A Compositional Specification Theory for Component Behaviours

Taolue Chen[1], Chris Chilton[1], Bengt Jonsson[2], and Marta Kwiatkowska[1]

[1] Department of Computer Science, University of Oxford, UK
[2] Department of Information Technology, Uppsala University, Sweden

Abstract. We propose a compositional specification theory for reasoning about components that interact by synchronisation of input and output (I/O) actions, in which the specification of a component constrains the temporal ordering of interactions with the environment. Such a theory is motivated by the need to support composability of components, in addition to modelling environmental assumptions, and reasoning about run-time behaviour. Models can be specified operationally by means of I/O labelled transition systems augmented by an inconsistency predicate on states, or in a purely declarative manner by means of traces. We introduce a refinement preorder that supports safe-substitutivity of components. Our specification theory includes the operations of parallel composition for composing components at run-time, logical conjunction for independent development, and quotient for incremental development. We prove congruence properties of the operations and show correspondence between the operational and declarative frameworks.

Keywords: specification theory, compositionality, components, I/O automata, interface automata, logic LTS, refinement, conjunction, quotient.

1 Introduction

An important paradigm for developing complex reactive systems is component-based design, where systems are composed from components, which themselves can be realised by smaller components. Component-based design can be supported by a specification theory, which allows the mixing of specifications and implementations, admits refinement, and provides composition operators. A specification theory suitable for components should be equipped with a *refinement* preorder which is substitutive, to facilitate component reuse. As a minimum, the composition operators should include structural *parallel composition*, for inferring component interactions at run-time; *conjunction*, to facilitate independent development constrained by several specifications; and *quotienting*, which supports incremental development in the following sense: given a specification of the full system, together with components implementing part of that system, quotienting allows one to find the coarsest specification of the remaining portion of the system to be implemented. Further useful operators include: *disjunction*, which finds a common specification that a collection of components implement; and *hiding*, which supports abstraction of components.

In this paper, we consider systems of components that interact by synchronisation of input and output actions, in which outputs are non-blocking. A specification should describe properties on the ordering of a component's interactions with its environment; it should also describe the assumptions on the environment under which these properties are guaranteed, thereby supporting assume-guarantee reasoning. A number of proposals for such specification theories have been put forward. As detailed in the survey of related work below, we find that they suffer from limitations or unnecessary complications.

The main contribution of our paper is a comprehensive, compositional specification theory for components that generalises existing frameworks by supporting all the above-mentioned operators, while retaining conceptual simplicity and strong algebraic properties of the operations. The framework permits the mixing of abstract component specifications and I/O labelled transition systems (called Logic IOLTSs), without restricting to determinism as in [1]. Our refinement is based on traces, hence admitting a simpler formulation than similar notions based on e.g. alternating simulation [2] or modalities [3,4], so is more amenable to language-theoretic constructs. From this formulation, we demonstrate that the induced mutual refinement is a congruence for the operators.

In contrast to existing I/O automata [5] and interface automata [2], we are able to express: (1) *assumptions* on the input provided by the environment; (2) *underspecification*, meaning that it is uncertain what the allowable interactions are; and (3) various (run-time) *errors*, including communication mismatch and bad behaviour. We show that all these features can be expressed using only the single concept of *inconsistency*, which we have adapted from the Logic LTSs of Lüttgen and Vogler [6,7], where input and output actions are not distinguished. Inconsistency is a property of states or interaction traces, which represents the possibility of some abnormal condition. Once an inconsistency has occurred, there is no escaping from it. Following the lead of CSP [8], we thus allow for chaotic behaviour to ensue once an inconsistency has arisen.

Related work. Our specification theory, in particular Logic IOLTSs, is inspired by the Logic LTS framework due to Lüttgen and Vogler [6], a compositional theory that admits as specifications LTSs without I/O distinctions. Their inconsistency predicate is induced from inequality of ready-sets, rather than communication mismatches as in our case. Refinement is based on ready-simulation; alphabetised parallel and conjunction are considered, but not quotient.

The operational component model in our framework has been greatly influenced by I/O automata [5] and interface automata [2]: both are based on I/O LTSs, with the proviso that I/O automata must be input-enabled, meaning that each state of the automaton is willing to accept any input. We differ from I/O automata by not imposing input-enabledness and from interface automata by working with an explicit representation of inconsistencies. Another difference is refinement, which for interface automata is defined in terms of alternating simulation, rather than traces; the original definition in [2] is simplified in [9], but works only for input-deterministic interface automata. It should be noted that, unlike [2,9], we use an associative variant of parallel composition, which combines an input and output into an output (as in [10]). Furthermore, we provide a definition of conjunction corresponding to shared refinement of interface automata, which substantially generalises that of [11] for synchronous components. Moreover, our quotienting operator on Logic IOLTSs generalises that in [12] defined only for deterministic components.

There are a number of process-algebraic frameworks that deal with asynchronous I/O interaction. We mention a characterisation of I/O automata by De Nicola and Segala [13], which is actually a generalisation (and also applicable to interface automata), since the inconsistent process Ω allows to distinguish between good and bad inputs. Similarly to our approach, refinement in [13] is given by trace containment, but does not extend to inconsistent trace containment. This is because we allow a Logic IOLTS to become inconsistent after emitting an output, whereas a process can only become inconsistent through receiving a bad input. Finally, we remark that [13] supports a number of operators of a specification theory, but does not deal with conjunction or quotient.

Our work is also related to the ioco theory in model based testing [14]. The ioco relation is similar to our refinement, but lacks compositionality of operators, so is not well-suited to a specification theory for components.

There have been several CSP-based frameworks that deal with asynchronous communication; of these, the receptive process theory (RPT) [15] utilises a model of concurrency similar to ours in that outputs are non-blocking. RPT also considers quotient (referred to as factorisation), but for the restricted class of delay-insensitive networks [16] that differ from our setting.

A further class of component-based modelling formalisms is based on may/must modalities. A specification theory for components has been devised in [17] based on modalities [3,4], but the definition of quotient is more restrictive than ours. Larsen *et al.* have made an effort in relating modal transition systems with interface automata [1]. The approach of modal I/O automata is based on a game-like definition of refinement, which we claim to be more complex than ours, see, e.g., the discussion of parallel composition in [4]. The framework in [4] can support reasoning about liveness properties which our framework does not (although they both support reasoning of safety properties). However, our framework can be easily extended by introducing quiescent states, and additionally considering containment of quiescent traces to reason about liveness.

Outline. The paper begins by introducing declarative specifications in Section 2, before considering operational specifications in Section 3. We focus on three composition operators: parallel, conjunction and quotient; omitting disjunction and hiding for reasons of space. The paper ends with a statement of full-abstraction results in Section 4. Proofs can be found in the accompanying technical report [18].

2 A Declarative Theory of Components

In this section, we model components abstractly by means of declarative specifications. We introduce a substitutive refinement preorder together with three compositional operators on declarative specifications.

A declarative specification comes equipped with an interface, together with a set of behaviours over the interface. The interface is represented by a set of input actions and a set of output actions, which are necessarily disjoint, while the behaviour is characterised by traces.

Definition 1 (Declarative specification). *A* declarative specification $\mathcal{P}$ *is a tuple* $\langle \mathcal{A}^I_\mathcal{P}, \mathcal{A}^O_\mathcal{P}, T_\mathcal{P}, F_\mathcal{P} \rangle$ *in which* $\mathcal{A}^I_\mathcal{P}$ *and* $\mathcal{A}^O_\mathcal{P}$ *are disjoint sets referred to as inputs and outputs respectively (the union of which is denoted by* $\mathcal{A}_\mathcal{P}$*),* $T_\mathcal{P} \subseteq \mathcal{A}^*_\mathcal{P}$ *is a non-empty set of permissible traces, and* $F_\mathcal{P} \subseteq \mathcal{A}^*_\mathcal{P}$ *is a set of inconsistent traces. The trace sets must satisfy the constraints:*

1. $F_\mathcal{P} \subseteq T_\mathcal{P}$
2. *If* $t \in T_\mathcal{P}$ *and* $i \in \mathcal{A}^I_\mathcal{P}$*, then* $ti \in T_\mathcal{P}$
3. $T_\mathcal{P}$ *is prefix closed*
4. *If* $t \in F_\mathcal{P}$ *and* $t' \in \mathcal{A}^*_\mathcal{P}$*, then* $tt' \in F_\mathcal{P}$*.*

Outputs are under the control of the component, whereas inputs are issued by the environment. This means that, after any successful interaction between the component and the environment, the environment can issue any input i, even if it will be refused by the component. Naturally, if i is refused by the component after the trace t, we deem ti to be an inconsistent trace, since a communication mismatch has occurred. Given this treatment of inputs, we say that our theory is *not* input-enabled, even though $T_\mathcal{P}$ is closed under input-extensions.

Example 1. A drinks machine dispenses either a tea or a coffee after a coin has been inserted. The drinks machine has sufficient water to produce only 2 drinks, after which a further coin insertion renders the machine inoperable. This behaviour can be encoded by the declarative specification $DM = \langle \{\pounds\}, \{t, c\}, T, F_1 \cup F_2 \rangle$, where:

- $T = \{\epsilon, \pounds, \pounds(c+t), \pounds(c+t)\pounds, \pounds(c+t)\pounds(c+t)\} \cup F_1 \cup F_2$
- $F_1 = \pounds(c+t)\pounds(c+t)\pounds(\pounds+c+t)^*$ insertion of third coin after two dispensations
- $F_2 = (\epsilon + \pounds(c+t))\pounds\pounds(\pounds+c+t)^*$ insertion of second coin before dispensation.

From hereon let $\mathcal{P}$, $\mathcal{Q}$ and $\mathcal{R}$ be declarative specifications with signatures $\langle \mathcal{A}^I_\mathcal{P}, \mathcal{A}^O_\mathcal{P}, T_\mathcal{P}, F_\mathcal{P} \rangle$, $\langle \mathcal{A}^I_\mathcal{Q}, \mathcal{A}^O_\mathcal{Q}, T_\mathcal{Q}, F_\mathcal{Q} \rangle$ and $\langle \mathcal{A}^I_\mathcal{R}, \mathcal{A}^O_\mathcal{R}, T_\mathcal{R}, F_\mathcal{R} \rangle$ respectively.

2.1 Refinement

As refinement corresponds to safe substitutivity, for $\mathcal{Q}$ to be used in place of $\mathcal{P}$ we require that $\mathcal{Q}$ must exist safely in *any* environment that $\mathcal{P}$ can exist in safely.

Whether an environment is safe for a specification depends on the sequences of message exchanges afforded by the component. If an environment can prevent a component from performing an inconsistent trace, then the environment is said to be safe.

We do not insist that a component $\mathcal{Q}$ must have the same interface as the component $\mathcal{P}$ to be refined. Instead $\mathcal{Q}$ must be accepting of at least all of $\mathcal{P}$'s inputs, while restricting to a subset of $\mathcal{P}$'s outputs. This can be formalised by the covariant relationship $\mathcal{A}^I_\mathcal{P} \subseteq \mathcal{A}^I_\mathcal{Q}$ on inputs and the contravariant constraint $\mathcal{A}^O_\mathcal{Q} \subseteq \mathcal{A}^O_\mathcal{P}$ on outputs.

In order to establish that refinement holds, we perform a weak form of alphabet equalisation on the inputs of the component to be refined. We refer to this operation as lifting. Informally, lifting extends the trace sets of $\mathcal{P}$ by explicitly refusing any input in $\mathcal{A}^I_\mathcal{Q} \setminus \mathcal{A}^I_\mathcal{P}$, after which it allows for arbitrary behaviour.

Definition 2 (Lifting). *Let $\mathcal{P}$ be a declarative specification, and let $\mathcal{A}^I_\mathcal{Q}$ be a set of input actions. The lifting of trace sets $T_\mathcal{P}$ and $F_\mathcal{P}$ to $\mathcal{A}^I_\mathcal{Q}$, written as $T_\mathcal{P} \uparrow \mathcal{A}^I_\mathcal{Q}$ and $F_\mathcal{P} \uparrow \mathcal{A}^I_\mathcal{Q}$ respectively, is defined as:*

- $T_\mathcal{P} \uparrow \mathcal{A}^I_\mathcal{Q} = T_\mathcal{P} \cup \{tit' : t \in T_\mathcal{P},\ i \in \mathcal{A}^I_\mathcal{Q} \setminus \mathcal{A}^I_\mathcal{P} \text{ and } t' \in (\mathcal{A}^I_\mathcal{Q} \cup \mathcal{A}_\mathcal{P})^*\}$
- $F_\mathcal{P} \uparrow \mathcal{A}^I_\mathcal{Q} = F_\mathcal{P} \cup \{tit' : t \in T_\mathcal{P},\ i \in \mathcal{A}^I_\mathcal{Q} \setminus \mathcal{A}^I_\mathcal{P} \text{ and } t' \in (\mathcal{A}^I_\mathcal{Q} \cup \mathcal{A}_\mathcal{P})^*\}$.

Recall that an environment is safe for a component if the environment can prevent the component from performing an inconsistent trace. As outputs are under the control of the component itself, a safe environment must refuse to issue an input on any trace from which there is a sequence of output actions after the input that allows the trace to become inconsistent.

Under such an arrangement, for each declarative specification $\mathcal{P}$ we can define the safe declarative specification $\mathcal{E}(\mathcal{P})$ containing all of $\mathcal{P}$'s permissible and inconsistent traces, but also satisfying the additional property: if $t \in T_\mathcal{P}$ and there exists $t' \in (\mathcal{A}^O_\mathcal{P})^*$ such that $tt' \in F_\mathcal{P}$, then $t \in F_{\mathcal{E}(\mathcal{P})}$. This has the effect of forcing all inconsistent traces to become inconsistent on the environment's issue of a bad input. If the environment respects this safe specification, by not issuing any input that results in an inconsistent trace, then the component can never encounter an inconsistent trace. Note that if $\epsilon \in F_{\mathcal{E}(\mathcal{P})}$ then there is no environment that can prevent $\mathcal{P}$ from performing an inconsistent trace. However, for uniformity we still refer to $\mathcal{E}(\mathcal{P})$ as the safe specification of $\mathcal{P}$.

Definition 3 (Safe specification). *Let $\mathcal{P}$ be a declarative specification. The most general safe specification for $\mathcal{P}$ is a declarative specification $\mathcal{E}(\mathcal{P}) = \langle \mathcal{A}^I_\mathcal{P}, \mathcal{A}^O_\mathcal{P}, T_{\mathcal{E}(\mathcal{P})}, F_{\mathcal{E}(\mathcal{P})} \rangle$, where $T_{\mathcal{E}(\mathcal{P})} = T_\mathcal{P} \cup F_{\mathcal{E}(\mathcal{P})}$ and $F_{\mathcal{E}(\mathcal{P})} = \{tt' \in \mathcal{A}^*_\mathcal{P} : t \in T_\mathcal{P} \text{ and } \exists t'' \in (\mathcal{A}^O_\mathcal{P})^* \cdot tt'' \in F_\mathcal{P}\}$.*

We can now define our substitutive refinement preorder. From the safe specification associated with an arbitrary declarative specification, it is easy to see whether a declarative specification can be substituted safely in place of another. Note that $F_\mathcal{Q} \subseteq F_\mathcal{P} \uparrow \mathcal{A}^I_\mathcal{Q}$ would be too strong to use for the last clause, as we are only interested in trace containment up to the point where an environment can issue a bad input.

Definition 4 (Refinement). *For declarative specifications $\mathcal{P}$ and $\mathcal{Q}$, $\mathcal{Q}$ is said to be a refinement of $\mathcal{P}$, written $\mathcal{Q} \sqsubseteq_{dec} \mathcal{P}$, iff:*

1. $\mathcal{A}^I_\mathcal{P} \subseteq \mathcal{A}^I_\mathcal{Q}$
2. $\mathcal{A}^O_\mathcal{Q} \subseteq \mathcal{A}^O_\mathcal{P}$
3. $T_{\mathcal{E}(\mathcal{Q})} \subseteq T_{\mathcal{E}(\mathcal{P})} \uparrow \mathcal{A}^I_\mathcal{Q}$
4. $F_{\mathcal{E}(\mathcal{Q})} \subseteq F_{\mathcal{E}(\mathcal{P})} \uparrow \mathcal{A}^I_\mathcal{Q}$.

As refinement is based on an extension of language inclusion, its complexity is in P, assuming regularity of the trace sets. Note that lifting maintains regularity.

Equivalence of declarative specifications in our framework is defined in terms of mutual refinement.

Definition 5 (Equivalence). *Let $\mathcal{P}$ and $\mathcal{Q}$ be declarative specifications. Then $\mathcal{P}$ and $\mathcal{Q}$ are said to be equivalent, written $\mathcal{P} \equiv_{dec} \mathcal{Q}$, iff $\mathcal{P} \sqsubseteq_{dec} \mathcal{Q}$ and $\mathcal{Q} \sqsubseteq_{dec} \mathcal{P}$.*

Lemma 1 (Preorder). *Refinement is both reflexive and transitive.*

2.2 Parallel composition

The parallel composition operator on declarative specifications yields a declarative specification representing the combined effect of its operands running asynchronously. We do not consider synchronous parallel composition, as this does not make sense when dealing with non-blocking output actions. To preserve the effect that a single output from a component can be received by multiple components in the environment, we must define the parallel composition to repeatedly broadcast an output: this means that an input $a?$ and output $a!$ combine to form an output $a!$ (as in certain variants of I/O automata), rather than a hidden action τ as is the case in Milner's CCS.

Not all declarative specifications can be composed with one another; we restrict to those that are said to be *composable*. $\mathcal{P}$ and $\mathcal{Q}$ are composable for parallel composition only if $\mathcal{A}^O_\mathcal{P} \cap \mathcal{A}^O_\mathcal{Q} = \emptyset$. This restriction is meaningful if we consider inputs on an interface as buttons and outputs as lights. Given two distinct components, it is not possible for them to share a common light, whereas it is possible to push their buttons at the same time. In practice, issues of composability can be avoided by employing renaming, if this is considered to be appropriate.

Definition 6 (Parallel composition). *Let $\mathcal{P}$ and $\mathcal{Q}$ be declarative specifications such that $\mathcal{A}^O_\mathcal{P}$ and $\mathcal{A}^O_\mathcal{Q}$ are disjoint. Then $\mathcal{P} \mid\mid \mathcal{Q}$ is the declarative specification $\langle \mathcal{A}^I_{\mathcal{P}||\mathcal{Q}}, \mathcal{A}^O_{\mathcal{P}||\mathcal{Q}}, T_{\mathcal{P}||\mathcal{Q}}, F_{\mathcal{P}||\mathcal{Q}} \rangle$, where:*

- $\mathcal{A}^I_{\mathcal{P}||\mathcal{Q}} = (\mathcal{A}^I_\mathcal{P} \cup \mathcal{A}^I_\mathcal{Q}) \setminus (\mathcal{A}^O_\mathcal{P} \cup \mathcal{A}^O_\mathcal{Q})$
- $\mathcal{A}^O_{\mathcal{P}||\mathcal{Q}} = \mathcal{A}^O_\mathcal{P} \cup \mathcal{A}^O_\mathcal{Q}$
- $T_{\mathcal{P}||\mathcal{Q}} = \{t \in \mathcal{A}^*_{\mathcal{P}||\mathcal{Q}} : t \upharpoonright \mathcal{A}_\mathcal{P} \in T_\mathcal{P}$ *and* $t \upharpoonright \mathcal{A}_\mathcal{Q} \in T_\mathcal{Q}\} \cup F_{\mathcal{P}||\mathcal{Q}}$
- $F_{\mathcal{P}||\mathcal{Q}} = \{tt' \in \mathcal{A}^*_{\mathcal{P}||\mathcal{Q}} : t \upharpoonright \mathcal{A}_\mathcal{P} \in F_\mathcal{P}$ *and* $t \upharpoonright \mathcal{A}_\mathcal{Q} \in T_\mathcal{Q}$*, or* $t \upharpoonright \mathcal{A}_\mathcal{P} \in T_\mathcal{P}$ *and* $t \upharpoonright \mathcal{A}_\mathcal{Q} \in F_\mathcal{Q}\}$.

Informally, a trace is permissible in $\mathcal{P} \mid\mid \mathcal{Q}$ if its projection onto $\mathcal{A}_\mathcal{P}$ is a trace of $\mathcal{P}$ and its projection onto $\mathcal{A}_\mathcal{Q}$ is a trace of $\mathcal{Q}$. A trace is inconsistent if it has a prefix whose projection onto the alphabet of one of the components is inconsistent and the projection onto the alphabet of the other component is a permissible trace of that component.

We demonstrate the following result, a corollary of which is that mutual refinement is a congruence for parallel, subject to composability.

Theorem 1 (Compositionality of parallel). *Let $\mathcal{P}$, $\mathcal{Q}$ and $\mathcal{R}$ be declarative specifications such that $\mathcal{P}$ and $\mathcal{R}$ are composable for parallel composition, and $\mathcal{A}_\mathcal{Q} \cap \mathcal{A}_\mathcal{R} \cap \mathcal{A}_{\mathcal{P}||\mathcal{R}} \subseteq \mathcal{A}_\mathcal{P} \cap \mathcal{A}_\mathcal{R}$. If $\mathcal{Q} \sqsubseteq_{dec} \mathcal{P}$, then $\mathcal{Q} \mid\mid \mathcal{R} \sqsubseteq_{dec} \mathcal{P} \mid\mid \mathcal{R}$.*

2.3 Conjunction

The conjunction operator on declarative specifications can be thought of as finding a common implementation for a number of properties, each of which are represented by declarative specifications. Naturally, any implementation of these properties should be a refinement of each of the properties to be implemented. The conjunction (or shared refinement) of two declarative specifications $\mathcal{P}$ and $\mathcal{Q}$ is the *coarsest* declarative specification that refines both $\mathcal{P}$ and $\mathcal{Q}$. Thus conjunction is the meet operator on the refinement preorder.

As for parallel composition, conjunction can only be performed on composable components. $\mathcal{P}$ and $\mathcal{Q}$ are composable for conjunction only if the sets $\mathcal{A}^I_\mathcal{P} \cup \mathcal{A}^I_\mathcal{Q}$ and $\mathcal{A}^O_\mathcal{P} \cup \mathcal{A}^O_\mathcal{Q}$ are disjoint.

Definition 7 (Conjunction). *Let $\mathcal{P}$ and $\mathcal{Q}$ be declarative specifications such that $\mathcal{A}^I_\mathcal{P} \cup \mathcal{A}^I_\mathcal{Q}$ and $\mathcal{A}^O_\mathcal{P} \cup \mathcal{A}^O_\mathcal{Q}$ are disjoint. Then $\mathcal{P} \wedge \mathcal{Q}$ is the declarative specification $\langle \mathcal{A}^I_{\mathcal{P}\wedge\mathcal{Q}}, \mathcal{A}^O_{\mathcal{P}\wedge\mathcal{Q}}, T_{\mathcal{P}\wedge\mathcal{Q}}, F_{\mathcal{P}\wedge\mathcal{Q}} \rangle$, where:*

- $\mathcal{A}^I_{\mathcal{P}\wedge\mathcal{Q}} = \mathcal{A}^I_\mathcal{P} \cup \mathcal{A}^I_\mathcal{Q}$
- $\mathcal{A}^O_{\mathcal{P}\wedge\mathcal{Q}} = \mathcal{A}^O_\mathcal{P} \cap \mathcal{A}^O_\mathcal{Q}$
- $T_{\mathcal{P}\wedge\mathcal{Q}} = T_\mathcal{P} \uparrow \mathcal{A}^I_\mathcal{Q} \cap T_\mathcal{Q} \uparrow \mathcal{A}^I_\mathcal{P}$
- $F_{\mathcal{P}\wedge\mathcal{Q}} = F_\mathcal{P} \uparrow \mathcal{A}^I_\mathcal{Q} \cap F_\mathcal{Q} \uparrow \mathcal{A}^I_\mathcal{P}$.

Conjunction has strong connections with the logical ‘and’ operator in Boolean algebra, as shown below. Mutual refinement is a congruence for conjunction, subject to composability.

Theorem 2 (Properties of conjunction).

- *Conjunction is the greatest lower bound operator for $\sqsubseteq_{dec}$*
- *$\mathcal{R} \sqsubseteq_{dec} \mathcal{P}$ and $\mathcal{R} \sqsubseteq_{dec} \mathcal{Q}$ iff $\mathcal{R} \sqsubseteq_{dec} \mathcal{P} \wedge \mathcal{Q}$*
- *$\mathcal{P} \wedge \mathcal{Q} \equiv_{dec} \mathcal{Q}$ iff $\mathcal{Q} \sqsubseteq_{dec} \mathcal{P}$.*

Theorem 3 (Compositionality of conjunction). *Let $\mathcal{P}$, $\mathcal{Q}$ and $\mathcal{R}$ be declarative specifications such that $\mathcal{P}$ is composable with $\mathcal{R}$ for conjunction. If $\mathcal{Q} \sqsubseteq_{dec} \mathcal{P}$, then $\mathcal{Q} \wedge \mathcal{R} \sqsubseteq_{dec} \mathcal{P} \wedge \mathcal{R}$.*

2.4 Quotient

The final operation that we consider on the specification theory is that of quotienting, which has strong connections to synthesis. Given a specification for a system $\mathcal{R}$, together with a component $\mathcal{P}$ implementing part of $\mathcal{R}$, the quotient yields the *coarsest* specification for the remaining part of $\mathcal{R}$ to be implemented. Thus, the parallel composition of the quotient with $\mathcal{P}$ should be a refinement of the system-wide specification $\mathcal{R}$. Therefore, quotient can be thought of as the adjoint of parallel composition.

As $\mathcal{P}$ is a sub-component of $\mathcal{R}$, we make the reasonable assumption that $\mathcal{A}_{\mathcal{P}} \subseteq \mathcal{A}_{\mathcal{R}}$. Moreover, a necessary condition for the existence of the quotient is that $\mathcal{A}^O_{\mathcal{P}} \subseteq \mathcal{A}^O_{\mathcal{R}}$, otherwise refinement will fail on the alphabet containment checks.

Definition 8 (Quotient). *Let $\mathcal{P}$ and $\mathcal{R}$ be declarative specifications such that $\mathcal{A}^O_{\mathcal{P}} \subseteq \mathcal{A}^O_{\mathcal{R}}$ and $\mathcal{A}_{\mathcal{P}} \subseteq \mathcal{A}_{\mathcal{R}}$. The quotient of $\mathcal{P}$ from $\mathcal{R}$ is the specification $\mathcal{R}/\mathcal{P}$ with signature $\langle \mathcal{A}^I_{\mathcal{R}/\mathcal{P}}, \mathcal{A}^O_{\mathcal{R}/\mathcal{P}}, T_{\mathcal{R}/\mathcal{P}}, F_{\mathcal{R}/\mathcal{P}} \rangle$, where:*

- $\mathcal{A}^I_{\mathcal{R}/\mathcal{P}} = \mathcal{A}^O_{\mathcal{P}} \cup \mathcal{A}^I_{\mathcal{R}}$
- $\mathcal{A}^O_{\mathcal{R}/\mathcal{P}} = \mathcal{A}^O_{\mathcal{R}} \setminus \mathcal{A}^O_{\mathcal{P}}$
- $T_{\mathcal{R}/\mathcal{P}} = \{t \in \mathcal{A}^*_{\mathcal{R}} : \forall t' \text{ a prefix of } t \cdot L(t') \text{ and } \forall t'' \in {\mathcal{A}^I_{\mathcal{R}/\mathcal{P}}}^* \cdot L(tt'')\}$
- $F_{\mathcal{R}/\mathcal{P}} = \{t \in A^*_{\mathcal{R}} : (t \restriction \mathcal{A}_{\mathcal{P}} \in T_{\mathcal{P}} \implies t \in F_{\mathcal{E}(\mathcal{R})}) \text{ and } \forall t' \text{ a prefix of } t \cdot L(t')\}$
- $L(t) = (t \restriction \mathcal{A}_{\mathcal{P}} \in F_{\mathcal{P}} \implies t \in F_{\mathcal{E}(\mathcal{R})}) \text{ and } (t \restriction \mathcal{A}_{\mathcal{P}} \in T_{\mathcal{P}} \implies t \in T_{\mathcal{R}})$.

The alphabet of the quotient contains all of the actions from $\mathcal{A}_{\mathcal{R}}$ and $\mathcal{A}_{\mathcal{P}}$ so that $\mathcal{R}/\mathcal{P}$ can fully control $\mathcal{P}$ and emulate the behaviour of $\mathcal{R}$. Yet still, simple examples reveal that there may not exist a component $\mathcal{Q}$ over an interface consisting of inputs $\mathcal{A}^I_{\mathcal{R}/\mathcal{P}}$ and outputs $\mathcal{A}^O_{\mathcal{R}/\mathcal{P}}$ such that $\mathcal{P} \mid\mid \mathcal{Q} \sqsubseteq_{dec} \mathcal{R}$. Unfortunately, the existence of the quotient cannot be ascertained by a syntactic check on the alphabets of $\mathcal{P}$ and $\mathcal{R}$.

In Definition 8 we referred to $\mathcal{R}/\mathcal{P}$ as a specification, but not a declarative specification. As the following theorem shows, if $T_{\mathcal{R}/\mathcal{P}}$ is non-empty (a condition of being a declarative specification), then the quotient exists.

Theorem 4 (Existence of quotient). *Let $\mathcal{P}$ and $\mathcal{R}$ be declarative specifications such that $\mathcal{A}^O_{\mathcal{P}} \subseteq \mathcal{A}^O_{\mathcal{R}}$ and $\mathcal{A}_{\mathcal{P}} \subseteq \mathcal{A}_{\mathcal{R}}$. Then there exists a declarative specification $\mathcal{Q}$ with input actions $\mathcal{A}^I_{\mathcal{R}/\mathcal{P}}$ and output actions $\mathcal{A}^O_{\mathcal{R}/\mathcal{P}}$ such that $\mathcal{P} \mid\mid \mathcal{Q} \sqsubseteq_{dec} \mathcal{R}$ iff $T_{\mathcal{R}/\mathcal{P}} \neq \emptyset$.*

The next two theorems show that $\mathcal{R}/\mathcal{P}$ satisfies the required properties of quotient when $T_{\mathcal{R}/\mathcal{P}}$ is non-empty, and that quotient is well-behaved with respect to refinement.

Theorem 5 (Properties of quotient). *Let $\mathcal{P}$ and $\mathcal{R}$ be declarative specifications such that $\mathcal{A}^O_{\mathcal{P}} \subseteq \mathcal{A}^O_{\mathcal{R}}$ and $\mathcal{A}_{\mathcal{P}} \subseteq \mathcal{A}_{\mathcal{R}}$. If $T_{\mathcal{R}/\mathcal{P}} \neq \emptyset$, then $\mathcal{P} \mid\mid (\mathcal{R}/\mathcal{P}) \sqsubseteq_{dec} \mathcal{R}$ and for any declarative specification $\mathcal{Q}$ over inputs $\mathcal{A}^I_{\mathcal{R}/\mathcal{P}}$ and outputs $\mathcal{A}^O_{\mathcal{R}/\mathcal{P}}$ such that $\mathcal{P} \mid\mid \mathcal{Q} \sqsubseteq_{dec} \mathcal{R}$ it holds that $\mathcal{Q} \sqsubseteq_{dec} \mathcal{R}/\mathcal{P}$.*

Theorem 6 (Compositionality of quotient). *Let $\mathcal{P}$, $\mathcal{Q}$ and $\mathcal{R}$ be declarative specifications such that $\mathcal{Q} \sqsubseteq_{dec} \mathcal{P}$.*

- *If $\mathcal{Q}/\mathcal{R}$ and $\mathcal{P}/\mathcal{R}$ are defined, then $\mathcal{Q}/\mathcal{R} \sqsubseteq_{dec} \mathcal{P}/\mathcal{R}$.*
- *If $\mathcal{R}/\mathcal{Q}$ and $\mathcal{R}/\mathcal{P}$ are defined, and $(\mathcal{A}^I_{\mathcal{Q}} \cap \mathcal{A}^O_{\mathcal{R}}) \setminus \mathcal{A}_{\mathcal{P}} = \emptyset$, then $\mathcal{R}/\mathcal{Q} \sqsupseteq_{dec} \mathcal{R}/\mathcal{P}$.*

3 An Operational Theory of Components

In this section we take an operational view of components, by specifying their allowable interactions in terms of Logic IOLTSs, an I/O version of labelled transition systems augmented by an inconsistency predicate on states. We remain faithful to the trace-based substitutive preorder, and cast refinement at the operational level in terms of declarative refinement. For any operational model, we can derive an equivalent declarative specification, meaning that the observable safe interactions between the models and an arbitrary environment are indistinguishable.

To support a compositional theory of components, we define the operations of parallel composition, conjunction and quotient directly on our operational models. We further show that compositionality results for the operators on the declarative framework carry over to the operational framework as well.

An explicit definition of implementation is not provided for our models, although there are a number of candidates. One such suggestion for the characterisation of implementations would be the set of specifications in which no inconsistent states are reachable. We leave this for the user to decide.

We can now define the operational models formally. For a set $\mathcal{A}$, write $\mathcal{A}^\tau$ as shorthand for $\mathcal{A} \cup \{\tau\}$, where it is assumed that $\tau \notin \mathcal{A}$.

Definition 9 (IOLTS). *An* I/O labelled transition system (IOLTS) P *is a tuple $\langle S_{\mathsf{P}}, \mathcal{A}^I_{\mathsf{P}}, \mathcal{A}^O_{\mathsf{P}}, \longrightarrow_{\mathsf{P}} \rangle$, where S_{P} is a (possibly infinite) collection of processes (states), $\mathcal{A}^I_{\mathsf{P}}$ and $\mathcal{A}^O_{\mathsf{P}}$ are disjoint sets referred to as the inputs and outputs (the union of which we denote by $\mathcal{A}_{\mathsf{P}}$), and $\longrightarrow_{\mathsf{P}} \subseteq S_{\mathsf{P}} \times \mathcal{A}^\tau_{\mathsf{P}} \times S_{\mathsf{P}}$ is the transition relation.*

Note that since we do not insist on our components being fully input-enabled (unlike I/O automata [5]), meaning that at any stage a component can refuse to accept an input issued by the environment or another component, we must extend IOLTSs to reason about potential communication mismatches that occur during interactions. We accomplish this by augmenting IOLTSs with an inconsistency predicate for tracking mismatches. The resulting model, called a Logic IOLTS, takes its inspiration from the Logic LTSs of Lüttgen and Vogler [6,7], although we have a different interpretation of inconsistency.

Definition 10 (Logic IOLTS). *A* Logic IOLTS P *is a tuple $\langle S_{\mathsf{P}}, \mathcal{A}^I_{\mathsf{P}}, \mathcal{A}^O_{\mathsf{P}}, \longrightarrow_{\mathsf{P}}, F_{\mathsf{P}} \rangle$ in which $\langle S_{\mathsf{P}}, \mathcal{A}^I_{\mathsf{P}}, \mathcal{A}^O_{\mathsf{P}}, \longrightarrow_{\mathsf{P}} \rangle$ is an IOLTS, and $F_{\mathsf{P}} \subseteq S_{\mathsf{P}}$ is an inconsistency predicate on states.*

The inconsistency predicate annotates states that correspond to run-time errors such as communication mismatches or underspecificationRegardless of why a state is inconsistent, we assume that on encountering an inconsistency, unspecified behaviour can ensue. Consequently, inconsistent states are resemblant of the process CHAOS from CSP [8].

Figure 1 shows a number of Logic IOLTSs represented pictorially. We adopt the convention of enclosing the transition system within a box corresponding to the interface of the component. Labelled arrows pointing at the interface correspond to inputs, whereas arrows emanating from the interface correspond to outputs. As a matter of clarity, we only represent the states that are reachable by a sequence of transitions from the process we are interested in. States annotated with an F are deemed to be inconsistent.

We introduce nomenclature for handling stability and hidden τ-transitions. A relation $\overset{\epsilon}{\Longrightarrow}_{\mathsf{P}} \subseteq S_{\mathsf{P}} \times S_{\mathsf{P}}$ is defined by $p \overset{\epsilon}{\Longrightarrow}_{\mathsf{P}} p'$ iff $p(\overset{\tau}{\longrightarrow}_{\mathsf{P}})^* p'$. Generalising $\overset{\epsilon}{\Longrightarrow}_{\mathsf{P}}$ for visible actions $a \in \mathcal{A}$, we obtain $p \overset{a}{\Longrightarrow}_{\mathsf{P}} p'$ iff there exist p_a and p'_a such that $p \overset{\epsilon}{\Longrightarrow}_{\mathsf{P}} p_a \overset{a}{\longrightarrow}_{\mathsf{P}} p'_a \overset{\epsilon}{\Longrightarrow}_{\mathsf{P}} p'$, and $p \overset{\dashv a}{\Longrightarrow}_{\mathsf{P}} p'$ iff there exists p_a such that $p \overset{a}{\longrightarrow}_{\mathsf{P}} p_a \overset{\epsilon}{\Longrightarrow}_{\mathsf{P}} p'$. The extension to words $w = a_1 \ldots a_n$ is defined in the natural way by $p \overset{w}{\Longrightarrow}_{\mathsf{P}} p'$ iff $p \overset{a_1}{\Longrightarrow}_{\mathsf{P}} \ldots \overset{a_n}{\Longrightarrow}_{\mathsf{P}} p'$.

Furthermore, for a compositional operator $\oplus$, and sets A and B, we write $A \oplus B$ for the set $\{a \oplus b : a \in A \text{ and } b \in B\}$. This allows us to use a process-algebraic notation for states.

From hereon, let $\mathsf{P} = \langle S_{\mathsf{P}}, \mathcal{A}^I_{\mathsf{P}}, \mathcal{A}^O_{\mathsf{P}}, \longrightarrow_{\mathsf{P}}, F_{\mathsf{P}} \rangle$, $\mathsf{Q} = \langle S_{\mathsf{Q}}, \mathcal{A}^I_{\mathsf{Q}}, \mathcal{A}^O_{\mathsf{Q}}, \longrightarrow_{\mathsf{Q}}, F_{\mathsf{Q}} \rangle$ and $\mathsf{R} = \langle S_{\mathsf{R}}, \mathcal{A}^I_{\mathsf{R}}, \mathcal{A}^O_{\mathsf{R}}, \longrightarrow_{\mathsf{R}}, F_{\mathsf{R}} \rangle$ be three Logic IOLTSs, and let p_{P}, q_{Q} and r_{R} be processes in the Logic IOLTSs P, Q and R respectively.

3.1 Refinement

In keeping with the declarative framework, we wish refinement to correspond to safe-substitutivity. Hence, we cast refinement at the operational level in terms of refinement at the declarative level. To do this, we define a mapping $[\![\cdot]\!]^*$ from operational models to declarative models (Definition 13) that preserves the environments that the models can interact harmoniously with.

An essential feature of operational refinement is that the mapping from operational to declarative models preserves the safe traces of the component. For a declarative specification $\mathcal{P}$, a trace t is said to be immediately-safe iff t is permissible, but not inconsistent (i.e., t lies within $T_{\mathcal{P}} \setminus F_{\mathcal{P}}$). If t is contained within $T_{\mathcal{P}} \setminus F_{\mathcal{E}(\mathcal{P})}$, we say that t is safe. The calculation of the safe traces for a Logic IOLTS is slightly more involved, because it is necessary to deal with non-determinism and τ-transitions.

Definition 11 (Immediately-safe states). *The set of immediately-safe states that a process p_{P} can be in after following the trace t is given by $h_{p_{\mathsf{P}}}(t)$, where $h_{p_{\mathsf{P}}} : \mathcal{A}^*_{\mathsf{P}} \longrightarrow 2^{S_{\mathsf{P}}}$ is defined as:*

- $h_{p_{\mathsf{P}}}(\epsilon) = \begin{cases} \emptyset & \textit{if } p_{\mathsf{P}} \overset{\epsilon}{\Longrightarrow}_{\mathsf{P}} p' \textit{ with } p' \in F_{\mathsf{P}} \\ \{p' \in S_{\mathsf{P}} : p \overset{\epsilon}{\Longrightarrow}_{\mathsf{P}} p'\} & \textit{otherwise} \end{cases}$
- $h_{p_{\mathsf{P}}}(to) = \begin{cases} \emptyset & \textit{if } \exists p' \in h_{p_{\mathsf{P}}}(t) \textit{ such that } p' \overset{\dashv o}{\Longrightarrow}_{\mathsf{P}} p'' \textit{ with } p'' \in F_{\mathsf{P}} \\ \{p'' \in S_{\mathsf{P}} : \exists p' \in h_{p_{\mathsf{P}}}(t) \cdot p' \overset{\dashv o}{\Longrightarrow}_{\mathsf{P}} p''\} & \textit{otherwise} \end{cases}$

  when $o \in \mathcal{A}^O_{\mathsf{P}}$
- $h_{p_{\mathsf{P}}}(ti) = \begin{cases} \emptyset & \textit{if } \exists p' \in h_{p_{\mathsf{P}}}(t) \textit{ such that } p' \overset{\dashv i}{\Longrightarrow}_{\mathsf{P}} p'' \textit{ with } p'' \in F_{\mathsf{P}}, \textit{ or } p' \not\xrightarrow{i}_{\mathsf{P}} \\ \{p'' \in S_{\mathsf{P}} : \exists p' \in h_{p_{\mathsf{P}}}(t) \cdot p' \overset{\dashv i}{\Longrightarrow}_{\mathsf{P}} p''\} & \textit{otherwise} \end{cases}$

  when $i \in \mathcal{A}^I_{\mathsf{P}}$.

Definition 12 (Safe traces). *A trace t of p_{P} is immediately-safe iff $h_{p_{\mathsf{P}}}(t) \neq \emptyset$ and is safe iff $h_{p_{\mathcal{E}(\mathsf{P})}}(t) \neq \emptyset$, where $\mathcal{E}$ propagates inconsistencies backwards over output and τ transitions. The set of immediately-safe traces of p_{P} is denoted $IST(p_{\mathsf{P}})$, while the set of safe traces is denoted $ST(p_{\mathsf{P}})$.*

An immediately-safe trace t of a process p characterises a permissible exchange between p and an arbitrary environment, such that t will never encounter an inconsistent state under any resolution of p's non-determinism. Relating this intuition to Definitions 11 and 12, suppose p and the environment can safely communicate on the trace t. If from some state that p is in after following t it can perform an output o, and every o it can output will never make the system inconsistent, then the environment must be willing to accept that output. Conversely, the environment can only safely issue an input i after t if i can be accepted from every state the process is in after following t, without making the system inconsistent. We must impose these restrictions to account for the fact that the process cannot be expected to know how to resolve its non-determinism prior to its communication with the environment.

Definition 13 (Model mapping). *The* model mapping *function* $[\![\cdot]\!]^*$ *from Logic IOLTSs to declarative specifications is defined by* $[\![p_{\mathsf{P}}]\!]^* = \langle \mathcal{A}^I_{\mathsf{P}}, \mathcal{A}^O_{\mathsf{P}}, T_{[\![p_{\mathsf{P}}]\!]^*}, F_{[\![p_{\mathsf{P}}]\!]^*} \rangle$*, where:*

- $T_{[\![p_{\mathsf{P}}]\!]^*} = \{t : p_{\mathsf{P}} \overset{t}{\Longrightarrow}_{\mathsf{P}}\} \cup F \cup FI$
- $F_{[\![p_{\mathsf{P}}]\!]^*} = F \cup FI$
- $F = \{tt' : p_{\mathsf{P}} \overset{t}{\Longrightarrow}_{\mathsf{P}} p',\ p' \in F_{\mathsf{P}} \textit{ and } t' \in \mathcal{A}^*_{\mathsf{P}}\}$
- $FI = \{tit' : p_{\mathsf{P}} \overset{t}{\Longrightarrow}_{\mathsf{P}} p',\ i \in \mathcal{A}^I_{\mathsf{P}},\ p' \not\xrightarrow{i}_{\mathsf{P}} \textit{ and } t' \in \mathcal{A}^*_{\mathsf{P}}\}$.

Theorem 7 (Model mapping preserves safe traces). *For an arbitrary process p_{P}, $IST(p_{\mathsf{P}}) = T_{[\![p_{\mathsf{P}}]\!]^*} \setminus F_{[\![p_{\mathsf{P}}]\!]^*}$ and $ST(p_{\mathsf{P}}) = T_{[\![p_{\mathsf{P}}]\!]^*} \setminus F_{\mathcal{E}([\![p_{\mathsf{P}}]\!]^*)}$.*

Having defined a mapping from operational to declarative models, we can now define operational refinement in the obvious way.

Definition 14 (Operational refinement). *Process q_{Q} is said to be a refinement of process p_{P}, written $q_{\mathsf{Q}} \sqsubseteq_{op} p_{\mathsf{P}}$, iff $[\![q_{\mathsf{Q}}]\!]^* \sqsubseteq_{dec} [\![p_{\mathsf{P}}]\!]^*$.*

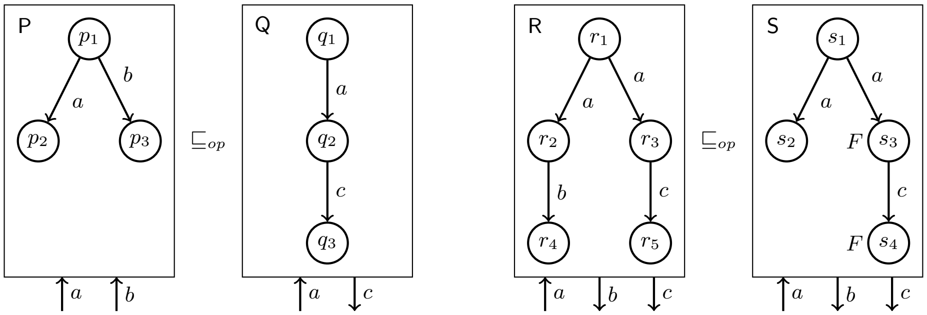

Fig. 1. Refinement of Logic IOLTSs.

Lemma 2 (Operational preorder). *Refinement is reflexive and transitive.*

Under the assumption of finiteness, we note that refinement checking is PSpace-complete. This is similar to traces refinement in CSP, where the worst-case is rarely observed in practice.

Definition 15 (Operational equivalence). *Processes p_{P} and q_{Q} are said to be equivalent, written $p_{\mathsf{P}} \equiv_{op} q_{\mathsf{Q}}$, iff $q_{\mathsf{Q}} \sqsubseteq_{op} p_{\mathsf{P}}$ and $p_{\mathsf{P}} \sqsubseteq_{op} q_{\mathsf{Q}}$.*

Looking at the refinements in Figure 1, from q_1 the environment can safely issue a, after which it must be willing to accept c. Clearly a can be safely accepted by p_1, and as p_2 does not issue a c output the environment will be perfectly happy. Moreover, as the environment is not permitted to issue a b in q_1 there is no harm in p_1 being able to handle this behaviour. Hence $p_1 \sqsubseteq_{op} q_1$. Now, $r_1 \sqsubseteq_{op} s_1$ as r_1 is willing to accept the input a from the environment, which is not the case in s_1. This is because we cannot trust s_1 to resolve its non-determinism on a in an optimistic way by always moving to s_2.

Example 2. To formally check $p_1 \sqsubseteq_{op} q_1$, it is necessary to resort to the definition of refinement on declarative specifications (Definition 4). It can easily be checked that all of the conditions of that definition hold by considering the sets below, obtained by computing the model mapping of the processes p_1 and q_1.

- $F_{[\![p_1]\!]^*} = F_{\mathcal{E}([\![p_1]\!]^*)} = (a+b)(a+b)^+$
- $F_{[\![q_1]\!]^*} = F_{\mathcal{E}([\![q_1]\!]^*)} = (a+ac)a(a+c)^*$
- $T_{[\![p_1]\!]^*} = (a+b)^*$
- $T_{[\![q_1]\!]^*} = \{\epsilon, a, ac\} \cup (a+ac)a(a+c)^*$
- $X \uparrow \mathcal{A}^I_{\mathcal{P}} = X \cup (\epsilon + a + ac + (aa+aca)(a+c)^*)b(a+b+c)^*$ for $X \in \{F_{[\![q_1]\!]^*}, T_{[\![q_1]\!]^*}\}$.

3.2 Error-completion

In order to simplify the definitions of the operators in our specification theory for the operational framework, we introduce the error-completion of a Logic

IOLTS. This is a transformation that leaves the mapping from a Logic IOLTS to a declarative specification unchanged.

The error-completion of a Logic IOLTS provides an explicit operational representation for the inconsistent traces that would arise in mapping the Logic IOLTS to its corresponding declarative specification. Consequently, an error-completed Logic IOLTS is closed under input extensions. It is this property that simplifies the definitions of the operators in our framework. We do not say that an error-completed Logic IOLTS is input-enabled, however, as we can distinguish good inputs from bad inputs.

Definition 16 (Error-completion). *Let P be a Logic IOLTS, and assume $f_{\mathsf{P}} \notin S_{\mathsf{P}}$. The* error-completion *of P is a Logic IOLTS $\mathsf{P}_\perp = \langle S_{\mathsf{P}_\perp}, \mathcal{A}^I_{\mathsf{P}}, \mathcal{A}^O_{\mathsf{P}}, \longrightarrow_{\mathsf{P}_\perp}, F_{\mathsf{P}_\perp}\rangle$, where:*

- $S_{\mathsf{P}_\perp} = S_{\mathsf{P}} \cup \{f_{\mathsf{P}}\}$
- $\longrightarrow_{\mathsf{P}_\perp} = \longrightarrow_{\mathsf{P}} \cup \{(f, a, f) : f \in F_{\mathsf{P}_\perp}$ *and* $a \in \mathcal{A}_{\mathsf{P}}\} \cup \{(s, a, f_{\mathsf{P}}) : a \in \mathcal{A}^I_{\mathsf{P}}$ *and* $\nexists s' \cdot s \xrightarrow{a}_{\mathsf{P}} s'\}$
- $F_{\mathsf{P}_\perp} = F_{\mathsf{P}} \cup \{f_{\mathsf{P}}\}$.

As remarked, the error-completion of a Logic IOLTS preserves the mapping from Logic IOLTSs to declarative specifications, as the next lemma shows. Note that the corresponding declarative specifications are equal, rather than declaratively equivalent.

Lemma 3 (Error-completion respects mappings). *For any process p_{P}, $[\![p_{\mathsf{P}}]\!]^* = [\![p_{\mathsf{P}_\perp}]\!]^*$.*

Besides simplifying the definition of the compositional operators in our specification theory, error-completion of a Logic IOLTS also simplifies the definition of the model mapping function.

Lemma 4 (Simplified model mapping). *Let p be a process in Logic IOLTS $\mathsf{P}_\perp$. Then $[\![p]\!]^* = \langle \mathcal{A}^I_{\mathsf{P}}, \mathcal{A}^O_{\mathsf{P}}, T_{[\![p]\!]^*}, F_{[\![p]\!]^*}\rangle$, where:*

- $T_{[\![p]\!]^*} = \{t : p \overset{t}{\Longrightarrow}_{\mathsf{P}_\perp}\}$
- $F_{[\![p]\!]^*} = \{tt' : p \overset{t}{\Longrightarrow}_{\mathsf{P}_\perp} p' \overset{t'}{\Longrightarrow}_{\mathsf{P}_\perp}$ *and* $p' \in F_{\mathsf{P}_\perp}\}$.

3.3 Parallel composition

As for declarative specifications, the parallel composition of Logic IOLTSs yields a Logic IOLTS representing the combined effect of its operands running asynchronously. We insist that any given output should be under the control of one component only. Therefore Logic IOLTSs P and Q are composable for parallel composition only if $\mathcal{A}^O_{\mathsf{P}} \cap \mathcal{A}^O_{\mathsf{Q}} = \emptyset$.

Definition 17 (Parallel composition). *Let P and Q be Logic IOLTSs composable for parallel composition. Then the parallel composition of P and Q is a Logic IOLTS $\mathsf{P} \mid\mid \mathsf{Q} = \langle S, \mathcal{A}^I, \mathcal{A}^O, \longrightarrow, F\rangle$, where:*

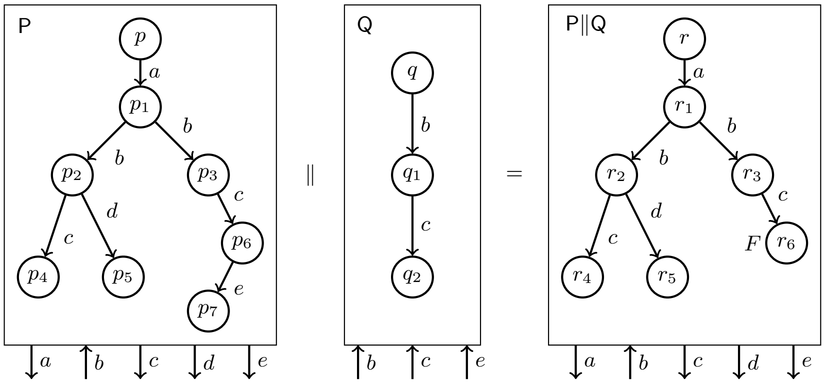

Fig. 2. Example of parallel composition on Logic IOLTSs.

- $S = S_{\mathsf{P}_\perp} \mid\mid S_{\mathsf{Q}_\perp}$
- $\mathcal{A}^I = (\mathcal{A}^I_{\mathsf{P}} \cup \mathcal{A}^I_{\mathsf{Q}}) \setminus (\mathcal{A}^O_{\mathsf{P}} \cup \mathcal{A}^O_{\mathsf{Q}})$
- $\mathcal{A}^O = \mathcal{A}^O_{\mathsf{P}} \cup \mathcal{A}^O_{\mathsf{Q}}$
- $\longrightarrow$ *is the smallest relation satisfying the following rules:*
  P1. If $p \xrightarrow{a}_{\mathsf{P}_\perp} p'$ *with* $a \in \mathcal{A}^\tau_{\mathsf{P}} \setminus \mathcal{A}_{\mathsf{Q}}$, *then* $p \mid\mid q \xrightarrow{a} p' \mid\mid q$
  P2. If $q \xrightarrow{a}_{\mathsf{Q}_\perp} q'$ *with* $a \in \mathcal{A}^\tau_{\mathsf{Q}} \setminus \mathcal{A}_{\mathsf{P}}$, *then* $p \mid\mid q \xrightarrow{a} p \mid\mid q'$
  P3. If $p \xrightarrow{a}_{\mathsf{P}_\perp} p'$ *and* $q \xrightarrow{a}_{\mathsf{Q}_\perp} q'$ *with* $a \in \mathcal{A}_{\mathsf{P}} \cap \mathcal{A}_{\mathsf{Q}}$, *then* $p \mid\mid q \xrightarrow{a} p' \mid\mid q'$.
- $F = (S_{\mathsf{P}_\perp} \mid\mid F_{\mathsf{Q}_\perp}) \cup (F_{\mathsf{P}_\perp} \mid\mid S_{\mathsf{Q}_\perp})$.

Conditions P1 to P3 ensure that the parallel composition of Logic IOLTSs interleave on independent actions and synchronise on common actions. For P3, given the parallel composability constraint, synchronisation can take place between an output and an input, or two inputs. Figure 2 shows how the parallel composition operator works in practice, although we omit non-enabled input transitions to inconsistent states. In particular, the example demonstrates how inconsistencies can be introduced through non-input enabledness, as in state r_6 corresponding to $p_6 \mid\mid q_2$.

Reassuringly, parallel composition of Logic IOLTSs yields a Logic IOLTS. The following theorem shows the relationship between parallel composition on Logic IOLTSs and parallel composition on declarative specifications.

Theorem 8 (Parallel correspondences). *Let* P *and* Q *be Logic IOLTSs composable for parallel composition. For processes* p_{P} *and* q_{Q}, *it holds that* $[\![p_{\mathsf{P}} \mid\mid q_{\mathsf{Q}}]\!]^* = [\![p_{\mathsf{P}}]\!]^* \mid\mid [\![q_{\mathsf{Q}}]\!]^*$.

3.4 Conjunction

In keeping with conjunction of declarative specifications, Logic IOLTSs P and Q are composable for conjunction only if the sets $\mathcal{A}^I_{\mathsf{P}} \cup \mathcal{A}^I_{\mathsf{Q}}$ and $\mathcal{A}^O_{\mathsf{P}} \cup \mathcal{A}^O_{\mathsf{Q}}$ are disjoint.

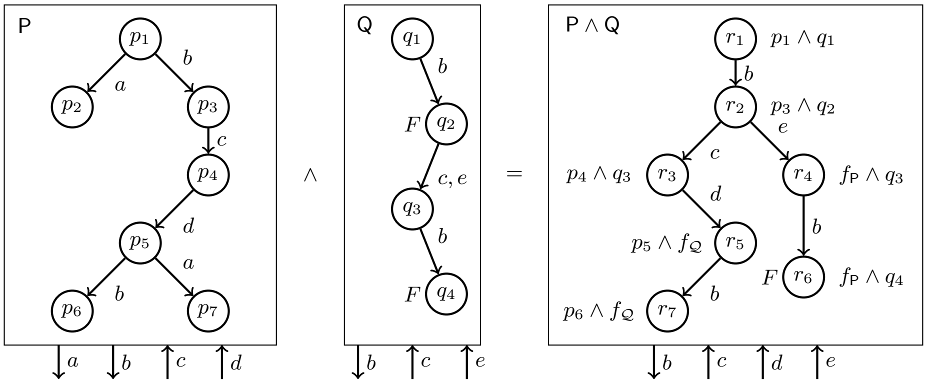

Fig. 3. Example of conjunction on Logic IOLTSs.

Definition 18 (Conjunction). *Let* P *and* Q *be Logic IOLTSs composable for conjunction. Then the conjunction of* P *and* Q *is a Logic IOLTS* $\mathsf{P} \wedge \mathsf{Q} = \langle S, \mathcal{A}^I_\mathsf{P} \cup \mathcal{A}^I_\mathsf{Q}, \mathcal{A}^O_\mathsf{P} \cap \mathcal{A}^O_\mathsf{Q}, \longrightarrow, F \rangle$*, where:*

- $S = S_{\mathsf{P}_\perp} \wedge S_{\mathsf{Q}_\perp}$
- $\longrightarrow$ *is the smallest relation satisfying the following rules:*
  C1. If $a \in \mathcal{A}^O_\mathsf{P} \cap \mathcal{A}^O_\mathsf{Q}$*,* $p \xrightarrow{a}_{\mathsf{P}_\perp} p'$ *and* $q \xrightarrow{a}_{\mathsf{Q}_\perp} q'$*, then* $p \wedge q \xrightarrow{a} p' \wedge q'$
  C2. If $a \in \mathcal{A}^I_\mathsf{P} \cap \mathcal{A}^I_\mathsf{Q}$*,* $p \xrightarrow{a}_{\mathsf{P}_\perp} p'$ *and* $q \xrightarrow{a}_{\mathsf{Q}_\perp} q'$*, then* $p \wedge q \xrightarrow{a} p' \wedge q'$
  C3. If $a \in \mathcal{A}^I_\mathsf{P} \setminus \mathcal{A}^I_\mathsf{Q}$ *and* $p \xrightarrow{a}_{\mathsf{P}_\perp} p'$*, then* $p \wedge q \xrightarrow{a} p' \wedge f_\mathsf{Q}$
  C4. If $a \in \mathcal{A}^I_\mathsf{Q} \setminus \mathcal{A}^I_\mathsf{P}$ *and* $q \xrightarrow{a}_{\mathsf{Q}_\perp} q'$*, then* $p \wedge q \xrightarrow{a} f_\mathsf{P} \wedge q'$
  C5. If $p \xrightarrow{\tau}_{\mathsf{P}_\perp} p'$*, then* $p \wedge q \xrightarrow{\tau} p' \wedge q$
  C6. If $q \xrightarrow{\tau}_{\mathsf{Q}_\perp} q'$*, then* $p \wedge q \xrightarrow{\tau} p \wedge q'$
- $F = F_{\mathsf{P}_\perp} \wedge F_{\mathsf{Q}_\perp}$.

The idea behind the definition of conjunction for $p \wedge q$ is that p and q must synchronise on common actions, interleave on τ-transitions, and on encountering independent input actions behave like the respective component to which the action belongs. On encountering a state $p \wedge q$ in which one of $p \in F_\mathsf{P}$ or $q \in F_\mathsf{Q}$ holds, let it be p, we know that whatever the behaviour of $p \wedge q$ it will always be a refinement of p. So the most general refinement of $p \wedge q$ will actually be q. This is supported by the fact that inconsistent states in the error-completed Logic IOLTS admit arbitrary behaviour.

Figure 3 shows the conjunction of processes p_1 and q_1 in the Logic IOLTSs P and Q (although for clarity we omit inputs leading to inconsistent states). In state r_1 corresponding to $p_1 \wedge q_1$, the b-output transitions of p_1 and q_1 synchronise. Independent output actions such as the a-transition in p_1 are not permitted to proceed, because it would not be the case that r_1 could be used safely in place of q_1 if this transition were to be permitted. State r_2 can evolve into r_3 by synchronising the c-inputs of p_3 and q_2, while it can also evolve into r_4 by proceeding on the independent input e of q_2. From this point, r_4 behaves like q_3, because e is an input-violation of p_3. Similar reasoning applies to r_3's evolution into r_5 by receiving the d-input.

As for parallel composition, there is a correspondence between conjunction at the operational and declarative levels.

Theorem 9 (Conjunction correspondences). *Let* P *and* Q *be Logic IOLTSs composable for conjunction. For processes* p_{P} *and* q_{Q}*, it holds that* $[\![p_{\mathsf{P}} \wedge q_{\mathsf{Q}}]\!]^* = [\![p_{\mathsf{P}}]\!]^* \wedge [\![q_{\mathsf{Q}}]\!]^*$.

3.5 Quotient

Non-determinism and τ-transitions arising in Logic IOLTSs make the definition of quotient more involved than the other operators we have considered on operational models. To ensure that the quotient is the coarsest specification, it is necessary to track the non-determinism of the system-wide specification and its partial implementation. This is because the non-determinism can affect the safe traces of a Logic IOLTS.

As for declarative specifications, we only compute the quotient of process p_{P} from r_{R} when $\mathcal{A}^O_{\mathsf{P}} \subseteq \mathcal{A}^O_{\mathsf{R}}$ and $\mathcal{A}_{\mathsf{P}} \subseteq \mathcal{A}_{\mathsf{R}}$. The quotient is the coarsest specification q over an interface consisting of inputs $\mathcal{A}^O_{\mathsf{P}} \cup \mathcal{A}^I_{\mathsf{R}}$ and outputs $\mathcal{A}^O_{\mathsf{R}} \setminus \mathcal{A}^O_{\mathsf{P}}$ such that $p_{\mathsf{P}} \mid\mid q \sqsubseteq_{op} r_{\mathsf{R}}$. If such a q exists, we denote it by $r_{\mathsf{R}}/p_{\mathsf{P}}$.

Before defining the quotient-construction, we introduce some functions and predicates that simplify the presentation.

Definition 19. *For Logic IOLTS* P*, set of states* $S \subseteq S_{\mathsf{P}_\perp}$ *and action* $a \in \mathcal{A}_{\mathsf{P}}$*, define:*

- $\mathsf{succ}^{\epsilon}_{\mathsf{P}}(S) = \{s' : s \overset{\epsilon}{\Longrightarrow}_{\mathsf{P}_\perp} s' \textit{ with } s \in S\}$
- $\mathsf{succ}^{a}_{\mathsf{P}}(S) = \{s' : s \overset{\dashv a}{\Longrightarrow}_{\mathsf{P}_\perp} s' \textit{ with } s \in S\}$.

Definition 20 (Quotient Logic IOLTS). *Let* P *and* R *be Logic IOLTSs such that* $\mathcal{A}^O_{\mathsf{P}} \subseteq \mathcal{A}^O_{\mathsf{R}}$ *and* $\mathcal{A}_{\mathsf{P}} \subseteq \mathcal{A}_{\mathsf{R}}$*. The* quotient *of* P *from* R *is the Logic IOLTS* $\mathsf{R/P} = \langle S_{\mathsf{R/P}}, \mathcal{A}^I_{\mathsf{R/P}}, \mathcal{A}^O_{\mathsf{R/P}}, \longrightarrow, F_{\mathsf{R/P}}\rangle$*, where:*

- $S_{\mathsf{R/P}} = \{R/P : R \subseteq S_{\mathsf{R}_\perp} \textit{ and } P \subseteq S_{\mathsf{P}_\perp}\}$
- $\mathcal{A}^I_{\mathsf{R/P}} = \mathcal{A}^O_{\mathsf{P}} \cup \mathcal{A}^I_{\mathsf{R}}$
- $\mathcal{A}^O_{\mathsf{R/P}} = \mathcal{A}^O_{\mathsf{R}} \setminus \mathcal{A}^O_{\mathsf{P}}$
- $\longrightarrow$ *is the smallest relation satisfying the following rules:*

  Q1. $R'/P' \xrightarrow{a} \mathsf{succ}^a_{\mathsf{R}}(R')/\mathsf{succ}^a_{\mathsf{P}}(P')$ *providing:*
  (a) $a \in \mathcal{A}^I_{\mathsf{P}} \cap \mathcal{A}^I_{\mathsf{R}}$ *implies* $\mathsf{succ}^a_{\mathsf{R}}(R') \cap F_{\mathcal{E}(\mathsf{R})} = \emptyset$
  (b) $a \in \mathcal{A}^O_{\mathsf{P}} \cap \mathcal{A}^O_{\mathsf{R}}$ *implies* $\mathsf{succ}^a_{\mathsf{R}}(R') \cap F_{\mathcal{E}(\mathsf{R})} = \emptyset$ *and* $\mathsf{succ}^a_{\mathsf{P}}(P') \neq \emptyset$
  (c) $a \in \mathcal{A}^I_{\mathsf{P}} \cap \mathcal{A}^O_{\mathsf{R}}$ *implies* $\mathsf{succ}^a_{\mathsf{P}}(P') \cap F_{\mathsf{P}} = \emptyset$ *and* $\mathsf{succ}^a_{\mathsf{R}}(R') \neq \emptyset$

  Q2. $R'/P' \xrightarrow{a} \mathsf{succ}^a_{\mathsf{R}}(R')/P'$ *providing:*
  (a) $a \in \mathcal{A}^I_{\mathsf{R}} \setminus \mathcal{A}_{\mathsf{P}}$ *implies* $\mathsf{succ}^a_{\mathsf{R}}(R') \cap F_{\mathcal{E}(\mathsf{R})} = \emptyset$
  (b) $a \in \mathcal{A}^O_{\mathsf{R}} \setminus \mathcal{A}_{\mathsf{P}}$ *implies* $\mathsf{succ}^a_{\mathsf{R}}(R') \neq \emptyset$.

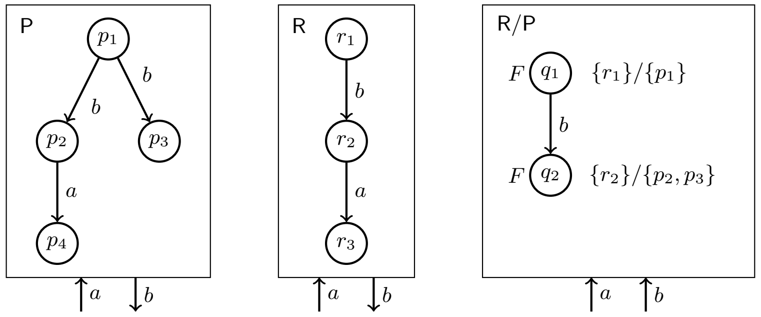

Fig. 4. Example showing non-existence of quotient on Logic IOLTSs.

- *$R'/P' \in F_{\mathsf{R/P}}$ iff at least one of the following rules holds:*
  F1. $R' = \emptyset$ or $P' = \emptyset$
  F2. $F_{\mathcal{E}(\mathsf{R}_\perp)} \cap R' \neq \emptyset$ or $F_{\mathsf{P}_\perp} \cap P' \neq \emptyset$
  F3. $R'/P' \xrightarrow{a} R''/P''$ with $a \in \mathcal{A}^I_{\mathsf{R/P}}$ and $R''/P'' \in F_{\mathsf{R/P}}$.

Definition 21 (Quotient). *Let P and R be Logic IOLTSs such that $\mathcal{A}^O_{\mathsf{P}} \subseteq \mathcal{A}^O_{\mathsf{R}}$ and $\mathcal{A}_{\mathsf{P}} \subseteq \mathcal{A}_{\mathsf{R}}$. The* quotient *of process p_{P} from process r_{R}, written $r_{\mathsf{R}}/p_{\mathsf{P}}$, is the process $\mathsf{succ}^{\epsilon}_{\mathsf{R}}(r_{\mathsf{R}})/\mathsf{succ}^{\epsilon}_{\mathsf{P}}(p_{\mathsf{P}})$ in the Logic IOLTS $\mathsf{R}//\mathsf{P}$ obtained from $\mathsf{R/P}$ by removing all transitions immediately leading to a state in $F_{\mathsf{R/P}}$, and removing all states R/P such that $R/P \in F_{\mathsf{R/P}}$ and $R \notin F_{\mathcal{E}(\mathsf{R})}$. If $\mathsf{succ}^{\epsilon}_{\mathsf{R}}(r_{\mathsf{R}})/\mathsf{succ}^{\epsilon}_{\mathsf{P}}(p_{\mathsf{P}})$ is not contained in $\mathsf{R}//\mathsf{P}$, then the quotient is not defined.*

As for declarative specifications, the quotient of p_{P} from r_{R} may not exist. The following theorem shows that definedness of the quotient according to the previous definition coincides precisely with the existence of such a quotient.

Theorem 10 (Existence of quotient). *Let P and R be Logic IOLTSs such that $\mathcal{A}^O_{\mathsf{P}} \subseteq \mathcal{A}^O_{\mathsf{R}}$ and $\mathcal{A}_{\mathsf{P}} \subseteq \mathcal{A}_{\mathsf{R}}$. Then $r_{\mathsf{R}}/p_{\mathsf{P}}$ is defined (i.e. $r_{\mathsf{R}} \in F_{\mathcal{E}(\mathsf{R})}$ or $r_{\mathsf{R}}/p_{\mathsf{P}} \notin F_{\mathsf{R/P}}$) iff there exists a process q in a Logic IOLTS with inputs $\mathcal{A}^I_{\mathsf{R/P}}$ and outputs $\mathcal{A}^O_{\mathsf{R/P}}$ such that $p_{\mathsf{P}} \mid\mid q \sqsubseteq_{op} r_{\mathsf{R}}$.*

Consequently, the constraint $r_{\mathsf{R}} \notin F_{\mathcal{E}(\mathsf{R})}$ and $r_{\mathsf{R}}/p_{\mathsf{P}} \in F_{\mathsf{R/P}}$ gives a precise characterisation of whether the quotient exists or not. When the quotient does exist, it behaves in exactly the same way as for declarative specifications.

Theorem 11 (Quotient correspondences). *Let P and R be Logic IOLTSs such that $\mathcal{A}^O_{\mathsf{P}} \subseteq \mathcal{A}^O_{\mathsf{R}}$ and $\mathcal{A}_{\mathsf{P}} \subseteq \mathcal{A}_{\mathsf{R}}$. If $r_{\mathsf{R}}/p_{\mathsf{P}} \notin F_{\mathsf{R/P}}$ or $r_{\mathsf{R}} \in F_{\mathcal{E}(\mathsf{R})}$, then $[\![r_{\mathsf{R}}/p_{\mathsf{P}}]\!]^* = [\![r_{\mathsf{R}}]\!]^*/[\![p_{\mathsf{P}}]\!]^*$.*

Figure 4 provides an example in which processes p_1 and r_1 have no quotient. This tallies with Theorem 10, as we have $\{r_1\}/\{p_1\} \in F_{\mathsf{R/P}}$ when $r_1 \notin F_{\mathcal{E}(\mathsf{R})}$. On the other hand, quotients exist for the processes p_1 and r_1 of Figures 5 and 6. This is also supported by Theorem 10, as $\{r_1\}/\{p_1\} \notin F_{\mathsf{R/P}}$ for the processes in both figures.

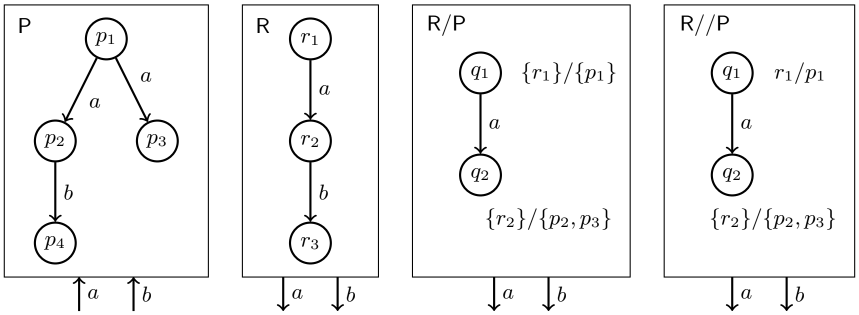

Fig. 5. Example of quotient on Logic IOLTSs with no inconsistencies.

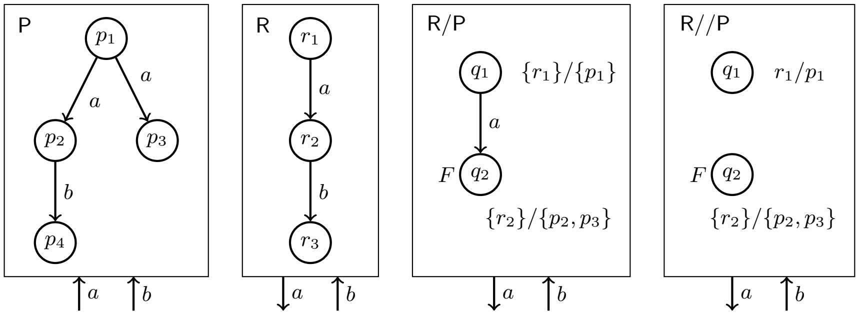

Fig. 6. Example of quotient on Logic IOLTSs with inconsistencies.

For Figure 6, the quotient is the single consistent state $\{r_1\}/\{p_1\}$. This is because in going from R/P to R//P we remove the transition labelled by the output a between the processes $\{r_1\}/\{p_1\}$ and $\{r_2\}/\{p_2, p_3\}$, as the latter state is inconsistent. Maintaining this transition would yield an invalid quotient as $p_1 \mid\mid (r_1/p_1)$ would be inconsistent when r_1 is consistent. It is safe to discard this transition only because it is an output. Recalling the definition of $\sqsubseteq_{op}$, for safe-substitutivity it is perfectly safe to suppress outputs on the left that would have occurred on the right.

4 Full-Abstraction Results

In this section we present full-abstraction results that relate our declarative and operational equivalences based on trace containment to a simple equivalence, which ensures that an inconsistent process must have an inconsistent specification. The result is shown by employing a testing scenario where processes are placed in parallel with an arbitrary composable process in order to establish their equivalence with regard to the observation of consistency.

Definition 22 (Declarative inconsistency equivalence). *Let $\mathcal{P}$ and $\mathcal{Q}$ be declarative specifications.* Declarative inconsistency equivalence*, denoted by $\equiv^F_{dec}$, is given by $\mathcal{P} \equiv^F_{dec} \mathcal{Q}$ iff $\mathcal{A}^I_{\mathcal{P}} = \mathcal{A}^I_{\mathcal{Q}}$, $\mathcal{A}^O_{\mathcal{P}} = \mathcal{A}^O_{\mathcal{Q}}$ and $\epsilon \in F_{\mathcal{P}} \iff \epsilon \in F_{\mathcal{Q}}$.*

Declarative equivalence can be established by placing each process in parallel with arbitrary composable tester processes and observing whether the simple inconsistency equivalence is maintained.

Theorem 12. *Let $\mathcal{P}$ and $\mathcal{Q}$ be declarative specifications. Then:*

$$\mathcal{P} \equiv_{dec} \mathcal{Q} \textit{ iff } \forall \mathcal{R} \cdot \mathcal{A}^O_{\mathcal{R}} \cap (\mathcal{A}^O_{\mathcal{P}} \cup \mathcal{A}^O_{\mathcal{Q}}) = \emptyset \implies \mathcal{E}(\mathcal{P} \mid\mid \mathcal{R}) \equiv^F_{dec} \mathcal{E}(\mathcal{Q} \mid\mid \mathcal{R}).$$

From this characterisation of $\equiv_{dec}$, we obtain a full-abstraction result with respect to parallel composition and $\equiv^F_{dec}$. Our definition of full-abstraction is taken from [19] (Definition 16), which means that $\equiv_{dec}$ is the coarsest congruence with respect to the operators of our specification theory and $\equiv^F_{dec}$.

Corollary 1 (Declarative full-abstraction). *Declarative equivalence $\equiv_{dec}$ is fully-abstract with respect to parallel, conjunction, quotient and $\equiv^F_{dec}$.*

We can now present analogous results for our operational models.

Definition 23 (Operational inconsistency equivalence). *Let p_{P} and q_{Q} be processes of Logic IOLTSs P and Q.* Operational inconsistency equivalence*, denoted by $\equiv^F_{op}$, is given by $p_{\mathsf{P}} \equiv^F_{op} q_{\mathsf{Q}}$ iff $\mathcal{A}^I_{\mathsf{P}} = \mathcal{A}^I_{\mathsf{Q}}$, $\mathcal{A}^O_{\mathsf{P}} = \mathcal{A}^O_{\mathsf{Q}}$ and $p_{\mathsf{P}} \in F_{\mathsf{P}} \iff q_{\mathsf{Q}} \in F_{\mathsf{Q}}$.*

Theorem 13. *Let p_{P} and q_{Q} be processes of Logic IOLTSs P and Q. Then:*

$$p_{\mathsf{P}} \equiv_{op} q_{\mathsf{Q}} \textit{ iff } \forall r_{\mathsf{R}} \cdot \mathcal{A}^O_{\mathsf{R}} \cap (\mathcal{A}^O_{\mathsf{P}} \cup \mathcal{A}^O_{\mathsf{Q}}) = \emptyset \implies \mathcal{E}(p_{\mathsf{P}} \mid\mid r_{\mathsf{R}}) \equiv^F_{op} \mathcal{E}(q_{\mathsf{Q}} \mid\mid r_{\mathsf{R}}),$$

where $\mathcal{E}$ applied to processes in Logic IOLTSs propagates the inconsistency predicate backwards over all output and τ labelled transitions.

Corollary 2 (Operational full-abstraction). *Operational equivalence $\equiv_{op}$ is fully-abstract with respect to parallel, conjunction, quotient and $\equiv^F_{op}$.*

5 Conclusion and Future Work

We have developed a compositional specification theory for components that may be modelled operationally, closely mirroring actual implementations, or in an abstract manner by means of declarative specifications. Both frameworks admit a simple refinement relation, defined in terms of traces, which corresponds to safe-substitutivity. We define asynchronous parallel composition, conjunction and quotient, and prove that the induced equivalence is a congruence for these operations. It is straightforward to extend our framework with disjunction and hiding. The simplicity of our formalism facilitates reasoning about the temporal ordering of interactions needed for assume-guarantee inference. Although not considered in this paper, our framework supports reasoning about safety properties in the context of assume-guarantee. Liveness properties may also be considered, but this requires the introduction of quiescence or infinite behaviours, the latter being achieved with the help of ω-automata techniques.

Acknowledgments. The authors are supported by EU FP7 project CONNECT and ERC Advanced Grant VERIWARE. We would also like to thank the anonymous reviewers for their insightful comments.

References

1. Larsen, K.G., Nyman, U., Wasowski, A.: Modal I/O automata for interface and product line theories. In Nicola, R.D., ed.: ESOP. Volume 4421 of LNCS., Springer (2007) 64–79
2. de Alfaro, L., Henzinger, T.A.: Interface automata. SIGSOFT Softw. Eng. Notes **26** (2001) 109–120
3. Raclet, J.B., Badouel, E., Benveniste, A., Caillaud, B., Passerone, R.: Why are modalities good for Interface Theories? In: Proc. 9th International Conference on Application of Concurrency to System Design. ACSD '09, IEEE Computer Society (2009) 119–127
4. Raclet, J.B., Badouel, E., Benveniste, A., Caillaud, B., Legay, A., Passerone, R.: Modal Interfaces: Unifying Interface Automata and Modal Specifications. In: Proc. 7th International Conference on Embedded Software. EMSOFT '09, ACM (2009) 87–96
5. Lynch, N.A., Tuttle, M.R.: An introduction to input/output automata. CWI Quarterly **2** (1989) 219–246
6. Lüttgen, G., Vogler, W.: Conjunction on processes: Full abstraction via ready-tree semantics. Theor. Comput. Sci. **373** (2007) 19–40
7. Lüttgen, G., Vogler, W.: Ready simulation for concurrency: It's logical! Inf. Comput. **208** (2010) 845–867
8. Brookes, S.D., Hoare, C.A.R., Roscoe, A.W.: A theory of communicating sequential processes. J. ACM **31** (1984) 560–599
9. de Alfaro, L., Henzinger, T.A.: Interface-based design. In Broy, M., Grünbauer, J., Harel, D., Hoare, T., eds.: Engineering Theories of Software Intensive Systems. Volume 195 of NATO Science Series II: Mathematics, Physics and Chemistry. Springer (2005) 83–104
10. de Alfaro, L.: Game models for open systems. In Dershowitz, N., ed.: Verification: Theory and Practice. Volume 2772 of LNCS. Springer (2004) 192–193
11. Doyen, L., Henzinger, T.A., Jobstmann, B., Petrov, T.: Interface theories with component reuse. In: Proc. 8th ACM international conference on Embedded software. EMSOFT '08, ACM (2008) 79–88
12. Bhaduri, P., Ramesh, S.: Interface synthesis and protocol conversion. Form. Asp. Comput. **20** (2008) 205–224
13. Nicola, R.D., Segala, R.: A process algebraic view of input/output automata. Theor. Comput. Sci. **138** (1995) 391–423
14. Tretmans, J.: Model-based testing and some steps towards test-based modelling. In Bernardo, M., Issarny, V., eds.: SFM. Volume 6659 of LNCS., Springer (2011) 297–326
15. Josephs, M.B.: Receptive process theory. Acta Inf. **29** (1992) 17–31
16. Josephs, M.B., Kapoor, H.K.: Controllable delay-insensitive processes. Fundam. Inf. **78** (2007) 101–130
17. Raclet, J.B.: Residual for component specifications. Electr. Notes Theor. Comput. Sci. **215** (2008) 93–110
18. Chen, T., Chilton, C., Jonsson, B., Kwiatkowska, M.: A Compositional Specification Theory for Component Behaviours. Technical Report CS-RR-12-01, Department of Computer Science, University of Oxford (2012)
19. Glabbeek, R.J.v.: Full abstraction in structural operational semantics (extended abstract). In: Proc. 3rd Intl Conference on Methodology and Software Technology: Algebraic Methodology and Software Technology, Springer (1994) 75–82