An I/O Automata-based Approach to Verify Component Compatibility: Application to the CyCab Car
Samir Chouali, Hassan Mountassir, Sebti Mouelhi

To cite this version:

HAL Id: hal-00563290
https://hal.archives-ouvertes.fr/hal-00563290
Submitted on 4 Feb 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers. L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
An I/O automata based approach to verify component compatibility: application to the CyCab car

Samir Chouali, Hassan Mountassir, Sebti Mouelhi

Laboratoire d’Informatique de l’Université de Franche-Comté - LIFC
16, route de Gray - 25030 Besançon cedex, France

Abstract
An interesting formal approach to specify component interfaces is interface automata based approach, which is proposed by L. Alfaro and T. Henzinger. These formalisms have the ability to model both the input and output requirements of components system. In this paper, we propose a method to enrich interface automata by the semantics of actions in order to verify components interoperability at the levels of signatures, semantics, and protocol interactions of actions. These interfaces consist of a set of required and offered actions specified by Pre and Post conditions. The verification of the compatibility between interface automata reuse the L.Alfaro and T.Henzinger proposed algorithm and adapt it by taking into account the action semantics. Our approach is illustrated by a case study of the vehicle CyCab.

Keywords: component based systems, interface compatibility, I/O automata.

1 Introduction

Interface formalisms play a central role in the component-based design of many types of systems. They are increasingly used thanks to their ability to describe, in terms of communicating interfaces, how a component of a system can be composed and connected to the others. An interface should describe enough information about the manner of making two or more components working together properly. Several approaches and models based on components have been proposed notably those of Szyperski [10] and Medvidovic [7]. Most of these models specify the components behaviors, the connectors ensuring their communications and the services provided or requested. Assembling components is performed by passing through different levels of abstraction, from the conception of the software architectures ADL until the implementation using platforms like CORBA, Fractal or .NET. The crucial question that arises to the developers is to know if the proposed assembling is correct or not.
In this paper, our interests concern components which are described by interface automata. These interfaces specify action protocols: scheduling calls of component actions. As some related works, we can mention the model in [3] where the protocols are associated to the component connectors. Others works as the ones in [9], the authors proposed a comparison between models at three grades of interoperability using the operation signatures, the interfaces protocols and the quality of service. The protocols in [6] based on transitions systems and concurrency including the reachability analysis. The composition operation is essential to define assembly and check the surety and vivacity properties. The approach in [8] aims to endow the UML components to specify interaction protocols between components. The behavioral description language is based on hierarchical automata inspired from StateCharts. It supports composition and refinement mechanisms of system behaviors. The system properties are specified in temporal logic. In [4], the authors define a component-based model Kmelia with abstract services, which does not take into account the data during the interaction. The behavior described by automata associated to services. This environment uses the tool MEC model-checker to verify the compatibility of components. Other works consider real-time constraints [5]. The idea is to determine the component characteristics and define certain criteria to verify the compatibility of their specifications using the tool Kronos.

The works of L.Alfaro and T.Henzinger [1,2], allows to specify component interfaces by interface automata. These interfaces are specified by automata which are labelled by input, output, and internal actions. The composition of interfaces is achieved by synchronizing actions. Our approach reuse this model and strengthening it by taking into account the action semantics to ensure a more reliable verification of components interoperability. The paper is organized as follows: In section 2, we describe the interface automata as well as the definitions and the algorithm used to verify the compatibility between component interfaces. In section 3, we present our approach to verify the interface compatibility, and we apply the approach to the case study of the vehicle CyCab in section 4. We conclude our work and present perspectives in section 5.

2 Input/Output Automata

The I/O automata are defined by Nancy A.Lynch and Mark.Tuttle [12] as a labelled transition systems. They are used to model distributed and concurrent systems. The set of actions in an I/O automata are composed of input actions, output actions, and hidden actions. These actions are enabled in every state of the system.

Definition 2.1 An I/O automata is a tuple $A = (Q, \Sigma_{\text{inp}}, \Sigma_{\text{out}}, \Sigma_{\text{int}}, \delta, I)$, such that:

- $Q$ is a set of states.
- $\Sigma_{\text{inp}}, \Sigma_{\text{out}}, \Sigma_{\text{int}}$, represent respectively a set of input action, a set of output action, and a set of internal actions. They are mutually disjoint. We denote by $\Sigma = \Sigma_{\text{inp}} \cup \Sigma_{\text{out}} \cup \Sigma_{\text{int}}$ a set of actions.
- $\delta \subseteq Q \times \Sigma \times Q$ is a set of transitions.
- $I$ is the non empty set of initial states.
2.1 Interface automata

The interface automata are introduced by L.Alfaro and T.Henzinger [1,2], to model component interfaces. These automata are I/O automata where it is not necessary to enable input actions in every system’s state. Every component is described by one interface automaton. In an interface automaton, output actions define the called actions by a component in his environment. They describe the required actions of a component. They are labelled by the symbol "!". Input actions describe the offered actions of a component. They are labelled by the symbol "?". Internal (or hidden) actions are enabled actions inside a component by the component himself. They are labelled by the symbol ";".

2.1.1 Composition and Compatibility

In this section we present the approach of L.Alfaro a T.Henzinger [1,2] to verify the compatibility of components which are specified by interface automata. The following definition presents the composition of two interface automata.

**Definition 2.2** Let $A_1 = \langle Q_1, \Sigma^I_1, \Sigma^O_1, \Sigma^H_1, \delta_1, I_1 \rangle$, $A_2 = \langle Q_2, \Sigma^I_2, \Sigma^O_2, \Sigma^H_2, \delta_2, I_2 \rangle$, be two interface automata. The set $\Sigma^I \cup \Sigma^O$ is denoted by $\text{share}(A_1, A_2)$. The automata $A_1, A_2$ are composable if $\langle (\Sigma^I_1 \cap \Sigma^O_2 = \emptyset) \land (\Sigma^I_2 \cap \Sigma^O_1 = \emptyset) \land (\Sigma^H_1 \cap \Sigma^H_2 = \emptyset) \rangle$.

Note that if two interface automata $A_1$ and $A_2$ are composable then $\text{share}(A_1, A_2) = (\Sigma^I_1 \cap \Sigma^O_2) \cup (\Sigma^I_2 \cap \Sigma^O_1)$.

The following definition presents the synchronized product of two interface automata.

**Definition 2.3** Let $A_1 = \langle Q_1, \Sigma^I_1, \Sigma^O_1, \Sigma^H_1, \delta_1, I_1 \rangle$, $A_2 = \langle Q_2, \Sigma^I_2, \Sigma^O_2, \Sigma^H_2, \delta_2, I_2 \rangle$, be two composable interface automata. The product $A_1 \times A_2$ is defined by $\langle Q_1 \times Q_2, \Sigma^\text{imp}, \Sigma^\text{out}, \Sigma^\text{int}, \delta, I_1 \times I_2 \rangle$ such that:

- $\Sigma^\text{imp} = (\Sigma^I_1 \cap \Sigma^I_2) \setminus \text{partage}(A_1, A_2)$,
- $\Sigma^\text{out} = (\Sigma^O_1 \cap \Sigma^O_2) \setminus \text{partage}(A_1, A_2)$,
- $\Sigma^\text{int} = (\Sigma^H_1 \cap \Sigma^H_2) \setminus \text{partage}(A_1, A_2)$,
- $((q_1, q_2), a, (q'_1, q'_2)) \in \delta$ if:
  - $a \notin \text{share}(A_1, A_2) \land (q_1, a, q'_1) \in \delta_1 \land q_2 = q'_2$
  - $a \notin \text{share}(A_1, A_2) \land q_1 = q'_1 \land (q_2, a, q'_2) \in \delta_2$
  - $a \in \text{share}(A_1, A_2) \land (q_1, a, q'_1) \in \delta_1 \land (q_2, a, q'_2) \in \delta_2$

In the following, we present the set of illegal state (deadlock states) in the product of two interface automata.

**Definition 2.4** Let $A_1 = \langle Q_1, \Sigma^I_1, \Sigma^O_1, \Sigma^H_1, \delta_1, I_1 \rangle$, $A_2 = \langle Q_2, \Sigma^I_2, \Sigma^O_2, \Sigma^H_2, \delta_2, I_2 \rangle$, be two interface automata. The set of illegal states in the product $A_1 \times A_2$ is denoted by $\text{Illegal}(A_1, A_2) = \{(q_1, q_2) \in Q_1 \times Q_2 | \exists a \in \text{share}(A_1, A_2) \text{ such that,} \}$

$$a \in \sigma^I_1 \land a \in \sigma^I_2 \land \exists a \in \sigma^I_1 \land a \in \sigma^I_2 \land \exists a \in \sigma^O_1 \land a \in \sigma^O_2 \land (q_1, a, q'_1) \in \delta_1 \land (q_2, a, q'_2) \in \delta_2 \land (a \in \sigma^O_1 \land a \in \sigma^O_2 \land (q_1, a, q'_1) \in \delta_1 \land (q_2, a, q'_2) \in \delta_2)$$

3
The set of illegal states describes the states in which the shared actions between the interface automata do not synchronize. So, on these states a component require (or offer) an action which is not offered (or required) by the environment.

In this approach, the verification of the compatibility between a component $C_1$ and a component $C_2$ is obtained by verifying the compatibility between their interface automata $A_1$ and $A_2$. Therefore, one verify if there is a helpful environment where it is possible to assemble correctly the components $C_1$ and $C_2$. So, one suppose the existence of such environment which accepts all the output actions of the automaton of the product $A_1 \times A_2$, and which do no not call any input actions in $A_1 \times A_2$.

The verification steps of the compatibility between $A_1$ and $A_2$ are listed below.

(i) verify that $A_1$ and $A_2$ are composable,
(ii) calculate the product $A_1 \times A_2$,
(iii) calculate the set of illegal in $A_1 \times A_2$,
(iv) calculate the bad states in $A_1 \times A_2$: the states from which the illegal state are reachable by enabling only the internal action or the output actions (one suppose the existence of a helpful environment),
(v) eliminate from the automaton $A_1 \times A_2$, the illegal state, the bad state, and the unreachable states from the initial states,
(vi) after performing the above step, if the automaton $A_1 \times A_2$ is empty then the interface automata $A_1$, $A_2$ are not compatible, therefore $C_1$ and $C_2$ can not assembled correctly in any environment.

The complexity of this approach is linear on the size of the both interface automata.

3 Considering action semantics in the verification of interface automata compatibility

We present an approach to verify the compatibility between component interfaces based on the I/O automata and the approach of L.Alfaro and T.Henzinger [1].

The contribution of our approach compared to the one presented in [1], is the consideration of the action semantics in the component interfaces and in the verification of the component compatibility. In [1], one verify component compatibility by considering only action signatures. We consider, that action signatures are not sufficient to decide on the component compatibility using an approach based on I/O automata.

We propose to annotate transition in interface automata by pre and post condition of actions. We adapt the compatibility verification algorithm presented in [1], to take into account pre and post of actions.

In the following definitions we formalise the adaptations on the L.Alfaro and T.Henzinger approach in order to introduce action semantics in the interface automata.

We introduce a finite set of variables $x \in V$ with their respective domain $D_x$.

The definition 3.1 presents interface automata with pre and post conditions of actions.
Definition 3.1 Let $A = \langle Q, \Sigma_{imp}, \Sigma_{out}, \Sigma_{int}, Pre, Post, \delta, I \rangle$, be an I/O automaton such that:

- $Q$ is a set of states.
- $\Sigma_{imp}, \Sigma_{out}, \Sigma_{int}$ represent respectively a set of input action, a set of output action, and a set of internal actions. They are mutually disjoint. We denote by $\Sigma = \Sigma_{imp} \cup \Sigma_{out} \cup \Sigma_{int}$ a set of actions.
- $Pre$ et $Post$ are respectively the set of preconditions and the set postconditions of a component actions. The preconditions and the postconditions are predicate over the set of variables $V$ and their respective domain $D_x$.
- $\delta \subseteq Q \times Pre \times \Sigma \times Post \times Q$ is the set of transitions.
- $I$ is the non empty set of initial states.

The following definition presents the condition to compose two interface automata.

Definition 3.2 Let $A_1 = \langle Q_1, \Sigma^I_1, \Sigma^O_1, \Sigma^H_1, Pre_1, Post_1, \delta_1, I_1 \rangle$, $A_2 = \langle Q_2, \Sigma^I_2, \Sigma^O_2, \Sigma^H_2, Pre_2, Post_2, \delta_2, I_2 \rangle$, be two interface automata. The set $\Sigma_1 \cup \Sigma_2$ is denote by $\text{share}(A_1, A_2)$. The automata $A_1, A_2$ are composable if: $(\Sigma^I_1 \cap \Sigma^I_2) = \emptyset \land (\Sigma^O_1 \cap \Sigma^O_2) = \emptyset \land (\Sigma^H_1 \cap \Sigma^H_2) = \emptyset$.

Note that if two interface automata $A_1$ and $A_2$ are composable then $\text{share}(A_1, A_2) = (\Sigma^I_1 \cap \Sigma^O_2) \cup (\Sigma^O_1 \cap \Sigma^I_2)$.

In the following definition we define the synchronized product of two interface automata.

Definition 3.3 Let $A_1 = \langle Q_1, \Sigma^I_1, \Sigma^O_1, \Sigma^H_1, Pre_1, Post_1, \delta_1, I_1 \rangle$, $A_2 = \langle Q_2, \Sigma^I_2, \Sigma^O_2, \Sigma^H_2, Pre_2, Post_2, \delta_2, I_2 \rangle$, be two composable interface automata. The product $A_1 \times A_2$ is defined by $\langle Q_1 \times Q_2, \Sigma_{imp}, \Sigma_{out}, \Sigma_{int}, Pre, Post, \delta, I_1 \times I_2 \rangle$ such that:

- $\Sigma_{imp} = (\Sigma^I_1 \cap \Sigma^I_2) \setminus \text{partage}(A_1, A_2)$,
- $\Sigma_{out} = (\Sigma^O_1 \cap \Sigma^O_2) \setminus \text{partage}(A_1, A_2)$,
- $\Sigma_{int} = (\Sigma^H_1 \cap \Sigma^H_2) \setminus \text{partage}(A_1, A_2)$,
- $((q_1, q_2), Pre, a, Post, (q'_1, q'_2)) \in \delta$ if:
  - $a \notin \text{share}(A_1, A_2) \land (q_1, Pre_1, a, Post_1, q'_1) \in \delta_1 \land q_1 = q'_1 \land Pre \equiv Pre_1 \land Post \equiv Post_1$
  - $a \notin \text{share}(A_1, A_2) \land (q_2, Pre_2, a, Post_2, q'_2) \in \delta_2 \land q_2 = q'_2 \land Pre \equiv Pre_2 \land Post \equiv Post_2$
  - $a \in \text{share}(A_1, A_2) \land ((q_1, Pre_1, a, Post_1, q'_1) \in \delta_1 \land a \in \Sigma^I_1) \land ((q_2, Pre_2, a, Post_2, q'_2) \in \delta_2 \land a \in \Sigma^O_2) \land Pre \equiv (Pre_2 \implies Pre_1) \land Post \equiv (Post_1 \implies Post_2)$
  - $a \in \text{share}(A_1, A_2) \land ((q_1, Pre_1, a, Post_1, q'_1) \in \delta_1 \land a \in \Sigma^O_1) \land ((q_2, Pre_2, a, Post_2, q'_2) \in \delta_2 \land a \in \Sigma^I_2) \land Pre \equiv (Pre_1 \implies Pre_2) \land Post \equiv (Post_2 \implies Post_1)$

The following definition presents the set of illegal states in the product of two interface automata.
Definition 3.4 Let $A_1 = \langle Q_1, \Sigma_1^I, \Sigma_1^O, \Sigma_1^H, \text{Pre}_1, \text{Post}_1, \delta_1, I_1 \rangle$, $A_2 = \langle Q_2, \Sigma_2^I, \Sigma_2^O, \Sigma_2^H, \text{Pre}_2, \text{Post}_2, \delta_2, I_2 \rangle$, be two interface automata.

the set of illegal state in the product $A_1 \times A_2$ is denoted by $\text{Illegal}(A_1, A_2) = \{ (q_1, q_2) \in Q_1 \times Q_2 \mid \exists a \in \text{share}(A_1, A_2) \text{ such that,} \\
(a \in \sigma_1^O \land a \in \sigma_2^O) \lor (a \in \sigma_1^O \land a \in \sigma_2^I) \lor (a \in \sigma_1^O \land a \in \sigma_2^I \land (q_1, \text{Pre}_1, a, \text{Post}_1, q_1') \in \delta_1 \land (q_2, \text{Pre}_2, a, \text{Post}_2, q_2') \in \delta_2 \land ((\text{Pre}_1 \Rightarrow \text{Pre}_2) \land (\text{Post}_2 \Rightarrow \text{Post}_1) \text{ is not valid}) \lor (a \in \sigma_1^I \land a \in \sigma_2^O \land (q_1, \text{Pre}_1, a, \text{Post}_1, q_1') \in \delta_1 \land (q_2, \text{Pre}_2, a, \text{Post}_2, q_2') \in \delta_2 \land ((\text{Pre}_2 \Rightarrow \text{Pre}_1) \land (\text{Post}_1 \Rightarrow \text{Post}_2) \text{ is not valid})) \}$.

The set of illegal states describes the states in which the shared actions between the interface automata do not synchronize. So, on these states we have two cases:

- a component requires (or offers) an action which is not offered (or required) by the environment.
- a component requires (or offers) an action which is offered (or required) by the environment but the actions required (or offered) by the component is incompatible at the semantic level with the action offered (or required) by the environment.

In this approach, the verification of the compatibility between a component $C_1$ and a component $C_2$ is obtained by verifying the compatibility between their interface automata $A_1$ and $A_2$. Therefore, one verify if there is a helpful environment where it is possible to assemble correctly the components $C_1$ and $C_2$. So, one suppose the existence of such environment which accepts all the output actions of the automaton of the product $A_1 \times A_2$, and which do no not call any input actions in $A_1 \times A_2$.

In order to verify the compatibility between two components $C_1$ and $C_2$, it is necessary to verify of the compatibility between their respective interface automata $A_1$ and $A_2$. So, one verify if there is a helpful environment (other components) where it is possible to assemble correctly the components $C_1$ and $C_2$. So, one suppose the existence of such environment which accepts all the output actions of the automaton of the product $A_1 \times A_2$, and which do no not call any input actions in $A_1 \times A_2$.

Remark 3.5 The verification steps in this approach are the same as the ones presented in the section 2.1.1(the same steps as in [1]). However, in our approach we consider the action semantics in :

- the interface automata definition,
- the product of two interface automata,
- the definition of the illegal states.

Consequently, our approach does not increase the linear complexity of the verification algorithm.

4 The CyCab case study

Several approaches have been proposed to study the concept of CyCab [11]. It is a new means of electrical transportation conceived basically for freestanding transport services. It is totally controlled by a computer system and it can be driven
automatically according to many modes.

The goal of the CyCab is to allow to a clients to use the vehicle to move from one station to another. To illustrate this concept, we consider the following environment requirements and functionalities of CyCab:

• A CyCab has and appropriate road where stations are marked by sensors.
• We propose that the driving of the CyCab is guided by the information received from the station, which allows to situate the CyCab compared the station.
• There is no obstacle in the road.
• The vehicle has a starter.
• The vehicle has also an emergency halt button.

Figure 1. The UML model of the CyCab components.

The CyCab and its environment can be seen as an abstract system composed of four components: the vehicle, the halt emergency button, the starter and the station. The figure 1 represents the UML model of component based system CyCab.

• The emergency halt button can be activated at every moment during the CyCab moves. It is specified by sending a signal emergency!.
• The starter allows to start the CyCab in order to move.
• The station is materialized by a sensor that receives signals from vehicle giving the vehicle position (position?). The station send as consequence a signal (far! or halt!) to the corresponding vehicle to indicate if it is far from the station or not.
The vehicle sends a signal \textit{position!} to the station to know if it is near from the station or not and it receives signals \textit{(far?) or \textit{halt?}} from the station as response. The vehicle sends also a signal \textit{reset!} to the component emergency halt button in order to reset the system after activating the emergency button.

In this section, as shown in the figure 2, we apply the proposed approach to specify firstly all interfaces of the four components. Secondly, we verify the compatibility between two components \textit{Vehicle} and \textit{Station}. Assume that \(A_v\) and \(A_s\) are respectively two interface automata associated to the components \textit{Vehicle} and \textit{Station}. Let \(\mathcal{V} = \{\text{car, station, starter, position}\}\) be the set of variables their respective domains are \{\text{moving, stopped}\}, \{\text{reached, notreached}\}, \{\text{active, inactive}\}, \{\text{known, unknown}\}.

The automaton \(A_v\) is given by the tuple \(\langle Q_v, \Sigma^I_v, \Sigma^O_v, \Sigma^H_v, \text{Pre}_v, \text{Post}_v, \delta_v, I_v \rangle\)

where:

- \(Q_v = \{1, 2, 3, 4\}\);
- \(\Sigma^I_v = \{\text{halt, start, emergency, far}\}\);
- \(\Sigma^O_v = \{\text{position, reset}\}\);
\(\Sigma^E \equiv \emptyset;\)

- \(\Sigma^S = \{VPrH, VPrS, VPrE, VPrF, VPrP, VPrR\}\)
  - \(VPrH \equiv \text{car} = \text{moving}\) is the precondition of the method \textit{halt};
  - \(VPrS \equiv \text{car} = \text{stopped} \land \text{starter} = \text{active}\) is the one of the method \textit{start};
  - \(VPrE \equiv \text{car} = \text{moving}\) is the the one of the method \textit{emergency};
  - \(VPrF \equiv \text{car} = \text{moving} \land \text{station} = \text{not reached}\) is the one of the method \textit{far};
  - \(VPrP \equiv \text{car} = \text{moving} \land \text{position} = \text{unknown}\) is the one of the method \textit{position};
  - \(VPrR \equiv \text{starter} = \text{inactive}\) is the one of the method \textit{reset};

- \(\text{Post}_v = \{VPsH, VPsS, VPsE, VPsF, VPsP, VPsR\}\)
  - \(VPsH \equiv \text{car} = \text{stopped} \land \text{starter} = \text{active}\) is the postcondition of the method \textit{halt};
  - \(VPsS \equiv \text{car} = \text{moving}\) is the one of the method \textit{start};
  - \(VPsE \equiv \text{car} = \text{stopped} \land \text{station} = \text{not reached} \land \text{starter} = \text{inactive}\) is the the one of the method \textit{emergency};
  - \(VPsF \equiv \text{station} = \text{not reached}\) is the one of the method \textit{far};
  - \(VPsP \equiv \text{car} = \text{moving} \land \text{position} = \text{known}\) is the one of the method \textit{position};
  - \(VPsR \equiv \text{starter} = \text{active}\) is the one of the method \textit{reset}.

- \(\delta_v\) is the set of transition relation;
- \(I_v = \{1\}\)

The automaton \(A_v\) is given by the tuple \(\langle Q_v, \Sigma^I_v, \Sigma^O_v, \Sigma^H_v, \text{Pre}_v, \text{Post}_v, \delta_v, I_v\\rangle\)

  - \(Q_v = \{1, 2\}\);
  - \(\Sigma^I_v = \{\text{position}\}\);
  - \(\Sigma^O_v = \{\text{halt, far}\}\);
  - \(\Sigma^H_v = \emptyset;\)
  - \(\text{Pre}_v = \{SPrP, SPrH, SPrF\}\)
    - \(SPrP \equiv \text{car} = \text{moving} \land \text{position} = \text{unknown}\) is the precondition of the method \textit{position};
    - \(SPrH \equiv \text{car} = \text{moving}\) is the one of the method \textit{halt};
    - \(SPrF \equiv \text{car} = \text{moving} \land \text{station} = \text{not reached}\) is the the one of the method \textit{far};
  - \(\text{Post}_v = \{SPsP, SPsH, SPsF\}\)
    - \(SPsP \equiv \text{car} = \text{moving} \land \text{position} = \text{known}\) is the precondition of the method \textit{position};
    - \(SPsH \equiv \text{car} = \text{stopped} \land \text{station} = \text{reached} \land \text{starter} = \text{active}\) is the one of the method \textit{halt};
    - \(SPsF \equiv \text{car} = \text{moving} \land \text{station} = \text{not reached}\) is the the one of the method \textit{far};
  - \(\delta_v\) is the set of transition relation;
  - \(I_v = \{1\}\)

The composition of the two interfaces \(A_v\) and \(A_s\) is possible because the set \(\text{Shared}(A_v, A_s) = \{\text{position, halt, far}\} \neq \emptyset\). The synchronized prod-
uct between them as shown in the figure 3, have as pre and post conditions of operations $Pre_{v \otimes s} = \{PrS, PrP, PrH, PrF, PrE, PrR\}$ and $Post_{v \otimes s} = \{PsS, PsP, PsH, PsF, PsE, PsR\}$ where

• $PrP \equiv VPrP \Rightarrow SpP$;
• $PrH \equiv SpP \Rightarrow VPrH$;
• $PrF \equiv SpP \Rightarrow VPrF$;
• $PrS \equiv VPrS$;
• $PrE \equiv VPrE$;
• $PrR \equiv VPrR$;
• $PsP \equiv SpP \Rightarrow VPsP$;
• $PsH \equiv VPsH \Rightarrow SpP$;
• $PsF \equiv VPsF \Rightarrow SpP$;
• $PsS \equiv VPsS$;
• $PrE \equiv VPsE$;
• $PrR \equiv VPsR$.

We apply the algorithm detailed in the previous section to compute the synchronized product automaton:

• After computing the set of illegal states in the product, we obtain the set $Illegal(A_v, A_s) = \{32, 12\}$. The state 32 is an illegal state because from the state 3 in the automaton Vehicle, the postcondition of the input shared action, $halt?$, do not imply the precondition of the corresponding output action, $halt!$, from the state 2 in the component Station ($VPsH \Rightarrow SpP$). In fact, the component Vehicle offer the actions $halt$ which provokes strictly the vehicle halt, while the component Station solicit an action $halt$ which provokes the the vehicle halt and
the station reach.

- Next, we compute by performing a backward reachability analysis from Illegal states which traverses only internal and output steps, all states thus reachable. The resulting set is \{11, 21, 42\} and so the set of unreachable states is \{41\}

- Finally, we remove all incompatible and unreachable states \{11, 21, 42, 32, 12, 41\} and from the product automaton to obtain their composite automaton \(A_v \parallel A_s\). The set of remaining states is empty and so, the two interfaces Vehicle and Station are not compatible.

Remark 4.1 If we apply the approach proposed by L. Alfaro and T. Henzinger [1] on the same use case, we can detect a compatibility between the components Vehicle and Station, which is contrasted by considering the semantics of the action halt.

5 Conclusion and perspectives

The proposed work in this paper is a methodology to analyze the compatibility between component interfaces. We are inspired by the method proposed by L. Alfaro and T. Henzinger where interfaces are described by protocols modeled by I/O automata. We improved these automata by pre and post conditions of component actions in order to handle the action semantics in the verification of interface compatibility. This verification is made up of two steps. The first determines if two components are composable or not by checking some conditions on the actions feasibility by considering their semantics. The second aims is to detect inconsistencies between the sequences of action calls given by communicating protocols. This phase is obtained by considering the synchronized product of interface automata. These results are applied on the case study of the autonomous vehicle CyCab.

In this context, we are interesting for two research directions. The first consists in implementing a verification tool which takes into account pre and post conditions of actions to check compatibility between interfaces. The second concerns composite components and their refinement to define under which conditions a set of assembled components satisfies constraints of the composite component.

References


