Fundamenta Informaticae 82 (2008) 1–48
IOS Press

Interface Automata with Complex Actions: Limiting Interleaving in Interface Automata

Shahram Esmaeilsabzali*, Nancy A. Day, Farhad Mavaddat
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, N2L 3G1 Canada
sesmaeil@cs.uwaterloo.ca; nday@cs.uwaterloo.ca; fmavaddat@cs.uwaterloo.ca

Abstract. Many formalisms use interleaving to model concurrency. To describe some system behaviours appropriately, we need to limit interleaving. For example, in a component-based system, we might wish to limit interleaving to force the inputs to a method to arrive together in order. In Web services, the arrival of XML messages consisting of multiple simple parts should not be interleaved with the behaviour of another component. We introduce *interface automata with complex actions* (IACA), which adds complex actions to de Alfaro and Henzinger's interface automata (IA). A complex action is a sequence of actions that may not be interleaved with actions from other components. The composition operation and refinement relation are more involved in IACA compared to IA, and we must sacrifice associativity of composition. However, we argue that the advantages of having complex actions make it a useful formalism. We provide proofs of various properties of IACA and discuss the use of IACA for modelling Web services.

1. Introduction

Interleaving is a common choice to model the concurrent behaviour of components of a system. Interleaving means that at each point in time only one component takes a step. The behaviours of the overall system consist of all possible interleavings of the actions of the components. Many formalisms, both algebraic and non-algebraic, have adopted interleaving semantics, *e.g.*, [20, 25, 23, 22, 21]. It provides an intuitive and elegant means for modelling and reasoning about systems' behaviours. Henzinger, Manna, and Pnueli, in [19], characterize interleaving semantics as *adequate*, meaning they can distinguish between systems with different behaviours, and *abstract*, meaning that unnecessary details of systems are ignored in modelling.

*Address for correspondence: David R. Cheriton School of Computer Science, University of Waterloo Waterloo, Ontario, Canada N2L 3G1

However, for some systems that are specified in an interleaving semantics, it is not possible to choose a uniform abstraction level for the atomicity of their actions. Some software artifacts have multiple constituent elements that represent a single unit, thus, we may wish to group multiple actions such that their behaviour cannot be interleaved with the behaviour of another component. In these cases, interleaving is not always appropriate to characterize system behaviour accurately because it is based on the assumption that the behaviours of a concurrent system include *all* possible orderings of actions in a system.

We introduce *interface automata with complex actions (IACA)*, designed to model component-based and service-oriented systems. An interface automaton with complex actions uses a limited version of interleaving for its concurrency semantics. A *complex action* consists of a sequence of distinct normal actions that cannot be interleaved with the behaviour of another component. In a component-based system, at its signature level, a method of a component can be characterized by the method's name and a set of parameters. Such parameters are elements of a single software artifact. Some formalisms choose to model a method by abstracting away its details using, for example, only its name, *e.g.*, [3]. To be able to model and reason about the details of the communication of the parameters, a formalism should be considered that does not interleave the arrival of the inputs of a component with another component, when they are composed. Thus, we require complex actions to model the semantics of the concurrent behaviour of a component-based system at this level of detail. As another example, in Web services[1], communication with service requesters and other Web services occurs through complex XML messages, which are streams of data delimited appropriately into multiple simple messages. We would like to model complex XML messages of Web services as non-interruptible software artifacts. By being able to model the details of artifacts such as complex XML messages of Web services and methods' parameters, we can precisely reason about the compatibility and composition of the systems that use these artifacts as their building blocks.

Various approaches have been proposed for grouping multiple actions together, which can be categorized into two major classes: *atomic actions* [8, 16][2], and *action refinement* [1, 15], which allow for refining an action into multiple actions. Many of these approaches are proposed in a process algebraic context. We are interested in defining an automata-based model with complex actions. Also, we want our model to comply as closely as possible with the class of *interface models*, introduced by de Alfaro and Henzinger [4, 3], which are a class of formalisms suitable for describing component-based systems. Interface models are designed to promote the compatibility of components such that different components or services can work together to achieve a desired behaviour. Interface models assume a *helpful environment*, which supplies needed inputs and receives all outputs. They also have well-formedness criteria that support *top-down design*, which means that a refinement of a component can be substituted for the original in the context of its composition with other components. Composition must be commutative and associative. The top-down design property makes it possible to refine an initial design into a more detailed design. One of our goals in IACA has been to maintain these properties of interface models.

Our IACA model is an extension of de Alfaro and Henzinger's *interface automata (IA)* [3, 5], which is an interface model. An interface automaton captures the temporal aspects of the functionality of a

[1]Throughout the paper we use the term "Web service" to refer to "an invocation" of the functionality of a Web service provider. Of course, many concurrent invocations of the same Web service provider can coexist. Our IACA model is meant to be suitable to model a single invocation of the functionality of a Web service provider, and reason about the compatibility of such invocations, rather than modelling Web service providers themselves.

[2]To avoid ambiguity, in our formalism, for actions that consist of multiple normal actions, we chose the term "complex actions" instead of "atomic actions."

component along with its assumptions about its environment. Syntactically, an IA is a transition system over disjoint sets of input, output, and hidden actions, similar to Input/Output Automata [22]. It captures what the component assumes about its environment via the temporal order of the input and output behaviours described by the transitions and states. Using this information about the environmental assumptions of the components, it is possible to reason about the compatibility of multiple components that are supposed to collaborate.

The distinguishing characteristics of IA are the way interface-based design and compatibility checking, in a top-down manner, are manifested in its composition operator and refinement relation. The composition of two IAs is a new IA, which combines their functionalities through an interleaving semantics. When the two components are ready to synchronize, *i.e.*, one has an output that is received by the other as an input (or vice versa), a hidden action is created in the composition. The environmental assumptions of the two components are combined to describe the environmental assumptions of the composition. In this interleaving, there is no notion of real time or a clock, only the temporal order of transitions are modelled. IA composition is both commutative and associative. Commutativity along with associativity allows for the *incremental* development of a system because the order in which different components are composed does not affect the result of the composition.

The refinement of an IA is defined based on the assumed helpful environment. The refinement relation is *conservative* in allowing only the refinements of a system that do not introduce new assumptions that the original system does not have. A refinement of an IA must have fewer input assumptions about its environment than its parent, and as such can replace the parent in all contexts. With respect to outputs, the refinement of an IA should not constrain the environment with new outputs that the parent IA did not issue to the environment. The composition operator and refinement relation for IA together establish the top-down design property of IA.

IACA extends IA by allowing the grouping of a sequence of transitions into a complex transition. Complex transitions provide the means for specifying: (i) the assumptions that a system might have about receiving a sequence of inputs from the environment in a non-interruptible manner; and (ii) that a non-interruptible sequence of output actions from a component should be sent to its environment. Both concepts cannot be modelled in IA, since an action cannot be decomposed into its constituent elements in IA; and furthermore, during the composition of two IAs, their states and transitions are interleaved, and the grouping of actions cannot be preserved. Not unexpectedly, the introduction of complex actions in IACA makes the definition of the composition and refinement more complicated than in IA.

Interleaving usually relies on a notion of *atomicity*, which is the idea that an action in the system is indivisible in time [24, 25]. In IA, for example, a transition on an input, output, or hidden action cannot be interrupted with another. An important question is how to choose the granularity of atomic actions when modelling a system. Usually, the granularity of actions can be chosen at a single, uniform level of abstraction. However, in component-based and service-oriented systems, we come across situations where a uniform level of abstraction is not "adequate" (using Henzinger *et al.*'s terminology in [19]).

In IACA, we are interested in having a model that allows us to model multiple levels of abstraction together. For example, consider the functionality of payment via a credit card, $pay(credit\_card\_no, amount)$. Assume that the value for the $credit\_card\_no$ parameter is received via a banking machine, and the value for the $amount$ parameter is received via a function that retrieves the price of an item. The challenge is how to choose an appropriate level of abstraction that is adequate for modelling the different functionalities of such a system. If we choose to consider pay, without its parameters, as an indivisible action, then our model is not precisely modelling the system because it does not allow for reasoning

about the way the parameters $credit\_card\_no$ and $amount$ must be received from the components that produce them. Alternatively, modelling pay as two actions, namely, $credit\_card\_no$ and $amount$ in a sequence, is not correct because they do not represent meaningful atomic actions, and furthermore, they can be interrupted when interleaved with actions of other components. What we really need is to have a *complex action*, which sequentially groups $credit\_card\_no$ and $amount$ as a single action, namely pay. The need to have complex actions arises because systems often have heterogeneous components that can only be modelled appropriately if different levels of abstraction for modelling are available. In our example, $credit\_card\_no$ can be an atomic output action of the banking machine, as well as, a part of the complex action pay of the payment functionality. As an alternative design, we could have considered an IACA model where a complex action is comprised of a set (or multi-set) of actions. But a sequential grouping of actions is needed, rather than a set (or multi-set), in order to capture the order between the constituent actions of the complex action.

Two IACAs can be composed with each other, if they are *composable*. The composability criteria for IACA composition extends IA's composability criteria by requiring that any two complex actions, belonging to two IACAs, either have disjoint constituent elements, or the order of actions in one complex action is a *subsequence* of the order of actions in the other. This extra composability requirement, along with the requirement that a complex action consists of distinct normal actions, is the least restrictive constraint that we could have defined and still guarantee the well-formedness of the IACA composition.

The main challenge for IACA is how to define a composition operator and a refinement relation that do not allow interleaving of complex actions, but support top-down design as much as possible. IACA composition, compared to IA composition, avoids all unnecessary interleavings within the complex actions, and thus yields a smaller result. Compared to other formalisms with complex actions, in IACA we must respect the helpful environment, which means that in composition a state should not be reached where one component would have to wait for communication from the other. With complex actions, synchronization is more involved than synchronization in IA, because there can be multiple constituent parameters that can potentially synchronize. We tried to define IACA composition in such a way that the necessary synchronization can happen and interface models' well-formedness criteria are also achieved. However, it became apparent to us that achieving associativity of the composition operator is not possible. Other approaches that have used types of interleaving with complex actions have also suffered from the loss of associativity, *e.g.*, A^2CCS [16]. Despite the non-associativity of composition, we believe that IACA is useful for modelling software artifacts with complex actions.

Complex actions complicate the composition operator of IACA, which, similar to other interface models, must combine the environmental assumptions of the components about their environments, as well as, modelling the way they affect their environments by issuing their outputs to it. The environmental assumption of an input complex action not only relies on the current state of the environment, but also, on whether all its input actions in its sequence of transitions can be satisfied. The requirement to consider all transitions in the sequence, means that a *binary* IACA composition operator cannot necessarily reason about the compatibility of more than two components; because there could always exist a third helpful (or unhelpful) component that would affect the compatibility and environmental assumptions of a complex action. As discussed later in Section 4.5, the inability to reason fully about the compatibility of components in the presence of complex actions, is the reason that IACA composition is not associative. In other words, in the presence of complex actions, to reason about the compatibility of components of a system, we need to consider all of them, rather than considering their compatibility in pairs.

Our definition of composition leads to a natural definition for the IACA refinement relation. IACA's refinement relation is intuitively comparable with concepts in programming languages such as subclasses and optional parameters for function calls. IACA refinement follows the IA refinement principle: a more refined IACA should not constrain its environment more than the original IACA does. As such, a more refined IACA should provide more inputs (*i.e.*, should be more controllable by its environment than the original IACA), and issue less outputs (*i.e.*, should constrain its environment less than the original IACA). Additionally, a more refined IACA, in comparison with the original IACA, can have complex actions with some more (optional) input elements at the end of its complex actions. Conversely, a more refined IACA can choose to omit some of the output elements at the end of the original IACA's complex actions.

The remainder of the paper is organized as follows. We begin by providing background information on IA. In Section 3, we describe IACA. In Sections 4 and 5, we present the IACA composition operator and the IACA refinement relation, respectively. In Section 6, we compare IACA with similar models that support complex actions, and summarize and discuss future work in Section 7. Appendix A includes the proofs of the lemmas and theorems stated in the paper. Throughout the paper, we provide examples of the use of IACA to model Web services. This paper is an extended version of our original work [13, 12]. Here, we have revised some of our design decisions, and have included a more thorough treatment of IACA composition and refinement, along with the outlines of the proofs of the properties of IACA.

2. Background: Interface Automata

Interface automata (IA) [3, 5], introduced by de Alfaro and Henzinger, is an automata-based model designed to be suitable for specifying component-based systems. IA is a part of the class of models called *interface models* [4], which are intended to specify concisely *how* systems can be used, and follow some well-formedness criteria that are suitable for component-based design. The two main characteristics of interface models are that they assume a *helpful environment* and support *top-down* design. A helpful environment for an interface provides the inputs it needs and always accepts all its outputs. Therefore, interfaces are optimistic, and do not usually specify all possible behaviours of the systems. For example, they often do not include fault scenarios. *Top-down design* is based on a notion of refinement, which relates two instances of a model. A refinement of a model can be substituted for the original. In a well-formed interface model, a binary *composition* operator and a *refinement* relation are defined. Composition is both commutative and associative. Top-down design means that for three interface models P, P', Q, and the composition of P and Q, $P \parallel Q$, if P' refines P, *i.e.*, $P' \preceq P$, then: $(P' \parallel Q) \preceq (P \parallel Q)$.

To provide the necessary context to present our IACA model, in the remainder of this section we provide a primer on IA, which is based on the presentation of IA in [3]. IAs are well-formed interface models. They are syntactically similar to Input/Output Automata [22], but have different semantics. Figure 1 shows two IAs. The arrows on top represent the inputs of the system and arrows at the bottom represent the outputs of the system. The initial state of an IA is designated by an arrow with no source. IA *Prod* is a component (service) that receives either an ISBN or a name of a book, and based on the request provides the price of the book in Canadian or US dollars. The author of a book is also an output of the system. Input actions are followed by "?", and output actions by "!". IA *Pay* carries out a credit card payment by receiving an amount in Canadian dollars and a credit card number, and produces either a reference number for a successful transaction or an error number.

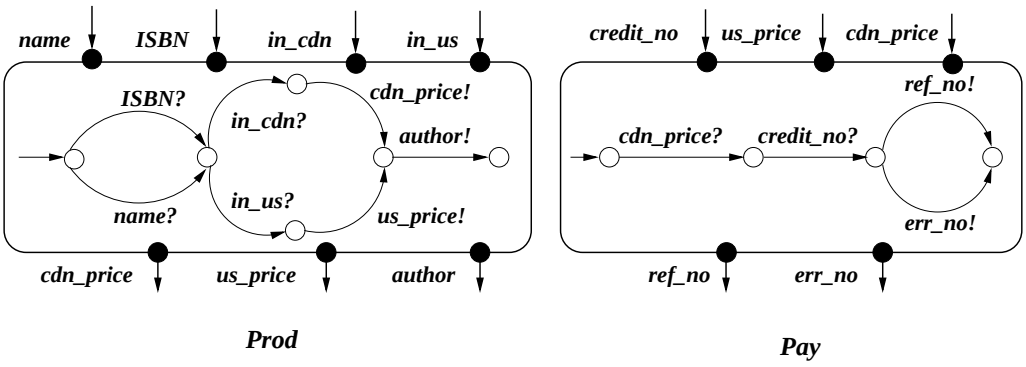


Figure 1: Two IAs: *Prod* and *Pay*.

Definition 2.1. An *interface automaton* (IA), $P = \langle \mathcal{V}_P, i_P, \mathcal{A}^I_P, \mathcal{A}^O_P, \mathcal{A}^H_P, \mathcal{T}_P \rangle$, consists of $\mathcal{V}_P$ a finite set of states, $i_P \in \mathcal{V}_P$ the initial state, $\mathcal{A}^I_P$, $\mathcal{A}^O_P$ and $\mathcal{A}^H_P$, which are disjoint sets of input, output, and hidden actions, respectively, and $\mathcal{T}_P$ the set of transitions between states such that $\mathcal{T}_P \subseteq \mathcal{V}_P \times \mathcal{A}_P \times \mathcal{V}_P$, where $\mathcal{A}_P = \mathcal{A}^I_P \cup \mathcal{A}^O_P \cup \mathcal{A}^H_P$. Well-formed IAs must be *deterministic* [6, 2], *i.e.,* for any two transitions (u, a, v) and (u, a, l), $v = l$.

For an IA P, and a state $u \in \mathcal{V}_P$: $\mathcal{A}^I_P(u)$, $\mathcal{A}^O_P(u)$, and $\mathcal{A}^H_P(u)$ represent the sets of input, output, and hidden actions, respectively, that have transitions with source u.

2.1. IA Composition

The composition of two IAs is a subset of all possible interleaved transitions of the two IAs, except for the actions that are shared. On a shared action (input of one IA and output of the other IA), the two IAs synchronize in the composition.

The composition in IA is defined only for two *composable* IAs.

Definition 2.2. IAs P and Q are *composable* if they do not take any of the same inputs and do not produce the same outputs, and their hidden actions do not overlap with other actions:

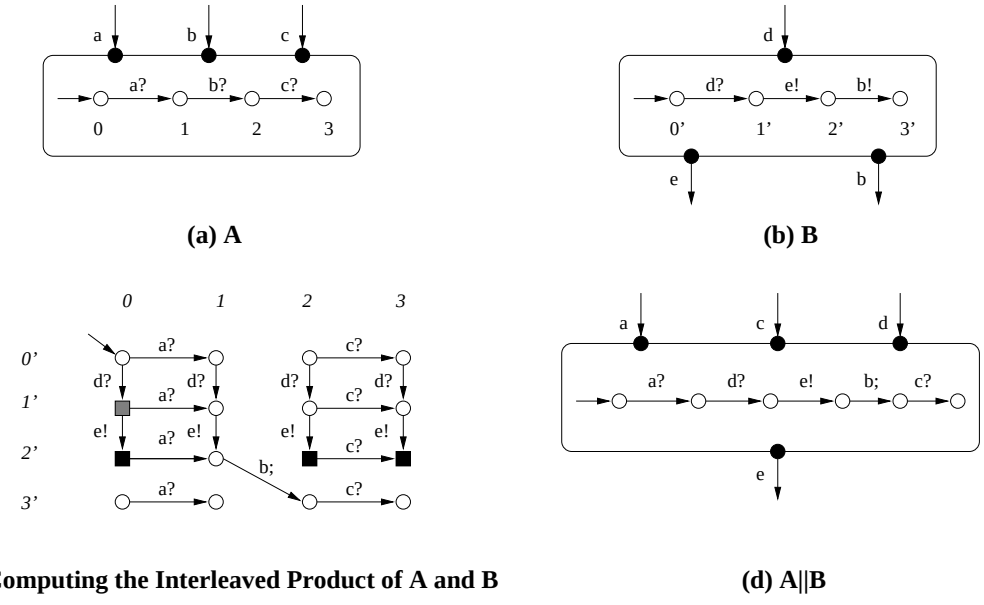
$$(\mathcal{A}^I_P \cap \mathcal{A}^I_Q) = (\mathcal{A}^O_P \cap \mathcal{A}^O_Q) = (\mathcal{A}^H_P \cap \mathcal{A}_Q) = (\mathcal{A}^H_Q \cap \mathcal{A}_P) = \emptyset$$

For two composable IAs, their set of shared actions is defined as follows.

Definition 2.3. Considering two IAs, P and Q, their set of *shared actions* is defined as:

$$Shared(P, Q) = \mathcal{A}_P \cap \mathcal{A}_Q$$

A hidden action in the composition is created by shared actions. During composition when an output action of one component is internally consumed by an input action of another component, a synchronization happens and the two actions are reduced to a hidden action on a single transition. To define composition, we first define the interleaved product of actions and states of two composable IAs:



(a) A **(b) B**

(c) Computing the Interleaved Product of A and B **(d) A||B**

Legend: ■ **Autonomously Reachable to Illegal State** ■ **Illegal State**

Figure 2: Two composable IAs, A and B, and their composition $A \parallel B$.

Definition 2.4. For two composable IAs, P and Q, their *interleaved product*, $P \otimes Q$, is defined as follows.

$$
\begin{array}{lcl}
\mathcal{V}_{P\otimes Q} & = & \mathcal{V}_P \times \mathcal{V}_Q \\
i_{P\otimes Q} & = & (i_P, i_Q) \\
\mathcal{A}^I_{P\otimes Q} & = & (\mathcal{A}^I_P \cup \mathcal{A}^I_Q) \backslash Shared(P,Q) \\
\mathcal{A}^O_{P\otimes Q} & = & (\mathcal{A}^O_P \cup \mathcal{A}^O_Q) \backslash Shared(P,Q) \\
\mathcal{A}^H_{P\otimes Q} & = & \mathcal{A}^H_P \cup \mathcal{A}^H_Q \cup Shared(P,Q) \\
\mathcal{T}_{P\otimes Q} & = & \left(\begin{array}{lll}
\{((p,q),a,(p',q)) & | & (p,a,p') \in \mathcal{T}_P \wedge a \notin Shared(P,Q) \wedge q \in \mathcal{V}_Q\} \\
\cup \ \{((p,q),a,(p,q')) & | & (q,a,q') \in \mathcal{T}_Q \wedge a \notin Shared(P,Q) \wedge p \in \mathcal{V}_P\} \\
\cup \ \{((p,q),a,(p',q')) & | & (p,a,p') \in \mathcal{T}_P \wedge (q,a,q') \in \mathcal{T}_Q \wedge a \in Shared(P,Q)\}
\end{array} \right)
\end{array}
$$

In Figure 2 (c), the matrix of states and transitions represents the interleaved product of IAs A and B. IAs A and B are composable and b is a shared action of the two IAs. Transitions on non-shared actions are interleaved, and a transition is created on a hidden action to represent the synchronization on b (from state (1,2') to state (2,3')). Hidden actions have ";" following their names.

The composition of two IAs is defined based on their interleaved product. In the composition of two composable IAs, because of the assumption of a helpful environment, neither component should have to wait to synchronize, *i.e.,* if one component is ready to issue an output action, the other should be ready to receive the action immediately. A state of the interleaved product where one component would have to wait is considered an *illegal state*. The composition of two IAs does not include their illegal states.

Definition 2.5. An *illegal state* of two composable IAs, P and Q, is a state in which one of the IAs has an output action, belonging to their set of shared actions, enabled in that state and the other IA does not have any transition using the corresponding action.

$$Illegal(P,Q) = \left\{ \quad (p,q) \in \mathcal{V}_P \times \mathcal{V}_Q \mid \exists a \in Shared(P,Q) \quad \cdot \left(\begin{array}{l} a \in \mathcal{A}_P^O(p) \wedge a \notin \mathcal{A}_Q^I(q) \\ \vee \\ a \in \mathcal{A}_Q^O(q) \wedge a \notin \mathcal{A}_P^I(p) \end{array} \right) \right\}$$

An illegal state is a combined state of the two components in which one is ready to issue an output and the other is not ready to receive it. If the interleaved product of two IAs is *open, i.e.*, there are some inputs that do not belong to the set of shared actions of the two IAs, then a helpful environment may be able to avoid an illegal state by not providing the inputs that lead the product to its illegal states. Inputs of an open system allow its environment to control it. However, if the interleaved product of two IAs is *closed, i.e.*, all actions are either output or hidden, and thus uncontrollable, then an environment cannot avoid the illegal states of the interleaved product, and the composition becomes empty.

The composition of two composable IAs, P and Q, can be formally defined based on a notion of *environment*.

Definition 2.6. An *environment* E for an IA P is itself an IA, and satisfies the following conditions:

- E and P are composable, and
- E is non-empty, and
- E can receive all of P's outputs, *i.e.*, $\mathcal{A}_E^I = \mathcal{A}_P^O$, and
- $Illegal(P,E) = \emptyset$.

Next, we define the notions of *compatible states* of two composable IAs, and then define composition of two composable IAs.

Definition 2.7. Given two composable IAs, P and Q, a pair $(p,q) \in \mathcal{V}_P \times \mathcal{V}_Q$ is *compatible* if there is an environment E for $P \otimes Q$ such that no state in $Illegal(P,Q) \times \mathcal{V}_E$ is reachable in $(P \otimes Q) \otimes E$ from the state $((p,q), i_E)$. The set $Cmp(P,Q)$ is the set of all such states.

Definition 2.8. Given two composable IAs, P and Q, the composition of P and Q, denoted as $P \parallel Q$, is an IA with the same actions as $P \otimes Q$, states $\mathcal{V}_{P\parallel Q} = Cmp(P,Q)$, initial state $i_{P\parallel Q} = \{i_{P\otimes Q}\} \cap Cmp(P,Q)$, and transitions $\mathcal{T}_{P\parallel Q} = \mathcal{T}_{P\otimes Q} \cap \{Cmp(P,Q) \times \mathcal{A}_{P\otimes Q} \times Cmp(P,Q)\}$.

If $i_{P\otimes Q} \in Cmp(P,Q)$, then IA $P \parallel Q$ is defined.

Let us look at our example in Figure 2 again. In state (0,2') of the interleaved product of A and B, in Figure 2 (c), B is immediately ready to send b but A is not yet ready to receive it. State (0,2') is an illegal state, as are (2,2') and (3,2'), shown in black boxes. These states are not included in the composition. States and transitions on paths that lead to these illegal states where the path consists *entirely* of output and hidden actions are also not included; such paths are called *autonomous paths*. The environment does not have any control over output and hidden transitions. State (0,1') (shown with a grey-filled box) is enabled with e! and as such can lead to an illegal state. Thus, state (0,1') is itself an illegal state, because

even in the presence of a helpful environment, execution may lead to illegal state (0,2'), from state (0,1'). The illegal states of an automaton are the undesired states that can be autonomously reached, without the help of the environment of the automaton; the environment cannot stop the automaton reaching them. The illegal states, as well as, unreachable states, along with their corresponding transitions, are eliminated from the composition of two IAs. The IA resulting from the composition of IAs A and B, $A \parallel B$, is shown in Figure 2 (d).

The composition of two composable IAs is non-empty if their interleaved product's initial state belongs to their set of compatible states. IA composition, in practice, can be computed by a simple backtracking algorithm, shown in Figure 3, which starts from all illegal states and considers any other states that can reach them through autonomous paths, as illegal states themselves. The states that survive the backtracking algorithm are compatible states of the interleaved product and will appear in the composition.

Algorithm RemoveIllegal(P, Q, $\mathcal{T}_{P\otimes Q}$, $\mathcal{V}_{P\otimes Q}$) ;
Variables : $\mathcal{T}_{P\|Q}$, $\mathcal{V}_{P\|Q}$, K, $temp$;
begin
  $K = Illegal(P, Q)$;
  repeat
    `/* Backtrack one transition to identify more illegal states */`
    $temp = \{(p,q) |\ \exists((p,q), a, (p',q')) \in \mathcal{T}_{P\otimes Q} \cdot (p',q') \in K\ \wedge a \in (\mathcal{A}^{O}_{P\otimes Q} \cup \mathcal{A}^{H}_{P\otimes Q})\}$;
    `/* Add to the set of illegal states */`
    $K = K \cup temp$;
  until $temp == \emptyset$;
  $\mathcal{V}_{P\|Q} = \mathcal{V}_{P\otimes Q} \backslash K$;
  $\mathcal{T}_{P\|Q} = \mathcal{T}_{P\otimes Q} \backslash \{((p,q), a, (p',q'))|\ ((p,q) \in K) \vee ((p',q') \in K)\}$;
  return $\mathcal{T}_{P\|Q}$, $\mathcal{V}_{P\|Q}$;
end

Figure 3: Algorithm for computing the composition of two composable IAs P and Q.

As another example, the composition of the IAs *Prod* and *Pay* in Figure 1 is shown in Figure 4. All states where *Prod* generates the output us_price and *Pay* is not ready to receive it are considered illegal states and are not included in the composition. In this example, by removing such illegal states, transitions on *in_us?* are removed. However, action *in_us* still appears as an input of the composition.

IA composition, for composable IAs, is both commutative and associative [3].

Theorem 2.1. For all IAs P and Q, either they are not composable, or $P \parallel Q$ is defined and is equal to $Q \parallel P$.

Theorem 2.2. For all IAs P, Q and R, either some of the IAs are not composable, or $(P \parallel Q) \parallel R = P \parallel (Q \parallel R)$.

2.2. Refinement

IA Q refines IA P, if Q provides the services of P. Q can have more inputs than P, but no more outputs. As an example, *GenPay* in Figure 5 refines *Pay* in Figure 1. *GenPay* provides more services than *Pay*

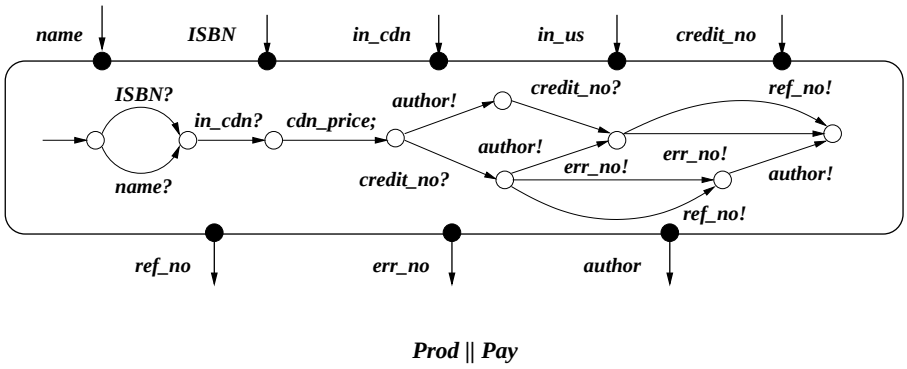


Figure 4: $Prod \parallel Pay$ is the composition of two composable IAs in Figure 1.

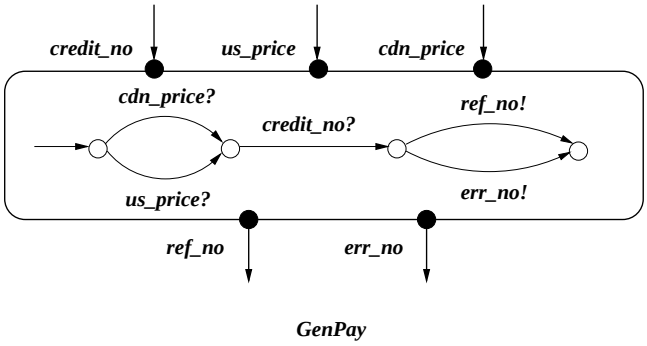


Figure 5: *GenPay* refines *Pay* in Figure 1.

since it can carry out payments in both Canadian and US dollars. As an interface model, top-down design guarantees that $(Prod \parallel GenPay)$ refines $(Prod \parallel Pay)$.

The refinement of IA is defined using a refinement relation between the states of two IAs. If IA Q refines IA P, denoted as $Q \preceq P$, then an *alternating simulation relation* [7] exists between the states of Q and P.[3] For simplicity, we refer to the alternating simulation relation as the refinement relation on states. For $q \in \mathcal{V}_Q$ and $p \in \mathcal{V}_P$, q refines p, denoted as $q \preceq p$, if q has more than or the same input actions as p, and less than or the same output actions as p. Also, for any state q' reachable from q, immediately or through hidden actions, there is a corresponding state p' similarly reachable from p such that $q' \preceq p'$. All states reachable from a state *only* through hidden transitions are considered the same state for the purposes of refinement. The initial state of Q must refine the initial state of P.

[3] An alternating simulation relation [7] is different from a regular simulation relation, in that an alternating simulation relation can be defined for composite systems consisting of multiple components. P' is related to P, a component of a composite system, by an alternating simulation, if P' can mimic the transitions of P, and furthermore it does not constrain the other components in the composite system more than P does. It is also possible to define an alternating simulation relation with respect to a set of components belonging to a composite system; *i.e.*, P can be a set of components. All other components not being involved in an alternating simulation, are considered as the environment.

Next, the IA refinement relation is formally described. For a state $u \in \mathcal{V}_P$, $closure_P(u)$ is a set containing u and all states that can be reached from u through internal transitions. The *externally enabled input* and *externally enabled output* actions of a state can then be defined.

Definition 2.9. For IA P, and state $u \in \mathcal{V}_P$, the set of *externally enabled input* ($ExtEn^I_P(u)$) and *externally enabled output* ($ExtEn^O_P(u)$) actions are:

$$\begin{aligned} ExtEn^I_P(u) &= \{a |\ \forall u' \in closure_P(u) \cdot a \in \mathcal{A}^I_P(u')\} \\ ExtEn^O_P(u) &= \{a |\ \exists u' \in closure_P(u) \cdot a \in \mathcal{A}^O_P(u')\} \end{aligned}$$

The externally enabled input and output actions are used to model the fact that an environment cannot detect the hidden transitions that may happen in a certain state of an IA. An environment does not distinguish between all *different* states that it may reach through hidden transitions. As such, to define a proper refinement for an IA P, for any of its states $u \in \mathcal{V}_P$, it is necessary to identify all the states that P can move through its hidden transitions, *i.e.*, identify $closure_P(u)$. A proper refinement of P, say Q, needs only to be receptive to input actions, *i.e.*, output actions from the environment, that are enabled in *all* states belonging to $closure_P(u)$. In other words, the environment of P is careful not to invoke an input that may not be enabled in one of the states in $closure_P(u)$, and Q would be a proper refinement, in its corresponding state for u, if it can handle all input actions (and possibly more) that belong to $ExtEn^I_P(u)$. Output transitions may be issued from any of the states in $closure_P(u)$, and therefore Q would be a proper refinement if, in its corresponding state for u, it does not issue more outputs to the environment than P does in any of the states belonging to $closure_P(u)$.

The refinement relation between two IAs holds, if the pair of their initial states are in that relation, and furthermore, all appropriate reachable states from a pair of states belonging to the refinement relation, also belong to the refinement relation. The following three definitions formally describe the IA refinement relation.

Definition 2.10. For a state u and an externally enabled action $a \in ExtEn^I_P(u) \cup ExtEn^O_P(u)$, the *externally reachable states* are defined as:

$$Dest_P(u, a) = \{u' |\ \exists (r, a, u') \in \mathcal{T}_P \cdot r \in closure_P(u)\}$$

Definition 2.11. The binary relation, *alternating simulation* $\preceq\ \subseteq \mathcal{V}_Q \times \mathcal{V}_P$, between two states $q \in \mathcal{V}_Q$ and $p \in \mathcal{V}_P$, denoted as $q \preceq p$, holds if the following conditions are true:

- $ExtEn^I_P(p) \subseteq ExtEn^I_Q(q)$
  (q has more, or the same, externally enabled inputs than p), and
- $ExtEn^O_P(p) \supseteq ExtEn^O_Q(q)$
  (q has less, or the same, externally enabled outputs than p), and
- $\forall\, a \in (ExtEn^I_P(p) \cup ExtEn^O_Q(q)) \cdot \forall\, q' \in Dest_Q(q, a) \cdot \exists\, p' \in Dest_P(p, a) \cdot q' \preceq p'$
  (For all common externally enabled actions at p and q, and all states reachable from q via those actions, there exists a state in P that simulates q's behaviours).

The above definition establishes a recursive refinement relation where a state in the refined IA can have more inputs and less outputs enabled, than the original interface. Furthermore, all neighboring states in the refined IA have a corresponding state in the original IA (the third condition of the relation). The first two conditions in the above definition guarantee that a refinement of an IA can be substituted with the original in all composition contexts, while maintaining the top-down design property. If the first two conditions are not met, then a refinement of an IA P, say P', when composed with another IA, say Q, could introduce some illegal states that P would not have introduced when composed with Q.

Definition 2.12. IA Q *refines* IA P, $Q \preceq P$, if:

- $\mathcal{A}_P^I \subseteq \mathcal{A}_Q^I$, and
- $\mathcal{A}_P^O \supseteq \mathcal{A}_Q^O$, and
- $i_Q \preceq i_P$.

The first two conditions maintain the composability and top-down design properties when an IA is replaced with its refinement, and the third condition propagates the simulation relation to appropriate states of the two IAs.

IA refinement is a reflexive and transitive relation.

3. Interface Automata with Complex Actions

Interface automata with complex actions (IACA) extends interface automata with the ability to declare a sequence of transitions to be a *complex action*. The transitions within a complex action are not interleaved with transitions from another component during IACA composition. Complex actions in an IACA are meant to model software artifacts such as methods or complex messages, which can have multiple constituent elements but should not be interleaved with other actions in composition. As an example, Figure 6 shows an IACA, *CompPay*, with a complex action *pay_in_cdn*, represented by the dashed transition. IACA *CompPay* is similar to IA *Pay* of Figure 1, except that input actions *cdn_price* and *credit_no* cannot be interleaved with actions from another component during composition with another IACA. Intuitively, we do not want *pay_in_cdn* to be interleaved since it represents a single method that has more than one parameter. From the perspective of interface models, the complex action *pay_in_cdn* captures an environmental assumption of IACA *CompPay*; namely, the assumption that *all* of the parameters of *pay_in_cdn* should arrive in the correct order.

A complex action represents either an input or an output behaviour, and thus should consist entirely of either inputs or outputs, possibly along with some hidden actions in the sequence.[4] Complex actions can only be a linear sequence of transitions. The states within a complex action are called *internal states* and are represented by grey-filled circles. In Figure 6, states 1, 3, and 4 are normal states of *CompPay* and state 2 is the only internal state. Internal states act as *delimiters* between constituent elements of a complex action. Compared to a normal state, an internal state has exactly one incoming and one outgoing transition.[5]

[4] Allowing for the mixture of input and output actions to appear in a complex action is unnecessary for our purpose, which is checking for the compatibility of the components of a system. If we were interested in modelling the "computation" of systems,

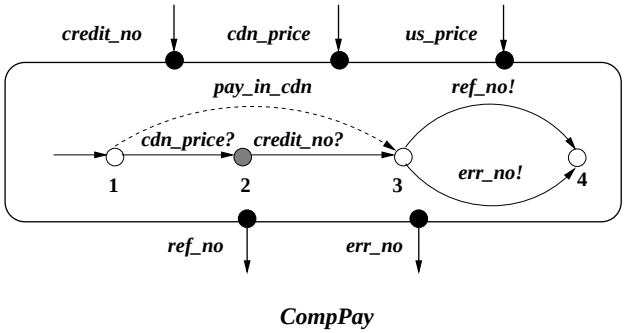


Figure 6: IACA *CompPay* represents similar functionality as IA *Pay* in Figure 1.

For convenience, we first introduce IACA and its components formally, and then separately, define the well-formedness criteria of an IACA. In the subsequent sections, we define the IACA composition operator and the IACA refinement relation.

Definition 3.1. An *interface automaton with complex actions* (IACA), $P = \langle \mathcal{V}_P^N, \mathcal{V}_P^{Int}, i_P, \mathcal{A}_P^I, \mathcal{A}_P^O, \mathcal{A}_P^H, \mathcal{A}_P^C, \mathcal{T}_P, \phi_P \rangle$, has the following elements:

- $\mathcal{V}_P^N$ is the set of normal states.
- $\mathcal{V}_P^{Int}$ is the set of internal states. $\mathcal{V}_P^N \cap \mathcal{V}_P^{Int} = \emptyset$. The internal states are the ones inside a complex action. We denote $\mathcal{V}_P = \mathcal{V}_P^N \cup \mathcal{V}_P^{Int}$ as the set of all states.
- i_P is the initial state. $i_P \in \mathcal{V}_P^N$.
- $\mathcal{A}_P^I, \mathcal{A}_P^O, \mathcal{A}_P^H$ are disjoint sets of input, output and hidden actions. These are normal, non-complex actions. We denote $\mathcal{A}_P^N = \mathcal{A}_P^I \cup \mathcal{A}_P^O \cup \mathcal{A}_P^H$.
- $\mathcal{A}_P^C$ is the set of complex actions where $\mathcal{A}_P^C \cap \mathcal{A}_P^N = \emptyset$. We denote $\mathcal{A}_P = \mathcal{A}_P^C \cup \mathcal{A}_P^N$.
- $\mathcal{T}_P \subseteq \mathcal{V}_P \times \mathcal{A}_P^N \times \mathcal{V}_P$ is the set of normal (non-complex) transitions. We require that each $v \in \mathcal{V}_P^{Int}$ is the source of *exactly* one transition and the destination of *exactly* one transition in $\mathcal{T}_P$.
- $\phi_P \subseteq \mathcal{V}_P^N \times \mathcal{A}_P^C \times \mathcal{V}_P^N$ is the set of complex transitions. There is a unique injective function frag_P mapping each complex transition in P to its sequence of non-complex transitions in $\mathcal{T}_P$, which is called its *complex fragment*. A complex fragment is an alternating sequence of states and normal actions, such that for $(u, c, v) \in \phi_P$, $\mathit{frag}_P(u, c, v) = \langle u, a_0, s_0, a_1, s_1, \ldots, s_{n-1}, a_n, v \rangle$, and
  - $\forall i \, (0 \leq i < n) \cdot s_i \in \mathcal{V}_P^{Int}$
    (all s_i's are internal states), and

instead of their interfaces, then complex actions consisting of a mixture of input and output actions would have been helpful.

[5]The distinction between normal and internal states is analogous to the distinction between "abstract"(observable) and "concrete"(invisible) states in action refinement for process algebras [17].

- $(\forall i\ (0 \leq i \leq n) \cdot a_i \in (\mathcal{A}_P^I \cup \mathcal{A}_P^H)) \vee (\forall i\ (0 \leq i \leq n) \cdot a_i \in (\mathcal{A}_P^O \cup \mathcal{A}_P^H))$
  (all actions either belong to the union of input and hidden normal actions, or belong to the union of output and hidden normal actions), and
- $\forall i, j\ (0 \leq i, j \leq n) \cdot (i \neq j) \Rightarrow (a_i \neq a_j)$
  (The constituent actions of a complex fragment are distinct actions), and
- $((u, a_0, s_0) \in \mathcal{T}_P) \wedge ((s_{n-1}, a_n, v) \in \mathcal{T}_P) \wedge \forall i\ (0 < i < n) \cdot (s_{i-1}, a_i, s_i) \in \mathcal{T}_P$
  (every transition is a non-complex transition).

As an example, in Figure 6, $\mathit{frag}_{CompPay}(1, \mathit{pay\_in\_cdn}, 3) = \langle 1, cdn\_price, 2, credit\_no, 3\rangle$.
We use the following notation:

- For an IACA P, and a state $s \in \mathcal{V}_P$: $\mathcal{A}_P^I(s)$, $\mathcal{A}_P^O(s)$, $\mathcal{A}_P^H(s)$, and $\mathcal{A}_P^C(s)$ represent the sets of input, output, hidden, and complex actions, respectively, that have transitions with source s. Set $\mathcal{A}_P^N(s)$ represents the set of normal actions that have transitions with source s. For $a \in \mathcal{A}_P^N(s)$ or $a \in \mathcal{A}_P^C(s)$, we say a is *enabled at* s, or s is *enabled with* a.

- For an IACA P, an alternating sequence of states and normal actions $\langle s_0, a_0, s_1, \cdots, s_n\rangle$ is an *execution* of P, if s_0 is reachable from i_P, and $(s_i, a_i, s_{i+1}) \in \mathcal{T}_P$, for all $0 \leq i < n$. A complex fragment of P is an "execution" of P that maps a complex transition to its execution.

- For a complex transition (u, c, v) in ϕ_P, we use functions $\mathit{sched}_P(u, c, v)$ and $\mathit{states}_P(u, c, v)$, to represent the *schedule* and the *states* of that complex transition, respectively. As an example, $\mathit{sched}_{CompPay}(1, \mathit{pay\_in\_cdn}, 3) = \langle cdn\_price, credit\_no\rangle$, and $\mathit{states}_{CompPay}(u, c, v) = \{1, 2, 3\}$.

- For internal states of an IACA P, we define function $\mathit{CompTran}_P$, which maps an internal state to its complex transition. As an example, $\mathit{CompTran}_{CompPay}(2) = (1, pay\_in\_cdn, 3)$.

An IACA P is well-formed if: each of its complex actions is associated with a unique schedule, each of its its internal states is associated with a complex transition, it is deterministic, and none of its states has more than one incoming transitions with the same action; *i.e.*, it is *incoming deterministic*. Formally,

Definition 3.2. An IACA $P = \langle \mathcal{V}_P^N, \mathcal{V}_P^{Int}, i_P, \mathcal{A}_P^I, \mathcal{A}_P^O, \mathcal{A}_P^H, \mathcal{A}_P^C, \mathcal{T}_P, \phi_P\rangle$ is *well-formed* if:

1. $(\forall (u, c, v) \in \phi_P \cdot \forall (u', c, v') \in \phi_P \cdot \mathit{sched}_P(u, c, v) = \mathit{sched}_P(u', c, v')) \wedge$
   $(\forall (u, c, v) \in \phi_P \cdot \forall (u', d, v') \in \phi_P \cdot \mathit{sched}_P(u, c, v) = \mathit{sched}_P(u', d, v') \Rightarrow d = c)$

   (Complex transitions with the same complex actions have the same schedules, and complex actions with the same schedules have the same complex action names), and

2. $\forall u \in \mathcal{V}_P^{Int} \cdot \mathit{CompTran}_P(u) \in \phi_P$

   (Every internal state is associated with a complex transition; *i.e.*, $\mathit{CompTran}_P$ is a total function), and

3. $\forall (u, a, v) \in \mathcal{T}_P \cdot \forall (u, a, v') \in \mathcal{T}_P \cdot (v = v')$

   (Similar to IA, IACA is deterministic), and

4. $\forall(u, a, v) \in \mathcal{T}_P \cdot \forall(u', a, v') \in \mathcal{T}_P \cdot (u \neq u') \Rightarrow (v \neq v')$

   (An IACA should be *incoming deterministic*; *i.e.*, two transitions with the same action should have distinct destinations).

In the remainder of the paper, we are only interested in the well-formed IACAs, and whenever we refer to an IACA, we mean a well-formed IACA.

The constraint that an internal state has exactly one incoming transition and exactly one outgoing transition, together with the first two IACA well-formedness constraints above, guarantees that a complex transition is associated with a unique sequence of non-complex transitions, and a unique complex action.

The third well-formedness constraint, *i.e.*, that an IACA is deterministic, is necessary to ensure that the composition of two composable IACAs would yield internal states that are the source of not more than one transition. In IA, the same constraint is necessary, but for a different reason, namely to ensure the associativity of the IA composition.

The fourth well-formedness constraint, *i.e.*, that an IACA is incoming deterministic, is necessary to ensure that the composition of two composable IACAs will not include internal states that have more than one incoming transition. More precisely, this constraint is meant to disallow transitions on the shared actions of two composable IACAs that make a certain state not incoming deterministic.[6] The fourth constraint disallows ambiguity in identifying the starting state of a complex transition, when two IACAs are composed.[7]

In some cases, the fourth well-formedness constraint can be avoided by transforming an IACA that is not incoming deterministic to an IACA that is equivalent to it, and is incoming deterministic. For example in Figure 7, IACA A, shown in part (a) of Figure 7, is not incoming deterministic, but can be transformed into IACA A', as shown in part (b) of Figure 7, which is incoming deterministic. The two IACAs are bisimilar in a process algebraic sense; which is a sufficient equivalence measure for our model. Similar techniques as are used in finite automaton minimization algorithms [9, 27], can be used to lump such states as 1 and 2 in A, into state $1'$ in A'. If such lumped states still yield a non-incoming deterministic IACA, then for acyclic IACAs, it is possible to eliminate the states that violate the incoming deterministic property. In part (d) of Figure 7, such states are eliminated by cloning states such as 3 and 4 in IACA B, shown in section (c) of Figure 7, into states $3\_1'$ and $3\_2'$, and $4\_1'$ and $4\_2'$ in IACA B', respectively. Again B and B' are bisimilar.[8]

Every IA is an IACA with empty sets of complex transitions and complex actions. We call the IA that consists of all parts of an IACA except the complex transitions and the complex actions, the *equivalent IA* to an IACA. Formally:

Definition 3.3. Given an IACA, $P = \langle \mathcal{V}_P^N, \mathcal{V}_P^{Int}, i_P, \mathcal{A}_P^I, \mathcal{A}_P^O, \mathcal{A}_P^H, \mathcal{A}_P^C, \mathcal{T}_P, \phi_P \rangle$, $Q = \langle \mathcal{V}_P, i_P, \mathcal{A}_P^I, \mathcal{A}_P^O, \mathcal{A}_P^H, \mathcal{T}_P \rangle$ is the *equivalent* IA for P.

[6] The set of shared actions of two composable IACAs P and Q, as will be defined later, similar to IA, is $(\mathcal{A}_P^N \cap \mathcal{A}_Q^N)$.

[7] The fourth well-formedness constraint for IACAs is analogous to the constraint of the Promela language [21], where `goto` statements referring to destinations within an `d_step` transitions are disallowed.

[8] For IACAs with cycles, this expensive procedure is not helpful since it may produce a non-deterministic IACA and/or could produce an IACA with more than one initial state. We believe that the question of constructing a bisimilar IACA that is incoming deterministic for an arbitrary IACA, which is not incoming deterministic, is an interesting research question.

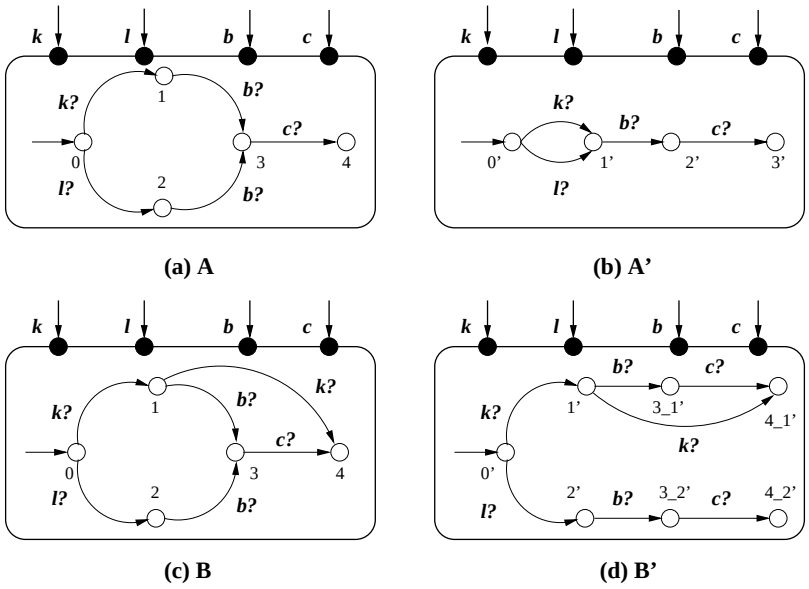


Figure 7: Eliminating states with more than one incoming transition with the same action.

4. IACA Composition

IACA composition is a binary function mapping two composable IACAs into a new IACA. The main difference between IACA and IA composition is that the transitions within a complex action are not interleaved in IACA composition. This behaviour is necessary to ensure all parameters of a method call or a message arrive together in the exact order required. Synchronization between actions of the two components may occur within a complex fragment, but each complex fragment in the two IACAs maintains its sequence of actions in the composition (possibly with some actions having become hidden actions). Synchronization, similar to IA, creates hidden transitions. In the composition of two IACAs, similar to IA composition, we combine the environmental assumptions of the two IACAs. The assumption that a complex fragment has about its environment, *i.e.,* that its schedule cannot be interleaved, is preserved when it is composed with another IACA, by requiring the other IACA to provide the appropriate actions, that belong to their shared actions, exactly in the order of the schedule of the complex fragment. Furthermore, either the whole complex fragment is present in the composition or the complex fragment should not appear in the composition at all. The interleaving in IACA composition of two composable IACAs is a subset of the interleaving in IA composition of the equivalent IAs of the two IACAs.

Figure 8 shows the composition of IACA *CompPay*, previously shown in Figure 6, and component *Prod* (now viewed as an IACA), previously shown as an IA in Figure 1. The transitions within the complex transition *pay_in_cdn* in *CompPay* are not interleaved with other actions, and *pay_in_cdn* remains a complex action in $Prod \parallel CompPay$. The composition involved a synchronization between the input *cdn_price* in *CompPay* and the output *cdn_price* in *Prod*, which results in a hidden action within the complex action *pay_in_cdn*. The output action *author* cannot appear between *cdn_price* and *credit_no* actions because they are elements of complex transition *pay_in_cdn*. Because of the way IACA composition limits interleaving, $Prod \parallel CompPay$ is smaller in size than the similar IA composition in Figure 4.

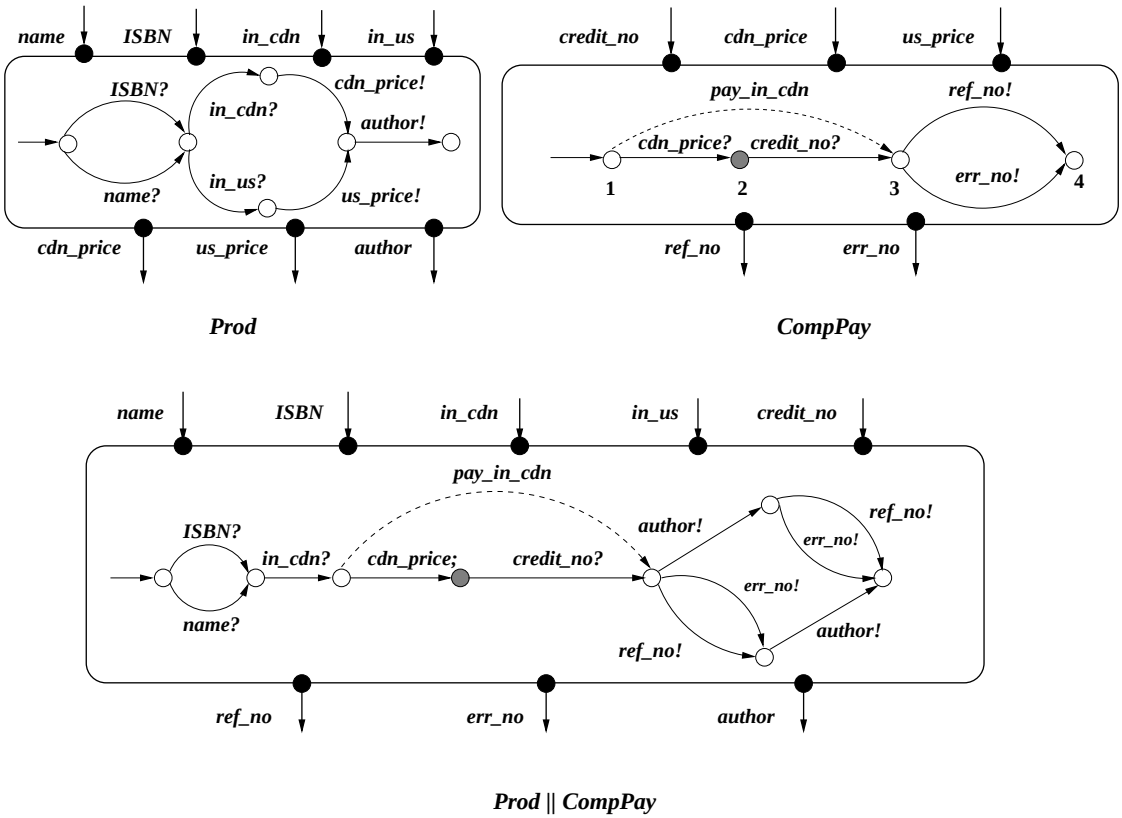


Figure 8: $Prod \parallel CompPay$

Before formally defining the composability criteria for the composition of two IACAs, we need to define the *subsequence* relation between two sequences.

Definition 4.1. We call a sequence r a *subsequence* of s, denoted $r \Subset s$, if r contains a subset of s elements in the same order as the elements appear in s.

The composability criteria in IACA includes the composability criteria of IA plus extra restrictions to handle complex transitions. If the schedules of two complex transitions, one from each IACA, overlap without the schedule of one being a subsequence of the schedule of the other, then the two IACAs are not composable.

Definition 4.2. Two IACAs are *composable*, if their equivalent IAs are composable, and for any pair of complex transitions belonging to the two IACAs, either they are *action-disjoint*, *i.e.*, their sets of schedules are disjoint, or the schedule of one is a *subsequence* of the other.[9]

[9]Initially in [13], we had a more restricted composability criteria, which required that for any two complex transitions belonging to two composable IACAs, either their schedules should not overlap or the schedule of one should be a prefix of another, rather than its subsequence. We thank the reviewer who suggested exploring alternative design decisions.

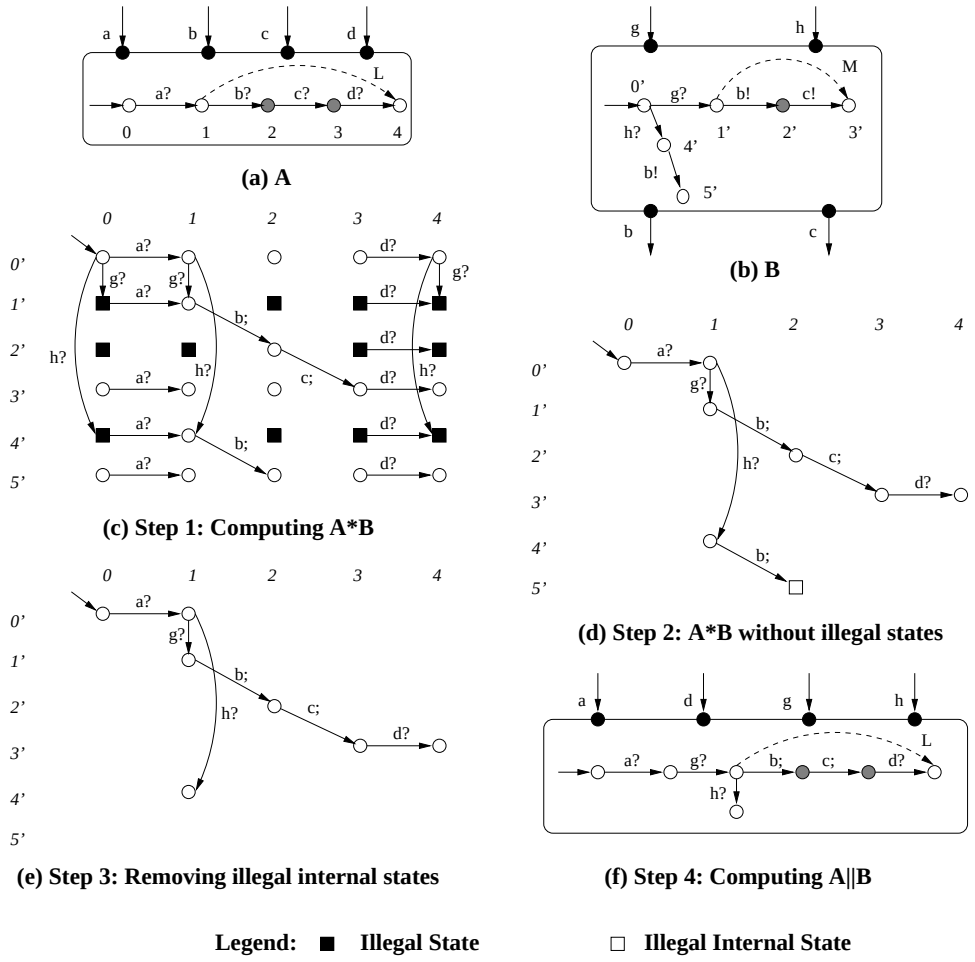


Figure 9: Computing IACA composition for two composable IACAs.

As an example, in Figure 9, the equivalent IAs of IACAs A and B are composable, and furthermore, the schedule of complex transition $(1', M, 3')$ of B, is a subsequence of the schedule of the complex transition $(1, L, 4)$, of A, and thus A and B are composable.

By requiring the subsequence relation between the schedules of any two complex transitions when they are composed, it is possible to guarantee that complex transitions in the result can be associated with the schedule of one of the complex transitions of the two IACAs. In Section 4.5, we discuss why we made this design choice from the alternatives, along with other design choices made so far.

For complex transitions belonging to two composable IACAs we define the following notation:

Definition 4.3. For two composable IACAs, P and Q, and complex transitions $(p, c, p') \in \phi_P$ and $(q, d, q') \in \phi_Q$, (p, c, p') *embeds* (q, d, q'), and (q, d, q') *is embedded* in (p, c, p'), if *sched*$(q, d, q') \Subset$ *sched*(p, c, p').

Similar to IA, the set of shared normal actions for two composable IACAs can be defined.

Definition 4.4. For two composable IACAs, P and Q, their set of *shared actions* is:

$$Shared(P,Q) = \mathcal{A}_P^N \cap \mathcal{A}_Q^N$$

We define the composition of two IACAs using the following sequence of constructive steps:

1. Compute the *interleaved product* of two IACAs, $P * Q$. The IACA interleaved product differs from the IA interleaved product, in that interleavings of transitions within complex actions are not allowed.
2. Remove *illegal states* from the interleaved product, as for IA.
3. Remove *illegal internal states* from the result of (2). We call the result the *legal interleaved product*, $P \circledast Q$.
4. Compute the complex transitions. The result is $P \parallel Q$.

Next, we define these steps formally, and use the simple IACAs of Figure 9 to illustrate these steps. All proofs are presented in Appendix A.

4.1. Step 1: Interleaved Product for IACA

In the first step, we compute the *interleaved product* of two composable IACAs P and Q, $P * Q$. Part (c) of Figure 9 illustrates the first step of computing the composition of two IACAs in our example.

Definition 4.5. For two composable IACAs, P and Q, we define their *interleaved product*, $P * Q$, as follows:

$$\mathcal{V}_{P*Q} = \mathcal{V}_P \times \mathcal{V}_Q$$
$$\mathcal{V}^{Int}_{P*Q} = \{(p,q) \in (\mathcal{V}_P \times \mathcal{V}_Q) |\ (p \in \mathcal{V}_P^{Int}) \vee (q \in \mathcal{V}_Q^{Int})\}$$
$$i_{P*Q} = (i_P, i_Q)$$
$$\mathcal{A}^I_{P*Q} = (\mathcal{A}^I_P \cup \mathcal{A}^I_Q) \setminus Shared(P,Q)$$
$$\mathcal{A}^O_{P*Q} = (\mathcal{A}^O_P \cup \mathcal{A}^O_Q) \setminus Shared(P,Q)$$
$$\mathcal{A}^H_{P*Q} = \mathcal{A}^H_P \cup \mathcal{A}^H_P \cup Shared(P,Q)$$
$$\mathcal{A}^C_{P*Q} = \mathcal{A}^C_P \cup \mathcal{A}^C_Q$$
$$\mathcal{T}_{P*Q} =$$

$$\{((p,q),a,(p',q)) \mid (p,a,p') \in \mathcal{T}_P \wedge a \notin Shared(P,Q) \wedge p \in \mathcal{V}_P^N \wedge q \in \mathcal{V}_Q^N\} \quad (1)$$
$$\cup\ \{((p,q),a,(p,q')) \mid (q,a,q') \in \mathcal{T}_Q \wedge a \notin Shared(P,Q) \wedge q \in \mathcal{V}_Q^N \wedge p \in \mathcal{V}_P^N\} \quad (2)$$
$$\cup\ \{((p,q),a,(p',q')) \mid (p,a,p') \in \mathcal{T}_P \wedge (q,a,q') \in \mathcal{T}_Q \wedge a \in Shared(P,Q)\} \quad (3)$$
$$\cup\ \{((p,q),a,(p',q)) \mid (p,a,p') \in \mathcal{T}_P \wedge a \notin Shared(P,Q) \wedge p \in \mathcal{V}_P^{Int}\} \quad (4)$$
$$\cup\ \{((p,q),a,(p,q')) \mid (q,a,q') \in \mathcal{T}_Q \wedge a \notin Shared(P,Q) \wedge q \in \mathcal{V}_Q^{Int}\} \quad (5)$$

We call the sets $\mathcal{V}_{P*Q}$ and $\mathcal{T}_{P*Q}$, the set of *interleaved states* and *interleaved transitions* of P and Q, respectively. Throughout the paper, whenever we refer to set `(1)`, `(2)`, `(3)`, `(4)`, or `(5)`, we mean the corresponding set of transitions in Definition 4.5.

The interleaved product of two composable IACAs is not an IACA. In particular, we cannot define ϕ_{P*Q} for the interleaved product of P and Q. We can define a new IACA only after following the four steps of constructing the composition.

For each state $(p, q) \in \mathcal{V}_{P*Q}$, if either p or q is an internal state, then (p, q) is an internal state. For example, in part (c) of Figure 9, all states associated with state 3 are internal states, and have transitions on action $d?$.

Set `(3)` consists of synchronizations that can happen between two IACAs on shared actions. The composability criteria guarantee that each action is an input action of one IACA and an output of the other IACA. Set `(1)` consists of the transitions that do not involve shared actions, and are transitions that exit normal states of P. Set `(4)` is for transitions originating from internal states of P; in the reachable part, these are not interleaved with the transitions from Q, because Q is in a normal state. Sets `(2)` and `(5)` are similarly defined as sets `(1)` and `(4)`, but for the transitions of IACA Q.

The definition of sets `(4)` and `(5)` allow for situations where some internal states could have two outgoing transitions. In the interleaved product, all such internal states belong to the internal states of two action-disjoint complex transitions, and are all unreachable, as will be shown in Lemma 4.1. Sets `(4)` and `(5)` are meant to allow a complex transition, belonging to P or Q respectively, to follow their schedules when a non-shared action is enabled on one of their internal states. Both sets disallow the transitions on the normal states of the other IACA to interleave their executions within a complex fragment.

Definition 4.5 guarantees that each internal state of the interleaved product has a maximum of one transition entering it, and a maximum of one transition exiting it. We defer the proof of this claim to the step where we remove all illegal states, in Section 4.3, and then prove a stronger claim for internal states.

4.2. Step 2: Remove illegal states

Similar to IA composition, the states where a shared output action is enabled, but there is no input ready from the other IACA are illegal states, and are not included in the composition. In this step, we remove the *illegal states* from the interleaved product. The illegal states of two IACAs are defined the same as the illegal states for their equivalent IAs, as defined in Definition 2.5. For IACA, in addition to illegal states that comprise of pairs of normal states, there could also exist illegal states whose pairs of states comprise of one or two internal states.

Part (c) of Figure 9 shows the illegal states of the two IACAs as black-filled boxes in the interleaved product. Part (d) of Figure 9 shows the result of removing illegal states for the interleaved product of A and B. Similar to IA, we remove all illegal states, as well as, transitions that are on paths to illegal states that consist *entirely* of output and hidden actions, *i.e.*, autonomous paths. We then remove all unreachable states and transitions. A similar algorithm to the one in Figure 3, which we used for removing illegal states for IAs, can be used to remove illegal states in IACA.

4.3. Step 3: Remove illegal internal states

By removing illegal states of an interleaved product, some internal states do not appear in the interleaved product of two IACAs. As such, there could exist some partial complex fragments that cannot be associated with any complex actions. The internal states of the interleaved product that have no outgoing transitions are called *illegal internal states*, and should be removed. Illegal internal states cannot be part of a complex action because all complex fragments must terminate in a normal state. For example, the empty box in part (d) of Figure 9 is an illegal internal state.

We use a backtracking algorithm to remove all illegal internal states. Removing an illegal internal state can create a new illegal internal state. The algorithm in Figure 10 needs to backtrack until no

illegal internal states exist in the interleaved product of two IACAs. However, unlike removing illegal states, we do not have to remove autonomous paths that lead to illegal internal states, because a helpful environment can avoid creating partial complex fragments by not issuing certain inputs. In fact, in Step 2, we have already removed such autonomous paths, and at this stage a helpful environment is able to avoid the undesired states. Part (e) of Figure 9 shows the result of removing illegal internal states of the interleaved product of part (d). Next, we define the set of illegal internal states and present the algorithm for removing them.

Definition 4.6. Given two composable IACAs, P and Q, and their sets of interleaved transitions, $\mathcal{T}_{P*Q}$, and interleaved states, $\mathcal{V}_{P*Q}$, the set of *illegal internal states* of P and Q, $IllegalInternal(P, Q)$, is computed by the algorithm in Figure 10, and represents all internal states of an interleaved product that have no outgoing transitions.

Algorithm RemoveIllegalInternal(P, Q, $\mathcal{T}_{P*Q}$, $\mathcal{V}_{P*Q}$) ;
Variables : $\mathcal{T}_{P\circledast Q}$, $\mathcal{V}_{P\circledast Q}$, K, $temp$, $IllegalInternal(P, Q)$;

begin
$K = \{(p, q) \in \mathcal{V}^{Int}_{P*Q} \mid \nexists((p, q), a, (p', q')) \in \mathcal{T}_{P*Q}\}$;
repeat
`/* Backtrack to identify more illegal internal states */`
$temp = \{(p, q) \in \mathcal{V}^{Int}_{P*Q} \mid \exists((p, q), a, (p', q')) \in \mathcal{T}_{P*Q} \cdot (p', q') \in K\}$;
`/* Add to the set of illegal internal states */`
$K = K \cup temp$;
until $temp == \emptyset$;
$IllegalInternal(P, Q) = K$;
$\mathcal{V}_{P\circledast Q} = \mathcal{V}_{P*Q} \backslash IllegalInternal(P, Q)$;
$\mathcal{T}_{P\circledast Q} = \mathcal{T}_{P*Q} \backslash$
$\{((p, q), a, (p', q'))| \ ((p, q) \in IllegalInternal(P, Q)) \vee ((p', q') \in IllegalInternal(P, Q))\}$;
return $\mathcal{T}_{P\circledast Q}$, $\mathcal{V}_{P\circledast Q}$;
end

Figure 10: Algorithm for removing illegal internal states and creating the legal interleaved product of two composable IACAs P and Q.

For two composable IACAs, P and Q, we call the result of removing all illegal internal states and transitions defined on them, as carried out in the algorithm in Figure 10, their *legal interleaved product*, denoted as $P \circledast Q$. Other elements of $P \circledast Q$ are defined similar to the ones defined for their interleaved product in Definition 4.5, with the necessary adjustments for the states and transitions that do not belong to $\mathcal{V}_{P\circledast Q}$ and $\mathcal{T}_{P\circledast Q}$ anymore, or have become unreachable. We still need to compute $\phi_{P\circledast Q}$.

The important property of an internal state in a legal interleaved product is that it is the source of exactly one transition and the destination of exactly one transition. We start by showing that an internal state can be the source of not more than one transition. Before stating the corresponding lemma, we need to prove the following lemma.

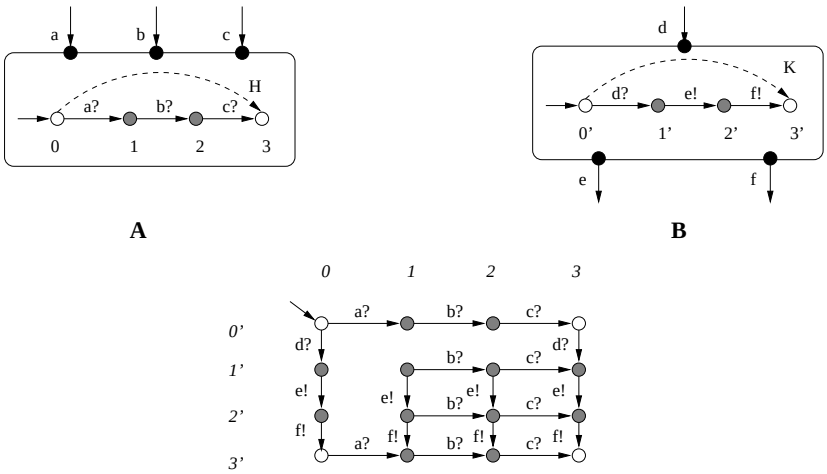


Figure 11: Interleaved product of two IACAs when their complex transition are action-disjoint. The unreachable states of $A * B$ are characterized in Lemma 4.1.

Lemma 4.1. For two composable IACAs, P and Q, and two action-disjoint complex transitions $(p, c, p') \in \phi_P$ and $(q, d, q') \in \phi_Q$, where $\mathit{frag}_P(p, c, p') = \langle p, c_0, p_0, \ldots, p_{n-1}, c_n, p' \rangle$ and $\mathit{frag}_Q(q, d, q') = \langle q, d_0, q_0, \ldots, q_{m-1}, d_m, q' \rangle$, all internal states $(p_i, q_j) \in \mathcal{V}^{Int}_{P*Q}$, where $(0 \leq i < n)$ and $(0 \leq j < m)$, are unreachable in $P * Q$.

Figure 11 illustrates an example of the result of interleaving two action-disjoint complex transitions, according to Definition 4.5, and the resulting unreachable internal states.

Lemma 4.2. Given two composable IACAs, P and Q, and their legal interleaved product $P \circledast Q$, for an internal state $(p, q) \in \mathcal{V}^{Int}_{P \circledast Q}$, there is exactly one transition in $\mathcal{T}_{P \circledast Q}$ with source (p, q).

Having shown that an internal state of the legal interleaved product of two composable IACAs can be the source of exactly one transition, next, we show that it is also the destination of exactly one transition.

Lemma 4.3. Given two composable IACAs, P and Q, and their legal interleaved product $P \circledast Q$, for an internal state $(p, q) \in \mathcal{V}^{Int}_{P \circledast Q}$, there is exactly one transition in $\mathcal{T}_{P \circledast Q}$ with destination (p, q).

Next, in Lemma 4.4, we show that each internal state $(p, q) \in \mathcal{V}^{Int}_{P \circledast Q}$ belongs to an *execution fragment*. We later, in Section 4.4, show that such an execution fragment indeed represents a complex fragment of P or Q.

Lemma 4.4. Given two composable IACAs, P and Q, and their legal interleaved product $P \circledast Q$, for an internal state $(p, q) \in \mathcal{V}^{Int}_{P \circledast Q}$, there exists exactly one execution $\langle (p_0, q_0), a_0, (p_1, q_1), \ldots, (p_n, q_n) \rangle$, called *the execution fragment* of state (p, q), denoted as $\Delta(p, q)$, and all of the following conditions are true:

- $\exists(p_j, q_j) \cdot (0 < j < n) \wedge (p_j = p) \wedge (q_j = q)$
  ((p, q) belongs to the execution fragment), and
- $\forall i \cdot (0 < i < n) \Rightarrow ((p_i, q_i), a_i, (p_{i+1}, q_{i+1})) \in \mathcal{T}_{P \circledast Q}$
  (All transitions of the execution fragment are in the legal interleaved product), and
- $(p_0, p_n \in \mathcal{V}_P^N) \wedge (q_0, q_n \in \mathcal{V}_Q^N) \wedge (\forall i \cdot (0 < i < n) \Rightarrow (p_i, q_i) \in \mathcal{V}_{P \circledast Q}^{Int})$
  (The source and destination states of the sequence are normal states and the other states are internal states).

To derive the complex transitions of a legal interleaved product, we need to prove that an execution fragment is not *partially overlapped*.[10] The non-partially overlapped property of an execution fragment ensures that during IACA composition, the schedule of complex fragments are properly embedded within each other. In the presence of partially overlapped execution fragments, it is not possible to ensure that the result of a composition yields only complex fragments that can be mapped to a complex action of the two composed IACAs.

Before proceeding to state the non-partially overlapped property of execution fragments in the legal interleaved product of two composable IACAs, let us first formally define a partially overlapped execution fragment.

Definition 4.7. Given two IACAs, P and Q, an internal state $(p, q) \in \mathcal{V}_{P \circledast Q}^{Int}$, its execution fragment $\Delta(p, q) = \langle (p_0, q_0), a_0, (p_1, q_1), \ldots, a_{n-1}, (p_n, q_n) \rangle$, $\Delta(p, q)$ is *partially overlapped*, if there exists a pair of complex transitions: $(u, c, u') \in \phi_P$ and $(v, d, v') \in \phi_Q$, with complex fragments $\mathit{frag}_P(u, c, u') = \langle u, c_0, u_0 \ldots, u_{x-1}, c_x, u' \rangle$ and $\mathit{frag}_Q(v, d, v') = \langle v, d_0, v_0, \ldots, v_{y-1}, d_y, v' \rangle$, and the following condition is true:

$$
\begin{array}{ll}
(\ (\exists(p_i, q_i) \cdot \exists(p_j, q_j) \cdot (0 \leq i, j < n) \wedge (i < j) \wedge (p_i = u) \wedge (q_j = v)) \wedge & \texttt{(a)} \\
(\exists(p_k, q_k) \cdot \exists(p_l, q_l) \cdot (0 < k, l \leq n) \wedge (j < k < l) \wedge (p_k = u') \wedge (q_l = v')) \wedge & \texttt{(b)} \\
(p_j \in \mathit{states}_P(u, c, u')) \wedge (q_k \in \mathit{states}_Q(v, d, v')) \ ) & \texttt{(c)} \\
\vee & \\
(\ (\exists(p_i, q_i) \cdot \exists(p_j, q_j) \cdot (0 \leq i, j < n) \wedge (i < j) \wedge (q_i = v) \wedge (p_j = u)) \wedge & \texttt{(d)} \\
(\exists(p_k, q_k) \cdot \exists(p_l, q_l) \cdot (0 < k, l \leq n) \wedge (j < k < l) \wedge (q_k = v') \wedge (p_l = u')) \wedge & \texttt{(e)} \\
(q_j \in \mathit{states}_Q(v, d, v')) \wedge (p_k \in \mathit{states}_P(u, c, u')) \ ) & \texttt{(f)}
\end{array}
$$

Figure 12 illustrates an informal picture of a partially overlapped execution fragment. Figure 12 is only provided for illustrative purposes, and as shown in the next lemma, the situation shown in the figure cannot happen for the legal interleaved product of two composable IACAs.

Part `(a)` of the above predicate specifies an execution fragment where complex transition $(u, c, u') \in \phi_P$ starts executing its transitions before $(v, d, v') \in \phi_Q$ starts. Part `(b)` of the above predicate specifies that complex transition (u, c, u') finishes its execution before complex transition (v, d, v') does. Parts `(a)` and `(b)` of the predicate implicitly require that: $p_j \in \mathcal{V}_P^{Int}$ and $q_k \in \mathcal{V}_Q^{Int}$, because otherwise (p_j, q_j) and (p_k, q_k) cannot belong to $\mathcal{V}_{P \circledast Q}^{Int}$. Part `(c)` of the above predicate ensures that the execution fragment $\Delta(p, q)$ includes the complex fragments of the two complex transitions, because each normal

[10] The notion of "execution fragment" is defined in Lemma 4.4.

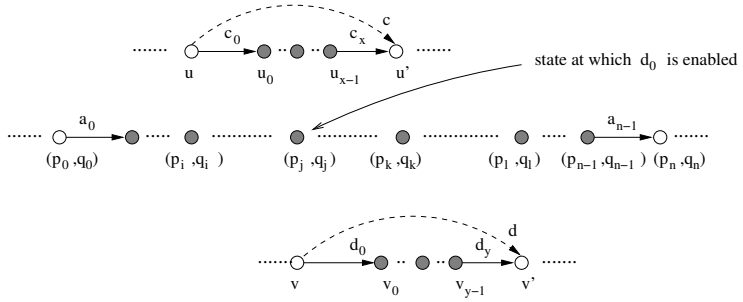


Figure 12: An informal illustration of a partially overlapped execution fragment, according to Definition 4.7.

state, such as u and v, can possibly be enabled with multiple complex and/or non-complex transitions. Parts `(d)`, `(e)`, and `(f)` of the above predicate describe another possibility for partially overlapped execution fragments, but this time for the case when complex transition $(v, d, v') \in \phi_Q$ starts executing its transitions before $(u, c, u') \in \phi_P$.

Lemma 4.5. Given two composable IACAs, P and Q, and their legal interleaved product $P \circledast Q$, for an internal state $(p, q) \in \mathcal{V}^{Int}_{P \circledast Q}$, its execution fragment $\Delta(p, q) = \langle (p_0, q_0), a_0, (p_1, q_1), \ldots, (p_n, q_n) \rangle$ is non-partially overlapped.

In the next section, we show how complex transitions can be derived from the legal interleaved product of two composable IACAs.

4.4. Step 4: Deriving the complex transitions

The fourth and final step in computing the composition of two composable IACAs, is to determine the complex transitions that are associated with execution fragments. First, we define some necessary notation.

Definition 4.8. Given two composable IACAs, P and Q, an internal state $(p, q) \in \mathcal{V}^{Int}_{P \circledast Q}$, and its execution fragment $\Delta(p, q) = \langle (p_0, q_0), a_0, (p_1, q_1), \ldots, a_{n-1}, (p_n, q_n) \rangle$, $\Delta(p, q)$ can be projected into two alternating sequence of states and actions: $\pi_P(\Delta(p, q)) = \langle p_0, a_0, p_1, \ldots\ a_{n-1}, p_n \rangle$ and $\pi_Q(\Delta(p, q)) = \langle q_0, a_0, q_1, \ldots\ a_{n-1}, q_n \rangle$.

Definition 4.9. Given two composable IACAs, P and Q, an internal state $(p, q) \in \mathcal{V}^{Int}_{P \circledast Q}$, and its execution fragment $\Delta(p, q) = \langle (p_0, q_0), a_0, (p_1, q_1), \ldots, a_{n-1}, (p_n, q_n) \rangle$, function 1^{st}*state*$(\Delta(p, q))$ returns i, where $(0 < i \leq n)$, if internal state $(p_i, q_i) \in \mathcal{V}^{Int}_{P \circledast Q}$ is the first state in the sequence of states in $\Delta(p, q)$, such that either $p_i \in \mathcal{V}^{Int}_P$ or $q_i \in \mathcal{V}^{Int}_Q$, but not both; or it returns 0, if such a state does not exist.

For an execution fragment, we prove the following:

Lemma 4.6. Considering two composable IACAs, P and Q, an internal state $(p, q) \in \mathcal{V}^{Int}_{P \circledast Q}$, its execution fragment $s = \Delta(p, q)$, and the projections of s, $\pi_P(s)$ and $\pi_Q(s)$, then at least one of the following is true:

- $\exists!(u, c, u') \in \phi_P \cdot \mathit{frag}_P(u, c, u') = \pi_P(s)$, or

- $\exists!(v, d, v') \in \phi_Q \cdot \mathit{frag}_Q(v, d, v') = \pi_Q(s)$.

where $\exists!$ means "there exists a unique."

Using Lemma 4.6, we can define a mapping function that maps an internal state $(p, q) \in \mathcal{V}^{Int}_{P \circledast Q}$ into a complex transition.

Definition 4.10. Given two composable IACAs, P and Q, an internal state $(p, q) \in \mathcal{V}^{Int}_{P \circledast Q}$, and its corresponding execution fragment $\Delta(p, q) = \langle (p_0, q_0), a_0, (p_1, q_1), \ldots, (p_n, q_n) \rangle$, we define function $\mathit{complex}_{P \circledast Q}(p, q)$, which maps (p, q) into exactly one complex transition:

$$\mathit{complex}_{P \circledast Q}(p, q) = \begin{cases} ((p_0, q_0), a, (p_n, q_n)) : & if\ (1^{st}\mathit{state}(\Delta(p, q)) \neq 0)\ \wedge \\ & \mathit{CompTran}_P(p_{1^{st}\mathit{state}(\Delta(p,q))}) = (p_0, a, p_n) \\ ((p_0, q_0), b, (p_n, q_n)) : & if\ (1^{st}\mathit{state}(\Delta(p, q)) \neq 0)\ \wedge \\ & \mathit{CompTran}_Q(q_{1^{st}\mathit{state}(\Delta(p,q))}) = (q_0, b, q_n) \\ ((p_0, q_0), c, (p_n, q_n)) : & if\ (1^{st}\mathit{state}(\Delta(p, q)) = 0)\ \wedge \\ & (\ \exists\ c_j \in set(\mathit{sched}(p_0, c, p_n)) \cdot \\ & (c_j \in \mathcal{A}^I_P) \wedge (p \in \mathit{states}_P(p_0, c, p_n))\ ) \\ ((p_0, q_0), d, (p_n, q_n)) : & if\ (1^{st}\mathit{state}(\Delta(p, q)) = 0)\ \wedge \\ & (\ \exists\ d_j \in set(\mathit{sched}(q_0, d, q_n)) \cdot \\ & (d_j \in \mathcal{A}^I_Q) \wedge (q \in \mathit{states}_Q(q_0, d, q_n))\ ) \end{cases}$$

The four conditions above are mutually exclusive and consider all possibilities, therefore, the result is exactly one complex transition. The first two conditions consider the cases where one complex transition embeds the other one. For these two cases, the name of the derived complex action is the same as the embedding complex action. The third and fourth conditions consider the cases when two complex fragments of two complex transitions overlap entirely, and the name of their corresponding complex actions are different. In that situation, we pick the name of the complex action that has some input actions in its schedule. In practice, we can use only one of the internal states (the first one) of a complex fragment and pass it to function *complex* to compute its complex transition.

Considering our example in Figure 9, part (f) of Figure 9 shows the composition of A and B. The complex transition of $A \parallel B$ is on complex action named L.

We are now ready to define the composition operation.

Definition 4.11. The *composition* of two composable IACAs, P and Q, denoted as $P \parallel Q$, is an IACA defined as:

$$
\begin{aligned}
\mathcal{V}_{P\|Q} &= \mathcal{V}_{P\circledast Q} \\
\mathcal{V}^{Int}_{P\|Q} &= \mathcal{V}^{Int}_{P\circledast Q} \\
i_{P\|Q} &= i_{P\circledast Q} \\
\mathcal{A}^{I}_{P\|Q} &= \mathcal{A}^{I}_{P\circledast Q} \\
\mathcal{A}^{O}_{P\|Q} &= \mathcal{A}^{O}_{P\circledast Q} \\
\mathcal{A}^{H}_{P\|Q} &= \mathcal{A}^{H}_{P\circledast Q} \\
\mathcal{A}^{C}_{P\|Q} &= \mathcal{A}^{C}_{P\circledast Q} \\
\mathcal{T}_{P\|Q} &= \mathcal{T}_{P\circledast Q} \\
\phi_{P\|Q} &= \{\mathit{complex}_{P\circledast Q}(p,q) |\ (p,q) \in \mathcal{V}^{Int}_{P\|Q}\}
\end{aligned}
$$

The composition of two composable IACAs results in a well-formed IACA.

Theorem 4.1. Given two composable IACAs, P and Q, $P \parallel Q$ is a well-formed IACA.

Finally, we prove that IACA composition is commutative.

Theorem 4.2. Given two composable IACAs, P and Q, $P \parallel Q = Q \parallel P$.

4.5. Discussion

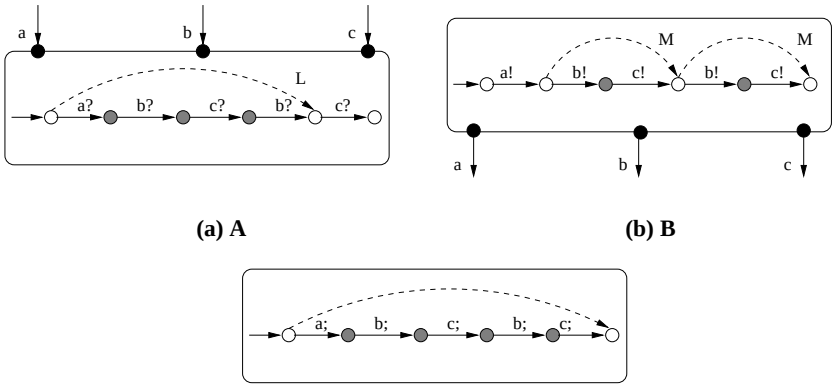
In this section, we discuss the rationale of some of our design choices for IACA, as well as, the reason for non-associativity in IACA composition.

4.5.1. IACA Design Choices

In this section, we justify our design choices in IACA by comparing these choices to the alternatives. In particular, we look at the well-formedness and composability criteria for IACA. The purpose of all well-formedness (Definition 3.2) and composability (Definition 4.2) criteria, is to guarantee that the composition of two IACAs will yield execution fragments that can be mapped to a complex fragment of one of the IACAs involved in the composition. If an execution fragment of an IACA composition $P \parallel Q$ cannot be mapped to a complex fragment of P or Q, it would mean that we have created a new complex action, which neither belongs to P nor to Q, which is not acceptable as the result of a composition; such a new complex action would impose an environmental assumption not found in P and Q, which is incorrect. Next, we consider some alternative design decisions for IACA.

- *Duplicate Actions in a Complex Fragment*: Figure 13 illustrates the composition of IACA A, with duplicate actions in its complex action L, with IACA B. The result is a complex action in the composition that cannot be associated with either L or M. Duplicate actions in a complex action are disallowed by the well-formedness criteria.[11]

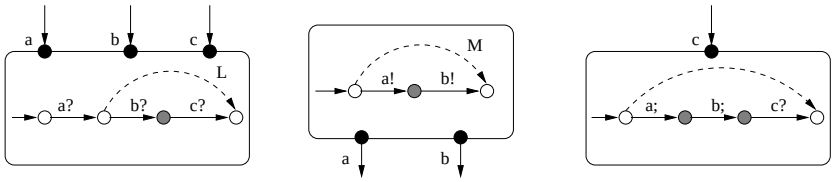
[11] Some non-interleaving models of concurrency, such as "modelling concurrency with partial orders" [26], allow for explicit distinction between occurrences of the same event, by using multi-sets rather than sets of events. The introduction of multi-sets, rather than sets, make many decision problems about such models of concurrency very difficult [14].



(a) A **(b) B**

(c) A||B

Figure 13: Duplicate actions in a complex fragment of an IACA may create undesired new complex actions, when it is composed with another IACA.



(a) A **(b) B** **(c) A||B**

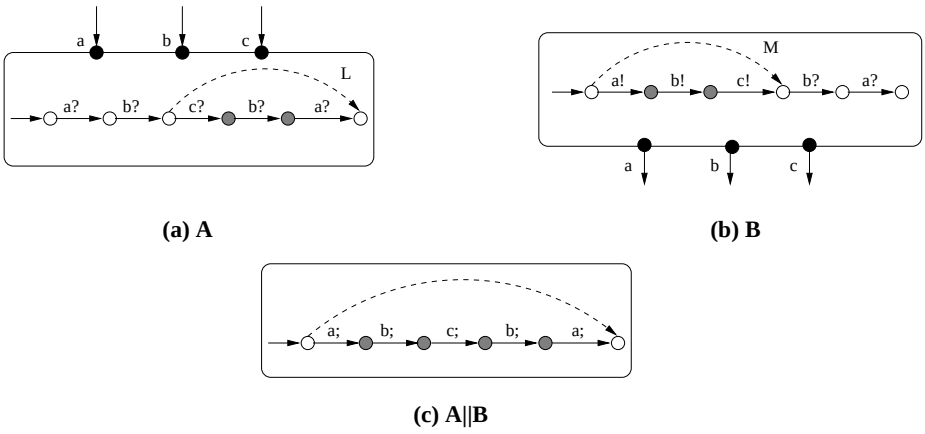
Figure 14: Composition of two IACAs with complex fragments with non-disjoint schedules creates an IACA with a new undesired complex action.

- *Complex Fragments with Incompatible Sequences of Actions*: There are two possibilities for the schedules of two complex transitions of two IACAs that IACA composability criteria disallow, and we show, through examples, the rationale of why we disallow these possibilities.

  Figure 14 illustrates the composition of two IACAs, A and B, with complex transitions whose schedules are neither disjoint nor one is the subsequence of the other, which violates the composability criteria. The result of the composition is an IACA with a complex fragment that cannot be mapped to the complex actions of A or B.

  Figure 15 illustrates the composition of two IACAs with complex fragments whose sets of schedules are the same, but neither is a subsequence of the other. The result is a complex action that cannot be associated with either IACA.

Our composability criteria for two IACAs is the most liberal criteria, which guarantees the well-formedness of the resulting IACA of the composition.



(a) A **(b) B**

(c) A||B

Figure 15: Composition of two IACAs with incompatible sequences of actions in their schedules, creates an IACA with a new undesired complex action.

4.5.2. Associativity in IACA Composition

IACA is not a full-fledged interface model because composition is not associative. Lack of associativity in IACA composition is unavoidable because within a complex transition, there could exist multiple normal actions that can be synchronized through composition(s) with other IACA(s). Such normal actions can only synchronize if their preceding normal actions in the schedule of a complex transition have already synchronized. As such, the order that we consider for composition of multiple IACAs can matter in the success or failure of synchronization for normal transitions within a complex fragment.

Figure 16 shows an example of three composable IACAs and the different possible orders for their compositions, which yields different results. IACA $(A \parallel B) \parallel C$ in part (f) of Figure 16 is not equal to IACA $A \parallel (B \parallel C)$ in part (g) of the Figure. In part (d) of Figure 16, $(A \parallel B)$ shows how a normal transition of A on action b can synchronize with a transition in B that belongs to complex action L. In part (e) of Figure 16, IACA $(B \parallel C)$ cannot synchronize on action d, and thus removes the whole complex transition on L. As such, associativity in IACA composition, which is necessary to be able to incrementally carry out the composition of more than two IACAs, cannot be achieved because it could be the case that a certain order of composition can satisfy some complex transitions' environmental assumptions, while a different order cannot.

Let us look more closely at why the IA composition operator is associative but IACA's is not. A binary composition operator in an interface model, *e.g.*, IA, needs to consider the environmental assumptions of two models and combine those assumptions in such a way that the assumptions of both models about their environments can be mutually satisfied. The composition operator for an interface model is "optimistic", meaning that the environment is helpful by providing the right inputs to the components, and receiving all their outputs. When two composable IAs, P and Q, are composed, IA composition is responsible for resolving the environmental assumptions of the two IAs that involve their shared actions. In other words, P and Q act as each others' environments, by sending input shared actions and/or receiving output shared actions to and from one another. The result of the composition of two composable models embodies the assumptions about an environment that allow them to collaborate. If such an envi-

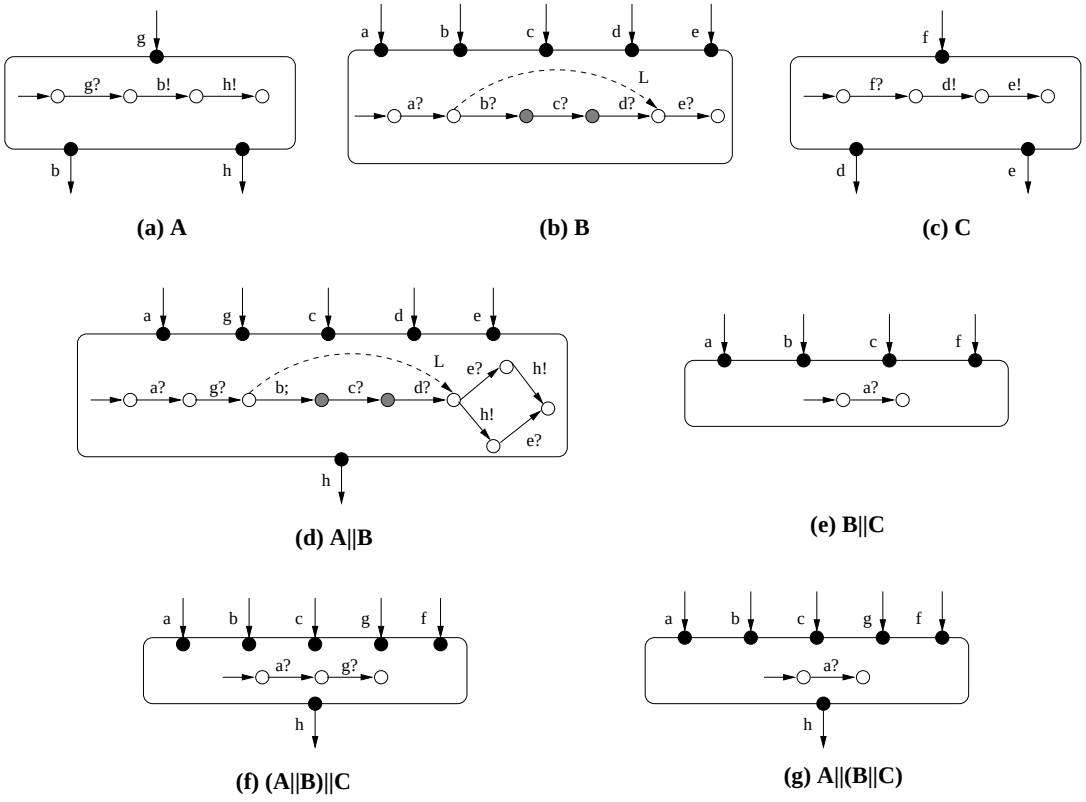


Figure 16: The results of composing three composable IACAs in different orders.

ronment does not exist, then the result of the composition is empty. Therefore, associativity for a binary composition operator of an interface model means that the order in which the compatibility of environmental assumptions of multiple composable models are considered does not affect the environmental assumptions needed for all components to work together.

Associativity cannot be achieved in IACA composition because the environmental assumptions of a complex transition require a sequence of non-interruptible inputs, rather than a single input from the environment (or a sequence of outputs, rather than a single output). Unlike IAs, which consider only the synchronization of a single action, in IACAs, the related synchronizations of a sequence of actions may be satisfied by more than one collaborating IACA. Thus a binary composition operator in the spirit of interface models is not possible in IACA.

Considering Web services, for example, if a complex XML message is supposed to be received, we can only afford to receive the elements of that XML message if they arrive as a stream in a correct order. But, the order of arrival of messages, in a Web service (IACA), depends on the order of composition between multiple Web services (IACAs). In other words, the success of the synchronization for a complex transition may rely on the order of the composition of more than two IACAs, and as such, a binary associative composition operator cannot be achieved.

The major consequence of the lack of associativity is that we cannot reason about the composition of multiple IACAs in an arbitrary order of composition. Any other model that supports a variation of atomicity, along with an interleaving semantics and a rendezvous communication, would suffer from the same non-associativity in its binary composition operator; *e.g.*, the composition operator in A^2CCS [16] is not associative. In the absence of shared actions among multiple IACAs, their composition is associative. To achieve associativity for a binary composition operator, a concurrency model other than interleaving and rendezvous communication could be considered.

5. IACA Refinement

A refined version of an IACA may replace it in a composition. As with IA, a refined IACA may have more inputs and less outputs than the model it refines. For Q to refine P, there must be an alternating simulation relation between the states of Q and P. A state q refines a state p if q has more than or the same inputs as p and less than or the same outputs as p. Additionally, for all states q' reachable from q immediately or through hidden transitions, there must be a p' reachable from p such that q' refines p'. For the complex actions of IACA, a refinement may have additional inputs at the end of the complex fragment or fewer outputs from the end of the complex fragment. This restriction ensures that other IACAs that synchronize with an IACA during composition are still able to synchronize with the refined version of that component.

As an example, the IACA in Figure 17 is the refinement of IACA *CompPay* in Figure 6. IACA *GenCompPay* is capable of carrying out payments in Canadian and US dollars (more inputs), however, it only provides a reference number as output and does not provide an error number as output (less outputs). Furthermore, the credit card payment accepts the province to determine appropriate taxation (more inputs at the end of a complex action).

Our goal in introducing IACA is to capture the idea of parameters to methods or complex messages in Web services using complex actions. IACA refinement matches the programming languages' concepts of subclasses (similar to IA), and optional parameters. In programming languages such as C/C++,

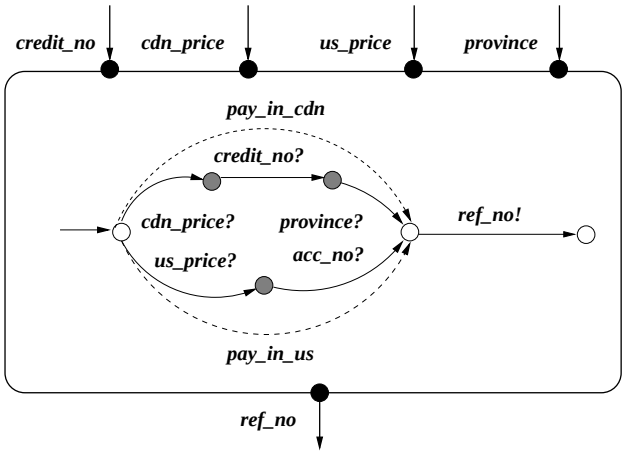


Figure 17: *GenCompPay* is the refinement of the IACA in Figure 6.

conventionally, optional parameters must appear at the end of a function signature. In programming languages such as Java and C++, a subclass of a class can have additional methods, but also has the methods of its parent. Similarly, a refined version of an IACA provides all of the original IACA's complex actions and possibly more.

To define IACA refinement for an IACA P, we first need to partition its complex actions into three sets based on whether a complex fragment has: (1) input and hidden actions (C_P^I), (2) output and hidden actions (C_P^O), and (3) only hidden actions (C_P^H). The definition of refinement is as follows:

Definition 5.1. IACA Q *refines* IACA P, denoted as $Q \preceq P$, if:

1. $\mathcal{A}_P^I \subseteq \mathcal{A}_Q^I$
   (Q has the same or more normal inputs than P), and
2. $\mathcal{A}_P^O \supseteq \mathcal{A}_Q^O$
   (Q has the same or fewer normal outputs than P), and
3. $C_P^I \subseteq C_Q^I$
   (Q has the same or more complex inputs than P,) and
4. $C_P^O \supseteq C_Q^O$
   (Q has the same or fewer complex outputs than P), and
5. $i_Q \preceq i_P$
   (there is an alternating simulation relation $\preceq$ between the initial states of Q and P).

Constraint (5) above propagates the alternating simulation relation to apply to all states of the two IACAs. By starting from the initial states of two IACAs, the alternating simulation relation is checked on all corresponding states.

Before defining the refinement relation, we need to introduce some notation. First, we define the set of states that are reachable immediately or through hidden transitions from a normal state:

Definition 5.2. For an IACA P, for each normal state, $p \in \mathcal{V}_P^N$, the set $closure_P(p)$ is defined as the set of *reachable states* of p that contains p itself and the normal states that can be reached from p through normal transitions with hidden actions. These may include transitions of a complex action.

Next, we define the sets of *enabled* normal and complex actions, which are the actions that occur on transitions immediately exiting a state or the states reachable from a state through hidden actions. States that are reachable through hidden actions are considered the same for the purpose of refinement. We consider only the inputs that are enabled at *all* of these states (because the environment may send an input to any of such states without knowing which of them exactly receives it, and thus all of them should be receptive to the input), but consider all outputs from these states (because the environment should accept any of such outputs).

In the following definition, for a normal state state $p \in \mathcal{V}_P^N$, the functions $\mathcal{A}_P^I(p)$, $\mathcal{A}_P^O(p)$, and $\mathcal{A}_P^C(p)$ return the set of all input, output, and complex actions, respectively, that p is enabled with.

Definition 5.3. Given an IACA P, for each normal state, $p \in \mathcal{V}_P^N$,

- The sets of *enabled normal input* and *enabled normal output* actions are:
$$EnNorm_P^I(p) = \{a \mid \forall p' \in closure_P(p) \cdot a \in \mathcal{A}_P^I(p')\}, and$$
$$EnNorm_P^O(p) = \{a \mid \exists p' \in closure_P(p) \cdot a \in \mathcal{A}_P^O(p')\}.$$
- The sets of *enabled complex input* and *enabled complex output* actions are:
$$EnComp_P^I(p) = \{a \in C_P^I \mid \forall p' \in closure_P(p) \cdot a \in \mathcal{A}_P^C(p')\}, and$$
$$EnComp_P^O(p) = \{a \in C_P^O \mid \exists p' \in closure_P(p) \cdot a \in \mathcal{A}_P^C(p')\}.$$

We denote $En_P^O(p) = EnNorm_P^O(p) \cup EnComp_P^O(p)$ and $En_P^I(p) = EnNorm_P^I(p) \cup EnComp_P^I(p)$.

Having defined the sets of actions that can be expected from a state p and the states reachable from p through hidden transitions, *i.e.*, states belonging to $closure_P(p)$, we now define the sets of states that can be reached from a normal state, as well as its reachable states, through transitions via a certain action.

Definition 5.4. For IACA P, a normal state $p \in \mathcal{V}_P^N$, and action $a \in En_P^I(p) \cup En_P^O(p)$, the set of *reachable states of p by a* is:

$$Dest_P(p, a) = \{p' \in \mathcal{V}_P^N \mid \exists\, r \in closure_P(p) \cdot (\exists\, (r, a, p') \in \mathcal{T}_P) \vee (\exists\, (r, a, p') \in \phi_P)\}$$

Before defining the refinement relation, we need to define one further notation, for specifying *prefix* relation between two sequences.

Definition 5.5. We call a sequence $r = \langle r_1, r_2, \cdots, r_m \rangle$ a *prefix* of sequence $s = \langle s_1, s_2, \cdots, s_n \rangle$, denoted $r \sqsubseteq s$, if $m \leq n$, and $r_i = s_i$, for all $1 \leq i \leq m$.

Finally, we can define the refinement relation between two states. This relation intuitively says that for every state $p \in \mathcal{V}_P^N$, there is an alternating simulation through a state $q \in \mathcal{V}_Q^N$. State q is receptive to all inputs, normal or complex, to which p is receptive, and q does not issue outputs that p does not.

Definition 5.6. For two IACAs, P and Q, the binary relation *alternating simulation* $\preceq \subseteq \mathcal{V}_Q^N \times \mathcal{V}_P^N$ between two states $q \in \mathcal{V}_Q^N$ and $p \in \mathcal{V}_P^N$ holds, denoted as $q \preceq p$, if all of the following conditions are true:

- $EnNorm_P^I(p) \subseteq EnNorm_Q^I(q)$
  (q may have the same or more normal inputs), and
- $EnNorm_P^O(p) \supseteq EnNorm_Q^O(q)$
  (q may have the same or fewer normal outputs), and
- $EnComp_P^I(p) \subseteq EnComp_Q^I(q)$
  (q may have the same or more complex inputs), and
- $EnComp_P^O(p) \supseteq EnComp_Q^O(q)$
  (q may have the same or fewer complex outputs), and
- $\forall a \in EnComp_P^I(p) \cdot \forall (m,a,n) \in \phi_P \cdot m \in closure_P(p) \Rightarrow$
  $\exists (r,a,s) \in \phi_Q \cdot r \in closure_Q(q) \wedge \mathit{sched}_P(m,a,n) \sqsubseteq \mathit{sched}_Q(r,a,s)$
  (For every enabled complex input action a at p, there is the same enabled complex input action at q. Furthermore, the schedule of complex action a in q can have some more input optional parameters at its end), and
- $\forall a \in EnComp_Q^O(q) \cdot \forall\, (r,a,s) \in \phi_Q \cdot r \in closure_P(q) \Rightarrow$
  $\exists (m,a,n) \in \phi_P \cdot m \in closure_P(p) \wedge \mathit{sched}_Q(r,a,s) \sqsubseteq \mathit{sched}_P(m,a,n)$
  (For every enabled complex output action a at q, there is the same enabled complex output action at p. Furthermore, the schedule of complex action a in q may omit some output parameters at its end), and
- $\forall a \in En_P^I(p) \cup En_Q^O(q) \cdot \forall\, q' \in Dest_Q(q,a) \Rightarrow \exists p' \in Dest_P(p,a) \cdot q' \preceq p'$
  ($\preceq$ holds for all reachable normal states under inputs for p and outputs for q).

Intuitively, an input complex action can be refined to input complex actions that have some extra input elements at the end of its fragment, and an output complex action can be refined to an output complex action that has some output elements missing from its end.

Choosing the subsequence relation to hold between the schedules of two complex fragments, instead of the prefix relation, would not have allowed for top-down design, which is the purpose of refinement.

Theorem 5.1. Given three IACAs, P, Q and P', such that $P' \preceq P$, P and Q are composable, and P' and Q are composable, $(P' \parallel Q) \preceq (P \parallel Q)$, if the following conditions hold:

1. $Shared(P,Q) = Shared(P',Q)$
   (P' and P communicate with Q through the same set of shared actions), and
2. $\forall (p',p) \cdot (p' \preceq p) \Rightarrow$
   $((\mathcal{A}_{P'}^I(p') \backslash \mathcal{A}_P^I(p)) \cap Shared(P,Q) = \emptyset) \wedge$
   $((\mathcal{A}_P^O(p) \backslash \mathcal{A}_{P'}^O(p')) \cap Shared(P,Q) = \emptyset)$
   (States of P' that are in the simulation relation with P do not introduce extra (nor eliminate) actions that belong to the shared actions of P and Q), and

3. $\forall(p', p) \cdot (p' \preceq p) \Rightarrow$
   $\forall(p, c, u) \in \phi_P \cdot (\exists(p', c, v) \in \phi_{P'} \wedge (c \in C^O_{P'}))$
   $\Rightarrow ((set(\mathsf{sched}_P(p, c, u)))\backslash(set(\mathsf{sched}_{P'}(p', c, v)))) \cap Shared(P, Q)) = \emptyset$

   (States of P' that are in the simulation relation with P should not have output complex actions that are shared with P, and miss some normal output actions in their schedules that belong to the shared actions of P and Q), and

4. $\forall(p', p) \cdot (p' \preceq p) \Rightarrow$
   $\forall(p', c, v) \in \phi_{P'} \cdot (\exists(p, c, u) \in \phi_P \wedge (c \in C^I_P))$
   $\Rightarrow ((set(\mathsf{sched}_{P'}(p', c, v)))\backslash(set(\mathsf{sched}_P(p, c, u)))) \cap Shared(P, Q) = \emptyset$

   (States of P' that are in the simulation relation with P should not have input complex actions shared with P that introduce new simple input actions in their schedules that belong to the shared actions of P and Q).

5.1. Discussion

The conditions of Theorem 5.1 require that P' preserves the same shared actions that P has with Q, and requires P' to behave in accordance to P on the shared actions of P and Q. In other words, we require that P' neither increases nor decreases the shared actions that P and Q have. Additionally, we also require that at the state level, the refined state and the original state use the same set of shared actions. Comparing IACA's top-down design criteria with IA, IA is more lenient. IA only requires $Shared(P', Q) \subseteq Shared(P, Q)$ for a similar top-down design result as in Theorem 5.1. Our restriction arises from the fact that we not only deal with illegal states (as with IA), but also deal with illegal internal states. To support top-down design, P', a refinement of P, should behave in such a way that it does not cause new illegal internal states that the composition of P and Q does not create. To avoid new illegal internal states, we should ensure that if P and Q have a chance to synchronize on actions of their complex transitions, P' and Q have the same chance.

In the above theorem we require P' and Q to be composable, because there is no guarantee that since P and Q are composable, P' and Q would be composable too. This requirement is necessary because P', as a refinement of P, can have extra complex transitions at its states, and there is no way to guarantee that such extra complex actions observe the composability criteria between the complex transitions of two IACAs. Alternatively, we could have defined our refinement relation in such a way that it would have disallowed the extra complex actions, and hence could have avoided requiring explicitly that P' and Q be composable. We prefer our definition of refinement, because it allows for specifying extra complex actions, *i.e.*, extra methods or complex messages, for a refinement of a component. The extra cost to pay is to require explicitly the refinement of a component to be composable with the component it is getting composed with.

Figure 18 illustrates the composition $(GenCompPay \parallel Prod)$. Considering the composition of *GenCompPay*, in Figure 17, with IACA of *Prod* in Figure 1, since $GenCompPay \preceq CompPay$ then $(GenCompPay \parallel Prod) \preceq (CompPay \parallel Prod)$. IACA $(CompPay \parallel Prod)$ is shown in Figure 8.

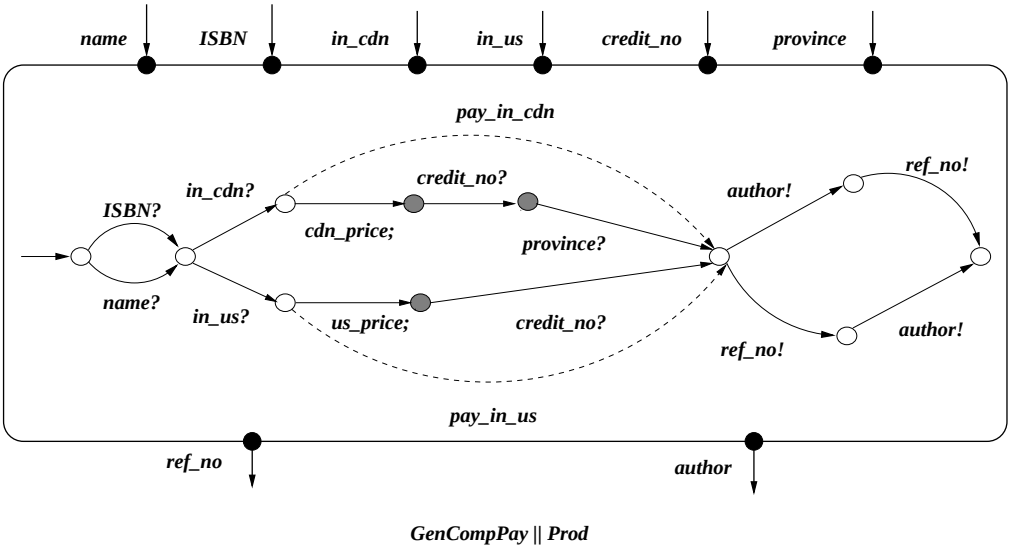


Figure 18: Composition of IACA *GenCompPay*, in Figure 17, with IACA equivalent IACA of IA *Prod* in Figure 1.

6. Related Work

The idea of grouping activities in a sequential, non-interruptible manner is common in many contexts. For example, in databases, the concept of a transaction is pivotal and resembles our complex actions. Within the context of concurrency theory, different approaches have been proposed to augment process algebraic-like languages to support non-interruptible sequences of actions. Such approaches can be generally categorized into two groups: (1) *atomic actions* (*e.g.*, [16, 8]), and (2) *action refinement* (*e.g.*, [1]).

In the work most comparable to ours, Gorrieri, Marchetti, and Montanari enhance CCS [25] to support non-interruptible actions [16]. Their proposed composition operator is non-associative and they suggest that non-associativity may be an intrinsic property of handling complex actions.

Action refinement approaches allow *stepwise refinements* of models into their more concrete equivalents. For a comprehensive treatment of action refinement, readers can refer to [17, 15].

Promela, the language of the Spin model checker [21], implements complex actions using the keywords `atomic` and `d_step`. Promela's atomic sequences may block and allow interleaving if an input is not available or an output cannot be consumed. `d_step` sequences must be deterministic and do not allow interleaving. A run-time error will occur if actions grouped in a `d_step` cannot synchronize when necessary. Our complex action is similar to `d_step`. In Promela, there is neither a syntax for a binary composition operator, nor a syntax for a refinement relation between processes. The way processes are run concurrently in Spin, implies an n-ary interleaving composition operator where all processes in a model are run during the composition. As such, associativity in composition is not relevant for Promela, because it does not have a binary composition operator. Assuming that Promela would have allowed for a binary composition operator, similar to many other semantics that support atomicity, `d_step` statements can create non-associative compositions. In Spin, `d_step` statements that cannot continue their executions will issue a run-time error. An example of non-associativity that can arise if Promela would

have supported binary composition is when a `d_step` statement needs to do two rendezvous receive operations, which include communication with two different processes. In this case, the non-deterministic order that is considered for composition, would either make the `d_step` announce an error, or the execution would continue gracefully.

While our approach has the same goals as much of the work mentioned above, we differ because we have created an automata-based interface model with complex actions that has most of the properties of interface models, which are designed to be a concise way to specify component-based systems.

7. Conclusion and Future Work

We have introduced *interface automata with complex actions (IACA)*, which add complex actions to de Alfaro and Henzinger's interface automata. The transitions within a complex action are not interleaved with transitions from another component in composition. Complex actions allow us to model non-interruptible behaviour, which is needed to describe parameters of methods or Web services complex messages. IACA has all the properties of an interface model except for associativity of composition.

An immediate application for IACA is in modelling Web services. Web services communicate with other Web services and their service requesters through input and output XML messages. Complex XML messages are streams of data items that should not be interleaved with other messages. *Web Service Description Language (WSDL)* [10, 11], the Web services standard for specifying Web services messages and their communication patterns, allows for specification of the functionality of Web services by: (1) specifying input and output messages of Web services, each with potentially multiple elements, and (2) specifying the temporal order of message exchanges in Web services. IACA is a natural formalism for modelling WSDL and reasoning about the compatibility of Web services. First, complex actions are a good way to model the input and output XML messages of Web services, and secondly, IACA, as an automata model, can effectively capture the temporal order of messages in Web services; and thirdly, IACA rendezvous communication mechanism suits the point-to-point communication mechanism of Web services, *e.g.,* SOAP communication framework for Web services [18].

In our future work, we plan to investigate how we can overcome the challenge of the lack of associativity for composition in IACA. We may need to relax the way elements of complex actions synchronize. Such a relaxation could involve choosing a different communication mechanism for our model and/or choosing a different concurrency model. Alternatively, it may be useful to define an n-ary composition operator, but that approach does not entirely follow the well-formedness criteria of interface models.

Acknowledgements

We would like to thank the reviewers of this paper for their insightful comments, which significantly improved our work.

References

[1] Aceto, L.: *Action Refinement in Process Algebras*, Cambridge University Press, 1992, ISBN 0-521-43111-5.

[2] de Alfaro, L.: Game Models for Open Systems, *Verification: Theory and Practice*, 2772, Springer, 2003, ISBN 3-540-21002-4.

[3] de Alfaro, L., Henzinger, T. A.: Interface Automata, *Proceedings of the Joint 8th European Software Engeneering Conference and 9th ACM SIGSOFT Symposium on the Foundation of Software Engeneering (ESEC/FSE-01)*, 26, 5, ACM Press, 2001, ISSN 0163-5948.

[4] de Alfaro, L., Henzinger, T. A.: Interface Theories for Component-Based Design, *Proceedings of the 1st International Workshop on Embedded Software*, 2211, Lecture Notes in Computer Science 2211, Springer-Verlag, 2001, ISSN 0302-9743.

[5] de Alfaro, L., Henzinger, T. A.: Interface-Based Design, *Proceedings of the Marktoberdorf Summer School, Kluwer*, Engineering Theories of Software Intensive Systems, 2004.

[6] de Alfaro, L., Stoelinga, M.: Interfaces: A Game-Theoretic Framework for Reasoning About Component-Based Systems, *Electronic Notes in Theoretical Computer Science*, **97**, 2004, 3–23.

[7] Alur, R., Henzinger, T. A., Kupferman, O., Vardi, M. Y.: Alternating refinement relations, *Proceeding of the 9th Conference on Concurrency Theory*, 1466, Springer-Verlag, 1998.

[8] Boudol, G.: Atomic Actions (note), *Bulletin of the European Association for Theoretical Computer Science*, **38**, 1989, 136–144, Technical Contributions.

[9] Brzozowski, J. A.: Canonical regular expressions and minimal state graphs for definite events, in: *Mathematical theory of Automata*, Polytechnic Press, Polytechnic Institute of Brooklyn, N.Y., 1962, 529–561, Volume 12 of MRI Symposia Series.

[10] Chinnici, R., Gudgin, M., Moreau, J.-J., Schlimmer, J., Weerawarana, S.: Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, 2006.

[11] Chinnici, R., Haas, H., Moreau, J.-J., Orchard, D., Weerawarana, S.: Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts, 2006.

[12] Esmaeilsabzali, S., Day, N. A., Mavaddat, F.: *Interface Automata with Complex Actions - Extended Version*, Technical Report CS-2005-26, University of Waterloo, David R. Cheriton School of Computer Science, 2005.

[13] Esmaeilsabzali, S., Mavaddat, F., Day, N. A.: Interface Automata with Complex Actions, *Electronic Notes in Theoretical Computer Science*, **159**, 2006, 79–97.

[14] Feigenbaum, J., Kahn, J. A., Lund, C.: Complexity Results for POMSET Languages, *SIAM Journal on Discrete Mathematics*, **6**(3), August 1993, 432–442, ISSN 0895-4801 (print), 1095-7146 (electronic).

[15] van Glabbeek, R., Goltz, U.: Refinement of actions and equivalence notions for concurrent systems, *Acta Informatica*, **37**(4-5), 2000, 229–327, ISSN 0001-5903.

[16] Gorrieri, R., Marchetti, S., Montanari, U.: A^2CCS: atomic actions for CCS, *Theoretical Computer Science*, **72**(2-3), 1990, 203–223, ISSN 0304-3975.

[17] Gorrieri, R., Rensink, A.: Action Refinement, in: *Handbook of Process Algebra* (J. A. Bergstra, A. Ponse, S. A. Smolka, Eds.), chapter 16, Elsevier, 2001, 1047–1147.

[18] Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., Nielsen, H. F.: SOAP Version 1.2 Part 1: Messaging Framework, W3C Recommendation, 2003.

[19] Henzinger, T. A., Manna, Z., Pnueli, A.: An interleaving model for real time, *JCIT: Proceedings of 5th Jerusalem conference on information technology*, IEEE Computer Society Press, 1990, ISBN 0-8186-2078-1.

[20] Hoare, C. A. R.: *Communicating Sequential Processes*, Prentice-Hall, 1985, ISBN 0-13-153271-5.

[21] Holzmann, G. J.: The model checker Spin, *IEEE Transaction on software Engineering*, **23**(5), 1997, 279–295, ISSN 0098-5589.

[22] Lynch, N. A., Tuttle, M. R.: Hierarchical correctness proofs for distributed algorithms, *Proceedings of the 6th ACM Symposium on Principles of Distributed Computing*, ACM Press, 1987.

[23] Manna, Z., Pnueli, A.: *The Temporal Logic of Reactive and Concurrent Systems: Specification*, Springer-Verlag, 1991.

[24] Milner, R.: Calculi of Synchrony and Asynchrony, *Theoretical Computer Science*, **25**, 1983, 267–310.

[25] Milner, R.: *Communication and Concurrency*, International Series in Computer Science, Prentice Hall, 1989.

[26] Pratt, V. R.: Modelling Concurrency With Partial Orders, *International Journal of Parallel Programming*, **15**(1), 1986, 33–71.

[27] Watson, B. W.: *A taxonomy of finite automata minimization algorithms*, Report, Department of Mathematics and Computing Science, Eindhoven University of Technology, The Netherlands, 1994.

A. Proofs

In this appendix, we present our proofs for the lemmas and theorems stated in sections 4 and 5.

Lemma 4.1 For two composable IACAs, P and Q, and two action-disjoint complex transitions (p, c, p') $\in \phi_P$ and $(q, d, q') \in \phi_Q$, where $\mathit{frag}_P(p, c, p') = \langle p, c_0, p_0, \ldots, p_{n-1}, c_n, p' \rangle$ and $\mathit{frag}_Q(q, d, q') = \langle q, d_0, q_0, \ldots, q_{m-1}, d_m, q' \rangle$, all internal states $(p_i, q_j) \in \mathcal{V}^{Int}_{P*Q}$, where $(0 \leq i < n)$ and $(0 \leq j < m)$, are unreachable in $P * Q$.

Proof:
To prove our claim, we first prove that (p_0, q_0) is unreachable, and then show that any other internal state, *i.e.*, any state belonging to the set of states $M = \{(p_i, q_j) \in \mathcal{V}^{Int}_{P*Q} | (0 < i < n) \wedge (0 < j < m)\}$, is reachable, if (p_0, q_0) is reachable. Figure 11, on page 22, shows an example of the situation that we are considering in this lemma.

Considering the definition of $\mathcal{T}_{P*Q}$ in Definition 4.5, state (p_0, q_0) can only be reached through the transitions originating from states (p, q_0), (p_0, q), and (p, q). But (p_0, q_0) cannot be reached from any of these states because $p \in \mathcal{V}^N_P$, $q \in \mathcal{V}^N_Q$, and $c_0 \neq d_0$. Therefore, (p_0, q_0) is unreachable.

Let us now consider an internal state $(p_i, q_j) \in M$. If state (p_i, q_j) is reachable, then according to Definition 4.5, it should be reachable via transitions originating from (p_{i-1}, q_j) and/or (p_i, q_{j-1}). Similarly, (p_{i-1}, q_j) can only be reached via transitions originating from states (p_{i-1}, q_{j-1}) and/or (p_{i-2}, q_j). Also, state (p_i, q_{j-1}) can only be reached via transitions originating from states (p_{i-1}, q_{j-1}) and/or (p_i, q_{j-2}). By following the chain of transitions that could make state (p_i, q_j) reachable, in a similar fashion as above, eventually we will reach at set of states $X = \{(y, q_0) | y \in \{p_1, p_2, \cdots, p_n\}\}$ and $Y = \{(p_0, x) | x \in \{q_1, q_2, \cdots, q_m\}\}$. Let us consider a state $(p_l, q_0) \in X$. It can only be reached via a transition initiating from (p_{l-1}, q_0). The other alternative state for reaching (p_l, q_0) is state (p_l, q), but state (p_l, q) cannot initiate such a transition since q is a normal state. Similar observation holds for a state in $(p_0, q_k) \in Y$; *i.e.*, it can only be reached via a transition initiating from (p_0, q_{k-1}). As such, all states in X and Y can only be reached via paths that initiate from state (p_0, q_0), because all (p_0, q_k) and (p_l, q_0) states are only reachable from state (p_0, q_0). Therefore, if an arbitrary internal state $(p_i, q_j) \in M$ is reachable, so is state (p_0, q_0).

Having shown that (p_0, q_0) is not reachable, and a state in M is reachable only if state (p_0, q_0) is reachable, we conclude that all internal states $(p_i, q_j) \in \mathcal{V}^{Int}_{P*Q}$, where $0 \leq i < n$ and $0 \leq j < m$, are unreachable. □

Lemma 4.2 Given two composable IACAs, P and Q, and their legal interleaved product $P \circledast Q$, for an internal state $(p, q) \in \mathcal{V}^{Int}_{P \circledast Q}$, there is exactly one transition in $\mathcal{T}_{P \circledast Q}$ with source (p, q).

Proof:
The way sets $\mathcal{V}_{P \circledast Q}$ and $\mathcal{T}_{P \circledast Q}$ are computed, in the algorithm in Figure 10, guarantees that the only internal states in $\mathcal{V}_{P*Q}$ that are not removed are those that, first, appear as the source of *at least* one transition, and second, appear as the destination of *at least* one transition, otherwise they are removed. Therefore, we only need to prove that for each internal state (p, q), there exists *maximum* one transition with source (p, q).

We first remark that the transitions in $\mathcal{T}_{P*Q}$, with their sources being internal states, can only be created by the disjoint sets `(3)`, `(4)`, and `(5)`. Furthermore, for an internal state $(p, q) \in \mathcal{V}^{Int}_{P*Q}$, each of the sets can create maximum one transition with source (p, q). If one of the sets introduces more than one transition in $\mathcal{T}_{P*Q}$ for (p, q), it would mean that, depending on whether $p \in \mathcal{V}^{Int}_{P}$, $q \in \mathcal{V}^{Int}_{Q}$, or both, more than one transition initiate from p, q, or both p and q, respectively; which is not possible.

Therefore, for (p, q) to have more more than one transition initiating from it, it is necessary that at least two sets among sets `(3)`, `(4)`, and `(5)`, introduce transitions initiating from it. But for internal state (p, q), depending on whether the enabled actions on p and q are shared or not, either set `(3)` can introduce a transition, or sets `(4)` and `(5)`.

If both p and q are enabled with a shared action, then since P and Q are deterministic and at least one of p and q is an internal state and is enabled with only one action, it is is impossible for (p, q) to be the source of more than one transition.

If both p and q are internal states of P and Q, and they are not enabled with a shared action, then it is possible for both sets `(4)` and `(5)` to introduce transitions with source (p, q) in $\mathcal{T}_{P*Q}$. As such, to prove that all internal states are the source of not more than one transition, it suffices to consider only the internal states of $\mathcal{V}_{P*Q}$ where both sets `(4)` and `(5)` can create transitions on them. Let us characterize all such states as set:

$$\begin{aligned} M \quad = \quad & \{(p, q) \in \mathcal{V}^{Int}_{P*Q} | \exists (p, a, p') \in \mathcal{T}_P \cdot \exists (q, b, q') \in \mathcal{T}_Q \cdot \\ & (p \in \mathcal{V}^{Int}_{P}) \wedge (q \in \mathcal{V}^{Int}_{Q}) \wedge (a, b \notin Shared(P, Q))\}. \end{aligned}$$

Considering a state $(p, q) \in M$, since p and q are internal states, and they are enabled with different actions, we can conclude that p and q belong to the complex fragments of two action-disjoint complex transitions of P and Q, respectively. But in Lemma 4.1, we proved that all such (p, q) states are unreachable, and so we can conclude all internal states $(p, q) \in \mathcal{V}^{Int}_{P \circledast Q}$ have exactly one outgoing transition. □

Lemma 4.3 Given two composable IACAs, P and Q, and their legal interleaved product $P \circledast Q$, for an internal state $(p, q) \in \mathcal{V}^{Int}_{P \circledast Q}$, there is exactly one transition in $\mathcal{T}_{P \circledast Q}$ with destination (p, q).

Proof:
Let us consider a transition $t_1 = ((p_1, q_1), a, (p, q)) \in \mathcal{T}_{P \circledast Q}$. We show that if there exists another transition $t_2 = ((p_2, q_2), b, (p, q)) \in \mathcal{T}_{P \circledast Q}$, then one of them cannot exist.

For $(p, q) \in \mathcal{V}_{P*Q}^{Int}$, either p, q, or both are internal states. Next, we consider the three possibilities separately.

- $p \in \mathcal{V}_P^{Int}$ and $q \in \mathcal{V}_Q^N$: We prove our claim for this case in two steps. The first step considers the case where $p_1 \neq p$, and the second step considers the case where $p_1 = p$.

  1. $p_1 \neq p$: According to Definition 4.5, transition t_1 can be created either by set `(1)`, `(3)`, or `(4)`, and there should exist a transition $(p_1, a, p) \in \mathcal{T}_P$. We consider two subcases, namely $p_2 = p$ and $p_2 \neq p$, separately, and show that in both cases transition t_2, as characterized above, cannot exist.

     (a) If $p_2 = p$ then, t_2 can only be created by set `(5)`. As such: (i) $q_2 \in \mathcal{V}_Q^{Int}$, and (ii) $(q_2, b, q) \in \mathcal{T}_Q$. Let us consider state $p_2 \in \mathcal{V}_P^{Int}$, which is the same state as p. There should exist an action $h \in \mathcal{A}_P^N(p_2)$. If $h \notin Shared(P, Q)$, then state (p_2, q_2) would have two outgoing transitions, which makes (p_2, q_2) unreachable according to Lemma 4.1, and thus t_2 cannot exist. If $h \in Shared(P, Q)$ and $h \in \mathcal{A}_P^O$, then (p_2, q_2) is an illegal state, and again t_2 cannot exist. The only remaining situation is when $h \in Shared(P, Q)$ and $h \in \mathcal{A}_P^I$. Let us identify the complex fragments that p_2 and q_2 belong to. We denote $A = \mathit{CompTran}_P(p_2)$, and $B = \mathit{CompTran}_Q(q_2)$, and denote $\langle x, a_0, x_0, \cdots, x_{n-1}, a_n, x' \rangle$ and $\langle y, b_0, x_0, \cdots, y_{m-1}, b_m, y' \rangle$, to be $\mathit{frag}_P(A)$ and $\mathit{frag}_Q(B)$, respectively. We assume that $p_2 = x_k$, where $(0 \leq k < n)$; we already know that $q_2 = y_{m-1}$, because $q \in \mathcal{V}_Q^N$. If A and B are action-disjoint, then (p_2, q_2) is not reachable, by Lemma 4.1, which supports our claim. Otherwise, A must be embedded in B, because $b \notin set(\mathit{sched}_P(A))$.[12] We denote $y_l \in \mathit{states}_Q(B)$, where $(0 \leq l < m-1)$, to be the internal state that $h \in \mathcal{A}_Q^O$ is enabled at; y_l exists because A is embedded in B. Since both $p_2 = x_k$ and $q_2 = y_{m-1}$ are internal states, an execution that reaches (p_2, q_2) must pass through a state (x_i, y_l), where $0 \leq i < k$. But all such (x_i, y_l) states are illegal states because y_l has h enabled at them. Thus (p_2, q_2) is not reachable, and t_2 cannot exist.[13]

     (b) If $p_2 \neq p$ then, according to Definition 4.5, for transition t_2 to exist, transition (p_2, b, p) should exist in $\mathcal{T}_P$. But we know that $(p_1, a, p) \in \mathcal{T}_P$. If (p_2, b, p) and (p_1, a, p) are the same transitions, then there does not exist such a distinct t_2. If they are distinct transitions, then it would mean that $p \in \mathcal{V}_P^{Int}$ has two incoming transitions, which is not possible. So, we can conclude that t_2 cannot exist when $p_1 \neq p$. Next, we consider the case where $p_1 = p$.

  2. $p_1 = p$: According to Definition 4.5, transition t_1 can be created only by set `(5)`, and thus: $q_1 \in \mathcal{V}_Q^{Int}$ and $(q_1, a, q) \in \mathcal{T}_Q$. Again, we consider two separate subcases: $p_2 = p$ and $p_2 \neq p$.

     (a) If $p_2 = p$, then t_2 can be created only by set `(5)`, and thus: $q_2 \in \mathcal{V}_Q^{Int}$ and $(q_2, b, q) \in \mathcal{T}_Q$. But $q_1 \in \mathcal{V}_Q^{Int}$, and $(q_1, a, q) \in \mathcal{T}_Q$ too, which is impossible because q_1 and q_2, as

[12] In Definition 4.3, we defined the "embedded" relation between two complex transitions.

[13] All such (x_i, y_l) states are illegal states, because schedule of complex actions are unique. State (x_k, y_l) is not an illegal state, but cannot reach (p_2, q_2) because it can only generate transitions by using set `(3)` in Definition 4.5.

internal states of Q, cannot have transitions to the same state, *i.e.*, to state q. Therefore, there is no such a t_2.

(b) For the case where $p_2 \neq p$, we prove that transition t_1 *cannot* exist, if transition t_2 exists. Furthermore, we show that there cannot exist more than one such t_2 transitions. Let us consider state (p_1, q_1). Assume there exists $h \in \mathcal{A}_P^N(p_1)$. If action h, the action which state $p_1 \in \mathcal{V}_P^{Int}$ has a transition enabled on, does not belong to $Shared(P, Q)$, then it would mean that the internal state (p_1, q_1) is enabled with two transitions, one on b and one on h, which means that state (p_1, q_1) is not reachable, according to Lemma 4.1. If $h \in Shared(P, Q)$ and $h \in \mathcal{A}_P^O$, then (p_1, q_1) is an illegal state, and again cannot be reached.

The only other possibility, for the case when $p_2 \neq p$, is when $h \in Shared(P, Q)$ and $h \in \mathcal{A}_P^I$. Let us identify the complex fragment that p_1, which is the same state as p, and q_1 belong to. We denote $A = \mathit{CompTran}_P(p_1)$, $B = \mathit{CompTran}_Q(q_1)$. Also we let $\langle x, a_0, x_0, \cdots, x_{n-1}, a_n, x' \rangle$ and $\langle y, b_0, x_0, \cdots, y_{m-1}, b_m, y' \rangle$, to be $\mathit{frag}_P(A)$ and $\mathit{frag}_Q(B)$, respectively. We assume that $p_1 = x_k$, where $(0 \leq k < n)$; we already know that $q_1 = y_{m-1}$, because $q \in \mathcal{V}_Q^N$. If A and B are action-disjoint, then (p_1, q_1) is not reachable, by Lemma 4.1, which supports our claim. Otherwise, A must be embedded in B, because $b \notin set(\mathit{sched}_P(A))$. We let $y_l \in \mathit{states}_Q(B)$, where $(0 \leq l < m - 1)$, to be the internal state that $h \in \mathcal{A}_Q^O$ is enabled at; y_l exists because A is embedded in B. Since both $p_1 = x_k$ and $q_1 = y_{m-1}$ are internal states, an execution that reaches (p_1, q_1) must pass through a state (x_i, y_l), where $0 \leq i < k$. But all such (x_i, y_l) states are illegal states because y_l has h enabled at them. Thus, (p_1, q_1) is not reachable, and t_1 cannot exist.

We still need to prove that there does not exist more than one such t_2 transition. If $b \notin Shared(P, Q)$, then the only transition with destination (p, q) is $((p_2, q), b, (p, q)) \in \mathcal{T}_{P \circledast Q}$, because $p \in \mathcal{V}_P^{Int}$, and there is only one transition with source p_2 in P, namely $(p_2, b, p) \in \mathcal{T}_p$. If $b \in Shared(P, Q)$, then since well-formed IACAs are incoming deterministic, then there could not exist more than one transition on action b in Q, with destination q. Therefore, there is exactly one t_2 with destination (p, q).

For an internal state $(p, q)\mathcal{V}_{P \circledast Q}^{Int}$, where $p \in \mathcal{V}_P^{Int}$ and $q \in \mathcal{V}_Q^N$, we considered all different scenarios and proved that there could not exist more than one transition with destination (p, q). Next, we consider other types of internal states in $\mathcal{V}_{P \circledast Q}^{Int}$.

- $p \in \mathcal{V}_P^N$ and $q \in \mathcal{V}_Q^{Int}$: This case is symmetric with the previous case.
- $p \in \mathcal{V}_P^{Int}$ and $q \in \mathcal{V}_Q^{Int}$: Based on whether p_1 and p_2 are different states from p, we consider the following four cases, and prove that in none of the cases there could exist more than one transition with destination in state (p, q).
  1. $(p1 \neq p)$ and $(p_2 = p)$: Transition t_2 can only be created by set `(5)`. Thus, $q_2 \in \mathcal{V}_Q^{Int}$ and $(q_2, b, q) \in \mathcal{T}_Q$. By following exactly the same proof for unreachability of (p_2, q_2) in case 1a above, we can prove that (p_2, q_2) is not reachable in this case as well. Therefore, there is exactly one transition with destination (p, q).
  2. $(p1 = p)$ and $(p_2 \neq p)$: This case is symmetric with the previous case.

3. $(p1 = p)$ and $(p_2 = p)$: Transitions t_1 and t_2 can only be created by set `(5)`. Furthermore, two distinct transitions (q_1, a, q) and (q_2, b, q) should exist in $\mathcal{T}_Q$, which is impossible because $q \in \mathcal{V}_Q^{Int}$. Therefore, such a t_2 cannot exist.
4. $(p_1 \neq p)$ and $(p_2 \neq p)$: Transitions t_1 and t_2 can be created by either of sets `(1)`, `(3)`, and `(4)`. However, in all cases, for t_1 and t_2 to be distinct transitions, two distinct transitions (p_1, a, p) and (p_2, b, p) should exist in $\mathcal{T}_P$, which is impossible because $p \in \mathcal{V}_p^{Int}$. Therefore, such a t_2 cannot exist.

For all possible cases, we showed that there could not exist more than one transition with destination (p, q), which concludes our proof.

□

Lemma 4.4 Given two composable IACAs, P and Q, and their legal interleaved product $P \circledast Q$, for an internal state $(p, q) \in \mathcal{V}_{P \circledast Q}^{Int}$, there exists exactly one execution $\langle (p_0, q_0), a_0, (p_1, q_1), \ldots, (p_n, q_n) \rangle$, called *the execution fragment* of state (p, q), denoted as $\Delta(p, q)$, and all of the following conditions are true:

- $\exists (p_j, q_j) \cdot (0 < j < n) \wedge (p_j = p) \wedge (q_j = q)$
  ((p, q) belongs to a complex fragment), and
- $\forall i \cdot (0 < i < n) \Rightarrow ((p_i, q_i), a_i, (p_{i+1}, q_{i+1})) \in \mathcal{T}_{P \circledast Q}$
  (All transitions of the complex fragment are in the legal interleaved product), and
- $(p_0, p_n \in \mathcal{V}_P^N) \wedge (q_0, q_n \in \mathcal{V}_Q^N) \wedge (\forall i \cdot (0 < i < n) \Rightarrow (p_i, q_i) \in \mathcal{V}_{P \circledast Q}^{Int})$
  (The source and destination states of the sequence are normal states and the other states are internal states).

Proof:
From Lemma 4.2 and Lemma 4.3, it follows that for each internal state $(p, q) \in \mathcal{V}_{P \circledast Q}^{Int}$, there is exactly one transition with source (p, q), and exactly one transition with destination (p, q). As such, each internal state $(p, q) \in \mathcal{V}_{P \circledast Q}^{Int}$ belongs to a unique alternating sequence of states and actions. We should show that such a sequence follows the criteria mentioned in the lemma. Let us consider the alternating sequence of states and actions that includes internal state (p, q). Since there are finite number of internal states, such an alternating sequence of states and actions eventually terminates in a normal state, *e.g.*, in a state (u, v) where $u \in \mathcal{V}_P^N$ and $v \in \mathcal{V}_Q^N$. Such a (u, v) in fact represents the state (p_n, q_n) in $\Delta(p, q)$.

There also exists a state (p_0, q_0) as characterized in $\Delta(p, q)$. Let us assume that there does not exist such a (p_0, q_0). Then, for any reachable internal state $(p, q) \in \mathcal{V}_{P \circledast Q}$, there always exists an initial state (i_P, i_Q), as a normal state, such that (p, q) can be reached from that state, and such a state can be considered as (p_0, q_0) for $\Delta(p, q)$.

Since we chose an arbitrary $(p, q) \in \mathcal{V}_{P \circledast Q}^{Int}$, we can conclude that for any internal state belonging to $\mathcal{V}_{P \circledast Q}^{Int}$, there exists an execution fragment $\langle (p_0, q_0), a_0, (p_1, q_1), \ldots, (p_n, q_n) \rangle$ such that all the above conditions hold. Furthermore, the execution fragment of an internal state (p, q) is unique, because all internal states have exactly one outgoing and one incoming transition, allowing them only to belong to a unique execution fragment. Therefore, each internal state in the legal interleaved product of two composable IACAs is associated with a unique execution fragment, and this concludes our proof. □

Lemma 4.5 Given two composable IACAs, P and Q, and their legal interleaved product $P \circledast Q$, for an internal state $(p, q) \in \mathcal{V}^{Int}_{P \circledast Q}$, its execution fragment $\Delta(p, q) = \langle (p_0, q_0), a_0, (p_1, q_1), \ldots, (p_n, q_n) \rangle$ is non-partially overlapped.

Proof:
Let us assume that the execution fragment of the internal state $(p, q) \in \mathcal{V}^{Int}_{P \circledast Q}$ is partially overlapped, we show by contradiction that this is not possible. If the execution $\Delta(p, q) = \langle (p_0, q_0), a_0, (p_1, q_1), \ldots, (p_n, q_n) \rangle$ is partially overlapped, it means that there exist two complex transitions, say $(u, c, u') \in \phi_P$ and $(v, d, v') \in \phi_Q$, such that the condition in Definition 4.7 holds for them; *i.e.,* there exists states (p_i, q_i), (p_j, q_j), (p_k, q_k), and (p_l, q_l) as required in the definition. We consider the situation where a complex transition in $\mathcal{T}_P$ appears before a transition in $\mathcal{T}_Q$; *i.e.,* the predicate including parts `(a)`, `(b)` and `(c)` in Definition 4.7 is true. The proof for the other case is symmetric to this case.

We present our proof in three stages, based on whether: (i) the schedules of (u, c, u') and (v, d, v') are action-disjoint, (ii) (u, c, u')'s schedule is embedded in (v, d, v')'s, or (iii) (v, d, v')'s schedule is embedded in (u, c, u')'s.

1. Let us first consider the case when the schedules of $(u, c, u') \in \phi_P$ and $(v, d, v') \in \phi_Q$ are action-disjoint. For this case, we show that such a (p_k, q_k) cannot be reached from state (p_j, q_j). Let us consider state (p_j, q_j). Since $p_j \in \mathcal{V}^{Int}_P$, according to Definition 4.5, state (p_j, q_j) can only be the source of the transitions that are created by sets `(3)` and `(4)`. If set `(3)` creates transition $((p_j, q_j), a_j, (p_{j+1}, q_{j+1}))$, then it means that at state $q_j \in \mathcal{V}^{N}_Q$, action a_j is enabled, but we know that $a_j \neq d_0$, because the schedule of (u, c, u') and (v, d, u') are disjoint, and thus state $q_{j+1} \notin (\mathit{states}_Q(v, d, v') - \{v, v'\})$, which means it is not possible to reach state (p_k, q_k), such that $q_k \in (\mathit{states}_Q(v, d, v') - \{v, v'\})$. Similarly, if set `(4)` can create transition $((p_j, q_j), a_j, (p_{j+1}, q_j))$, then state (p_{j+1}, q_j) is in a similar situation as state (p_j, q_j), and there does not exist a sequence of transitions that would allow (p_{j+1}, q_j) to reach state (p_k, q_k), where $q_k \in (\mathit{states}_Q(v, d, v') - \{v, v'\})$. As such, if the schedules of two complex transitions are action-disjoint, such a (p_k, q_k) cannot exist. Next we consider the two cases where the schedules of two complex transitions are not disjoint; *i.e.,* one is a subsequence of the other.

2. If (u, c, u') is embedded in (v, d, v'), then again, we prove that state (p_k, q_k) cannot be reached from (p_j, q_j). Let us again consider state (p_j, q_j). State (p_j, q_j) can only initiate transitions that are created by set `(3)`, because $c \Subset d$. But we know that $q_j \in \mathcal{V}^{Int}_Q$ is not enabled with action a_j, because $c \Subset d$, and the schedule of a complex transition consists of transitions on distinct actions. As such, depending on whether: (i) $a_j \in \mathcal{A}^O_P$ and q_j not enabled with a_j, (ii) $a_j \in \mathcal{A}^I_P$ and q_j not enabled with a_j, or (iii) q_j enabled with a_j, (p_j, q_j) will be either: (i) an illegal state, (ii) an internal state without an outgoing transition, or (iii) would initiate $((p_j, q_j), a_j, (p_{j+1}, q_{j+1}))$ transition where $q_{j+1} \notin (\mathit{states}_Q(v, d, v') - \{v, v'\})$, respectively. For all cases, it is impossible to reach a state (p_k, q_k), where $q_k \in (\mathit{states}_Q(v, d, v') - \{v, v'\})$, and thus such a (p_k, q_k) cannot exist.

3. The last case is when (u, c, u') embeds (v, d, v'), or (v, d, v') is embedded in (u, c, u'). We consider two subcases for this case, namely the case where $a_j = d_0$ and the case where $a_j \neq d_0$, and show for both subcases that state (p_k, q_k) cannot exist.

(a) If $a_j = d_0$ then since (v, d, v') is embedded in (u, c, u'), from state (p_j, q_j) by following the sequence of transitions that are created by sets `(3)` and `(4)`, finally state (p_l, q_l) can be reached; furthermore, this sequence is the only sequence of transitions that can be created. But this means that we either reach state (p_l, q_l) before reaching (p_k, q_k) or $(p_l, q_k) = (p_l, q_l)$, which are both unacceptable scenarios according to Definition 4.7. Therefore, state (p_k, q_k), as characterized in Definition 4.7, cannot exist.

(b) If $a_j \neq d_0$ then, there are two possibilities: (i) from state (p_j, q_j), following a sequence of transitions, that are created by sets `(3)` and `(4)`, and eventually arriving at state (p_l, q_l), before reaching state (p_k, q_k); and (ii) arriving at an illegal state (p_e, q_e), where $(j \leq e < n)$, before reaching state (p_l, q_l). But in both cases, state (p_k, q_k) cannot exist.

We considered different possible scenarios for a pair of complex transitions, and showed that in all cases state (p_k, q_k), as characterized in Definition 4.7 cannot exist. Therefore, the legal interleaved product of two composable IACAs cannot create partially overlapped execution fragments, and this concludes our proof. □

Lemma 4.6 Considering two composable IACAs, P and Q, an internal state $(p, q) \in \mathcal{V}^{Int}_{P \circledast Q}$, its execution fragment $s = \Delta(p, q)$, and the projections of s, $\pi_P(s)$ and $\pi_Q(s)$, then at least one of the following is true:

- $\exists!(u, c, u') \in \phi_P \cdot \mathit{frag}_P(u, c, u') = \pi_P(s)$, or
- $\exists!(v, d, v') \in \phi_Q \cdot \mathit{frag}_Q(v, d, v') = \pi_Q(s)$.

where $\exists!$ means "there exists a unique."

Proof:
Let us consider the execution fragment $\Delta(p, q) = \langle (p_0, q_0), a_0, (p_1, q_1), \ldots, (p_n, q_n) \rangle$, and $1^{st}\mathit{state}(\Delta(p, q))$. Let us assume that $1^{st}\mathit{state}(\Delta(p, q)) = i \neq 0$. Without loss of generality, let us assume that $p_i \in \mathcal{V}^{Int}_P$ and $q_i \in \mathcal{V}^{N}_Q$. A complex transition can be associated with internal state p_i, namely $\mathit{CompTran}_P(p_i) = (u, c, u')$. We let $\mathit{frag}_P(u, c, u') = \langle u, c_0, u_0, \ldots, u_{x-1}, a_x, u' \rangle$ and $p_i = u_l$, where $(0 \leq l < x)$. We prove that $\mathit{frag}_P(u, c, u') = \pi_P(s)$. We present our proof in two steps: (i) we show that all states (p_j, q_j), where $(0 \leq j < i)$, follow their corresponding states in $\mathit{frag}_P(u, c, u')$, and (ii) we show that all states (p_k, q_k), where $(i \leq k \leq n)$, follow their corresponding states in $\mathit{frag}_P(u, c, u')$.

1. Let us first consider the (p_j, q_j) states as characterized above. We first notice that state (p_i, q_i), or alternatively (u_l, q_i), can be reached by a transition, $((p_{i-1}, q_{i-1}), a_{i-1}, (p_i, q_i))$, that is created by either set `(1)` or set `(3)`.[14] Regardless of whether set `(1)` or `(3)` creates transition

[14]The reason the other sets cannot create such a transition is as follows. Set `(2)` cannot create such a transition because $p_i \in \mathcal{V}^{Int}_P$. Set `(4)` cannot create such a transition either, because if it could, then state (p_{i-1}, q_{i-1}) would have been the the first state as characterized above, instead of state (p_i, q_i). The case for set `(5)` is more complicated. Let us denote $\mathit{CompTran}_Q(q_{i-1}) = (v, d, v')$ and $\mathit{frag}_Q(v, d, v') = \langle v, d_0, v_0, \ldots, v_{y-1}, d_y, v' \rangle$. There are two possibilities for complex transition (u, c, u') and (v, d, v'). If their schedules are action-disjoint, then state (p_{i-1}, q_{i-1}) is not reachable according to

$((p_{i-1}, q_{i-1}), a_{i-1}, (p_i, q_i))$, there should exist a transition $(p_{i-1}, a_{i-1}, p_i) \in \mathcal{T}_P$; but since $p_i = u_l$, we have: $(u_{l-1}, c_l, u_l) \in \mathcal{T}_P$, $p_{i-1} = u_{l-1}$ and $c_l = a_{i-1}$.

Let us now consider state (p_{i-1}, q_{i-1}). If $(i-1) = 0$, then $u = p_0$, and we have successfully shown that all (p_j, q_j)s follow the complex fragment of complex transition (u, c, u'). If $(i-1) \neq 0$, then there should exist a transition $((p_{i-2}, q_{i-2}), a_{i-2}, (p_{i-1}, q_{i-1}))$, which can be created either by set `(3)`, or set `(4)`, and furthermore $p_{i-1} \in \mathcal{V}_P^{Int}$ and $q_{i-1} \in \mathcal{V}_Q^{Int}$.[15] Since transition $((p_{i-2}, q_{i-2}), a_{i-2}, (p_{i-1}, q_{i-1}))$ is created either by set `(3)` or `(4)`, then there should exist $(p_{i-2}, a_{i-2}, p_{i-1}) \in \mathcal{T}_P$; but since $p_{i-1} = u_{l-1}$, we have: $(u_{l-2}, c_{l-1}, u_{l-1}) \in \mathcal{T}_P$, $p_{i-2} = u_{l-2}$, and $c_{l-1} = a_{i-2}$. In the same fashion, we can trace back the remaining (p_j, q_j)s until we reach state (p_0, q_0); all such transitions and states comply with the complex fragment of complex transition (u, c, u'). Since (p_i, q_i) is the first internal state that either $p_i \in \mathcal{V}_P^{Int}$ or $q_i \in \mathcal{V}_Q^{Int}$, it has to be the case that $u = p_0$ and $q_0 \in \mathcal{V}_Q^N$.

We still need to consider the states in $\Delta(p, q)$ that appear after (p_i, q_i), namely (p_k, q_k) states, where $(i \leq k \leq n)$. Next, we prove that (p_k, q_k)s also comply with the complex fragment of complex transition (u, c, u').

2. Considering state (p_{i+1}, q_{i+1}), it can be reached from state (p_i, q_i) through a transition $((p_i, q_i), a_i, (p_{i+1}, q_{i+1}))$ that can be created only by set `(3)` or `(4)`. As such there should exist a transition $(p_i, a_i, p_{i+1}) \in \mathcal{T}_P$. But since $p_i = u_l$, we have: $(u_l, c_{l+1}, u_{l+1}) \in \mathcal{T}_P$, $p_{i+1} = u_{l+1}$, and $c_{l+1} = a_i$. We proceed our proof with considering four separate subcases, depending on whether p_{i+1} or q_{i+1} are internal states or not.

   (a) $p_{i+1} \in \mathcal{V}_P^N$ and $q_{i+1} \in \mathcal{V}_Q^N$: Arriving at such a (p_{i+1}, q_{i+1}) state means that all (p_k, q_k) states follow the complex fragment of (u, c, u'), as desired. Furthermore, it should be the case that $p_n = u'$ and $q_n \in \mathcal{V}_Q^N$, otherwise $\Delta(p, q)$ will be a partially overlapped execution fragment, which is not possible according to Lemma 4.5.

   (b) $p_{i+1} \in \mathcal{V}_P^{Int}$ and $q_{i+1} \in \mathcal{V}_Q^N$: It can be shown, with the same reasoning as mentioned above for state (p_i, q_i), that state (p_{i+1}, q_{i+1}), by following the complex fragment of complex action (u, c, u'), will reach state (p_{i+2}, q_{i+2}).

   (c) $p_{i+1} \in \mathcal{V}_P^N$ and $q_{i+1} \in \mathcal{V}_Q^{Int}$: This subcase is not possible, because it would mean that $\Delta(p, q)$ is a partially overlapped execution fragment, which according to Lemma 4.5 is not possible.

Lemma 4.1. The other possibility is that complex transition (u, c, u') is embedded in complex transition (v, d, v'). Complex transition (v, d, v') cannot be embedded in complex transition (u, c, u'), because at internal state (p_{i-1}, q_{i-1}), set `(5)` has created a transition, which would mean that $c \Subset d$. As such, it should be the case that $a_i \in Shared(P, Q)$ and $a_i \in set(sched_Q(v, d, v'))$. We denote the internal state in complex fragment of (v, d, v') that is enabled with a_i, as v_r. Internal state v_r appears before q_{i-1} in complex fragment of (v, d, v'). If $a_i \in \mathcal{A}_P^O$, then state (p_i, q_i) is an illegal state, and it is not reachable. If $a_i \in \mathcal{A}_Q^O$, then either internal state (p_i, v_r) is an internal state of $\Delta(p, q)$, which means (p_i, q_i) is not reachable in $\Delta(p, q)$, or an internal state (p_s, v_r), where $p_s \neq p_i$, is an internal state of $\Delta(p, q)$. But state (p_s, v_r) is an illegal state, and again state (p_i, q_i) is not reachable. As such, set `(5)` cannot create such a transition, because state (p_i, q_i) becomes unreachable, which is not possible.

[15] We should note that set `(5)` cannot possibly create such a transition. The reason is similar to the reason above for why set `(5)` cannot create transition $((p_{i-1}, q_{i-1}), a_{i-1}, (p_i, q_i))$.

(d) $p_{i+1} \in \mathcal{V}_P^{Int}$ and $q_{i+1} \in \mathcal{V}_Q^{Int}$: State (p_{i+2}, q_{i+2}) can be reached through transition $((p_{i+1}, q_{i+1}), a_{i+1}, (p_{i+2}, q_{i+2}))$, which can be created by either set `(3)` or set `(4)`.[16] But since both sets `(3)` and `(4)` follow the complex fragment of complex transition (u, c, u'), so does complex state (p_{i+1}, q_{i+1}).

Having shown that all states possibly reachable from (p_i, q_i) follow the complex fragment of complex transition (u, c, u'), it means that all (p_k, q_k) states indeed follow the complex fragment of complex transition (u, c, u').

It is also possible that (p_i, q_i), as characterized above, is comprised of states $p_i \in \mathcal{V}_Q^N$ and $q_i \in \mathcal{V}_P^{Int}$; for that case, the proof is symmetric, but state q_i and its corresponding complex transition, say $\mathit{CompTran}_Q(q_i) = (v, d, v')$, is considered. As such, we proved that regardless of whether p_i or q_i is an internal state, there is a complex transition in P or Q that has the same schedule as $\pi_P(\Delta(p, q))$ or $\pi_Q(\Delta(p, q))$, respectively.

Lastly, we need to prove the case that there does not exist such a (p_i, q_i) state, *i.e.*, $1^{st}\mathit{state}(\Delta(p, q)) = 0$. If $1^{st}\mathit{state}(\Delta(p, q)) = 0$, then all p_ms and q_ms, where $0 < m < n$, are internal states. But such states can only be reached via transitions created by set `(3)`, and follow the schedule of two complex transitions in P and Q that have exactly the same schedules, which is an acceptable case of our lemma. Therefore, we conclude that: given an internal state (p, q), and its corresponding execution fragment $\Delta(p, q)$, the sequence of states and actions in $\Delta(p, q)$ can be associated with exactly one complex fragment in P, Q, or both, which concludes our proof.

□

Theorem 4.1 Given two composable IACAs, P and Q, $P \parallel Q$ is a well-formed IACA.

Proof:
Definition 4.11 defines the elements of an IACA for $P \parallel Q$ properly. To prove the well-formedness of $P \parallel Q$, it suffices to prove that: (i) $\mathit{frag}_{P\parallel Q}$ is defined for all complex transitions in $\phi_{P\parallel Q}$, and (ii) $P \parallel Q$ follows the well-formedness criteria in Definition 3.2.

Let us first consider $\mathit{frag}_{P\parallel Q}$. The complex fragment of a complex transition $((p, q), c, (p', q'))$ is defined by simply identifying *an* internal state $(x, y) \in \mathcal{V}_{P\parallel Q}^{Int}$ such that $\mathit{complex}_{P\circledast Q}(x, y) = ((p, q), c, (p', q'))$. We can then define $\mathit{frag}_{P\parallel Q}((p, q), c, (p', q')) = \Delta(x, y)$. State (x, y) exists, since otherwise complex transition $((p, q), c, (p', q'))$ does not exist. Furthermore, complex fragment of $((p, q), c, (p', q'))$, *i.e.*, $\Delta(x, y)$, follows the criteria defined for complex fragments: first, according to Lemma 4.4, an execution fragment $\Delta(x, y)$ follows the structure of a complex fragment, and second, according to Lemma 4.6, it also follows the complex fragment of a complex transition in P, Q, or both.

To prove the well-formedness of $P \parallel Q$ as an IACA, we need to prove that: (i) complex transitions in $P \parallel Q$ that have same schedules, will have the same complex actions, and vice versa; (ii) every internal state is associated with a complex transitions; (iii) $P \parallel Q$ is deterministic, and (iv) $P \parallel Q$ is incoming deterministic.

[16] Similar to the rationale above, in part 1 of our proof, it can be shown that set `(5)` cannot create such a $((p_{i+1}, q_{i+1}), a_{i+1}, (p_{i+2}, q_{i+2}))$ transition.

The way we derive complex actions in Definition 4.10 guarantees that criterion (i) holds. If criterion (i) does not hold then it would mean that either P, Q, or both, are not well-formed IACAs, which is not possible. The criterion (ii) holds because of Lemma 4.4, which guarantees that each internal state is associated with a unique complex transition.

Criteria (iii) and (iv) hold because the way transitions are computed in Definition 4.5. The only scenario that a state in $(u, v) \in \mathcal{V}_{P\|Q}$ could have more than one outgoing and/or incoming transitions with the same action, is when either u, v, or both have more than one incoming and/or outgoing transitions in $\mathcal{T}_P$ and $\mathcal{T}_Q$, respectively, which is not possible because P and Q are well-formed IACAs. Having shown that $P \parallel Q$ follows all criteria of being a well-formed IACA, we conclude that it is indeed a well-formed IACA.

□

Theorem 4.2 Given two composable IACAs, P and Q, $P \parallel Q = Q \parallel P$.

Proof:
By inspecting the composability criteria, and the four steps of the composition, it can be observed that the composition operation is defined entirely symmetric with respect to P and Q, and thus is commutative.

□

Theorem 5.1 Given three IACAs, P, Q and P', such that $P' \preceq P$, P and Q are composable, and P' and Q are composable, $(P' \parallel Q) \preceq (P \parallel Q)$, if the following conditions hold:

1. $Shared(P, Q) = Shared(P', Q)$

   (P' and P communicate with Q through the same set of shared actions), and

2. $\forall (p', p) \cdot (p' \preceq p) \Rightarrow$
   $((\mathcal{A}^I_{P'}(p') \backslash \mathcal{A}^I_P(p)) \cap Shared(P, Q) = \emptyset) \wedge$
   $((\mathcal{A}^O_P(p) \backslash \mathcal{A}^O_{P'}(p')) \cap Shared(P, Q) = \emptyset)$

   (States of P' that are in the simulation relation with P do not introduce extra (nor eliminate) actions that belong to the shared actions of P and Q), and

3. $\forall (p', p) \cdot (p' \preceq p) \Rightarrow$
   $\forall (p, c, u) \in \phi_P \cdot (\exists (p', c, v) \in \phi_{P'} \wedge (c \in C^O_{P'}))$
   $\Rightarrow ((set(\mathsf{sched}_P(p, c, u))) \backslash (set(\mathsf{sched}_{P'}(p', c, v)))) \cap Shared(P, Q)) = \emptyset$

   (States of P' that are in the simulation relation with P should not have output complex actions that are shared with P, and miss some normal output actions in their schedules that belong to the shared actions of P and Q), and

4. $\forall (p', p) \cdot (p' \preceq p) \Rightarrow$
   $\forall (p', c, v) \in \phi_{P'} \cdot (\exists (p, c, u) \in \phi_P \wedge (c \in C^I_P))$
   $\Rightarrow ((set(\mathsf{sched}_{P'}(p', c, v))) \backslash (set(\mathsf{sched}_P(p, c, u)))) \cap Shared(P, Q) = \emptyset$

   (States of P' that are in the simulation relation with P should not have input complex actions shared with P that introduce new simple input actions in their schedules that belong to the shared actions of P and Q).

Proof:
To prove that $(P' \parallel Q) \preceq (P \parallel Q)$, we should show that the five conditions for refinement, as stated in Definition 5.1, hold. To show that condition (1) holds, consider $\mathcal{A}^I_{P\|Q}$. By the definition of composition, $\mathcal{A}^I_{P\|Q} = (\mathcal{A}^I_P \cup \mathcal{A}^I_Q) \backslash Shared(P, Q)$. Since $Shared(P', Q) = Shared(P, Q)$ and $\mathcal{A}^I_{P'} \supseteq \mathcal{A}^I_P$, it follows that $\mathcal{A}^I_{P'\|Q} \supseteq \mathcal{A}^I_{P\|Q}$. Similarly it can be shown that condition (2) holds, *i.e.*, $\mathcal{A}^O_{P'\|Q} \subseteq \mathcal{A}^O_{P\|Q}$. To show that condition (3) holds, we should show that $C^I_{P'\|Q} \supseteq C^I_{P\|Q}$, which is true, because $P' \preceq P$, $C_{P'} \supseteq C_P$, and $C_{P'\|Q} = C_{P'} \cup C_Q$. Similarly it can be shown that condition (4) in the refinement definition holds.

As for condition (5) of the refinement relation definition, we can construct an alternating simulation relation from the states of $(P' \parallel Q)$ to $(P \parallel Q)$. We define our simulation relation as $(p', q) \preceq' (p, q)$ for all reachable states $(p, q) \in \mathcal{V}^N_{P\|Q}$ and all states p' such that $p' \preceq p$. All such $(p', q) \in \mathcal{V}^N_{P\|Q}$ are reachable states because for all such states, based on the conditions of the theorem, if (p', q) is an illegal state, then (p, q) should be an illegal state as well. Condition 2 of the theorem does not allow P' not to provide the shared actions that P provides, and therefore it is impossible for (p', q) to be an illegal state, when (p, q) is not. Furthermore, conditions 3 and 4 of the theorem guarantee that $P' \parallel Q$ does not have more illegal internal states, on the common complex actions between P' and P, than $P \parallel Q$ does. If $P' \parallel Q$ has an illegal internal state (u', v) that $P \parallel Q$ does not have, then it means that $P \parallel Q$ has an internal state (u, v) belonging to a complex transition of $P \parallel Q$, and (u, v) can synchronize on a shared action, and (u', v) cannot synchronize on the same shared action in $P' \parallel Q$. This situation is impossible, because conditions 3 and 4 of the theorem disallow this situation. As such, for any (p, q), we have (p', q) which can simulate (p, q), and this concludes our proof. □