Interface Automata with Complex Actions - Extended Version

Shahram Esmaeilsabzali   Nancy A. Day   Farhad Mavaddat

David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1
{sesmaeil,nday,fmavaddat}@cs.uwaterloo.ca

Technical Report: CS-2005-26 (Revised May 15th, 2006)

Abstract. Many formalisms use interleaving to model concurrency. To describe some system behaviours appropriately, we need to limit interleaving. For example, in component-based systems, we 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 add 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 and refinement operations 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.,* [16, 14, 13, 12]. It provides an intuitive and elegant means for modelling and reasoning about systems' behaviours. Henzinger, Manna, and Pnueli, in [11], 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.

However, for some systems, interleaving is not always adequate. Some software artifacts have multiple constituent elements but 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 interleaving with complex actions as its semantics for concurrency. A *complex action* consists of multiple normal actions that cannot be interleaved with the behaviour of another component. In component-based systems, 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.,* [6]. To model the details of the parameter communication, the arrival of the inputs should not be interleaved with the behaviour of another component. Thus, we require complex actions to model the semantics of the concurrent behaviour of a component-based system at this level of detail. In Web services, 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.

Various approaches have been proposed for grouping multiple actions together (atomic actions)[1][3, 10] or refining a single action into multiple actions (action refinement) [1, 17]. Many of these approaches are proposed in a process algebraic context. We are interested in defining an automata-based model with complex actions. We want our model to comply as closely as possible with the class of *interface models*, introduced by de Alfaro and Henzinger [7, 6], which are a class of formalism suitable for describing component-based systems.

The paradigm of component-based development is based upon the principle of reusing already developed components to create new systems. It should be possible to design each component independently from the other components to make it reusable in different contexts. However, no matter how generically a component is designed, to be useful in practice, assumptions need to be made about its environment. Assuming a less restrictive environment will make the component applicable in more contexts.

Interface models are designed to facilitate 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 propertiers of interface models.

Our IACA model is an extension of de Alfaro and Henzinger's *interface automata (IA)* [6, 8], which is an interface model. An interface automaton captures aspects of the functionality of a component along with its assumptions about its environment. 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. In IA, the system is described via sets of input and output actions. As an automaton-based model, an IA captures the required temporal order of the input and output behaviours of a component. In this way, it captures what the component assumes about its environment.

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 *incremental*

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

development of a system because the order in which different components are composed does not affect the result of composition.

The refinement of an IA is defined based on the assumed helpful environment. A refinement of an IA must have fewer 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 and refinement operators for IA together establish the top-down design property of IAs.

IACA extends IA with complex actions, which are needed to model actions that should not be interleaved in composition, such as parameters of methods and complex messages of Web services. Interleaving usually relies on a notion of *atomicity*, which is the idea that an action in the system is indivisible in time [15, 16]. 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 [11]).

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, $pay(credit\_card\_no, amou$
$nt)$. 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 now 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 action, then our model is not precisely modelling the system. On the other hand, we cannot choose to model *pay* as two actions: *credit_card_no* and *amount* in a sequence; because they do not represent meaningful atomic actions. What we really need is to have a *complex action*, which groups *credit_card_no* and *amount* as a single action, namely *pay*. The need to have complex actions arises because systems often tend to 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.

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. 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 [10]. Despite the non-associativity of composition, we belive that IACA is useful for modelling software artifacts with complex actions.

Our definition of composition leads to a natural definition for the IACA refinement relation. IACA's refinement relation is intuitively comparable with programming languages' concepts such as subclasses and optional parameters for function calls. IACA refinement follows the IA refinement principles: 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 report is organized as follows. We begin by providing background on IA. Next, we describe IACA and its composition operator and refinement relation. In Section 4, we compare IACA with similar models that support complex actions, and summarize and discuss future work in Section 5. Throughout the report, we provide examples of the use of IACA to model Web services. This report is an extended version of our original work [9]. Here, we have included a more thorough and formal treatment of IACA composition and refinement, and provide outlines of the proofs of the properties of IACA.

2 Background: Interface Automata

Interface automata (IA) [6, 8], 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* [7], which are intended to specify concisely *how* systems can be used and to adhere to certain well-formedness criteria which make them appropriate for modelling component-based systems. 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: $(P' \parallel Q) \preceq (P \parallel Q)$.

IAs are well-formed interface models. They are syntactically similar to *Input/Output Automata* [13], 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.

Definition 1. *An* interface automaton *(IA), $P = \langle V_P, i_P, \mathcal{A}_P^I, \mathcal{A}_P^O, \mathcal{A}_P^H, \tau_P\rangle$, consists of V_P a finite set of states, $i_P \in V_P$ the initial state, $\mathcal{A}_P^I$, $\mathcal{A}_P^O$ and $\mathcal{A}_P^H$, which are disjoint sets of input, output, and hidden actions, respectively, and τ_P the set of transitions between states such that $\tau_P \subseteq V_P \times \mathcal{A}_P \times V_P$, where $\mathcal{A}_P = \mathcal{A}_P^I \cup \mathcal{A}_P^O \cup \mathcal{A}_P^H$. Well-formed IAs must be input deterministic [8], i.e., for any two input transitions (u, a, v) and (u, a, l), $v = l$.*

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

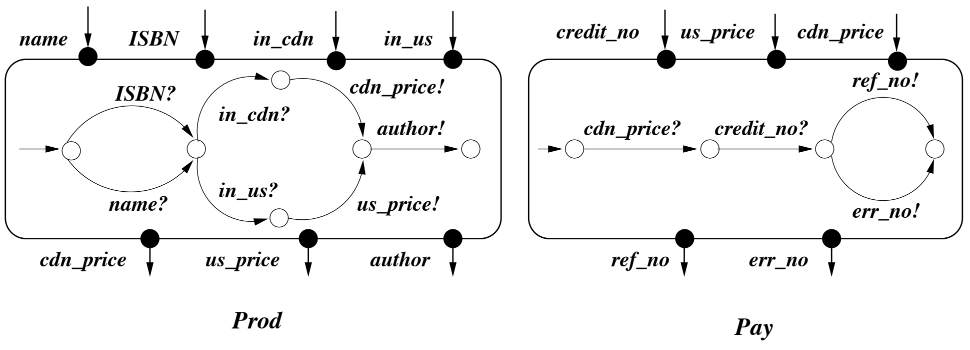

Fig. 1: Two IAs: *Prod* and *Pay.*

2.1 IA Composition

The composition of two IAs consists of all possible interleaved transitions of the two IAs, except for those 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.

First, we define when two IAs are composable.

Definition 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:*

$$(\mathcal{A}_P^I \cap \mathcal{A}_Q^I) = (\mathcal{A}_P^O \cap \mathcal{A}_Q^O) = (\mathcal{A}_P^H \cap \mathcal{A}_Q) = (\mathcal{A}_Q^H \cap \mathcal{A}_P) = \emptyset$$

For two composable IAs, we can define their set of shared actions.

Definition 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 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 begin by defining the interleaved product of actions and states of two composable IAs:

Definition 4. *For two composable IAs, P and Q, their* interleaved product*, $P \otimes Q$, is defined as follows.*

$$\begin{aligned}
V_{P\otimes Q} &= V_P \times V_Q \\
i_{P\otimes Q} &= i_P \times i_Q \\
\mathcal{A}^I_{P\otimes Q} &= \mathcal{A}^I_P \cup \mathcal{A}^I_Q \backslash \mathit{Shared}(P,Q) \\
\mathcal{A}^O_{P\otimes Q} &= \mathcal{A}^O_P \cup \mathcal{A}^O_Q \backslash \mathit{Shared}(P,Q) \\
\mathcal{A}^H_{P\otimes Q} &= \mathcal{A}^H_P \cup \mathcal{A}^H_Q \cup \mathit{Shared}(P,Q) \\
\tau_{P\otimes Q} &= \left(\begin{array}{l}
\quad \{((p,q),a,(p',q)) \mid (p,a,p') \in \tau_P \wedge a \notin \mathit{Shared}(P,Q) \wedge q \in V_Q\} \\
\cup \{((p,q),a,(p,q')) \mid (q,a,q') \in \tau_Q \wedge a \notin \mathit{Shared}(P,Q) \wedge p \in V_P\} \\
\cup \{((p,q),a,(p',q')) \mid (p,a,p') \in \tau_P \wedge (q,a,q') \in \tau_Q \wedge a \in \mathit{Shared}(P,Q)\}
\end{array}\right)
\end{aligned}$$

In Figure 2, 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 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.*

$$\mathit{Illegal}(P,Q) = \left\{ (v,u) \in V_P \times V_Q \mid \exists a \in \mathit{Shared}(P,Q) \cdot \begin{pmatrix} a \in \mathcal{A}^O_P(v) \wedge a \notin \mathcal{A}^I_Q(v) \\ \vee \\ a \in \mathcal{A}^O_Q(u) \wedge a \notin \mathcal{A}^I_P(v) \end{pmatrix} \right\}$$

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.

Composition of two composable IAs, P and Q, can be formally defined based on a notion of a *legal environment* for them.

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.,* $\mathcal{A}^I_E = \mathcal{A}^O_P$, and

- $Illegal(P, E) = \emptyset$.

In the following three definitions, we formalize the notions of: *legal environment*, *compatible states* of two composable IAs, and finally the composition of two composable IAs.

Definition 6. *Given two composable IAs, P and Q, a* legal environment *for the pair (P, Q) is an environment E for $P \otimes Q$ such that no state in $Illegal(P, Q) \times E$ is reachable in $(P \otimes Q) \otimes E$.*

We can now define the set of compatible states for two composable IAs as the existence of a legal environment for their product.

Definition 7. *Given two composable IAs P and Q, a pair $(v, u) \in V_P \times V_Q$ is* compatible *if there is an environment E for $P \otimes Q$ such that no state in $Illegal(P, Q) \times V_E$ is reachable in $(P \otimes Q) \otimes E$ from the state $((v, u), i_E)$. $Cmp(P, Q)$ is the set of all such states.*

Having defined the compatibility of two composable IAs, we can now formally define the composition of two IAs.

Definition 8. *Given two composable IAs, P and Q, $P \parallel Q$ is an IA with the same actions as $P \otimes Q$, states $V_{P\|Q} = Cmp(P, Q)$, initial state $i_{P\|Q} = \{i_{P\otimes Q}\} \cap Cmp(P, Q)$, and transitions $\tau_{P\|Q} = \tau_{P\otimes Q} \cap \{Cmp(P, Q) \times \mathcal{A}_{P\otimes Q} \times Cmp(P, Q)\}$.*

Let us look at our example in Figure 2 again. In state (0,2') of the interleaved product of A and B, IA B is immediately ready to send b but IA 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; we call such paths *autonomous paths*. The environment does not have any control over output and hidden transitions. State (0,1') (shown with filled in circle) 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'). Non-reachable states are also eliminated. The IA resulting from the composition of IAs A and B is labelled $A \parallel B$ in Figure 2.

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.

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, *in_us*? still appears as an input of the composition.

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

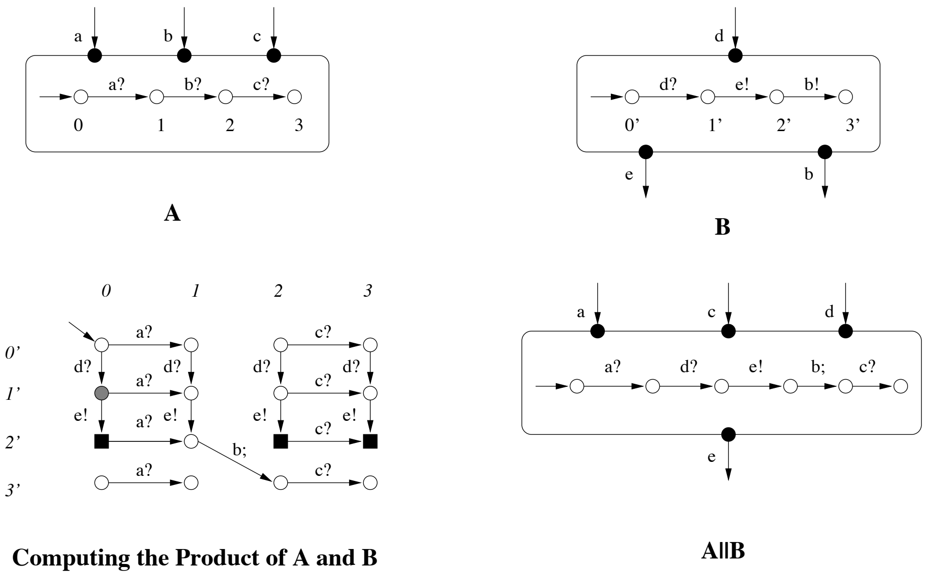

Fig. 2: Two composable IAs, A and B, and their composition $A \parallel B$.

```
Algorithm RemoveIllegal($P$, $Q$, $\tau_{P\otimes Q}$, $V_{P\otimes Q}$) ;
Variables  : $\tau_{P\|Q}$, $V_{P\|Q}$, $L[\,]$, $i$, $temp$ ;
begin
    $i = 0$ ;
    $L_i = Illegal(P, Q)$ ;
    repeat
        /* Backtrack one transition to identify more illegal states */
        $temp = \{(p, q) \mid \exists((p, q), a, (p', q')) \in \tau_{P\otimes Q} \cdot (p', q') \in L_i \ \wedge a \in (\mathcal{A}^O_{P\otimes Q} \cup \mathcal{A}^H_{P\otimes Q})\}$ ;
        /* Add to the set of illegal states */
        $L_{i+1} = \ L_i \cup temp$ ;
    until $L_i == L_{i+1}$;
    $V_{P\|Q} = V_{P\otimes Q} \backslash L_i$;
    $\tau_{P\|Q} = \tau_{P\otimes Q} \backslash \{((p, q), a, (p', q')) \mid ((p, q) \in L_i) \vee ((p', q') \in L_i)\}$ ;
    return $\tau_{P\|Q}$, $V_{P\|Q}$;
end
```

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

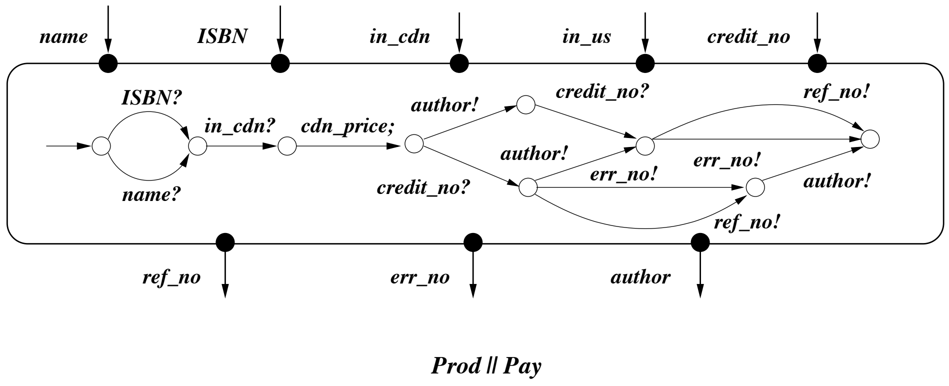

Fig. 4: $Prod \parallel Pay$ is the composition of two composable IAs in Figure 1.

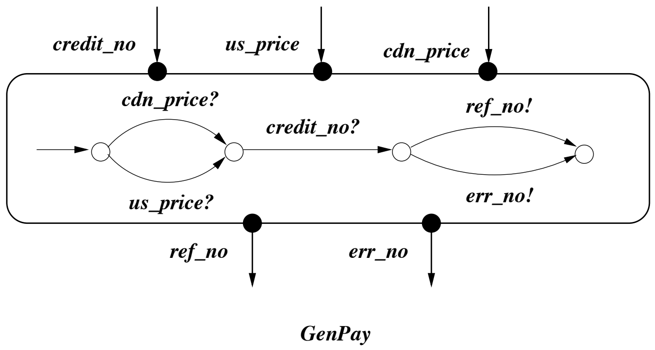

Fig. 5: $GenPay$ refines Pay in Figure 1.

more services than *Pay* 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, stated as $Q \preceq P$, then an *alternating simulation relation* [2] exists between the states of Q and P.[2] For simplicity, we refer to the alternating simulation relation as the refinement relation on states. For $q \in V_Q$ and $p \in V_P$, q refines p, $q \preceq p$, 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.

In the following, we formally describe the IA refinement relation. For a state $u \in V_P$, $closure_P(u)$ 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 9. *For IA P, and state $u \in V_P$, the set of* externally enabled input *($ExtEn^I_P(p)$) and* externally enabled output *($ExtEn^O_P(p)$) actions are:*

$$ExtEn^I_P(u) = \{a \mid \forall u' \in closure_P(u) \cdot a \in \mathcal{A}^I_P(u')\}$$
$$ExtEn^O_P(u) = \{a \mid \exists u' \in closure_P(u) \cdot a \in \mathcal{A}^O_P(u')\}$$

The externally enabled input and output actions are used to model the fact that an environment cannot detect the hidden transitions which 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 $u \in V_P$, we need to identify all the states that P can move through its hidden transitions, *i.e.,* identify $closure_P(u)$. A proper refinement of P, IA Q, needs to only 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 output to the environment than P does in any of the states belonging to $closure_P(u)$.

To check for refinement relation between two IAs, we should check for the existence of a refinement relation which includes the pair of the initial states of the two IAs, and furthermore, appropriately propagates the refinement relation to the other states of the two IAs. The following three definitions formalize the refinement relation.

[2] An alternating simulation relation can be distinguished 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.

Definition 10. *For a state p and an externally enabled action $a \in ExtEn^I_P(p) \cup ExtEn^O_P(p)$, the externally reachable states are defined as:*

$$Dest_P(p, a) = \{p' | \; \exists (r, a, p') \in \tau_P \cdot r \in closure_P(p)\}$$

Definition 11. *The binary relation,* alternating simulation $\preceq \; \subseteq V_Q \times V_P$*, between two states $q \in V_Q$ and $p \in V_P$ (represented 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.)
- $ExtEn^O_P(p) \supseteq ExtEn^O_Q(q)$
  (q has less, or the same, externally enabled outputs than p.)
- $\forall \; a \in (ExtEn^I_P(p) \cup ExtEn^O_Q(q)) \cdot \forall \; q' \in ExtDest_Q(q, a) \cdot \exists \; p' \in ExtDest_P(p, a) \cdot q' \preceq p'$
  (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.)

This relation essentially defines the basics of 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).

Definition 12. *IA Q* refines *IA P, $Q \preceq P$,if:*

- $\mathcal{A}^I_P \subseteq \mathcal{A}^I_Q$
- $\mathcal{A}^O_P \supseteq \mathcal{A}^O_Q$
- $i_Q \preceq i_P$

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 the 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 the environmental assumption of IACA *CompPay*, that *all* of its parameters should arrive in the correct order.

Complex actions represent either input or output behaviours, and thus should consist entirely of either inputs or outputs, possibly along with some hidden actions in the beginning of the sequence of actions of a certain complex action. Complex actions can only be a linear sequence

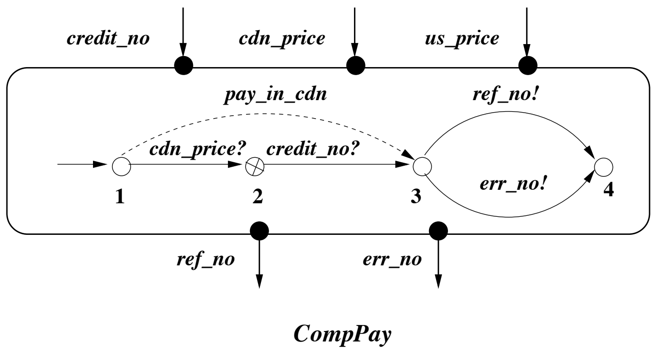

Fig. 6: IACA *CompPay* represents similar functionality as IA *Pay* in Figure 1.

of transitions. The states within a complex action are called *internal states* and are represented by circles with an "x" in them. In Figure 6, states 1, 3, and 4 are normal states of *CompPay* and state 2 is the only internal state.

In the remainder of this section, we first introduce IACA formally, and then define its composition operator and refinement relation.

Definition 13. *An* interface automaton with complex actions *(IACA) $P = \langle V_P^N, V_P^{int}, i_P, \mathcal{A}_P^I, \mathcal{A}_P^O, \mathcal{A}_P^H, \mathcal{A}_P^C, \tau_P, \phi_P \rangle$, has the following elements:*

- *V_P^N is the set of normal states.*
- *V_P^{Int} is the set of internal states. $V_P^N \cap V_P^{Int} = \emptyset$. The internal states are the ones inside a complex action. We denote $V_P = V_P^N \cup V_P^{Int}$ as the set of all states.*
- *i_P is the initial state. $i_P \in V_P^N$.*
- *$\mathcal{A}_P^I, \mathcal{A}_P^O, \mathcal{A}_P^H$ 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 $\mathcal{A}_P = \mathcal{A}_P^C \cup \mathcal{A}_P^N$.*
- *$\tau_P \subseteq V_P \times \mathcal{A}_P^N \times V_P$ is the set of normal (non-complex) transitions. We require that each $v \in V_P^{Int}$ is the source of* exactly *one transition and the destination of* exactly *one transition in τ_P.*
- *$\phi_P \subseteq V_P^N \times \mathcal{A}_P^C \times V_P^N$ is the set of complex transitions. Every $(u, c, v) \in \phi_P$ is associated with a sequence of non-complex transitions in τ_p called a* complex fragment*, shown as frag(u, c, v). A complex fragment is defined as an alternating sequence of states and normal actions: frag$(u, c, v) = \langle u, a_0, s_0, a_1, s_1, \ldots, s_{n-1}, a_n, v \rangle$ where*
  - *$\forall i \cdot s_i \in V_P^{Int}$ (all s_i's are internal states.), and*
  - *$(\forall i \cdot (a_i \in (\mathcal{A}_P^I \cup \mathcal{A}_P^H))) \vee (\forall i \cdot (a_i \in (\mathcal{A}_P^O \cup \mathcal{A}_P^H)))$ (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*
  - *$((u, a_0, s_0) \in \tau_P) \wedge ((s_{n-1}, a_n, v) \in \tau_P) \wedge \forall i (0 < i < n) \cdot (s_{i-1}, a_i, s_i) \in \tau_P$ (every transition is a non-complex transition.)*

As an example, in Figure 6, $frag(1, pay\_in\_cdn, 3) = \langle 1, cdn\_price?, 2, credit\_no?, 3 \rangle$. We use the function $sched(u, c, v)$ to represent the *schedule* of a complex transition as the sequence of normal actions that appear in the complex transition. As an example, $\mathit{sched}(1, \mathit{pay\_in\_cdn}, 3) = \langle \mathit{cdn\_price}?, \mathit{credit\_no}?\rangle$.

An IACA is well-formed if, first, each complex action is associated with a unique schedule, and second, it is input deterministic.

Definition 14. *An IACA* $P = \langle V_P^N, V_P^{int}, i_P, \mathcal{A}_P^I, \mathcal{A}_P^O, \mathcal{A}_P^H, \mathcal{A}_P^C, \tau_P, \phi_P\rangle$ *is* well-formed *if:*

- $(\forall(u,c,v) \in \phi_P \cdot \forall(u',c,v') \in \phi_P \cdot \mathit{sched}(u,c,v) = \mathit{sched}(u',c,v')) \wedge$
  $(\forall(u,c,v) \in \phi_P \cdot \forall(u',d,v') \in \phi_P \cdot \mathit{sched}(u,c,v) = \mathit{sched}(u',d,v') \Rightarrow d = c)$
  (Complex transitions with the same complex actions have same schedules, and complex actions with same schedules have the same complex action names.)
- $\forall(u,a,v) \in \tau_P, (u,a,v') \in \tau_P \cdot a \in \mathcal{A}_P^I \Rightarrow (v = v')$
  (Similar to IA, IACA is input deterministic.)

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

The constraint that internal states have exactly one incoming transition and exactly one outgoing transition, together with the constraints on ϕ_P above guarantee that a complex transition is associated with a unique sequence of non-complex transitions, and such a sequence is associated with a unique complex action.

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 15. *Given an IACA,* $P = \langle V_P^N, V_P^{int}, i_P, \mathcal{A}_P^I, \mathcal{A}_P^O, \mathcal{A}_P^H, \mathcal{A}_P^C, \tau_P, \phi_P\rangle$, $Q = \langle V_P, i_P, \mathcal{A}_P^I, \mathcal{A}_P^O, \mathcal{A}_P^H, \tau_P\rangle$ *is the* equivalent *IA for P.*

3.1 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 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, create hidden transitions, however, unlike IA, with a complex fragment, no interleaving can happen. In the composition of two IACAs, similar to IA composition, we combine the environmental assumptions of two IACAs. The assumption that a complex fragment has about its environment, *i.e.,* its schedule cannot be interleaved, is translated as the assumption that the other IACA in the composition provides 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 IACA composition of two composable IACAs is a subset of the IA composition of the equivalent IAs for the two IACAs.

Figure 7 shows the composition of *CompPay,* in Figure 6, and component *Prod* (now viewed as an IACA), 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*.

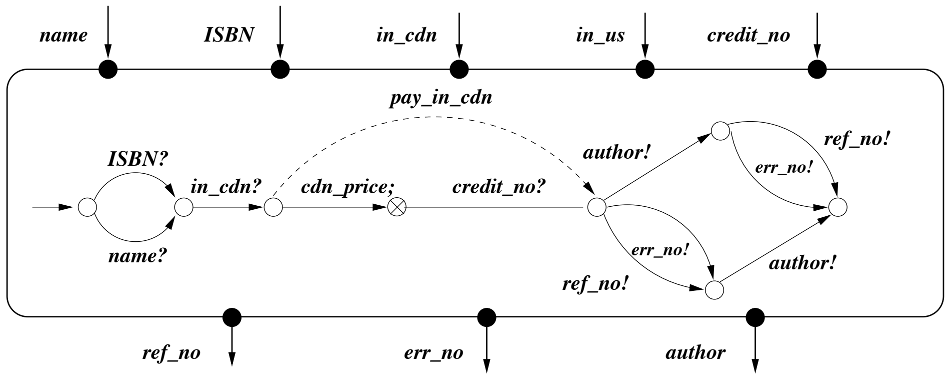

Fig. 7: $Prod \parallel CompPay$

The composability criteria in IACA includes the composability criteria of IA plus extra restrictions to handle complex transitions. If the schedule of two complex transitions, one from each IA, overlap without the schedule of one being a prefix of the schedule of the other, then it is not possible to maintain the sequentiality of either complex action. Furthermore, such a resulting complex transition cannot be associated with any of the complex transitions belonging to the two IACAs. As an example, in Figure 8 the complex action M of IACA B is a prefix of the complex action L of IACA A, *i.e.,* $M \sqsubseteq L$, and thus A and B are composable.

Definition 16. *Two IACAs are* composable*, if their equivalent IAs are composable, and for any given pair of complex transitions belonging to two IACAs, either their schedules do not overlap, or the schedule of one is a prefix of the other.*

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

Definition 17. *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 normal states* from the interleaved product as for IA.
3. Remove *illegal internal states* from the interleaved 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 mathematically, and use the simple IACAs of Figure 8 to illustrate these steps.

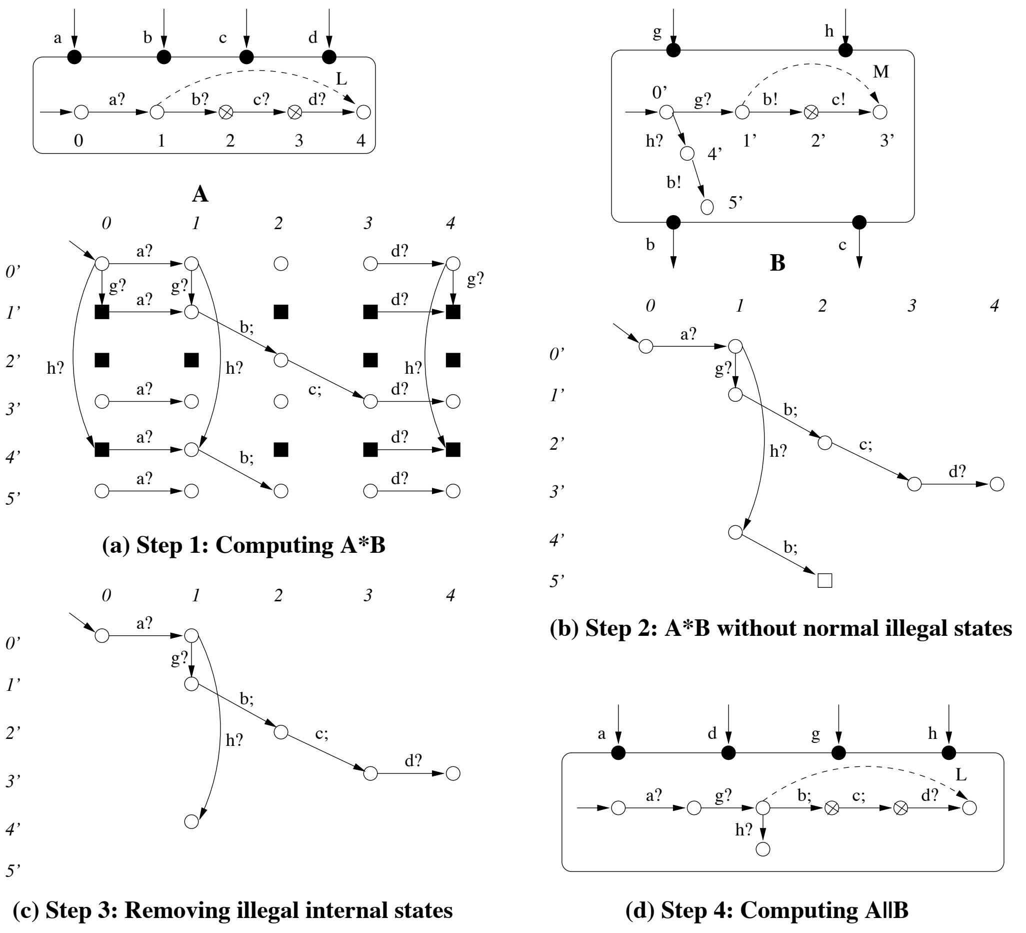
A
B
L
M
(a) Step 1: Computing A*B
(b) Step 2: A*B without normal illegal states
(c) Step 3: Removing illegal internal states
(d) Step 4: Computing A||B

Fig. 8: Computing IACA composition for two composable IACAs.

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 (a) of Figure 8 illustrates the first step of computing the composition of two IACAs in our example.

Definition 18. *For two composable IACAs, P and Q, we define their* interleaved product*, $P * Q$, as follows:*

$$V_{P*Q} = V_P \times V_Q$$
$$V^{Int}_{P*Q} = \{(p,q) \in (V_P \times V_Q) |\ (p \in V^{Int}_P) \vee (q \in V^{Int}_Q)\}$$
$$i_{P*Q} = (i_P, i_Q)$$
$$\mathcal{A}^I_{P*Q} = (\mathcal{A}^I_P \cup \mathcal{A}^I_Q) \setminus Shared(P,Q)$$
$$\mathcal{A}^O_{P*Q} = (\mathcal{A}^O_P \cup \mathcal{A}^O_Q) \setminus Shared(P,Q)$$
$$\mathcal{A}^H_{P*Q} = \mathcal{A}^H_P \cup \mathcal{A}^H_P \cup Shared(P,Q)$$
$$\mathcal{A}^C_{P*Q} = \mathcal{A}^C_P \cup \mathcal{A}^C_Q$$
$$\begin{aligned}
\tau_{P*Q} \quad = & \\
& \{((p,q),a,(p',q)) \ | (p,a,p') \in \tau_P \wedge a \notin Shared(P,Q) \wedge p \in V^N_P \wedge q \in V^N_Q\} && (1)\\
\cup & \{((p,q),a,(p,q')) \ | (q,a,q') \in \tau_Q \wedge a \notin Shared(P,Q) \wedge q \in V^N_Q \wedge p \in V^N_P\} && (2)\\
\cup & \{((p,q),a,(p',q')) \,| (p,a,p') \in \tau_P \wedge (q,a,q') \in \tau_Q \wedge a \in Shared(P,Q)\} && (3)\\
\cup & \{((p,q),a,(p',q)) \ | (p,a,p') \in \tau_P \wedge a \notin Shared(P,Q) \wedge p \in V^{Int}_P \wedge q \in V^N_Q\} && (4)\\
\cup & \{((p,q),a,(p,q')) \ | (q,a,q') \in \tau_Q \wedge a \notin Shared(P,Q) \wedge q \in V^{Int}_Q \wedge p \in V^N_P\} && (5)
\end{aligned}$$

We call the sets V_{P*Q} and τ_{P*Q}, the set of *interleaved states* and *interleaved transitions* of P and Q, respectively.

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 creating the composition.

For each state $(p,q) \in V_{P*Q}$, if either p or q is an internal state, then (p,q) is an internal state. For example, in part (a) of Figure 8, all states associated with state 3 are internal states, and have transitions only on action $d?$. Internal states of $P * Q$ do not interleave with other actions.

The third set of interleaved transitions, in Definition 18, consists of synchronizations that can happen between two IACAs on shared actions. The composability criteria guarantee that each action a is an input action of one IACA and an output of the other IACA. The first set of interleaved transitions consists of those transitions that do not involve shared actions, and are transitions that exit normal states of P. The fourth set is for transitions originating from internal states of P; these are not interleaved with the transitions from Q. The second and fifth sets of the interleaved transitions are similarly defined as the first and fourth sets, but for IACA Q's transitions.

The definition of the fourth and fifth sets of interleaved transitions exclude the transitions where two internal states of two IACAs *collide*. Two internal states, belonging to two complex transitions of two IACAs, collide when each internal state wants to enforce its schedule, and as such none of the them can enforce its schedule. If two internal states have the same shared action enabled, they do not collide, and can synchronize to create a mutual hidden transition, as allowed by the third set of interleaved transitions in the definition. As an example of a collision of internal states, state $(3,2')$, in part (a) of Figure 8, does not have any transitions because both 3 and 2 are internal states, and synchronization is not possible for their enabled transitions.

Definition 18 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 when we remove all illegal states, in Section 3.1, and then prove a stronger claim for internal states.

Step 2: Remove illegal normal 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 normal states* from the interleaved product. The illegal normal states of two IACAs are the same as the illegal states for their equivalent IAs, as defined in Definition 5. Illegal normal states can include internal states of the interleaved product.

Part (a) of Figure 8 shows the illegal normal states of the two IACAs as black-filled boxes in the interleaved product. Part (b) of Figure 8 shows the result of removing illegal normal states for the interleaved product of A and B. Similar to IA, we remove all illegal normal 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 non-reachable 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 normal states in IACA.

Step 3: Remove illegal internal states Since in step 2, we treat normal and internal states similarly, and also since some of the internal states of two composable IACAs may collide, some of the 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. Those 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 (b) of Figure 8 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 9 needs to backtrack until no illegal internal states exist in the interleaved product of two IACAs. However, unlike removing illegal normal states, we do not have to remove autonomous paths which 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 (c) of Figure 8 shows the result of removing illegal internal states of the interleaved product of Part (b). In the following, we formally define the set of illegal internal states, and then present the algorithm for removing those states.

Definition 19. *Given two composable IACAs P and Q and their sets of interleaved transitions, τ_{P*Q}, and interleaved states, V_{P*Q}, the set of* illegal internal states *of P and Q, $IllegalInternal(P,Q)$, is defined as follows:*

$$IllegalInternal(P,Q) = \{(p,q) \in V^{Int}_{P*Q} \mid \nexists((p,q),a,(p',q')) \in \tau_{P*Q}\}$$

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 9, 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 18, with the necessary adjustments for those states and transitions that do not belong to $V_{P\circledast Q}$ and $\tau_{P\circledast Q}$ anymore. Again, we still cannot compute $\phi_{P\circledast Q}$ at this stage.

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.

```
Algorithm RemoveIllegalInternal($P$, $Q$, $\tau_{P*Q}$, $V_{P*Q}$) ;
Variables  : $\tau_{P\circledast Q}$, $V_{P\circledast Q}$, $L[\,]$, $i$, $temp$ ;
begin
    $i = 0$ ;
    $L_i = IllegalInternal(P, Q)$ ;
    repeat
        /* Backtrack to identify more illegal internal states */
        $temp = \{(p, q) \in V^{Int}_{P*Q} \mid \exists((p, q), a, (p', q')) \in \tau_{P*Q} \cdot (p', q') \in L_i\}$ ;
        /* Add to the set of illegal internal states */
        $L_{i+1} = L_i \cup temp$ ;
    until $L_i == L_{i+1}$;
    $V_{P\circledast Q} = V_{P*Q} \setminus L_i$;
    $\tau_{P\circledast Q} = \tau_{P*Q} \setminus \{((p, q), a, (p', q'))| ((p, q) \in L_i) \vee ((p', q') \in L_i)\}$ ;
    return $\tau_{P\circledast Q}$, $V_{P\circledast Q}$;
end
```

Fig. 9: Algorithm for removing illegal internal states and creating the legal interleaved product of two composable IACAs P and Q.

Lemma 1. *For each state $(p, q) \in V_{P\circledast Q}$, if (p, q) is an internal state, i.e., $(p, q) \in V^{Int}_{P\circledast Q}$, then there is exactly one transition in $\tau_{P\circledast Q}$ with source (p, q), and exactly one transition in $\tau_{P\circledast Q}$ with destination (p, q).*

Proof. The way $V_{P\circledast Q}$ and $\tau_{P\circledast Q}$ are computed, in the algorithm in Figure 9, guarantees that the only internal states in 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), and *maximum* one transition with destination (p, q). We first show that for any $(p, q) \in V^{Int}_{P*Q}$, there exists maximum one transition in τ_{P*Q} with source (p, q), and then show that there also exists maximum one transition in τ_{P*Q} with destination (p, q).

We show by contradiction that there does not exist more than one transition with source state (p, q). We first remark that the five sets of transitions for defining τ_{P*Q}, in Definition 18, are disjoint sets, because of the constraints on the source states, and actions being shared or non-shared. Also, the transitions with source that are internal states can be only created by sets (3), (4), and (5), in Definition 18. Furthermore, for $(p, q) \in V^{Int}_{P*Q}$, we claim that only one of the sets (3), (4), or (5) can define a transition with source (p, q). This claim is true because, first, sets (4) and (5) represent disjoint source states for their transitions, and second, if set (3), along with either set (4) or (5), can create an extra transition, it would mean that an internal state, either p or q, or both, have two transitions enabled on them, one being a shared and one a non-shared action of P and Q, which cannot be true.[3] As such, for $(p, q) \in V^{Int}_{P*Q}$, if there are more than one transition with source (p, q), then all of them should be created by the same set, *i.e.,* either (3), (4), or (5). However, for that to happen, either P or Q, or both, should have internal states that are the source of more than one transition, which cannot be true.

Having shown that (p, q) can be the source of exactly one transition, next, we show that there could not exist more than one transition with destination (p, q). For state $(p, q) \in V_{P*Q}$, either p, q, or both are internal states. In the following, we consider the three possibilities separately.

[3] For the general case when $(p, q) \notin V^{Int}_{P*Q}$, mutual exclusiveness among sets (3), (4), and (5) does not hold.

1. $p \in V_P^{Int}$ and $q \in V_Q^N$: According to Definition 18, only sets (1), (3), and (4) can contribute to creating transitions with destinations in (p, q). We argue that only one of the three sets can create transitions with destination (p, q).
   Let us consider a transition $((p', q'), a, (p, q)) \in \tau_{P*Q}$. By the definition of sets (1), (3), and (4) in Definition 18, $((p', q'), a, (p, q))$ exists only if $(p', a, p) \in \tau_P$. Furthermore, (p', a, p) is a part of the complex fragment of a complex transition in P, because $p \in V_P^{Int}$. As such, there exists only one transition in P with destination p. Furthermore, considering the definition of sets (1) and (4), for transition $((p', q'), a, (p, q)) \in \tau_{P*Q}$, we have $q = q'$, and since there exists exactly one $(p', a, p) \in \tau_P$, sets (1) and (4) can create exactly one, and the same transition, *i.e.,* $((p', q'), a, (p, q)) \in \tau_{P*Q}$.
   As for set (3), we first remark that similar to the justification mentioned above, there could not exist more than one transition created by set (3) with destination (p, q) in τ_{P*Q}. Furthermore, set (3) can only contribute to τ_{P*Q}, when neither set (1) nor (4) can contribute, because there is only one $(p', a, p) \in \tau_P$, and action a can be either a shared or non-shared action.[4]
   As such, we showed that either sets (1) and (4) can contribute a transition to τ_{P*Q}, or set (3) can, and none of the sets can create more than one transition with destination (p, q).
2. $p \in V_P^N$ and $q \in V_Q^{Int}$: This case is symmetric with the previous case.
3. $p \in V_P^{Int}$ and $q \in V_Q^{Int}$: Such a (p, q) is the destination of the transitions that can only be created by set (3). Therefore, any such transition $((p', q'), a, (p, q)) \in \tau_{P*Q}$, is a hidden transition. Considering internal state (p, q), if there exists more than one transition in τ_{P*Q} with destination (p, q), then we need to have more than one transition in P and Q, with destination $p \in V_P^{Int}$ and $q \in V_Q^{Int}$, respectively, which is not possible. Thus, there is exactly one transition in τ_{P*Q} with destination (p, q).
   For all possible cases, we showed that there could not exist more than one transition with destination (p, q), which concludes this part of the proof.

We showed that for any internal state $(p, q) \in V_{P \circledast Q}^{Int}$ there is exactly one transition in $\tau_{P \circledast Q}$ with source (p, q), and exactly one transition in $\tau_{P \circledast Q}$ with destination (p, q), which concludes our proof.

Next, in Lemma 2, we show that each internal state $(p, q) \in V_{P \circledast Q}^{Int}$ belongs to a complex fragment. We then, in Section 3.1, show that such a complex fragment indeed represents a complex fragment of P or Q.

Lemma 2. *For each internal state $(p, q) \in V_{P \circledast Q}^{Int}$, there exists exactly one sequence $\langle (p_0, q_0), a_0, (p_1, q_1), \ldots, (p_n, q_n) \rangle$, represented as $\Delta(p, q)$, such that 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.)
- $\forall i \cdot (0 < i < n) \Rightarrow ((p_i, q_i), a_i, (p_{i+1}, q_{i+1})) \in \tau_{P \circledast Q}$
  (All transitions of the complex fragment are in the legal interleaved product.)
- $(p_0, p_n \in V_P^N) \wedge (q_0, q_n \in V_Q^N) \wedge (\forall i \cdot (0 < i < n) \Rightarrow (p_i, q_i) \in V_{P \circledast Q}^{Int})$
  (The source and destination states of the sequence are normal states and the other states are internal states.)

[4] Again, this mutual exclusiveness holds only in this particular case.

Proof. From Lemma 1, it follows that for any internal state $(p, q) \in V^{Int}_{P \circledast Q}$, there is exactly one transition with source (p, q), and one transition with destination (p, q). As such, any internal state $(p, q) \in V^{Int}_{P \circledast Q}$ belongs to a unique alternating sequence of states and actions. We should now show that such sequences follow the criteria mentioned in the lemma. Let us now consider the alternating sequence of states and actions that contains 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, *i.e.,* in a state (u, v) where $u \in V^N_P$ and $v \in V^N_Q$. Such a (u, v) in fact represents the state (p_n, q_n) in $\Delta(p, q)$.

Next, we show that 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 $(p, q) \in V_{P \circledast Q}$, 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 $(p, q) \in V^{Int}_{P \circledast Q}$, we can conclude that for any state in $V^{Int}_{P \circledast Q}$ there exists a complex 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.

Step 4: Deriving the complex transitions The fourth and final step in computing the composition of two composable IACAs, is to determine the complex action associated with each complex fragment. Given two composable IACAs, P and Q, we introduced Δ as a function that returns a complex fragment associated with an internal state of the interleaved product. Each complex fragment $s = \langle (p_0, q_0), a_0, (p_1, q_1), \ldots, (p_n, q_n) \rangle$ returned by the function Δ can itself be projected into two alternating sequence of states and actions, one belonging to P the other to Q. We define $\pi_P(s) = \langle p_0, a_0, p_1, \ldots\ p_n \rangle$ and π_Q similarly. For these complex fragments, we prove the following:

Lemma 3. *For a reachable internal state $(p, q) \in V^{Int}_{P \circledast Q}$, its complex fragment $s = \Delta(p, q)$, and the projections of s, $\pi_P(s)$ and $\pi_Q(s)$, one of the following is true:*

- $\exists!(p, d, p') \in \phi_P \cdot \mathit{frag}(p, d, p') = \pi_P(s)$
- $\exists!(q, e, q') \in \phi_Q \cdot \mathit{frag}(q, e, q') = \pi_Q(s)$

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

Proof. Consider $s = \Delta(p, q)$ to be an alternating sequence of states and actions $\langle (p_0, q_0), a_0, (p_1, q_1)$
$, \ldots,$
$(p_n, q_n) \rangle$. To prove our claim, we identify the complex transition, which either belongs to P or Q, and s follows its schedule.

According to the definition of Δ, for all (p_i, q_i), where $(0 < i < n)$, either $p_i \in V^{Int}_P$ or $q_i \in V^{Int}_Q$. Consider the first internal state (p_j, q_j), such that either $p_j \in V^{Int}_P$ or $q_j \in V^{Int}_Q$, but not both, if such a state exists at all. According to Definition 18, (p_j, q_j) can be the source of transitions that are created via sets (3), (4), or (5). Depending on whether p_j or q_j is an internal state, according to the definition of sets (3), (4), and (5), the sequence s at (p_j, q_j) will either follow the only transition with source p_j in τ_P, or will following the only transition with source q_j in τ_Q, respectively. Furthermore, the next state in s, *i.e.,* (p_{j+1}, q_{j+1}), where $(j + 1) < n$, depending on whether p_j or q_j is an internal state, should certainly have either p_{j+1} or q_{j+1} as an internal state of P or Q, respectively.[5] It can be observed that for all (p_k, q_k), where $j < k < n$, if p_j (or alternatively q_j) is an internal state then all p_ks (or respectively all q_ks) are internal states and follow the complex fragment that p_j (or q_j) follows in P (or Q). We notice that a state (p_k, q_k), where $j < k < n$, could have both p_k and q_k as internal states. Such states have hidden transitions enabled, and do not violate our above observation that all (p_k, q_k)s follow the complex fragment that p_j or q_j belong to, depending on whether $p_j \in V_P^{Int}$ or $q_j \in V_Q^{Int}$.

Next, we show that all states (p_l, q_l), where $(0 \leq l < j)$, follow the same complex fragment that (p_k, q_k)s, where $j < k < n$, follow. Consider (p_{j-1}, q_{j-1}) where $(j-1) \geq 0$. Let us assume that p_j is an internal state, then according to τ_{P*Q} definition, $(p_{j-1}, a_{j-1}, p_j) \in \tau_P$, and $((p_{j-1}, q_{j-1}), a_{j-1}, (p_j, q_j)) \in \tau_{P*Q}$. Similarly, if we assume that q_j is an internal state, then $(q_{j-1}, a_{j-1}, q_j) \in \tau_Q$ and $((p_{j-1}, q_{j-1}), a_{j-1}, (p_j, q_j)) \in \tau_{P*Q}$. In other words, the action of transition $((p_{j-1}, q_{j-1}), a_{j-1}, (p_j, q_j))$, depending on whether p_j or q_j is an internal state, can be identified by the action of the transition with source p_{j-1} or q_{j-1} in P or Q, respectively. Similarly, for transition $((p_{j-2}, q_{j-2}), a_{j-2}, (p_{j-1}, q_{j-1}))$, where $(j-2) \geq 0$, its action, depending on whether p_{j-1} or q_{j-1} is an internal state, can be identified by the action of the transition with source p_{j-2} or q_{j-2} in P or Q, respectively. But the action of the transition with source state (p_{j-1}, q_{j-1}) was identified by whether $p_j \in V_P^{Int}$ or $q_j \in V_Q^{Int}$, and so is (p_{j-2}, q_{j-2}). Similarly, we can show that for all other internal states (p_l, q_l), the transition with source (p_l, q_l), depending on whether p_j or q_j is an internal state, has the same action that p_l or q_l has in P or Q, respectively.[6]

Let us now consider state (p_1, q_1).[7] Based on p_j or q_j being an internal state, there exists (p_0, a_0, p_1), or respectively (q_0, a_0, q_1) in τ_P or τ_Q. Furthermore, if $p_j \in V_P^{Int}$, then (p_0, a_0, p_1) is the first transition in the complex fragment that p_j belongs to, and similarly for q_j.

As such we can conclude that all states in s follow the states in complex fragments of p_j or q_j, depending whether $p_j \in V_P^{Int}$ or $q_j \in V_Q^{Int}$. Furthermore, there could not exist any transitions in the complex fragment that contains p_j or q_j, that do not have a counterpart in s, because if there were some missing transitions, some states in s would have become illegal internal states and would have already been removed in step 3 of the composition. Also, since an internal state in P or Q can be uniquely identified by a complex transition, then the sequence s can also be uniquely identified by the same complex transition.

Lastly, we notice that, there may not exist such a (p_j, q_j), as we described above, meaning that all p_ms and q_ms, where $0 < m < n$, are internal states. According to τ_{P*Q} definition, in Definition 18, only set (3) can have transitions with such (p_m, q_m)s, which implies that two complex transitions from P and Q completely overlap each other and this situation is an acceptable case according to the lemma.

Using Lemma 3, we can define a mapping function that maps an internal state $(p, q) \in V_{P \circledast Q}^{Int}$ into a complex transition.

[5] Note that if $p_j \in V_P^{Int}$, or if $q_j \in V_Q^{Int}$, it is not possible that $p_{j+1} \notin V_P^{Int}$ and $q_{j+1} \in V_Q^{Int}$, or $q_{j+1} \notin V_Q^{Int}$ and $p_{j+1} \in V_Q^{Int}$, respectively. The composablity criteria for IACAs, which states that the schedule of complex actions should not partially overlap, disallows such a situation.

[6] It is well to note that all transitions appearing before state (p_j, q_j) are created by set (3) in Definition 18. As such, all transitions appearing before (p_j, q_j) in s are hidden transitions.

[7] It is possible that (p_1, q_1) is the same as (p_j, q_j), *i.e.,* there are no (p_l, q_l) states.

Definition 20. *Given an internal state $(p, q) \in V^{Int}_{P \circledast Q}$, the injective function $complex_{P \circledast Q}(p, q)$ maps (p, q) into exactly one complex transition $((p_0, q_0), c, (p_n, q_n))$ where the action c belongs to P, Q or both.*

The complex action, c, is uniquely identified by Lemma 3 except in the case where both conditions of the lemma are true, *i.e.,* when two complex fragments overlap entirely but the name of their corresponding complex actions are different. In that situation, we pick the name of the complex action that has normal inputs. Considering our example in Figure 8, part (d) of Figure 8 shows the composition of A and B. The complex transition of $A \parallel B$ is on complex action L.

We are now ready to define the composition operation formally.

Definition 21. *The* composition *of two composable IACAs P and Q, $P \parallel Q$, is an IACA defined as follows:*
$V_{P\parallel Q} = V_{P\circledast Q}$
$V^{Int}_{P\parallel Q} = V^{Int}_{P\circledast Q}$
$i_{P\parallel Q} = i_{P\circledast Q}$
$\mathcal{A}^I_{P\parallel Q} = \mathcal{A}^I_{P\circledast Q}$
$\mathcal{A}^O_{P\parallel Q} = \mathcal{A}^O_{P\circledast Q}$
$\mathcal{A}^H_{P\parallel Q} = \mathcal{A}^H_{P\circledast Q}$
$\mathcal{A}^C_{P\parallel Q} = \mathcal{A}^C_{P\circledast Q}$
$\tau_{P\parallel Q} = \tau_{P\circledast Q}$
$\phi_{P\parallel Q} = \{complex_{P\circledast Q}(p, q) \mid (p, q) \in V^{Int}_{P\parallel Q}\}$

In practice, we can use only one of the internal states (the first one) of a complex fragment to compute the complex action rather than all of them.

Finally, we prove that IACA composition is commutative.

Theorem 1. *Given two composable IACAs P and Q, $P \parallel Q = Q \parallel P$.*

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

Discussion IACA is not a full-fledged interface model because composition is not associative. Lack of associativity in IACA is unavoidable, because within a complex transition, there could exist multiple normal actions that can be synchronized through composition(s) with other IACAs. Such normal actions can only synchronize if their preceding normal actions in the schedule of 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. 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 IACA, depends on the order of composition between multiple IACAs (Web services). In other words, the success of synchronization for a complex transition can 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. Instead, we have to consider multiple groupings of components. We plan to investigate ways to determine groupings for composition that would yield a maximal result, *i.e.,* choosing composition parenthesizations that would increase the chance of synchronization among different IACAs. In the absence of shared actions among multiple IACAs, their composition is associative.

3.2 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 10 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).

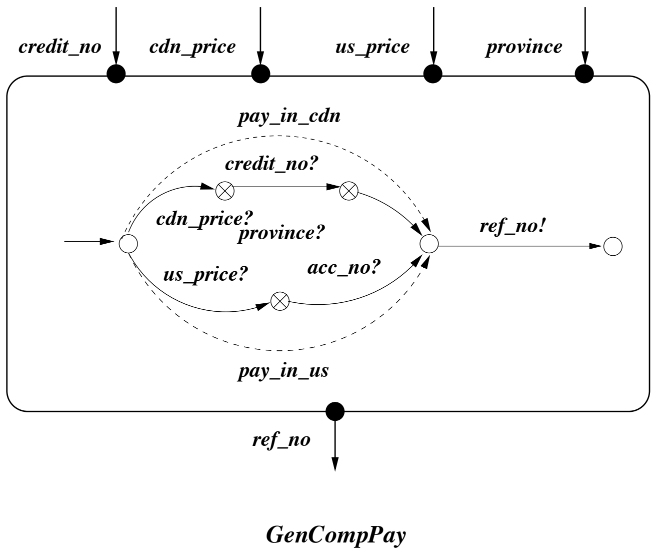

Fig. 10: *GenCompPay* is the refinement of the IACA in Figure 6.

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 optional parameters and subclasses. In programming languages, such as C/C++, 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, we first need to partition the complex actions into three sets based on whether the associated complex fragment has: (1) input and hidden actions (C^I_P), (2) output and hidden actions (C^O_P), and (3) only hidden actions (C^H_P). The definition of refinement is as follows:

Definition 22. *IACA Q* refines *IACA P, $Q \preceq P$, if:*

1. $\mathcal{A}^I_P \subseteq \mathcal{A}^I_Q$ *(Q has the same or more normal inputs than P.)*
2. $\mathcal{A}^O_P \supseteq \mathcal{A}^O_Q$ *(Q has the same or fewer normal outputs than P.)*
3. $C^I_P \subseteq C^I_Q$ *(Q has the same or more complex inputs than P.)*
4. $C^O_P \supseteq C^O_Q$ *(Q has the same or fewer complex outputs than P.)*
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. Next, we define this relation.

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 23. *For each normal state $p \in V^N_P$, the set $closure_P(p)$ is defined as the set of* reachable states *of p which 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 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 state without knowing which of them exactly receives it and so 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). The functions $\mathcal{A}^I_P(p), \mathcal{A}^O_P(p), \mathcal{A}^C_P(p)$ return the set of input, output, and complex actions, respectively, on transitions exiting normal state $p \in V^N_P$.

Definition 24. *For each normal state $p \in V^N_P$ of IACA P,*

- *The sets of* enabled normal input *and* enabled normal output *actions are:*
  $EnNorm^I_P(p) = \{a \mid \forall p' \in closure_P(p) \cdot a \in \mathcal{A}^I_P(p')\}$
  $EnNorm^O_P(p) = \{a \mid \exists p' \in closure_P(p) \cdot a \in \mathcal{A}^O_P(p')\}$

- *The sets of* enabled complex input *and* enabled complex output *actions are:*
  $EnComp^I_P(p) = \{a \in C^I_P |\ \forall p' \in closure_P(p) \cdot a \in \mathcal{A}^C_P(p')\}$
  $EnComp^O_P(p) = \{a \in C^O_P |\ \exists p' \in closure_P(p) \cdot a \in \mathcal{A}^C_P(p')\}$
- $En^O_P(p) = EnNorm^O_P(p) \cup EnComp^O_P(p)$
- $En^I_P(p) = EnNorm^I_P(p) \cup EnComp^I_P(p)$

Having defined the sets of actions that can be expected from a state p and 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 through transitions via a certain action.

Definition 25. *For IACA P, a normal state $p \in V^N_P$, and action $a \in En^I_P(p) \cup En^O_P(p)$, the set of* reachable states of p by a *is:*

$$Dest_P(p,a) = \{p' \in V^N_P |\ \exists\ r \in closure_P(p) \cdot (\exists\ (r,a,p') \in \tau_P) \vee (\exists\ (r,a,p') \in \phi_P)\}$$

Finally, we can define the refinement relation between two states. This relation intuitively says that for every state $p \in V^N_P$, there is an alternating simulation through state $q \in V^N_Q$. 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 26. *For two IACAs, P and Q, the binary relation* alternating simulation $\preceq\ \subseteq V^N_Q \times V^N_P$ *between two states $q \in V^N_Q$ and $p \in V^N_P$ holds if all of the following conditions are true:*

- $EnNorm^I_P(p) \subseteq EnNorm^I_Q(q)$
  (q may have the same or more normal inputs.)
- $EnNorm^O_P(p) \supseteq EnNorm^O_Q(q)$
  (q may have the same or fewer normal outputs.)
- $EnComp^I_P(p) \subseteq EnComp^I_Q(q)$
  (q may have the same or more complex inputs.)
- $EnComp^O_P(p) \supseteq EnComp^O_Q(q)$
  (q may have the same or fewer complex outputs.)
- $\forall a \in EnComp^I_P(p) \cdot \forall (m,a,n) \in \phi_P \cdot m \in closure_P(p) \Rightarrow$
  $\exists (r,a,s) \in \phi_Q \cdot r \in closure_Q(q) \wedge sched(m,a,n) \sqsubseteq sched(r,a,s)$
  (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.)
- $\forall a \in EnComp^O_Q(q) \cdot \forall\ (r,a,s) \in \phi_Q \cdot r \in closure_P(q) \Rightarrow$
  $\exists (m,a,n) \in \phi_P \cdot m \in closure_P(p) \wedge sched(r,a,s) \sqsubseteq sched(m,a,n)$
  (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.)
- $\forall a \in En^I_P(p) \cup En^O_Q(q) \cdot \forall\ q' \in Dest_Q(q,a) \Rightarrow \exists p' \in Dest_P(p,a) \cdot q' \preceq p'$
  ($\preceq$ holds for all reachable normal states under inputs for p and outputs for q.)

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

The purpose of refinement is to support top-down design, the following theorem states this property.

Theorem 2. *Given three IACAs, P, Q and P', such that $P' \preceq P$, P and Q are composable, and P' and Q are composable, then $(P' \parallel Q) \preceq (P \parallel Q)$, 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)
2. $\forall (p', p) \cdot (p' \preceq p) \Rightarrow$
   $((\mathcal{A}^I_{P'}(p') \backslash \mathcal{A}^I_P(p)) \notin Shared(P, Q)) \wedge$
   $((\mathcal{A}^O_P(p) \backslash \mathcal{A}^O_{P'}(p')) \notin Shared(P, Q))$
   (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.)
3. $\forall (p', p) \cdot (p' \preceq p) \Rightarrow$
   $(\forall (p, c, u) \in \phi_P \cdot \exists (p', c, v) \in \phi_{P'} \wedge (c \in C^O_{P'})$
   $\Rightarrow ((set(sched(p, c, u))) \backslash (set(sched(p', c, v))) \cap (Shared(P, Q))) = \emptyset)$
   (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.)
4. $\forall (p', p) \cdot (p' \preceq p) \Rightarrow$
   $(\forall (p', c, v) \in \phi_{P'} \cdot \exists (p, c, u) \in \phi_P \wedge (c \in C^I_P)$
   $\Rightarrow ((set(sched(p', c, v))) \backslash (set(sched(p, c, u))) \cap (Shared(P, Q))) = \emptyset)$
   (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 22, hold. To show that condition (1) holds consider $\mathcal{A}^I_{P\parallel Q}$, by the definition of composition, $\mathcal{A}^I_{P\parallel Q} = (\mathcal{A}^I_P \cup \mathcal{A}^I_Q) \backslash Shared(P, Q)$. Since $Shared(P', Q) = Shared(P, Q)$ and $\mathcal{A}^I_{P'} \supseteq \mathcal{A}^I_P$, it follows that $\mathcal{A}^I_{P'\parallel Q} \supseteq \mathcal{A}^I_{P\parallel Q}$. Similarly it can be shown that condition (2) holds, *i.e.,* $\mathcal{A}^O_{P'\parallel Q} \subseteq \mathcal{A}^O_{P\parallel Q}$. To show that condition (3) holds, we should show that $C^I_{P'\parallel Q} \supseteq C^I_{P\parallel Q}$, which is true since because $P' \preceq P$ we know that $C_{P'} \supseteq C_P$, and also $C_{P'\parallel Q} = C_{P'} \cup C_Q$. Similarly it can be shown that condition (4) of 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 $(p, q) \in V^N_{P\parallel Q}$ and all states p' such that $p' \preceq p$. All such $(p', q) \in V^N_{P\parallel Q}$ are reachable states because for all such states, based on the conditions of the theorem, if (p', q) is an illegal normal state, then (p, q) should be an illegal normal 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 normal state while (p, q) is not. Furthermore, conditions 3 and 4 of the theorem guarantee that $P' \parallel Q$ does not have more illegal internal states than $P \parallel Q$ does, on the common complex actions between P' and P. 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 disallows this situation. As such, for any (p, q), we have (p', q) which can simulate (p, q), and this concludes our proof.

Discussion The conditions of Theorem 2 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 2. Our restriction arises from the fact that we not only deal with illegal normal 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.

Another issue is that in our theorem we require that P' and Q are composable, but there is no guarantee that since P and Q are composable, then P' and Q should be composable as well. This requirement in our theorem is necessary because P', as refinement of P, can have extra complex actions at its states, and there is no way to guarantee that such extra complex actions observe the overlapping composability criteria of IACAs. We could have instead defined our refinement relation in such a way that it would have disallowed the extra complex actions, and hence avoided requiring explicitly that P' and Q be composable. However, since it is very well likely for P' and Q to be composable, we prefer our definition of refinement, which allows extra complex actions.

Figure 11 illustrates the composition ($GenCompPay \parallel Prod$). Considering the composition of *GenCompPay*, in Figure 10, with IACA of *Prod* in Figure 1, since $GenCompPay \preceq CompPay$ then $(GenCompPay \parallel Prod) \preceq (CompPay \parallel Prod)$.

4 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.,* [10, 3]), and (2) *action refinement* (*e.g.,* [1]).

In the work most comparable to ours, Gorrieri, Marchetti, and Montanari enhance CCS [16] to support non-interruptible actions [10]. 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 recent comprehensive treatment of action refinement, in a not entirely algebraic setting, readers can refer to [17].

Promela, the language of the Spin model checker [12], 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 actions are similar to `d_step`. Composition in Promela is an n-ary operator and there is no defined notion of refinement. As such, associativity in its composition is irrelevant.

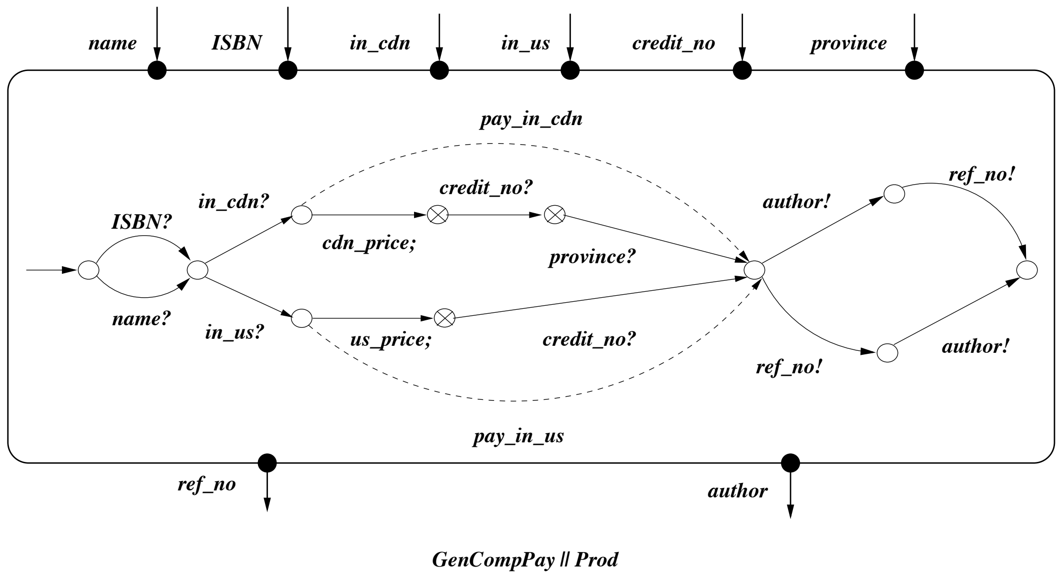

Fig. 11: Composition of IACA *GenCompPay*, in Figure 10, with IACA equivalent IACA of IA *Prod* in Figure 1.

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.

5 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)* [4, 5], 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 exchange 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 Web service input and output XML messages, and second, IACA, as an automaton model, can effectively capture the temporal order of messages in Web service.

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. 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. Also, for Web services, we may need to combine services in response to a search query. Through heuristics, we may be able to reduce the need to search all possible associativity orderings.

References

1. Luca Aceto. *Action Refinement in Process Algebras*. Cambridge University Press, 1992.
2. Rajeev Alur, Thomas A. Henzinger, Orna Kupferman, and Moshe Y. Vardi. Alternating refinement relations. In *Proceeding of the 9th Conferance on Concurrency Theory*, volume 1466 of *Lecture Notes in Computer Science*, pages 163–178. Springer-Verlag, 1998.
3. Gérard Boudol. Atomic actions (note). *Bulletin of the European Association for Theoretical Computer Science*, 38:136–144, 1989. Technical Contributions.
4. Roberto Chinnici, Martin Gudgin, Jean-Jacques Moreau, Jeffrey Schlimmer, and Sanjiva Weerawarana. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, 2006.
5. Roberto Chinnici, Hugo Haas, Jean-Jacques Moreau, David Orchard, and Sanjiva Weerawarana. Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts, 2006.
6. Luca de Alfaro and Thomas A. Henzinger. Interface Automata. In Volker Gruhn, editor, *Proceedings of the Joint 8th European Software Engeneering Conference and 9th ACM SIGSOFT Symposium on the Foundation of Software Engeneering (ESEC/FSE-01)*, volume 26, 5 of *Software Engineering Notes*, pages 109–120. ACM Press, 2001.
7. Luca de Alfaro and Thomas A. Henzinger. Interface Theories for Component-Based Design. In *Proceedings of the First International Workshop on Embedded Software*, volume 2211, pages 148–165. Lecture Notes in Computer Science 2211, Springer-Verlag, 2001.
8. Luca de Alfaro and Thomas A. Henzinger. Interface-Based Design. In *Proceedings of the Marktoberdorf Summer School, Kluwer*, Engineering Theories of Software Intensive Systems, 2004.
9. Shahram Esmaeilsabzali, Farhad Mavaddat, and Nancy A. Day. Interface automata with complex actions. In *Proceeding of IPM International Workshop on Foundations of Software Engineering (FSEN)*, volume 159, pages 79–97. Electronic Notes in Theoretical Computer Science, 2006.
10. Roberto Gorrieri, Sergio Marchetti, and Ugo Montanari. A^2CCS: atomic actions for CCS. *Theoretical Computer Science*, 72(2-3):203–223, 1990.
11. Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. An interleaving model for real time. In *JCIT: Proceedings of the fifth Jerusalem conference on Information technology*, pages 717–730, Los Alamitos, CA, USA, 1990. IEEE Computer Society Press.
12. Gerard J. Holzmann. The model checker Spin. *IEEE Trans. Soft. Eng.*, 23(5):279–295, 1997.
13. Nancy A. Lynch and Mark R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the 6th ACM Symposium on Principles of Distributed Computing*, pages 519–543, 1987.
14. Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York, 1991.
15. Robin Milner. Calculi of synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.
16. Robin Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989. SU Fisher Research 511/24.
17. Rob van Glabbeek and Ursula Goltz. Refinement of actions and equivalence notions for concurrent systems. *Acta Inf.*, 37(4-5):229–327, 2000.