigma_2|_k)) \models guar_2 \\
\Rightarrow & \quad \text{\% Premise } pre \Rightarrow pre_1 \wedge pre_2 \text{ and} \\
 & \quad \quad \text{existential quantification on flags} \\
 & (s_0.\sigma, tr(\sigma|_0)) \models pre_1 \wedge (s_0.\sigma, tr(\sigma|_0)) \models pre_2 \\
 & \forall k : (s_k.\sigma, tr(\sigma|_k)) \models rely \\
 & \forall k : (s_k.\sigma_1, tr(\sigma_1|_k)) \models \exists \overline{fl} : guar_1 \\
 & \forall k : (s_k.\sigma_2, tr(\sigma_2|_k)) \models \exists \overline{fl} : guar_2 \\
\equiv & \quad \text{\% Definition of } \otimes : \; s_k.\sigma = s_k.\sigma_1 = s_k.\sigma_2 \\
 & \quad \quad \quad \quad \quad \quad tr(\sigma|_k) = tr(\sigma_1|_k) = tr(\sigma_2|_k) \\
 & (s_0.\sigma_1, tr(\sigma_1|_0)) \models pre_1 \wedge (s_0.\sigma_2, tr(\sigma_2|_0)) \models pre_2 \\
 & \forall k : (s_k.\sigma_1, tr(\sigma_1|_k)) \models (rely \wedge \exists \overline{fl} : guar_2) \\
 & \forall k : (s_k.\sigma_2, tr(\sigma_2|_k)) \models (rely \wedge \exists \overline{fl} : guar_1) \\
\Rightarrow & \quad \text{\% Premises } rely \wedge guar_1 \Rightarrow rely_2, rely \wedge guar_2 \Rightarrow rely_1. \\
 & \quad \quad \text{and the restriction that no flag occurs in } rely, rely_1, rely_2 \\
 & (s_0.\sigma_1, tr(\sigma_1|_0)) \models pre_1 \wedge (s_0.\sigma_2, tr(\sigma_2|_0)) \models pre_2 \\
 & \forall k : (s_k.\sigma_1, tr(\sigma_1|_k)) \models rely_1 \\
 & \forall k : (s_k.\sigma_2, tr(\sigma_2|_k)) \models rely_2 \\
\equiv & \quad \text{\% Definition of } PRE_1, PRE_2, RELY_1, RELY_2, \mathbf{SA}_1, \mathbf{SA}_2 \\
 & \sigma_1 \in \mathbf{SA}_1 \wedge \sigma_2 \in \mathbf{SA}_2
\end{array}$$

Before considering the proof of semantic premise (ii) we must define the interpretation for *all* flags: For each k, the interpretation η_k of flags is:

$$
\begin{array}{ll}
\eta_k \models en(*) \equiv (P_k.\sigma, s_k.\sigma) \models en(*) & \eta_k \models u \equiv (P_k.\sigma, s_k.\sigma) \models u \\
\eta_k \models en(*_1) \equiv (P_k.\sigma_1, s_k.\sigma_1) \models en(*) & \eta_k \models u_1 \equiv (P_k.\sigma_1, s_k.\sigma_1) \models u \\
\eta_k \models en(*_2) \equiv (P_k.\sigma_2, s_k.\sigma_2) \models en(*) & \eta_k \models u_2 \equiv (P_k.\sigma_2, s_k.\sigma_2) \models u
\end{array}
$$
$$
\begin{array}{c}
\eta_k \models en(C_1?) \equiv (P_k.\sigma_1, s_k.\sigma_1) \models en(C_1?) \\
\eta_k \models en(C_2?) \equiv (P_k.\sigma_2, s_k.\sigma_2) \models en(C_2?) \\
\eta_k \models en(D_1!) \equiv (P_k.\sigma_1, s_k.\sigma_1) \models en(D_1!) \\
\eta_k \models en(D_2!) \equiv (P_k.\sigma_2, s_k.\sigma_2) \models en(D_2!)
\end{array}
$$

where $C_1 \in I_1$, $C_2 \in I_2$, $D_1 \in O_1$, $D_2 \in O_2$. Then:

$$
\begin{array}{ll}
 & \sigma_1 \in \mathbf{SC}_1 \wedge \sigma_2 \in \mathbf{SC}_2 \\
\equiv & \quad \text{\% Definition of } \mathbf{SC}_1, \mathbf{SC}_2, GUAR_1, GUAR_2 \\
 & \forall k : (P_k.\sigma_1, s_k.\sigma_1, tr(\sigma_1|_k)) \models guar_1 \\
 & \forall k : (P_k.\sigma_2, s_k.\sigma_2, tr(\sigma_2|_k)) \models guar_2 \\
\equiv & \quad \text{\% Definition of } \otimes, \eta_k, guar_1 \text{ is over } \beta_1, guar_2 \text{ is over } \beta_2 \\
 & \forall k : (\eta_k, s_k.\sigma, tr(\sigma|_k)) \models guar_1[u_1/u, en(*_1)/en(*)] \\
 & \forall k : (\eta_k, s_k.\sigma, tr(\sigma|_k)) \models guar_2[u_2/u, en(*_2)/en(*)] \\
\equiv & \quad \text{\% Definition of } \otimes, \eta_k \text{ and definition of ready sets} \\
 & \forall k : (\eta_k, s_k.\sigma, tr(\sigma|_k)) \models guar_1[u_1/u, en(*_1)/en(*)] \\
 & \forall k : (\eta_k, s_k.\sigma, tr(\sigma|_k)) \models guar_2[u_2/u, en(*_2)/en(*)] \\
 & \forall k : (\eta_k, s_k.\sigma, tr(\sigma|_k)) \models Merge \\
\Rightarrow & \quad \text{\% Conjunction and Premise :} \\
 & \quad Merge \wedge guar_1[u_1/u, en(*_1)/en(*)] \wedge guar_2[u_2/u, en(*_2)/en(*)] \\
 & \quad \Rightarrow guar \\
 & \forall k : (\eta_k, s_k.\sigma, tr(\sigma|_k)) \models guar \\
\equiv & \quad \text{\% Definition of } \eta_k, guar \text{ is over } \beta \\
 & \forall k : (P_k.\sigma, s_k.\sigma, tr(\sigma|_k)) \models guar \\
\equiv & \quad \text{\% Definition of } GUAR, \mathbf{SC} \\
 & \sigma \in \mathbf{SC}
\end{array}
$$

We postpone the proof of semantic premise (*iii*) and consider intermediate results. First, by definition of $\otimes$, we have:

$$[_{\beta_1 \otimes \beta_2} : \sigma_1 \in CONV \wedge \sigma_2 \in CONV \Rightarrow \sigma \in CONV]$$

and

$$P_k.\sigma = \Xi \Rightarrow P_k.\sigma_1 = \Xi \wedge P_k.\sigma_2 = \Xi$$

The proof of the former proceeds by contradiction: if $\sigma \in DIV$ then $\sigma_1 \in DIV$ or $\sigma_2 \in DIV$. Using these preliminary results, semantic premise (*iii*) is now proved:

$$
\begin{array}{ll}
 & \sigma \in \mathbf{SA} \wedge \sigma_1 \in \mathbf{OC}_1 \wedge \sigma_2 \in \mathbf{OC}_2 \\
\Rightarrow & \quad \text{\% Definition of } \mathbf{OC}_1, \mathbf{OC}_2, POST_1, POST_2 \\
 & \sigma_1 \in CONV \wedge \sigma_2 \in CONV \\
 & \forall k : P_k.\sigma_1 = \Xi \Rightarrow (s_k.\sigma_1, tr(\sigma_1|_k)) \models post_1 \\
 & \forall k : P_k.\sigma_2 = \Xi \Rightarrow (s_k.\sigma_2, tr(\sigma_2|_k)) \models post_2 \\
\equiv & \quad \text{\% Definition of } \otimes \\
 & \sigma_1 \in CONV \wedge \sigma_2 \in CONV \\
 & \forall k : P_k.\sigma_1 = \Xi \Rightarrow (s_k.\sigma, tr(\sigma|_k)) \models post_1 \\
 & \forall k : P_k.\sigma_2 = \Xi \Rightarrow (s_k.\sigma, tr(\sigma|_k)) \models post_2 \\
\Rightarrow & \quad \text{\% Preliminary results} \\
 & \sigma \in CONV \\
 & \forall k : P_k.\sigma = \Xi \Rightarrow (s_k.\sigma, tr(\sigma|_k)) \models (post_1 \wedge post_2) \\
\Rightarrow & \quad \text{\% Premise } post_1 \wedge post_2 \Rightarrow post \\
 & \sigma \in CONV \\
 & \forall k : P_k.\sigma = \Xi \Rightarrow (s_k.\sigma, tr(\sigma|_k)) \models post \\
\equiv & \quad \text{\% Definition of } POST, \mathbf{OC} \\
 & \sigma \in \mathbf{OC}
\end{array}
$$

□

6 Discussion

The semantic rule is based on the interpretation $\mathbf{SA} @\!\!\underset{\rightarrow}{\_} \mathbf{C}$ of assumption-commitment specifications: the commitments are required to hold when the assumptions hold ($\mathbf{SA} \longrightarrow \mathbf{C}$) and moreover, whenever the assumptions hold at step k of a computation, the safety commitments are required to hold at step $k + 1$ ($\mathbf{SA}$ @ $\mathbf{SC}$). This interpretation is classical in (synchronous) message-based concurrency [18, 21, 31] but is less usual in state-based concurrency. In the latter case, only the part $\mathbf{SA} \longrightarrow \mathbf{C}$ is retained [2, 14, 26, 27, 30]. However, in state-based concurrency, $\mathbf{SA} @\!\!\underset{\rightarrow}{\_} \mathbf{C}$ and $\mathbf{SA} \longrightarrow \mathbf{C}$ often admit the same set of implementations (Theorem 3).

This work has been influenced by Abadi and Lamport's previous work [2] on composing assumption-commitment specifications at the semantic level. The composition rule of [2] is based on the interpretation $\mathbf{SA} \longrightarrow \mathbf{C}$; it certainly covers the specifications of state-based processes in Sect. 4 but its additional hypotheses do *not* hold for the specifications of message-based processes in Sect. 5. In their subsequent work [3], Abadi and Lamport have proposed a new rule, based on the interpretation $\mathbf{SA} @\!\!\underset{\rightarrow}{\_} \mathbf{C}$ where $\mathbf{SC}$ is the smallest safety set greater than $\mathbf{OC} \cap \mathbf{SC}$. In order to obtain the latter from our semantic rule, we first observe that, in their TLA approach, composition is conjunction. Semantically, it means that the merging operator ${}_{\beta_1}\!\otimes_{\beta_2} (\sigma_1, \sigma_2, \sigma)$ can be defined as $\sigma = \sigma_1 = \sigma_2$. Consequently, the premises of Rule (3) become:

$$
\begin{array}{l}
\mathbf{SA} \cap \mathbf{SC}_1 \cap \mathbf{SC}_2 \subseteq \mathbf{SA}_1 \cap \mathbf{SA}_2 \\
\mathbf{SC}_1 \cap \mathbf{SC}_2 \subseteq \mathbf{SC} \\
\mathbf{SA} \cap \mathbf{OC}_1 \cap \mathbf{OC}_2 \subseteq \mathbf{OC}
\end{array}
$$

Then, we observe (see last proof step of Theorem 1 in the appendix) that the second premise above can be replaced with $\mathbf{SA}_+ \cap \mathbf{SC}_1 \cap \mathbf{SC}_2 \subseteq \mathbf{SC}$ where $\mathbf{SA}_+$ [3] captures the 'one step delay': $\sigma|_k \in \mathbf{SA}_+ \equiv \sigma|_{k-1} \in \mathbf{SA}$.

The choice of a more abstract model (there are many instances of computations and many corresponding instances for $\otimes$ that match the loose definitions in Sect. 2) has allowed us to derive syntactic parallel composition rules for specifications of both state-based and message-based processes. As highlighted by the the proofs carried out in Sects. 4 and 5, the transformation of the semantic operator $\otimes$ into disjunction or conjunction is due to the nature of the *rely* and *guar* conditions and to the observation that state transitions are interleaved (leading to disjunction) whereas communications are conjoined (leading to conjunction).

Our goal was to unify the syntactic rules presented in Sects. 4 and 5; the development of more general semantic rules that cover other styles of assumption-commitment still requires further work. Nevertheless, we believe that other instances of our semantic rule can be derived. In particular, the commitment may include (liveness) temporal formulas like $\Box(P \Rightarrow \Diamond Q)$; examples are given in [1, 11]. Previous work by Pandya and Joseph in message-based concurrency [19, 21] indicates that *asynchronous* channels might be incorporated at a reasonable cost: configurations record the sequence of buffered messages and specifications distinguish between traces of sent messages and traces of received ones. Another possible extension of this work is the comparison of this semantic rule with rules for assumption-commitment specifications [28] of stream processing functions [9]. Indeed, stream processing functions define traces (that are 'sequences') and the composition of functions corresponds to operations on traces (instances of $\otimes$).

Although it is sufficient for our purpose, a main restriction of the semantic rule in Sect. 2 is that it applies to safety assumptions only. Other rules have been devised to cope with liveness in the assumptions [10, 20, 22, 25]. In [20, 22], the mutual dependency problem is solved by the explicit construction of an ordering between assumption-commitment specifications; the premises then correspond to a proof by induction on that ordering. In [10, 25] the mutual dependency problem is solved by defining an acyclicity condition on the assumptions and commitments. However, the exact relation between rules with and without liveness assumptions is still unclear. Nevertheless, Pandya [20] has shown that Misra-Chandy's rule for safety assumptions [18] can be derived from his rule.

7 Summary

This paper has highlighted the relation between the parallel rules for $(pre, rely, wait, guar, post)$ and $(pre, rely, guar, post)$ specifications of state-based and message-based processes respectively. It has been shown that both are instances of the same semantic rule. The latter is based on an abstract definition of computations and on the existence of a merging operator $\otimes$ that relates the computations of parallel processes. The transformation of semantic rules into syntactic rules proceeds by first providing concrete definitions for computations and the operator $\otimes$ and then showing that the semantic premises follow from the syntactic ones.

Acknowledgement. We are very grateful to Willem-Paul de Roever who suggested the topic of this paper, encouraged our cooperation in investigating it together and made many comments on prehistoric versions. Special thanks go to Ketil Stølen and Qiwen Xu for the fruitful discussion we had during visits in Munich and Kiel respectively. Thanks are also due to Michel Sintzoff for his suggestions. Last but not least we thank all referees for their useful comments.

References

1. Abadi, M., Lamport, L.: An old-fashioned recipe for real time. In: de Bakker, J.W., Huizing, C., de Roever, W.-P., Rozenberg, G. (eds.) Real time: theory in practice. (Lect. Notes Comput. Sci., vol. 600, pp 1-27) Springer-Verlag 1992
2. Abadi, M., Lamport, L.: Composing specifications. ACM Trans. on Prog. Lang. and Syst. **15** (1), 73-132 (1993)
3. Abadi, M., Lamport, L.: Conjoining specifications, Digital Equipment Corporation Systems Research Center, Research Report 118, 1993.
4. Abadi, M., Plotkin, G.D.: A logical view of composition. TCS **114** 3-30 (1993)
5. Aczel, P.: On an inference rule for parallel composition. Unpublished, University of Manchester 1983
6. Barringer, H., Kuiper, R., Pnueli, A.: Now you may compose temporal logic specifications. In: Proc. 16th ACM Symposium on Theory of Computing, pp. 51-63 1984
7. Barringer, H., Kuiper, R., Pnueli, A.: A compositional temporal approach to a CSP-like language. In: Neuhold, E.J., Chroust, G. (eds.) Formal Models of Programming. Elsevier 1985
8. Brookes, S.D., Hoare, C.A.R., Roscoe, A.W.: A theory of communicating sequential processes. J. ACM **31**, 560-599 (1984)
9. Broy, M.: Functional specification of time-sensitive communicating systems. ACM Trans. on Soft. Eng. and Meth. **2** (1), 1-46 (1993)
10. Cau, A., Kuiper, R., de Roever, W.-P.: Formalizing Dijkstra's development strategy within Stark's formalism. In: Jones, C.B., Shaw, R.C. Denvir, T. (eds.) Proc. 5th BCS-FACS Refinement Workshop, pp 4-42. Springer-Verlag 1992
11. Collette, P.: Application of the composition principle to UNITY-like specifications. In: Gaudel, M.-C., Jouannaud, J.-P., (eds.) Proc. TAPSOFT '93. (Lect. Notes Comput. Sci., vol 668, pp 230-242) Springer-Verlag 1993
12. Collette,P.: An explanatory presentation of composition rules for assumption-commitment specifications. Inf. Proc. Letters **50**, 31-35 (1994)
13. Collette, P.: Design of compositional proof systems based on assumption-commitment specifications – Application to UNITY. Ph.D. Thesis, Université Catholique de Louvain, 1994.
14. Grønning, P., Nielsen, T.Q., Lovengreen, H.H.: Refinement and composition of transition-based rely-guarantee specifications with auxiliary variables. In: Veni Madhavan, C.E., Nori, K.V. (eds.) Proc. Foundations of Software Technology and Theoretical Computer Science. (Lect. Notes Comput. Sci., vol 472, pp 332-348) Springer-Verlag 1991
15. Hennessy, M.: The semantics of programming languages. Wiley 1990
16. Jones, C.B.: Development methods for computer programs including a notion of interference. Ph.D. Thesis, Oxford University, 1981.
17. Jones, C.B.: Tentative steps towards a development method for interfering programs. ACM Trans. on Prog. Lang. and Syst. **5** (4), 596-619 (1983)
18. Misra, J., Chandy, K.M.: Proofs of networks of processes. IEEE Trans. on Soft. Eng., **7** (4), 417-426 (1981)
19. Pandya, P.K.: Compositional verification of distributed programs. Ph.D. Thesis, University of Bombay 1988.
20. Pandya, P.K.: Some comments on the assumption-commitment framework for compositional verification of distributed programs. In: de Bakker, J.W., de Roever, W.-P., Rozenberg, G. (eds.) Stepwise refinement of distributed systems. (Lect. Notes Comput. Sci., vol 430, pp 622-640) Springer-Verlag 1990
21. Pandya, P.K., Joseph, M.: P–A logic - a compositional proof system for distributed programs. Distrib. Comp. **5**, 37-54 (1991)
22. Pnueli, A: In transition from global to modular temporal reasoning about programs. In: Apt, K.R. (ed.) Logics and models of concurrent systems. (NATO ASI Series, vol F 13, pp 123-144) Springer-Verlag 1985
23. de Roever, W.-P.: The quest for compositionality - a survey of assertion based proof systems for concurrent programs, Part I. In Proc. of IFIP Working Conference: The Role of Abstract Models in Computer Science, North-Holland, 1985.
24. de Roever, W.-P., Hooman, J., de Boer, F., Lakhneche, Y., Xu, Q., Pandya, P.: State-based proof theory of concurrency: from noncompositional to compositional methods. Book manuscript, 350 pages, Christian-Albrechts-Universität zu Kiel, Germany, 1994

25. Stark, E.W.: A proof technique for rely/guarantee properties. In: Proc. Foundations of Software Technology and Theoretical Computer Science. (Lect. Notes Comput. Sci., vol 206, pp 369-391) Springer-Verlag 1985
26. Stirling, C.: A generalization of Owicki-Gries Hoare logic for a concurrent while language. TCS **58**, 347-359 (1988)
27. Stølen, K.: A method for the development of totally correct shared-state parallel programs. In: Baeten, J.C.M., Groote, J.F., (eds.), Proc. Concur '91. (Lect. Notes Comput. Sci., vol 527, pp 510-525.) Springer-Verlag 1991
28. Stølen, K., Dederichs, F., Weber, R.: Assumption/commitment rules for networks of asynchronously communicating agents. Technical Report SFB 342/2/93, Technical University of Munich 1993
29. Woodcock, J.C.P., Dickinson, B.: Using VDM with rely and guarantee-conditions, in Bloomfield, R., Marshall, L., Jones, R. (eds.). Proc. VDM '88, The Way Ahead. (Lect. Notes Comput. Sci., vol 328, pp 434-458.) Springer-Verlag 1988
30. Xu, Q., He, J.: A theory of state-based parallel programming: Part I. In Morris, J. (ed.) Proc. 4th BCS-FACS Refinement Workshop, pp 326-359. Springer-Verlag 1991
31. Zwiers, J., de Bruin, A., de Roever, W.-P.: A proof system for partial correctness of dynamic networks of processes. In: Proc. Conference on Logics of Programs 1983. (Lect. Notes Comput. Sci., vol 164, pp 513-527) Springer-Verlag 1984
32. Zwiers, J., de Roever, W.-P., van Emde Boas,P.: Compositionality and concurrent networks: soundness and completeness of a proof system. Technical Report 57, University of Nijmegen 1984.
33. Zwiers, J.: Compositionality, concurrency and partial correctness. (Lect. Notes Comput. Sci., vol 321) Springer-Verlag 1989

A Soundness proof of the semantic composition rule

We first prove the basic rule:

$$\frac{[\beta_1 \otimes \beta_2 : \sigma \in \mathbf{SA} \wedge \sigma_1 \in \mathbf{SC}_1 \wedge \sigma_2 \in \mathbf{SC}_2 \Rightarrow \sigma_1 \in \mathbf{SA}_1 \wedge \sigma_2 \in \mathbf{SA}_2]}{[\beta_1 \otimes \beta_2 : \sigma \in \mathbf{SA} \wedge \sigma_1 \in (\mathbf{SA}_1 \ @\ \mathbf{SC}_1) \wedge \sigma_2 \in (\mathbf{SA}_2 \ @\ \mathbf{SC}_2) \Rightarrow \sigma_1 \in \mathbf{SA}_1 \wedge \sigma_2 \in \mathbf{SA}_2]}$$

Assume $\sigma \in \mathbf{SA} \wedge \sigma_1 \in (\mathbf{SA}_1 \ @\ \mathbf{SC}_1) \wedge \sigma_2 \in (\mathbf{SA}_2 \ @\ \mathbf{SC}_2)$. Since $\mathbf{SA}_1$ and $\mathbf{SA}_2$ are safety sets, we may prove $\sigma_1|_k \in \mathbf{SA}_1 \wedge \sigma_2|_k \in \mathbf{SA}_2$ by induction on k. First, let $k = 0$:

$$\begin{array}{ll}
 & \sigma \in \mathbf{SA} \wedge \sigma_1 \in (\mathbf{SA}_1 \ @\ \mathbf{SC}_1) \wedge \sigma_2 \in (\mathbf{SA}_2 \ @\ \mathbf{SC}_2) \\
\Rightarrow & \quad \% \ \ \mathbf{SA} \text{ is a safety set and definition of } @ \\
 & \sigma|_0 \in \mathbf{SA} \wedge \sigma_1|_0 \in \mathbf{SC}_1 \wedge \sigma_2|_0 \in \mathbf{SC}_2 \\
\Rightarrow & \quad \% \ \ \text{Premise and } \beta_1 \otimes \beta_2 \, (\sigma_1, \sigma_2, \sigma) \Rightarrow \beta_1 \otimes \beta_2 \, (\sigma_1|_0, \sigma_2|_0, \sigma|_0) \\
 & \sigma_1|_0 \in \mathbf{SA}_1 \wedge \sigma_2|_0 \in \mathbf{SA}_2
\end{array}$$

Then, let $k > 0$:

$$\begin{array}{ll}
 & \sigma \in \mathbf{SA} \wedge \sigma_1 \in (\mathbf{SA}_1 \ @\ \mathbf{SC}_1) \wedge \sigma_2 \in (\mathbf{SA}_2 \ @\ \mathbf{SC}_2) \\
\Rightarrow & \quad \% \ \ \mathbf{SA} \text{ is a safety set} \\
 & \sigma|_k \in \mathbf{SA} \wedge \sigma_1 \in (\mathbf{SA}_1 \ @\ \mathbf{SC}_1) \wedge \sigma_2 \in (\mathbf{SA}_2 \ @\ \mathbf{SC}_2) \\
\Rightarrow & \quad \% \ \ \text{Induction Hypothesis: } \sigma_1|_{k-1} \in \mathbf{SA}_1 \wedge \sigma_2|_{k-1} \in \mathbf{SA}_2 \\
 & \quad \text{Definition of } @ \\
 & \sigma|_k \in \mathbf{SA} \wedge \sigma_1|_k \in \mathbf{SC}_1 \wedge \sigma_2|_k \in \mathbf{SC}_2 \\
\Rightarrow & \quad \% \ \ \text{Premise and } \beta_1 \otimes \beta_2 \, (\sigma_1, \sigma_2, \sigma) \Rightarrow \beta_1 \otimes \beta_2 \, (\sigma_1|_k, \sigma_2|_k, \sigma|_k) \\
 & \sigma_1|_k \in \mathbf{SA}_1 \wedge \sigma_2|_k \in \mathbf{SA}_2
\end{array}$$

In fact, proving Rule (3) amounts to proving:

$$\frac{\begin{array}{ll}(i) & [_{\beta_1}\otimes_{\beta_2} : \sigma \in \mathbf{SA} \wedge \sigma_1 \in \mathbf{SC}_1 \wedge \sigma_2 \in \mathbf{SC}_2 \Rightarrow \sigma_1 \in \mathbf{SA}_1 \wedge \sigma_2 \in \mathbf{SA}_2] \\ (ii) & [_{\beta_1}\otimes_{\beta_2} : \sigma_1 \in \mathbf{SC}_1 \wedge \sigma_2 \in \mathbf{SC}_2 \Rightarrow \sigma \in \mathbf{SC}] \\ (iii) & [_{\beta_1}\otimes_{\beta_2} : \sigma \in \mathbf{SA} \wedge \sigma_1 \in \mathbf{OC}_1 \wedge \sigma_2 \in \mathbf{OC}_2 \Rightarrow \sigma \in \mathbf{OC}]\end{array}}{[_{\beta_1}\otimes_{\beta_2} : \sigma_1 \in (\mathbf{SA}_1 @_{\rightarrow} \mathbf{C}_1) \wedge \sigma_2 \in (\mathbf{SA}_2 @_{\rightarrow} \mathbf{C}_2) \Rightarrow \sigma \in (\mathbf{SA} @_{\rightarrow} \mathbf{C})]}$$

By definition, $\mathbf{SA} @_{\rightarrow} \mathbf{C} = (\mathbf{SA} \rightarrow \mathbf{C}) \cap (\mathbf{SA} \ @ \ \mathbf{SC})$. We first consider the proof of $\mathbf{SA} \rightarrow \mathbf{C}$, i.e. the proof of $\sigma \in \mathbf{SC} \wedge \sigma \in \mathbf{OC}$ from $\sigma \in \mathbf{SA}$:

$$\begin{array}{ll}
 & \sigma \in \mathbf{SA} \wedge \sigma_1 \in (\mathbf{SA}_1 @_{\rightarrow} \mathbf{C}_1) \wedge \sigma_2 \in (\mathbf{SA}_2 @_{\rightarrow} \mathbf{C}_2) \\
\equiv & \quad \% \text{ Definition of } @_{\rightarrow} \\
 & \sigma \in \mathbf{SA} \wedge \sigma_1 \in (\mathbf{SA}_1 \ @ \ \mathbf{SC}_1) \wedge \sigma_2 \in (\mathbf{SA}_2 \ @ \ \mathbf{SC}_2) \\
 & \sigma_1 \in (\mathbf{SA}_1 \rightarrow \mathbf{C}_1) \wedge \sigma_2 \in (\mathbf{SA}_2 \rightarrow \mathbf{C}_2) \\
\Rightarrow & \quad \% \text{ Basic rule above} \\
 & \sigma \in \mathbf{SA} \wedge \sigma_1 \in \mathbf{SA}_1 \wedge \sigma_2 \in \mathbf{SA}_2 \wedge \sigma_1 \in (\mathbf{SA}_1 \rightarrow \mathbf{C}_1) \wedge \sigma_2 \in (\mathbf{SA}_2 \rightarrow \mathbf{C}_2) \\
\Rightarrow & \quad \% \text{ Definition of } \rightarrow \\
 & \sigma \in \mathbf{SA} \wedge \sigma_1 \in \mathbf{SC}_1 \wedge \sigma_1 \in \mathbf{OC}_1 \wedge \sigma_2 \in \mathbf{SC}_2 \wedge \sigma_2 \in \mathbf{OC}_2 \\
\Rightarrow & \quad \% \text{ Premises } (ii), (iii) \\
 & \sigma \in \mathbf{SC} \wedge \sigma \in \mathbf{OC}
\end{array}$$

We now consider the proof of $\sigma \in (\mathbf{SA} \ @ \ \mathbf{SC})$. First, we prove $\sigma|_0 \in \mathbf{SC}$:

$$\begin{array}{ll}
 & \sigma_1 \in (\mathbf{SA}_1 \ @ \ \mathbf{SC}_1) \wedge \sigma_2 \in (\mathbf{SA}_2 \ @ \ \mathbf{SC}_2) \\
\Rightarrow & \quad \% \text{ definition of } @ \\
 & \sigma_1|_0 \in \mathbf{SC}_1 \wedge \sigma_2|_0 \in \mathbf{SC}_2 \\
\Rightarrow & \quad \% \text{ Premise } (ii) \text{ and } _{\beta_1}\otimes_{\beta_2} (\sigma_1, \sigma_2, \sigma) \Rightarrow {}_{\beta_1}\otimes_{\beta_2} (\sigma_1|_0, \sigma_2|_0, \sigma|_0) \\
 & \sigma|_0 \in \mathbf{SC}
\end{array}$$

Then, let $k > 0$. We assume $\sigma|_{k-1} \in \mathbf{SA}$ and prove $\sigma|_k \in \mathbf{SC}$:

$$\begin{array}{ll}
 & \sigma|_{k-1} \in \mathbf{SA} \wedge \sigma_1 \in (\mathbf{SA}_1 \ @ \ \mathbf{SC}_1) \wedge \sigma_2 \in (\mathbf{SA}_2 \ @ \ \mathbf{SC}_2) \\
\Rightarrow & \quad \% \ \mathbf{SA}_i \ @ \ \mathbf{SC}_i \text{ is a safety property} \\
 & \sigma|_{k-1} \in \mathbf{SA} \wedge \sigma_1|_{k-1} \in (\mathbf{SA}_1 \ @ \ \mathbf{SC}_1) \wedge \sigma_2|_{k-1} \in (\mathbf{SA}_2 \ @ \ \mathbf{SC}_2) \\
 & \sigma_1 \in (\mathbf{SA}_1 \ @ \ \mathbf{SC}_1) \wedge \sigma_2 \in (\mathbf{SA}_2 \ @ \ \mathbf{SC}_2) \\
\Rightarrow & \quad \% \text{ Basic rule and } _{\beta_1}\otimes_{\beta_2} (\sigma_1, \sigma_2, \sigma) \Rightarrow {}_{\beta_1}\otimes_{\beta_2} (\sigma_1|_{k-1}, \sigma_2|_{k-1}, \sigma|_{k-1}) \\
 & \sigma|_{k-1} \in \mathbf{SA} \wedge \sigma_1|_{k-1} \in \mathbf{SA}_1 \wedge \sigma_2|_{k-1} \in \mathbf{SA}_2 \\
 & \sigma_1 \in (\mathbf{SA}_1 \ @ \ \mathbf{SC}_1) \wedge \sigma_2 \in (\mathbf{SA}_2 \ @ \ \mathbf{SC}_2) \\
\Rightarrow & \quad \% \text{ Definition of } @ \\
 & \sigma|_{k-1} \in \mathbf{SA} \wedge \sigma_1|_k \in \mathbf{SC}_1 \wedge \sigma_2|_k \in \mathbf{SC}_2 \\
\Rightarrow & \quad \% \text{ Premise } (ii) \text{ and } _{\beta_1}\otimes_{\beta_2} (\sigma_1, \sigma_2, \sigma) \Rightarrow {}_{\beta_1}\otimes_{\beta_2} (\sigma_1|_k, \sigma_2|_k, \sigma|_k) \\
 & \sigma|_k \in \mathbf{SC}
\end{array}$$

This article was processed by the author using the LaTeX style file *pljour1* from Springer-Verlag.