Submodule construction for systems of I/O automata*

J. Drissi[1], G. v. Bochmann[2]

[1] *Dept. d'IRO, Université de Montréal, CP. 6128, Succ. Centre-Ville, Montréal, H3C 3J7, Canada, Phone: (514) 343-6161, Fax: (514) 343-5834, drissi@iro.umontreal.ca*
[2] *School of Information Technology & Engineering, University of Ottawa , Colonnel By Hall (A510), P.O.Box 450 Stn A,Ottawa,Ont.,K1N 6N5, Canada, Phone : (613) 562-5800 ext. 6205, Fax 562-5175 , bochmann@site.uottawa.ca*

Abstract. This paper addresses the problem of designing a submodule of a given system of communicating I/O automata. The problem may be formulated mathematically by the equation $(C||X)\ \mathfrak{R}\ A$ under the constraint $I_X=In$, where C represents the specification of the known part of the system, called the context, A represents the specification of the whole system, X represents the specification of the submodule to be constructed, || is a composition operator, $\mathfrak{R}$ is a conformance relation and In is the required set of inputs for X. As conformance relation, we consider the safe realization and the subtype relation. The subtype relation is a generalization of the well known criteria of trace equivalence, complete trace equivalence, quasi equivalence and reduction, while the weaker safe realization relation is implied by all those criteria. We propose two algorithms for solving the problem with respect to the safe realization and the subtype relation and we characterize the set of solutions in each case.

1 Introduction

One common problem, encountered in the hierarchical design of complex systems, in the synthesis of controllers and in the reuse of components, is the submodule construction problem, also called factorization problem or equation solving problem. The submodule construction problem (SCP) is to construct the specification of a submodule X when the specification of the system and all submodules but X are given. Such a problem may be formulated mathematically by the equation $(C||X)\ \mathfrak{R}\ A$, where C represents the specification of the known part of the system, A represent the specification of the whole system, || is a composition operator and $\mathfrak{R}$ is a conformance relation. The SCP was first formulated and treated in [13], where specifications are expressed in terms of execution sequences, and trace equivalence was used as conformance relation. In [19], the author uses Milner's Calculus of Communicating Systems to model the same problem. Many other works [7, 18] have been done using labelled transition systems as a model for the specifications and the

* This work was partially supported by the NSERC Strategic grant SRTGP200 "Methods for the systematic testing of distributed software systems" and an NSERC Research grant.

strong and/or the observational equivalences as conformance relations. In [4], we consider this problem in the context of the input/output Finite State Machine model (I/O FSM) [5]. The direct application of an approach based on the LTS model is not possible since the solutions obtained are not in general I/O FSMs. We have to add constraints on the environment behavior to obtain the system's behavior in the form of an I/O Finite State Machine. We have developed a method for constructing all the solutions when the specifications are given in the form of deterministic completely specified input/output Finite State Machines and the trace equivalence relation is used as conformance relation. This work was generalized in [16] to the case where the specifications are given in the form of nondeterministic completely specified input/output Finite State Machines and the reduction relation is used as conformance relation.

We will generalize our previous work by dealing with nondeterministic partially specified input/output Finite State Machines and by using other criteria such as complete trace equivalence, quasi-equivalence and reduction of nondeterminism. For this purpose, we consider partial I/O automata for systems specification, which is more general than input/output Finite State Machines. An I/O automata corresponding to a given input/output Finite State Machine can always be obtained by unfolding each atomic input/output transition *s-x/y->s'* into two consecutive transitions *s-x->s"* and *s"-y->s'* of the corresponding I/O automaton. A fundamental property of the model of I/O automata is that there is a very clear distinction between those actions that are performed under the control of the automaton and those actions that are performed under the control of its environment. An automaton's transitions are classified as either "input" or "output".

In typed object-oriented languages the notion of subtype, that is, a conformity relation between types, is defined. A type *P* conforms to another type *Q* if *P* provides at least the operations of *Q* (*P* may also provide additional operations). Moreover, the types of the results of *P*'s operations must conform to the types of the results of the corresponding operations of *Q*. Finally, the types of the arguments of *Q*'s operations must conform to those of *P*'s operations [2]. The idea behind the notion of subtype is the ability to use an instance of a subtype of a type *T* whenever an instance of type *T* is required to do a job.

While the subtyping relation of object-oriented languages are mainly concerned with the available operations and the types of their parameters, we are concerned in this paper with the dynamic behavior of objects, that is, of I/O automata, considering the allowed sequences of input and output operations. We will define a subtype criterion, denoted $\preceq$, for the same purpose as in object-oriented languages, i.e. the possibility to replace any subsystem by an instance of its subtypes without changing the system's behavior.

When composing a collection of partial I/O automata, problems due to unspecified reception may appear when a receiving I/O automaton does not have an input transition originating from the present state when the sending *IOA* executes a corresponding output transition. A composition of a collection of I/O automata is said to be safe if it does not contains unspecified receptions [8, 14].

We define the safe realization of an I/O automaton A by a composite *IOA* $B=B_1||B_2||...||B_n$, denoted $B \leq_{\$} A$ as follows: for any environment modeled by an I/O automaton E, if the composition of A and E is safe then the composition of B_1, B_2, ..., B_n and E is also safe. The safe realization criterion does not allow us to enforce mandatory output behaviors in certain given states, i.e. an *IOA B* which is a safe realization of an *IOA A*, will accept all the inputs accepted from the initial state of A and may produce no output. In the paper "*Modal Specifications*" [9], the author presents a theory of Modal Specifications which imposes restrictions on the transitions of possible implementations by telling which transitions are mandatory and which are admissible. This allows a refinement ordering between Modal Specifications to be defined. To deal with the problem of mandatory output behaviors and also to be able to represent the set of solutions to the equation $(C||X) \mathcal{A} A$ as the set of subtypes of a particular type (or model) in the same modelling framework, we enhance the model of I/O automata by allowing the imposing of conditions on the set of traces from a given state. We call such a model an "I/O automata with optional complete traces". With each state s we associate two sets: the first set contains sets of traces from s such that at least one trace in each set must be present in any implementation of this automaton; the second set is a subset of the traces from s and each time an execution, starting in s, is a prefix of an element of this set, the execution should progress to complete a trace in this set. We assume that an implementation is a normal (partial) I/O automaton. We will introduce a progress property which formalizes the preceding notion. Moreover, by requiring safe realization and realization of the progress property, we obtain the subtype relation. We show in this paper that the subtype relation is a generalization of the well known criteria trace equivalence, complete trace equivalence, quasi equivalence and reduction.

Since in a composition of a collection of *IOAs* the sets of inputs are not disjoint, we generalize the architecture by allowing the component that will be designed, to observe some interactions between the environment and the context. This is done by adding to the equation a constraint on the required set of inputs of the solution. For the two criteria, safe realization and subtype, we propose for each an algorithm that produces an I/O automaton solution to the equation $(C||X) \Re A$ under the constraint $I_X=In$ if such a solution exists. We prove that the set of possible solutions is then either the set of safe realizations or the set of subtypes of the obtained solution, depending on which criterion was in the equation.

This paper is structured as follows. In Section 2, we define basic notions. In Section 3, we introduce the safe realization and the subtype relations and we compare them with the trace equivalence, complete trace equivalence, quasi equivalence and reduction criteria. Section 4 presents the submodule construction problem and the architecture in which this problem will be solved. In Section 5, we propose an algorithm for solving the problem with respect to the safe realization relation and we characterize the set of solutions. In Section 6, we propose an algorithm to solve the problem with respect to the subtype relation and we characterize the set of solutions. In Section 7, we show that the submodule construction problem for non-deterministic partially specified input/output Finite State Machines is a particular case of the results in Section 6 for the trace equivalence, quasi equivalence and reduction criteria. Finally, in Section 8 we conclude the paper. The proofs of the theorems are in the annex.

2 Input-Output Automata

2.1 Basic notions and definitions

In this paper, an I/O automaton (briefly *IOA*) A, is a 5-tuple $(S_A, I_A, O_A, T_A, s_{0A})$ where S_A is a finite set of states with s_{0A} as the initial state, I_A is a non-empty, finite set of inputs, O_A is a non-empty, finite set of outputs with $I_A \cap O_A = \emptyset$ and $T_A \subseteq S_A \times (I_A \cup O_A) \times S_A$ is a transition set. An element $(s, u, s') \in T_A$ is denoted by $s\text{-}u\rightarrow s'$. If for each $s \in S_A$ and all $x \in I_A$ there exists $s' \in S_A$ such that $s\text{-}x\rightarrow s'$, then A is said to be **completely specified** or **input-enabled**; otherwise A is **partially specified**. An *IOA* A is said to be **nondeterministic** if there exist $s\text{-}u\rightarrow s'$, $s\text{-}u\rightarrow s''$ and $s' \neq s''$ for some s and u; otherwise A is **deterministic**. For a deterministic *IOA* A, the outgoing transitions of each state are uniquely labeled. For each state $s \in S_A$, we denote $inp(s)=\{x \in I_A \mid \exists s' \in S_A\ s\text{-}x\rightarrow s'\}$, $out(s)=\{y \in O_A \mid \exists s' \in S_A\ s\text{-}y\rightarrow s'\}$, $entering(s)=\{u \in (I_A \cup O_A) \mid \exists s' \in S_A\ s'\text{-}u\rightarrow s\}$, and $leaving(s)=inp(s) \cup out(s)$. The **input-enabled form** of an *IOA* A, denoted by $Ief(A)$, is obtained by adding to each state s transitions for the non-specified inputs $(I_A \backslash inp(s))$ leading to the special state *Fail*, i.e., $Ief(A)=(S_A, I_A, O_A, T_A \cup (\cup_{s \in S_A}(\{s\} \times (I_A \backslash inp(s))\{Fail\})), s_{0A})$.

If there exist states $s_1, ..., s_{k+1} \in S_A$ such that $(s_i, a_i, s_{i+1}) \in T_A$, for each $i = 1,...,k$, then the k-tuple $((s_1, a_1, s_2), (s_2, a_2, s_3), ...,(s_k, a_k, s_{k+1}))$ is said to be an **execution** starting in s_1. The sequence $\sigma = a_1...a_k \in (I_A \cup O_A)^*$ is said to be a **trace** from the state s_1. The set of traces from the state s is denoted $Tr_A(s)$ and we denote it Tr_A if $s = s_{0A}$. For a deterministic *IOA* A, a state s and a sequence $\sigma \in Tr_A(s)$ uniquely determine the final state of the trace σ which we denote s_σ. A state s' of an *IOA* A is **reachable** from a state s if there exists $\sigma \in Tr_A(s)$ such that $s_\sigma = s'$. If s is the initial state of A then s' is said to be reachable. The set $\{s_1, s_2, ..., s_k, s_{k+1}\}$, denoted by $S(s_1, \sigma)$, represents the set of states reachable from s_1 by a prefix of σ. For each sequence $\sigma \in \Sigma^*$ and a

subset Σ' of Σ, the **Σ'-projection** of σ, denoted $Pr_{\Sigma'}(\sigma)$, is obtained by deleting from σ each symbol which is not in Σ'. For simplicity, we denote also by $Pr_A(\sigma)$, the projection of σ over the alphabet ($I_A \cup O_A$) of the *IOA A*. For a set of traces *Y*, we denote by $Pr_{\Sigma'}(Y)$, the set containing the Σ'-projection of the elements in *Y*. For a set *X* containing sets of traces, we denote by $Pr_{\Sigma'}(X)$, the set $\{Pr_{\Sigma'}(Y) |\ Y \in X\}$.

The **connected component** of *A* containing the initial state is the *IOA* $CC(A)=(S_C, I_C, O_C, T_C, s_{oC})$ such that $S_C=\{s \in S_A |\ s \text{ reachable}\}$, $I_C=I_A$, $O_C=O_A$, $T_C=\{(s, u, s') \in T_A |\ s \in S_C\}$ and $s_{oC}=s_{oA}$. If $A=CC(A)$ then *A* is said to be **initially connected**.

We define a **chaos *IOA***, whose traces contain all the words over the alphabet, by $Ch=(\{ch\}, I, O, H, ch)$, where $H=\{(ch, t, ch) \mid t \in I \cup O\}$.

2.2 The composition of I/O automata

A system can be considered as a finite collection of *IOAs* communicating with one another and with the environment. The composition of *IOAs* is defined in the case of complete *IOAs* in [12], and in the case of partial *IOAs* in [8, 14]. In the second case problems due to unspecified reception may appear when a receiving *IOA* does not have an input transition originating from the present state when the sending *IOA* executes a corresponding output transition.

Definition 1 : Given a collection of *IOAs* $(A_i=(S_{A_i}, I_{A_i}, O_{A_i}, T_{A_i}, s_{oi}))_{1 \le i \le n}$, such that the sets $(O_{A_i})_{1 \le i \le n}$ are pairwise disjoint. The composition of $(A_i)_{1 \le i \le n}$, denoted $A_1 || A_2 || ... || A_n$, is defined as the connected component of the *IOA* $A=(S_A, I_A, O_A, T_A, s_{oA})$ where :

- $S_A=S_{A_1} \times S_{A_2} \times ... \times S_{A_n}$,
- $I_A=(I_{A_1} \cup I_{A_2} \cup ... \cup I_{A_n}) \setminus (O_{A_1} \cup O_{A_2} \cup ... \cup O_{A_n})$,
- $O_A=O_{A_1} \cup O_{A_2} \cup ... \cup O_{A_n}$,
- $((s_1, s_2, ..., s_n), u, (s_1', s_2', ..., s_n')) \in T_A$ iff for all $i \in \{1, 2, ..., n\}$, if $u \in (I_{A_i} \cup O_{A_i})$ then $(s_i, u, s_i') \in T_{A_i}$, else $s_i=s_i'$,
- $s_{oA}=(s_{o1}, s_{o2}, ..., s_{on})$.

The composition of *IOAs* is commutative and associative. This composition allows a number of *IOAs* to accept the same input simultaneously.

Following the work of [14], we define a safety property which formalizes the non-occurrence of an unspecified reception in the composition *A* of a collection of *IOA* $(A_i=(S_{A_i}, I_{A_i}, O_{A_i}, T_{A_i}, s_{oi}))_{1 \le i \le n}$. We denote by ε the empty word.

Definition 2 : Given a collection of *IOA* $(A_i=(S_{A_i}, I_{A_i}, O_{A_i}, T_{A_i}, s_{0i}))_{1\le i\le n}$, the composition $A=A_1||A_2||...||A_n$ is safe, written $\mathbb{S}(A)$, iff any word t in $(I_A\cup O_A)^*$ such that $Pr_{A_i}(t)\in Tr_{A_i}.(I_{A_i}\cup\{\varepsilon\})$ for all i, is a trace of A (i.e. $t\in Tr_A$).

We illustrate the preceding concepts with the following example (Figure 1, 2, 3). We consider the *IOAs* A_1 and A_2 with $I_{A_1}=\{x, x', z\}$, $O_{A_1}=\{u, y\}$, $I_{A_2}=\{u\}$ and $O_{A_2}=\{z\}$. For the composition $A=A_1||A_2$ we have $I_A=\{x, x'\}$ and $O_A=\{u, z, y\}$.

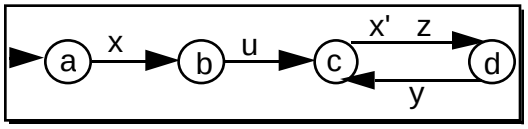

Figure 1 : The *IOA* A_1

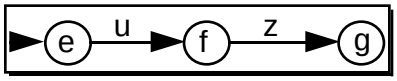

Figure 2 : The *IOA* A_2

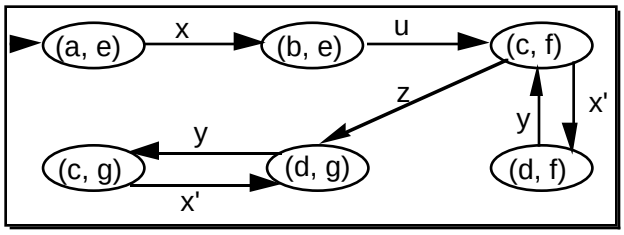

Figure 3 : The *IOA* $A_1||A_2$

We define now a hiding operator which allows us, for example, to hide actions considered to be internal in a composition. Given an *IOA* A and a subset Σ of $(I_A\cup O_A)$, we define $H_\Sigma(A)$ to be the *IOA* obtained from A by first replacing all the actions in Σ by the internal action τ and then determinizing the obtained automaton. In Figure 4, we have $D=H_{\{u, z\}}(A_1||A_2)$, $I_D=\{x, x'\}$ and $O_D=\{y\}$.

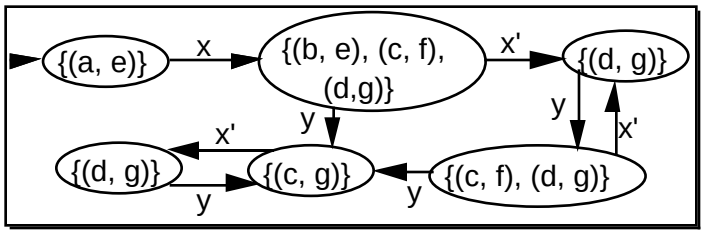

Figure 4 : The *IOA* $H_{\{u, z\}}(A_1||A_2)$

3 The conformance relations

3.1 The safe realization relation

Definition 3 : For an *IOA* A and a composite *IOA* $B=B_1||B_2||...||B_n$, with $I_A=I_B$, we say that B realizes A with safety, written $B\le_{\mathbb{S}} A$, iff for every *IOA* E, with $I_E=O_A$ and $O_E=I_A$, $\mathbb{S}(E||A)$ implies $\mathbb{S}(E||B_1||B_2||...||B_n)$.

Definition 3 means that for any environment E over the alphabet of A if the composition of A and E is safe then the composition of B_1, B_2, ..., B_n and E must also be safe, i.e. in any reachable state of the composition $E||B_1||B_2||...||B_n$, there is no unspecified reception.

Definition 4 : The reflection of an *IOA* $A=(S_A, I_A, O_A, T_A, s_{oA})$ is the *IOA* $\tilde{A}=(S_A, I_{\tilde{A}}, O_{\tilde{A}}, T_A, s_{oA})$ where $I_{\tilde{A}}=O_A$, $O_{\tilde{A}}=I_A$.

If for any state of A, we add some outputs then the composition of the obtained *IOA* with $\tilde{A}$ will be non safe. Also, if we add some outputs for any state of $\tilde{A}$, the composition of the obtained *IOA* with A will be non safe. Intuitively, $\tilde{A}$ represents the most liberal environment in with A is safe.

Lemma 1: For an *IOA* A and a composite *IOA* $B=B_1||B_2||...||B_n$, with $I_A=I_B$, the following propositions are equivalent :

i - $\mathbb{S}(\tilde{A}||B_1||B_2||...||B_n)$,

ii - for every *IOA* E, with $I_E=O_A$ and $O_E=I_A$, $\mathbb{S}(E||A) \Rightarrow \mathbb{S}(E||B_1||B_2||...||B_n)$.

In the example in Figure 3, the *IOA* $A=A_1||A_2$ is not safe since for the trace $t=xux'z$ in $(I_A \cup O_A)^*$ we have $Pr_{A_1}(t) \in Tr_{A_1}.(I_{A_1})$ and $Pr_{A_2}(t) \in Tr_{A_2}$, but we do not have $t \in Tr_A$. However, for the *IOA* S given in Figure 5, we have that the I/O automaton $A=A_1||A_2$ realizes S with safety, since S does not allow for the visible trace xx' corresponding to the above trace t.

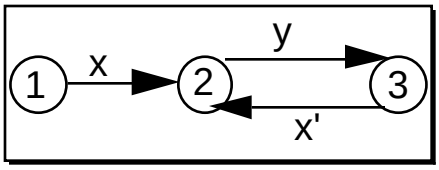

Figure 5 : The I/O automaton S

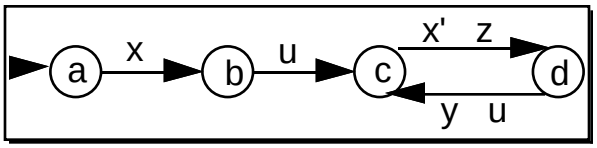

Figure 6 : The I/O automaton A_3

If we consider the *IOA* A_3 in Figure 6 with $I_{A_3}=I_{A_1}$ and $O_{A_3}=O_{A_1}$, we remark that $A_3||A_2=A_1||A_2$ but $A_3||A_2$ does not realize S with safety since for the trace $t'=xuzyx'u$ in $(I_A \cup O_A)^*$ we have $Pr_{A_3}(t') \in Tr_{A_3}$, $Pr_{A_2}(t') \in Tr_{A_2}.(I_{A_2})$ and $Pr_S(t') \in Tr_S$, but t' is not a trace of the composition $\widetilde{S}\ ||A_3||A_2$.

3.2 The subtype relation

The safe realization relation is a weak criterion since it allows an *IOA* B which is a safe realization of an *IOA* A, to accept all the inputs accepted from the initial state of A and to produce no output. To deal with the notion of task completion, which imposes conditions on the set of outputs produced after a given execution and also to be able to represent the set of solutions to the

equation $(C \| X) \mathbb{S} A$ as the set of subtypes of an element in the same model, we need to enhance the definition of an *IOA* to take into account these conditions. This leads to the definition of I/O automata with optional complete traces.

Definition 5 : An I/O automaton with optional complete traces *A* (briefly *IOAWOCT*), is a 3-tuple (IOA_A, MT_A, OCT_A) where IOA_A is an *IOA*, $MT_A = \{(s, MT_A(s)) | s \in S_A\}$ with $MT_A(s) \subseteq 2^{Tr_{IOA_A}(s)}$ and $OCT_A = \{(s, OCT_A(s)) | s \in S_A\}$ with $OCT_A(s) \subseteq Tr_{IOA_A}(s)$.

An *IOAWOCT A* can be considered to be a specification. The sets $MT_A(s)$ and $OCT_A(s)$ impose constraints on the traces of valid implementations of *A*. We consider implementations in the form of *IOA*. An element *Y* of $MT_A(s)$ imposes that at least one trace of *Y*, is a possible trace of the implementation in the state corresponding to *s*. Moreover, each time, an execution starting in the state corresponding to *s* in the implementation has a trace whose projection over the alphabet of IOA_A is a prefix of an element of $OCT_A(s)$, this execution should progress to complete some trace whose projection over the alphabet of IOA_A is in $OCT_A(s)$. We note that if $OCT_A(s)$ contains only elements of length equal to one, it does not impose any constraint on the implementations having the same alphabet as IOA_A. In the case where the implementation has a different alphabet from the alphabet of IOA_A, the elements of length equal to one in $OCT_A(s)$ prohibit traces without external output.

An *IOA A* can be viewed as an *IOAWOCT*, which we call A^{woct}, and which is constructed as follow : $IOA_{A^{woct}} = A$ and for each state *s* of *A* $MT_{A^{woct}}(s) = \{\{y\} | y \in out(s)\}$ and $OCT_{A^{woct}}(s) = \emptyset$. With this consideration, the set of *IOA*s is included in the set of *IOAWOCT*s.

Consider an *IOAWOCT A* and an environment *E* such that $\mathbb{S}(E \| IOA_A)$. If we want to replace *A* in the environment *E* by an *IOA B* which is a subtype of *A*, then *B* must be a safe realization of IOA_A, i.e. $B \leq_{\mathbb{S}} IOA_A$, moreover, since in each state *s* of IOA_A, there are some conditions on the set $Tr_{IOA_A}(s)$, then *B* must realize these conditions. In the following definition, we formalize this notion by what we call the progress property. We note *Pref*(*X*) the set of all non-empty prefixes of elements of a set of traces *X*.

Definition 6 : Given a deterministic *IOAWOCT A*, we say that the deterministic *IOA B* with $I_B = I_{IOA_A}$ realizes the progress property with respect to the *IOAWOCT A*, written $B \leq_{\mathbb{P}} A$, iff

If $MT_A(s_{0IOA_A}) \neq \emptyset$ then

i - for every *X* of $MT_A(s_{0IOA_A})$, $Pr_{IOA_A}(Tr_B(s_{0B})) \cap X \neq \emptyset$,

ii - for every $\sigma_1 \in Tr_B(s_{0B})$, if $Pr_{IOA_A}(\sigma_1) \in Pref(OCT_A(s_{0IOA_A}))$ then there exists $\sigma_2 \in Tr_B((s_{0B})_{\sigma_1})$ such that $Pr_{IOA_A}(\sigma_1\sigma_2) \in OCT_A(s_{0IOA_A})$.

And, for every $\sigma t \in Tr_B$ with $t \in (I_{IOA_A} \cup O_{IOA_A})$, if $\sigma' = Pr_{IOA_A}(\sigma t) \in Tr_{IOA_A}$ and $MT_A((s_{0IOA_A})_{\sigma'}) \neq \emptyset$ then

iii - for every X of $MT_A((s_{0IOA_A})_{\sigma'})$, $Pr_{IOA_A}(Tr_B((s_{0B})_{\sigma t})) \cap X \neq \emptyset$,

iv - for every $\sigma_1 \in Tr_B((s_{0B})_{\sigma t})$, if $Pr_{IOA_A}(\sigma_1) \in Pref(OCT_A((s_{0IOA_A})_{\sigma'}))$ then there exists $\sigma_2 \in Tr_B((s_{0B})_{\sigma t \sigma_1})$ such that $Pr_{IOA_A}(\sigma_1 \sigma_2) \in OCT_A((s_{0IOA_A})_{\sigma'})$.

The conditions (i) and (iii) impose that one trace from each element of $MT_A((s_{0IOA_A})_{\sigma'})$ is present in the projection over the alphabet of IOA_A of $Tr_B((s_{0B})_{\sigma t})$. The conditions (ii) and (iv) impose that each trace of $Tr_B((s_{0B})_{\sigma t})$ whose projection over the alphabet of IOA_A is a prefix of an element of $OCT_A((s_{0IOA_A})_{\sigma'})$, can progress to complete a trace in $Tr_B((s_{0B})_{\sigma t})$ whose projection over the alphabet of IOA_A is an element of $OCT_A((s_{0IOA_A})_{\sigma'})$.

Now, we give a formal definition for the notion that an *IOA* is a conforming implementation of an *IOAWOCT* by requiring safe realization and realization of the progress property.

Definition 7 : Given a deterministic *IOA B* and a deterministic *IOAWOCT A* with $I_B = I_{IOA_A}$, *B* is said to be a conforming implementation of *A*, written $B \leq_{conf} A$, iff $B \leq_{\mathbb{P}} A$ and $B \leq_{\mathbb{S}} IOA_A$.

We remark that in the case where IOA_A is completely specified, $B \leq_{conf} A$ and $I_B \subseteq I_A$ implies $Pr_{IOA_A}(Tr_B) \subseteq Tr_{IOA_A}$.

We say that an *IOAWOCT B* is a subtype of an *IOAWOCT A* if all conforming implementations of *B* are also conforming implementations of *A*.

Definition 8 : A deterministic *IOAWOCT B* is a subtype of a deterministic *IOAWOCT A*, written $B \preceq A$, if for any deterministic *IOA C* : $C \leq_{conf} B$ implies $C \leq_{conf} A$.

Lemma 2 : Given a deterministic *IOA B* and a deterministic *IOAWOCT A* with $I_B = I_{IOA_A}$. The following propositions are equivalent :

i - $B \leq_{conf} A$ ii - $B^{woct} \preceq A$.

3.3 Others conformance relations

We will compare some well known conformance relations with the subtype and the safe realization relations.

3.3.1 Trace equivalence

The *IOAs A* and *B* are said to be trace-equivalent, written $B \cong A$, iff $Tr_A = Tr_B$. Given an *IOA A*, a trace $\sigma \in Tr_A$ is said to be a complete trace iff $leaving((s_{oA})_\sigma) = \emptyset$. The set of complete traces is denoted CTr_A. The *IOA B* is said to be complete trace-equivalent to the *IOA A*, written $B =_{cte} A$, iff $Tr_A = Tr_B$ and $CTr_A = CTr_B$ [6]. In the case of deterministic *IOAs*, the complete trace-equivalence reduces to the trace-equivalence.

Lemma 3: Given two deterministic *IOAs A* and *B*, with $I_A = I_B$ and $O_A = O_B$. The following propositions are equivalent :

i - $B =_{cte} A$,

ii - $B \leq_{conf} A^{woct}$ and $A \leq_{conf} B^{woct}$.

3.3.2 Quasi-equivalence

The *IOA B* is said to be quasi-equivalent to the *IOA A*, written $B \leq_{qe} A$, iff

$$\forall \sigma \in Tr_A \ (\sigma \in Tr_B \wedge out((s_{oA})_\sigma) = out((s_{oB})_\sigma)).$$

The quasi-equivalence relation requires that $Tr_A \subseteq Tr_B$ and after any trace in Tr_A, *A* and *B* produce the same set of outputs [11, 17].

Lemma 4: Given two deterministic *IOAs A* and *B*, with $I_A = I_B$ and $O_A = O_B$. The following propositions are equivalent :

i - $B \leq_{qe} A$,

ii - $B \leq_{conf} A^{woct}$.

3.3.3 Reduction of nondeterminism

The *IOA B* is said to be a reduction of the *IOA A*, written $B \leq_{red} A$, iff for each trace σ in Tr_A if σ is in Tr_B then :

$$inp((s_{oA})_\sigma) \subseteq inp((s_{oB})_\sigma) \wedge out((s_{oB})_\sigma) \subseteq out((s_{oA})_\sigma) \wedge (out((s_{oA})_\sigma) \neq \emptyset \Rightarrow out((s_{oB})_\sigma) \neq \emptyset).$$

Lemma 5: Given two deterministic *IOAs A* and *B*, with $I_A = I_B$ and $O_A = O_B$. We denote by A^{red} the $IOAWOCT$ such that $IOA A^{red} = A$ and for each state s of A $MT A^{red}(s) = \{out(s)\}$ and $OCT A^{red}(s) = \emptyset$. The following propositions are equivalent :

i - $B \leq_{red} A$,

ii - $B \leq_{conf} A^{red}$.

Lemma 6: Each of the above conformance relations (trace equivalence, complete trace equivalence, quasi equivalence and reduction) implies the safe realization criterion, that is, if one of these relations holds between two deterministic *IOAs B* and *A* then *B* is a safe realization of *A*.

4 The design of a submodule

4.1 The architecture

We use the composition of two communicating components (Figure 7) to discuss problems related to the design of a component of a compound system.

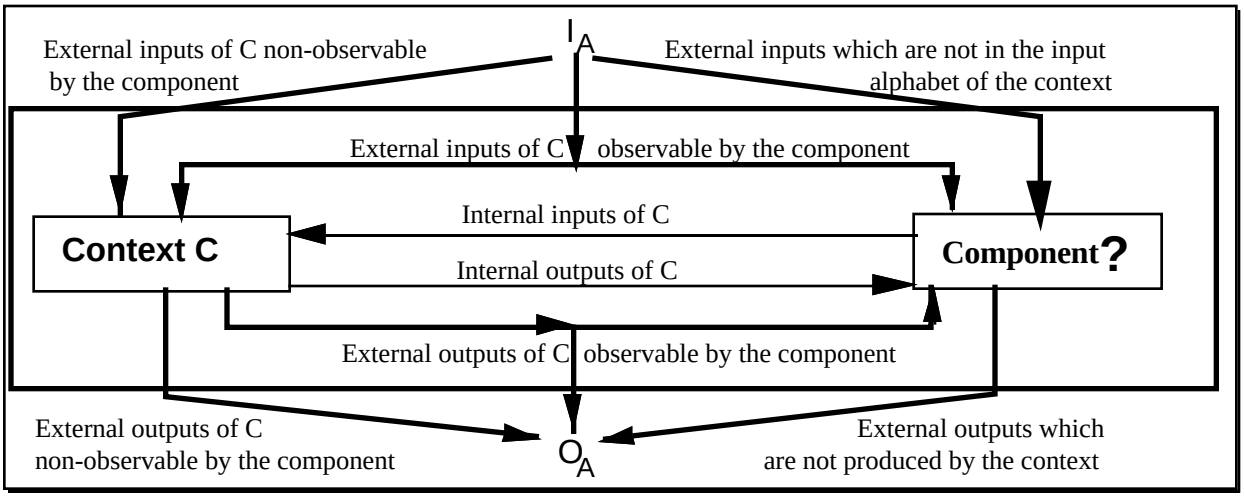

Figure 7: The composition of two communicating components *C* and *Comp*

We consider the class of systems which can be represented by two deterministic *IOA* that communicate with one another and with an environment. One deterministic *IOA*, called the *context C*, models the known part of the system, the behavior of which is given, while the other deterministic *IOA*, called the new component *Comp*, represents the behavior of a certain component of the system (the submodule to be designed). The set of inputs accepted by the system from the environment can be divided into three disjoint sets. The first is the set of inputs of the context *C* non observable by the component, the second is the set of inputs of the context observable by the component, and the third represents the set of inputs of the system which are not visible by the context. Similarly, the set of outputs delivered by the system to the environment can be divided in three disjoint sets. The first is the set of outputs delivered by the context *C* to the environment which are not observable by the new component, the second is the set of outputs delivered by the context *C* to the environment which are observable by the component, and the third represents the set of outputs of the component accepted by the environment and not delivered to the context.

4.2 The problem

The problem is known as the problem of submodule construction, redesign or equation solving, where an appropriate conformity criterion should hold between a designed system and its

given specification *A* and where the system consists of a given component *C* and a new component *X* to be designed. In this paper, we consider this problem as a problem of equation solution in the realm of *IOA* for the equation $(C||X)\ \Re\ A$ under the constraint $I_X=In$ with *X* being a free variable, $\Re$ is a conformance relation and *In* a given set of inputs.

In Section 5, we propose an algorithm which takes as input two deterministic *IOAs*, *A* and *C*, and a set *In* such that $(I_A \backslash I_C) \cup (O_C \backslash O_A) \subseteq In \subseteq I_A \cup O_C$, and produces as output an *IOA* $Sol_{\$}$ (if it exists) with $I_{Sol_{\$}}=In$ and $O_{Sol_{\$}}=(I_C \backslash I_A) \cup (O_A \backslash O_C)$, such that the composition of *C* with $Sol_{\$}$ is a safe realization of *A*. In Section 6, we propose an algorithm which takes as input a deterministic *IOA C*, a deterministic *IOAWOCT A*, and a set *In* such that $(I_A \backslash I_C) \cup (O_C \backslash O_A) \subseteq In \subseteq I_A \cup O_C$, and produces as output an *IOAWOCT Sol* (if it exists) with $I_{IOA_{Sol}}=In$ and $O_{IOA_{Sol}}=(I_C \backslash I_A) \cup (O_A \backslash O_C)$, such that the composition of *C* with IOA_{Sol} is a conforming implementation of *A*. The existence of a solution for a given equation depends on the selected set of inputs for the component.

Lemma 7 : If for a given set of inputs *In* there is no solution, then for any subset of *In* there is no solution.

5 The solution for the safe realization criterion

5.1 The proposed method

We use a chaos *IOA* which represents all the traces over the input alphabet $I_A \cup O_C$ and the output alphabet $(I_C \backslash I_A) \cup (O_A \backslash O_C)$; any solution of $(C||X) \leq_{\$} A$ is trace included in this chaos automaton. The main idea of our approach is to remove from the chaos automaton all the traces which combined with traces of the context *C* in the environment $\tilde{A}$ may cause a non-safe behavior. This will allow us to capture the set (if not empty) of the permissible traces of the component to be designed in the form of an *IOA* $Sol_{\$}$, called the generic safe solution. A permissible trace is a trace of a solution of $(C \parallel X) \leq_{\$} A$. We give in the next paragraph an algorithm which constructs the generic safe solution if it exists. As input, the algorithm requires two deterministic *IOAs C* and *A* and a set *In* such that $(I_A \backslash I_C) \cup (O_C \backslash O_A) \subseteq In \subseteq I_A \cup O_C$. The set *In* will be used as the input set for the generic solution.

To be able to characterize those traces of the composition of the chaos automaton and the context *C* in the environment $\tilde{A}$ which lead to a non conforming behavior with respect to *A*, we find in Step 1 the input-enabled forms of $\tilde{A}$ and *C*, and then construct the composed *IOA* $R=Ief(C)||Ch||Ief(\tilde{A})$. Since a state of *R* is a triplet of states of $Ief(C)$, *Ch* and $Ief(\tilde{A})$, each time we reach a state of *R* which contains $Fail_{Ief(\tilde{A})}$ or $Fail_{Ief(C)}$, we replace this state by $Fail_R$. The complexity in the worst case of Step 1 is polynomial in the number of states of *A* and *C* and the

number of elements in the alphabet of $A \| C$. In Step 2, we replace in R all the actions that are not in the alphabet of the component to be designed by the internal action τ and we determinize. Since a state of the obtained automaton, denoted R_1, corresponds to a subset of states of R, we declare any state of R_1 which contains $Fail_R$ as $Fail_{R_1}$. The complexity of Step 2 is in the worst case exponential in the number of states of A and C. In the third and last step, we remove recursively from R_1 all the traces which lead to the $Fail_{R_1}$ state. The complexity of step 3 is in the worst case polynomial in the number of states of R_1.

Algorithm 1

Input : The specification of the context C, the specification of the system A and a set of inputs In.
Output : An *IOA* $Sol_{\S}$ with $I_{Sol_{\S}}=In$ (equal to R_1) which satisfies the equation $(C \| Sol_{\S}) \leq_{\S} A$ if a solution exists.

Step 1 : $R := Ief(C) \| Ch \| Ief(\tilde{A})$.
Step 2 : $R_1 := \mathrm{H}_{(I_A \cup O_C) \setminus In}(R)$.
Step 3 : (*Remove-Fail-state*)
WHILE exist a transition s - $t \rightarrow Fail_{R_1}$ **DO**
IF $t \in (I_C \setminus I_A) \cup (O_A \setminus O_C)$ **THEN** Remove this transition;
ELSE
IF $s = s_{0R_1}$ **THEN** return "NO SOLUTION"; **STOP**;
ELSE Replace each transition c- $t' \rightarrow s$ by c- $t' \rightarrow Fail_{R_1}$;
$R_1 := CC(R_1)$;

Theorem 1 : Given two deterministic *IOAs* A and C, and given a set In such that $(I_A \setminus I_C) \cup (O_C \setminus O_A) \subseteq In \subseteq I_A \cup O_C$, if Algorithm 1 produces an *IOA* $Sol_{\S}$ then $(C \| Sol_{\S}) \leq_{\S} A$, else there is no solution for $(C \| X) \leq_{\S} A$ with the specified input alphabet In.

Example :

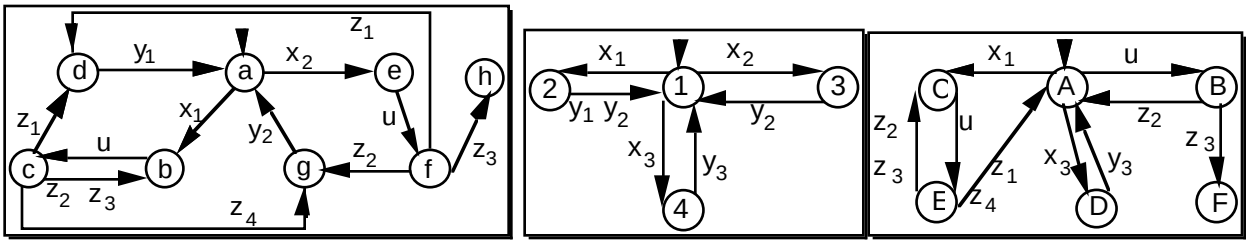

Figure 8 : the I/O automata C and the I/O automata A and the I/O automata $Sol_{\S}$

We consider for the context the *IOA* C shown in Figure 8 with $I_C=\{x_1, x_2, z_1, z_2, z_3, z_4\}$, $O_C=\{u, y_1, y_2\}$, and for the whole system the *IOA* A shown in Figure 9 with $I_A=\{x_1, x_2, x_3\}$,

$O_A=\{y_1, y_2, y_3\}$ and $In=\{x_1, x_3, u\}$. We obtain the solution $Sol_{\S}$ shown in Figure 10 with the output set $O_{Sol_{\S}}=\{z_1, z_2, z_3, z_4, y_3\}$.

5.2 The set of solutions

The solution obtained by the previous method is a generic one, which means that we can derive from it the set of solutions of the equation $(C \| X) \leq_{\S} A$.

Theorem 2 : Given two deterministic *IOAs* A and C, and given a set In such that $(I_A \backslash I_C) \cup (O_C \backslash O_A) \subseteq In \subseteq I_A \cup O_C$, an *IOA* B, with $I_B=In$ and $O_B=O_{Sol_{\S}}$, is a solution of the equation $(C \| X) \leq_{\S} A$ iff $B \leq_{\S} Sol_{\S}$.

Example : Figure 11 shows some other solutions for the example above.

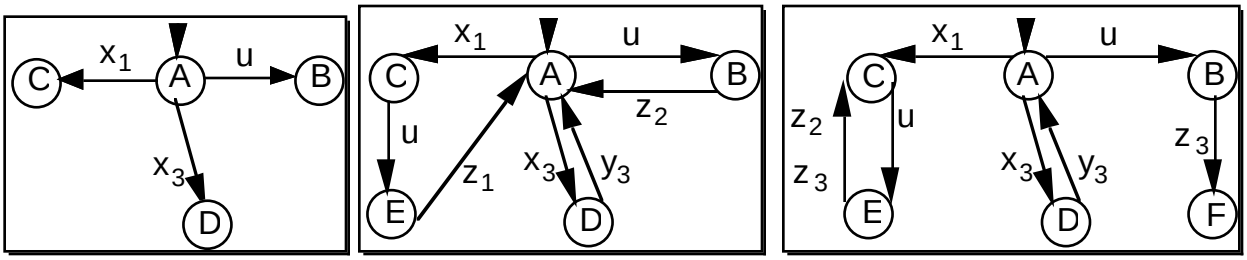

Figure 11 : Some safe realizations of $Sol_{\S}$

6 The solution for the subtype criterion

As discussed in Section 4.2, we are looking for a solution to the submodule construction problem when the conforming implementation relation is used. More formally, for a given *IOA* C and an *IOAWOCT* A and a set In such that $(I_A \backslash I_C) \cup (O_C \backslash O_A) \subseteq In \subseteq I_A \cup O_C$ we want to find an *IOA* X such that $(C \| X) \leq_{\text{conf}} A$ and $I_X=In$. We describe in Subsection 6.1 an algorithm which returns a generic solution in the form of an *IOAWOCT*. A generic solution is a solution from which all the solutions can be derived. We show in Subsection 6.2 that all the solutions to the equation $(C \| X) \leq_{\text{conf}} A$ under the constraint $I_X=In$ are the conforming implementations of the generic solution. We give in the next paragraph an algorithm which constructs the generic solution if it exists. As input, the algorithm requires two deterministic *IOA* C and $Sol_{\S}$, and an *IOAWOCT* A. To simplify the algorithm, we consider only the case where for each state s of IOA_A, $OCT_A(s)=\text{out}(s)$ and $MT_A(s) \subseteq 2^{out(s)}$. We denote this restricted class *IOAWO*, for I/O automaton with options. This means that each output of s is optional and at least one output of each element of $MT_A(s)$ is a mandatory output.

6.1 The Algorithm

We use the *IOA* $Sol_{\S}$ which represent the solution to the equation $(C \| X) \leq_{\S} IOA_A$ (see Section 5) as a starting point. Any solution of $(C \| X) \leq_{\text{conf}} A$ is trace included in $Sol_{\S}$. The main idea of our approach is to remove from $Sol_{\S}$ all those traces which, when combined with traces of the context C in the environment $\tilde{A}$, may cause a non-conforming behavior with respect to the mandatory traces MT_A or the optional complete traces OCT_A. This will allow us to capture the set of the permissible traces of the component to be designed in the form of an *IOAWOCT Sol*, called the generic solution (if this set is not empty). A permissible trace is a trace of an *IOA* which is a solution of $(C \| X) \leq_{\text{conf}} A$. The algorithm proceeds in six steps. In Step 1, we construct the composition of C, $Sol_{\S}$ and $\widetilde{IOA_A}$ then we associate to each state of the composition the mandatory constraints associated to the corresponding state of IOA_A. In Step 2, we process the states from which a constraint is not satisfied and the non-controlable transitions which lead to the *Fail* state, i.e. a transition labelled with an action which is not an output of the component to be designed. In Step 3, we associate with each constrained state *c* an *IOA* whose set of traces is equal to the subset of $Tr_{IOA_R}(c)$ which is the union of the traces that contain no external action and the traces that contain a single external action (an output action) as their last element. This *IOA* will allow us in the next steps to characterize the mandatory and the complete traces of the solution we are looking for. In Step 4, we hide the actions which are not in the alphabet of the solution and we associate to each state the corresponding constraints. In Step 5, we remove recursively all non-conforming traces. Finally, in Step 6 we construct the generic solution in the form of an *IOAWOCT Sol*.

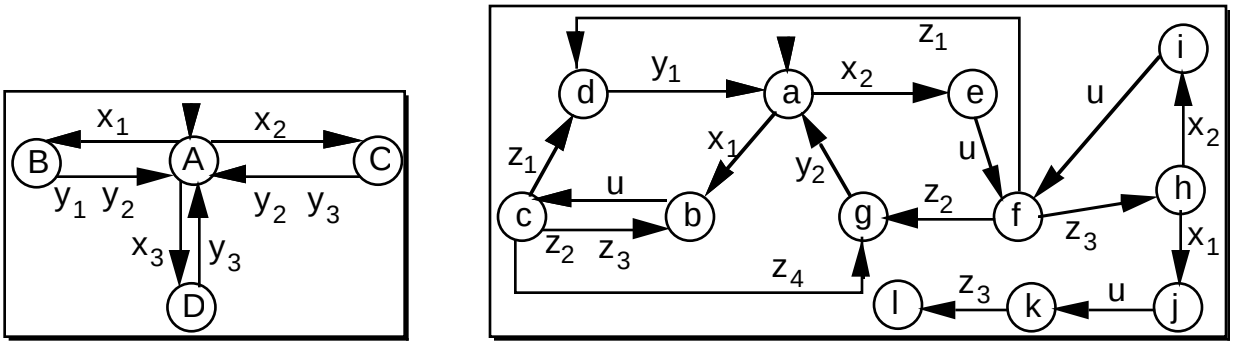

Figure 12 : The *IOA* IOA_A and the *IOA C*.

To illustrate the work done in each step of the algorithm, we use the following example. We consider for the context the *IOA C* shown in Figure 12 with $I_C=\{x_1, x_2, z_1, z_2, z_3, z_4\}$, $O_C=\{u, y_1, y_2\}$, and for the whole system the *IOAWO A* corresponding to the IOA_A of Figure 12, with $I_{IOA_A}=\{x_1, x_2, x_3\}$, $O_{IOA_A}=\{y_1, y_2, y_3\}$, and
$MT_A=\{(A, \emptyset), (B, \{\{y_1\}\}), (C, \{\{y_2\}\}), (D, \{\{y_3\}\})\}$, and
$OCT_A=\{(A, \emptyset), (B, \{\{y_1, y_2\}\}), (C, \{\{y_2, y_3\}\}), (D, \{\{y_3\}\})\}$, and $In=\{x_1, x_3, u\}$.

Since the algorithm requires as input the solution for the safe realization relation, we use first Algorithm 1 to obtain the *IOA* $Sol_{\S}$ shown in Figure 13.

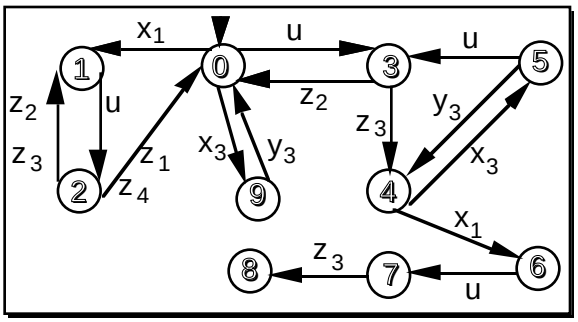

Figure 13 : The *IOA* $Sol_{\S}$.

Algorithm 2

Input : The specification of the context in the form of an *IOA C*, the specification of the system behavior in the form of an *IOAWO A* and a set of inputs *In***.**

Output : An *IOAWOCT Sol* with $I_{IOA_{Sol}}=In$ which satisfies the equation $(C||IOA_{Sol})\leq_{conf}A$ if a solution exists.

Step 1. In this step, we construct an *IOAWOCT R* such that IOA_R is the composition of *C*, $Sol_{\S}$ and $\widetilde{IOA_A}$, and we initialize the sets $MT_R(c)$ and $OCT_R(c)$ with the empty set for each state *c* of IOA_R. If $MT_A(s_{oIOA_A})$ is not empty, we assign it to $MT_R(s_{oR})$. For each state $c=(s_1, s_2, s_3)$ in $S_{IOA_R}\backslash\{s_{oIOA_R}\}$ such that $entering(c)\cap(I_{IOA_A}\cup O_{IOA_A})$ and $MT_A(s_3)$ are not empty, we assign to $MT_R(c)$ the set $MT_A(s_3)$. This means that for a state *c* of IOA_R where $MT_R(c)$ is not empty a progress property must hold. The complexity of this step is in the worst case polynomial in the number of states of IOA_A and *C*, and the number of elements in the alphabet of $IOA_A||C$ (i.e. $O(nmr^2)$ where n=number of states of *A*, m=number of states of *C* and r=number of element in the alphabet of $A||C$).

$IOA_R=C||Sol_{\S}||\widetilde{IOA_A}$;

FOR each state $c=(s_1, s_2, s_3)$ in S_{IOA_R} **DO** $MT_R(c):=\emptyset$; $OCT_R(c):=\emptyset$;

IF $MT_A(s_{oIOA_A})\neq\emptyset$ **THEN** $MT_R(s_{oIOA_R}):=MT_A(s_{oIOA_A})$;

FOR each state $c=(s_1, s_2, s_3)$ in $S_{IOA_R}\backslash\{s_{oIOA_R}\}$ such that $entering(c)\cap(I_{IOA_A}\cup O_{IOA_A})\neq\emptyset$ and $MT_A(s_3)\neq\emptyset$ **DO** $MT_R(c):=MT_A(s_3)$;

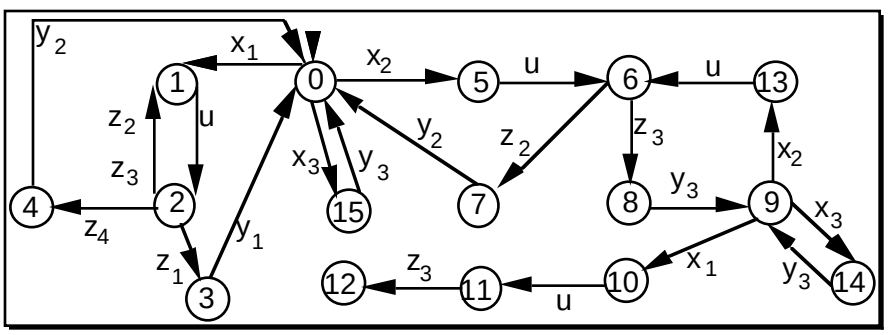

Figure 14 : The *IOA* IOA_R

For the example, we obtain at the end of Step 1 the *IOA* IOA_R shown in Figure 14 and the set of constraints MT_R={(0, Ø), (1, {{y_1}}), (2, Ø), (3, Ø), (4, Ø), (5, {{y_2}}), (6, Ø), (7, Ø), (8, Ø), (9, Ø), (10, {{y_1}}), (11, Ø), (12, Ø), (13, {{y_2}}), (14, {{y_3}}), (15, {{y_3}})}.

Step 2. In this step, we remove from IOA_R some non-conforming traces. For a state c of IOA_R, a state which can be reached from c with an internal trace is said to be an internal successor of c; the set containing all external outputs of its internal successors is denoted ext-out-after(c); the state c is said to be a silent state if ext-out-after(c) is empty. For a state c of IOA_R where $MT_R(c)$ is not empty, if ext-out-after(c) does not meet the constraints imposed by $MT_R(c)$ then we return "NO SOLUTION" in the case where c is the initial state, otherwise all the transitions s- t→ c labelled with an external action will be replaced by s- t→ $Fail_R$ and we replace IOA_R by its initially connected component; else for each silent internal successor c' of c we return "NO SOLUTION" in the case where c' is the initial state, otherwise we replace each transition s- t→ c' by s- t→ $Fail_R$ and we replace IOA_R by its connected component. If the intersection of *entering*($Fail_R$) and ($I_{IOA_A} \cup O_C$) is not empty, for each transition s- t→ $Fail_R$ with t in the intersection, we return "NO SOLUTION" in the case where s is the initial state, otherwise we replace each transition s'- t→ s by s'- t→ $Fail_R$ and we replace IOA_R by its connected component. This processing is repeated until the *IOA* IOA_R can not be updated. At the end of this step, any constrained state satisfies its constraints and may have only $Fail_R$ as silent internal successor; moreover all the transitions leading to $Fail_R$ are labelled with controllable actions, i.e. outputs of the component to be designed. To achieve this, we construct two sets : the set *NonSilentStates* containing those states of IOA_R where an external output occurs; and the set *ConstrainedStates* which initially contains all states c of IOA_R where $MT_R(c)$ is not empty. We repeat the following processing until the set *ConstrainedStates* is empty : we remove from *ConstrainedStates* an element c, and we determine the set of its internal successors, denoted *Succint*(c); if there exists an element in $MT_R(c)$ such that its intersection with ext-out-after(c) is empty, then we return "NO SOLUTION" in the case where c is the initial state, otherwise we replace each transition s- t→ c by s- t→ $Fail_R$ for each external action t and we assign the empty set to $MT_R(c)$ and we replace IOA_R by its connected component; now, if all the constraints imposed by $MT_R(c)$ are satisfied, i.e. $MT_R(c)$ is not empty, we turn our attention to the silent states in *Succint*(c); for each such state c', we return "NO SOLUTION" in the case where it is the initial state, otherwise we replace each transition s- t→ c' by s- t→ $Fail_R$ and we replace IOA_R by its connected component; if the intersection of *entering*($Fail_R$) and ($I_{IOA_A} \cup O_C$) is not empty, for each transition s- t→ $Fail_R$ with t in the intersection, we return "NO SOLUTION" in the case where s is the initial state, otherwise we replace each transition s'- t→ s by s'- t→ $Fail_R$ and we replace IOA_R by its connected component, then we update the set *ConstrainedStates* by assigning to it the set containing each state of IOA_R where $MT_R(c)$ is not

empty. The complexity of this step in the worst case is polynomial in the number of states of IOA_A and C, and the number of elements in the alphabet of $\widetilde{IOA_A} \| C$ (i.e. $O(n^3m^3r^3)$ where n=number of states of A, m=number of states of C and r=number of element in the alphabet of $A\|C$).

Procedure *Remove-some-non-conforming-traces*

IntActions :=$(I_C \cup O_C) \setminus (I_{IOA_A} \cup O_{IOA_A})$;
NonSilentStates=$\{s \in S_{IOA_R} \mid out(s) \cap O_{IOA_A} \neq \emptyset\}$;
ConstrainedStates := $\{c \in S_{IOA_R} \mid MT_R(c) \neq \emptyset\}$;
WHILE *ConstrainedStates* $\neq \emptyset$ **DO**
c **:=** an element of *ConstrainedStates* ;
ConstrainedStates := *ConstrainedStates* $\setminus \{c\}$;
Succint$(c)=\{c' \in S_R \setminus \{Fail_R\} \mid \exists \sigma \in$ IntActions* such that $c'=c_\sigma\}$;
ext-out-after$(c) := \cup_{c' \in Succint(c)} out(c') \cap O_{IOA_A}$;
TempMT :=$MT_R(c)$;
WHILE TempMT$\neq\emptyset$ **DO**
Y **:=** an element of TempMT;
TempMT := TempMT$\setminus\{Y\}$;
IF $Y \cap$*ext-out-after*$(c)=\emptyset$ **THEN** // the constraints for state c are not satisfied
IF $c=s_{0IOA_R}$ **THEN** return "NO SOLUTION"; **STOP**;
ELSE
replace *s*- $t \rightarrow c$ by *s*- $t \rightarrow Fail_R$ for each *t* in *entering*$(c) \cap (I_{IOA_A} \cup O_{IOA_A})$;
$MT_R(c):=\emptyset$; TempMT:=$\emptyset$; $IOA_R:=CC(IOA_R)$;
ENDWHILE TempMT$\neq\emptyset$
IF $MT_R(c)\neq\emptyset$ **THEN**
WHILE *Succint*$(c)\neq\emptyset$ **DO**
c' **:=** an element of *Succint*(c);
Succint(c) := *Succint*$(c)\setminus\{c'\}$;
IF *Succint*$(c') \cap$*NonSilentStates*$=\emptyset$ **THEN**
IF $c'=s_{0IOA_R}$ **THEN** return "NO SOLUTION"; **STOP**;
ELSE
replace *s*- $t \rightarrow c'$ by *s*- $t \rightarrow Fail_R$ for each *t* in *entering*(c');
$IOA_R:=CC(IOA_R)$; *Succint*(c) := *Succint*$(c) \cap S_{IOA_R}$;
ENDWHILE *Succint*$(c)\neq\emptyset$
IF *entering*$(Fail_R) \cap (I_{IOA_A} \cup O_C) \neq \emptyset$ **THEN**
TempTrans :=$\{(s, t, Fail_R) \in T_{IOA_R} \mid t \in (I_{IOA_A} \cup O_C)\}$;
WHILE TempTrans$\neq\emptyset$ **DO**

(s', t', $Fail_R$) **:=** an element of TempTrans;

IF $s'=s_{0IOA_R}$ **THEN** return "NO SOLUTION"; **STOP**;

ELSE

replace s'' - t'' -> s' by s'' - t'' -> $Fail_R$ for each t'' in *entering*(s');

IOA_R:=$CC(IOA_R)$;

TempTrans :={(s, t, $Fail_R$)$\in T_{IOA_R}$ | $t \in (I_{IOA_A} \cup O_C)$};

ENDWHILE TempTrans≠Ø

ConstrainedStates := {$c \in S_{IOA_R}$ | $MT_R(c)$≠Ø};

ENDWHILE *ConstrainedStates* ≠Ø

For our example, the work done in Step 2 is as follow. After state 10, the required external output y_2 is not possible, then this state is replaced by $Fail_R$. Since now there exists a transition labelled with a non-controllable action leading to $Fail_R$ (9-x_1->$Fail_R$) the state 9 is replaced by $Fail_R$. The obtained connected component is shown in Figure 15, and the updated set of constraints is MT_R={(0, Ø), (1, {{y_1}}), (2, Ø), (3, Ø), (4, Ø), (5, {{y_2}}), (6, Ø), (7, Ø), (8, Ø), (15, {{y_3}})}.

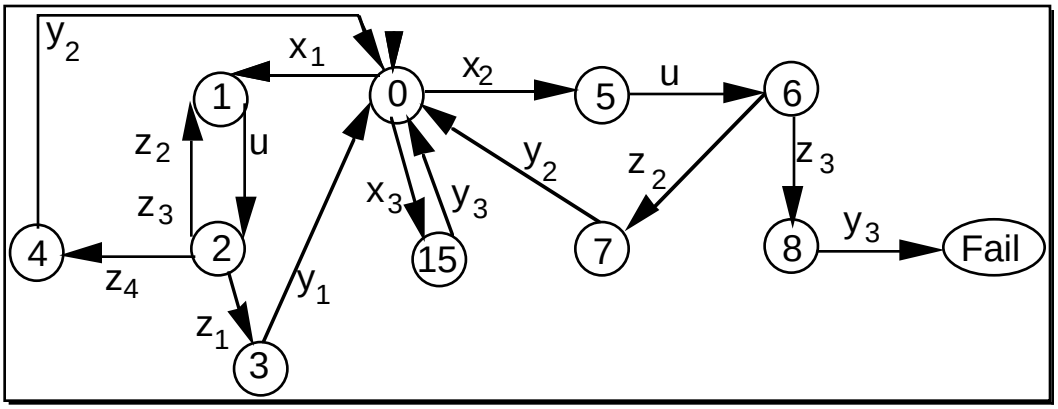

Figure 15 : the obtained *IOA* IOA_R at the end of Step 2

Step 3. In this step, we associate to each state c of IOA_R where $MT_R(c)$ is not empty, an *IOA* *CONST*(c) whose set of traces is equal to the subset of $Tr_{IOA_R}(c)$ which is the union of the traces that contain no external action and the traces that contain a single external action (an output action) as their last element. This *IOA* will allow us in the following steps of the algorithm to characterize the mandatory and the complete traces of the solution we are looking for. To be able to compose *CONST*(c) with the *IOA* obtained from IOA_R after hiding some actions and to preserve all the possible external outputs after c in *CONST*(c), we assign the empty set to $O_{CONST(c)}$ and the set containing all the actions present in IOA_R to $I_{CONST(c)}$. The complexity in the worst case of this step is polynomial in the number of states of IOA_R (i.e. $O(n^2m^2r)$ where n=number of states of A, m=number of states of C and r=number of element in the alphabet of $A||C$).

Procedure *Updating-information*

FOR each state c in $S_{IOA_R}\backslash\{Fail_R\}$ such that $MT_R(c)\neq\emptyset$ **DO**

IntActions :=$(I_C\cup O_C)\backslash(I_{IOA_A}\cup O_{IOA_A})$;

TempStates := $\{c'\in S_{IOA_R} \mid \exists\sigma\in$ IntActions* such that $c'=c_\sigma\}$;

TempTrans := $\{(s, t, s')\in Tr_{IOA_R} \mid s, s'\in$ TempStates and $t\in$ IntActions$\}$;

TempTransFinal := $\{(s, t, Final) \mid s\in$ TempStates and $t\in leaving(s)\cap O_{IOA_A}$ and $s_t\neq Fail_R\}$;

TempTransFail := $\{(s, t, Fail_R)\in Tr_{IOA_R} \mid s\in$ TempStates and $t\in O_{IOA_A}\}$;

IF TempTransFinal$\neq\emptyset$ **THEN** TempStates := TempStates$\cup\{Final\}$;

IF TempTransFail$\neq\emptyset$ **THEN** TempStates := TempStates$\cup\{Fail_R\}$;

TempTrans := TempTrans$\cup$TempTransFinal$\cup$TempTransFail;

$CONST(c)$:=(TempStates, $I_{IOA_R}\cup O_{IOA_R}$, $\emptyset$, TempTrans, c);

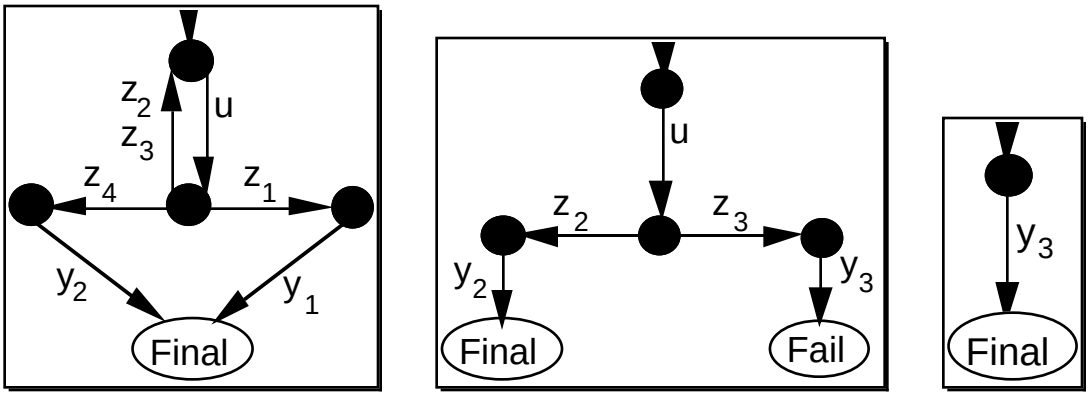

Figure 16 : The *IOA*s *CONST*(1), *CONST*(5) and *CONST*(15).

Step 4. In this step, we hide the actions that are in $(I_{IOA_A}\cup O_C)\backslash In$ and construct $H_{(I_{IOA_A}\cup O_C)\backslash In}(IOA_R)$. Since a state of the obtained automaton, denoted R_1, is a subset of states of IOA_R, we declare any state of R_1 which contains $Fail_R$ as $Fail_{R_1}$. Moreover, we assign to TempConstraint(s_{oR_1}) the set of pairs ($MT_R(c)$, $CONST(c)$) for c in s_{oR_1} such that $MT_R(c)$ is not empty. For each state s in $S_{R_1}\backslash\{s_{oR_1}, Fail_{R_1}\}$, if $entering(s)$ contains some external actions, we assign to TempConstraint(s) the set of pairs ($MT_R(c)$, $CONST(c)$) for c in s such that $MT_R(c)$ is not empty, otherwise we assign to TempConstraint(s)) the set of pairs ($MT_R(c)$, $CONST(c)$) for c in s such that $MT_R(c)$ is not empty and c is reachable in IOA_R from c'' in s with an external action in $(I_{IOA_A}\cup O_C)\backslash I_n$, i.e. there exists s' in R_1, c' in s' and c'' in s with $s=s'_t$, $c''=c'_{t.\sigma}$ in IOA_R and $c=c''_u$ for σ in $((I_{IOA_A}\cup O_C)\backslash I_n)^*$ and u in $(I_{IOA_A}\cup O_C)\backslash I_n$. The complexity in the worst case of this step is exponential in the number of states of IOA_R. We denote by ε the empty word.

Procedure *Hide-actions-in-*$(I_{IOA_A}\cup O_C)\backslash In$

ConstrainedStates := $\{c\in S_{IOA_R} \mid MT_R(c)\neq\emptyset\}$;

S_{R_1}:=$\emptyset$; T_{R_1}:=$\emptyset$; s_{oR_1}:=$\{c\in S_{IOA_R} \mid \exists\sigma\in((I_{IOA_A}\cup O_C)\backslash In)^*$ such that $c=(s_{oIOA_R})_\sigma\}$;

S_{R_1}:=$S_{R_1}\cup\{s_{oR_1}\}$; TempStates := $\{s_{oR_1}\}$; L :=$In\cup(I_C\backslash I_{IOA_A})\cup(O_{IOA_A}\backslash O_C)$;
TempConstraint(s_{oR_1}) :={($MT_R(c)$, $CONST(c)$)| $c\in s_{oR_1}\cap ConstrainedStates$}

WHILE TempStates≠Ø **DO**

s **:=** an element of TempStates;

TempStates:=TempStates\{s};

FOR each t in L **DO**

FOR each c in s **DO**

$Succh(c)=\{c'\in S_{IOA_R}|\ \exists\sigma\in((I_{IOA_A}\cup O_C)\backslash I_n)^+$ such that $c'=c_{t.\sigma}\}$;

s':=$\cup_{c\in s} Succh(c)$;

s'':=$s'\cup\{c_t \mid c\in s\}$;

IF $s''\neq\emptyset$ **THEN**

IF $Fail_R\in s''$ **THEN** T_{R_1}:=$T_{R_1}\cup\{(s, t, Fail_{R_1})\}$; S_{R_1}:=$S_{R_1}\cup\{Fail_{R_1}\}$;

ELSE

T_{R_1}:=$T_{R_1}\cup\{(s, t, s'')\}$;

IF $s''\notin S_{R_1}$ **THEN**

S_{R_1}:=$S_{R_1}\cup\{s''\}$; TempStates:=TempStates$\cup\{s''\}$;

TempConstraint(s''):=Ø;

IF $t\in((O_C\backslash O_{IOA_A})\cup(I_C\backslash I_{IOA_A}))$ **THEN**

TempConstraint(s''):={($MT_R(c)$, $CONST(c)$)| $c\in s'\cap ConstrainedStates$}
$\cup$TempConstraint(s'');

ELSE

TempConstraint(s''):={($MT_R(c)$, $CONST(c)$)| $c\in s''\cap ConstrainedStates$};

ENDWHILE TempStates≠Ø

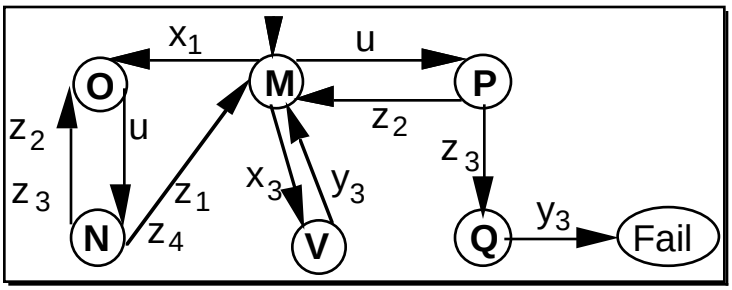

Figure 17 : The IOA R_1

At the end of Step 4 the obtained *IOA* R_1 for our example is shown in Figure 17 and we have the following constraints :
TempConstraint(**M**)={({{y_2}}, $CONST(5)$)}, TempConstraint(**O**)={({{y_1}}, $CONST(1)$)} and TempConstraint(**V**)={({{y_3}}, $CONST(15)$)}.

Step 5. In this step, we recursively remove the traces which lead to $Fail_{R_1}$. To achieve this, we initialize the set *ConstrainedStates* with the set containing each state c of R_1 where TempConstraint(c) is not empty, and we remove all the transitions which lead to $Fail_{R_1}$. We repeat the following processing until the set *ConstrainedStates* is empty : we remove from *ConstrainedStates* an element s, and we assign TempConstraint(s) to a temporary set Temp; while Temp is not empty, we remove from it an element ($MT_R(c)$, *CONST*(c)); we assign to *CONST*(c) its composition with the *IOA* obtained from R_1 where s is considered as the initial state; if there exists an element in $MT_R(c)$ such that its intersection with the set of external outputs present in *CONST*(c) is empty, then we return "NO SOLUTION" in the case where s is the initial state of R_1, otherwise we replace each transition s'-t→s by s'-t→$Fail_{R_1}$ and we replace R_1 by its initially connected component and we assign to Temp the empty set; if the intersection of *entering*($Fail_{R_1}$) and *In* is not empty, for each transition s''-t→$Fail_{R_1}$ with t in the intersection, we return "NO SOLUTION" in the case where s'' is the initial state, otherwise we replace each transition s'-t→s'' by s'-t→$Fail_{R_1}$ and we replace R_1 by its initially connected component. Then we redefine the set *ConstrainedStates* by assigning to it the set containing each state c of R_1 where TempConstraint(c) is not empty and we remove all the transitions which lead to $Fail_{R_1}$; if s remains in S_{R_1}, we turn our attention to those states in *CONST*(c), different from c and *Final*, from which we can not reach a state where an external output occurs; for each such state c' in $S_{CONST(c)}$, we replace each transition c''- t→ c' by c''- t→ $Fail_{CONST(c)}$; then we replace *CONST*(c) by its initially connected component; now for each transition c''- t→ $Fail_{CONST(c)}$, we determine a trace σ such that $c_\sigma = c''$ and we replace the transition s_σ-t→s' by s_σ-t→$Fail_{R_1}$; if the intersection of *entering*($Fail_{R_1}$) and *In* is not empty, for each transition s''- t→ $Fail_{R_1}$ with t in the intersection, we return "NO SOLUTION" in the case where s'' is the initial state, otherwise we replace each transition s'- t→ s'' by s'- t→ $Fail_{R_1}$ and we replace R_1 by its initially connected component. Then we redefine the set *ConstrainedStates* by assigning to it the set containing each state of R_1 where TempConstraint(c) is not empty and we remove all the transitions which lead to $Fail_{R_1}$. The complexity in the worst case of this step is polynomial in the number of states of R_1.

Procedure *Remove-Fail-state*

ConstrainedStates := $\{c \in S_{R_1} \mid \text{TempConstraint}(c) \neq \emptyset\}$;
I_{R_1} := *In* ; O_{R_1} := $(I_C \setminus I_{IOA_A}) \cup (O_{IOA_A} \setminus O_C)$;
T_{R_1} := $T_{R_1} \setminus \{(c, t, Fail_{R_1}) \in T_{R_1}\}$;
WHILE *ConstrainedStates* ≠∅ **DO**
  s **:=** an element of *ConstrainedStates* ;
  ConstrainedStates := *ConstrainedStates* \{s};
  Temp := TempConstraint(s);

WHILE Temp ≠Ø **DO**

($MT_R(c)$, $CONST(c)$) := an element of Temp;

Temp :=Temp\{($MT_R(c)$, $CONST(c)$)};

$CONST(c)$:= $CONST(c)$|| (S_{R_1}, I_{R_1}, O_{R_1}, T_{R_1}, s);

Temp1 :=$\cup_{c' \in S_{CONST(c)}}$ $out(c') \cap O_{IOA_A}$;

Temp2 :=$MT_R(c)$;

WHILE Temp2≠Ø **DO**

Y **:=** an element of Temp2;

Temp2 := Temp2\{Y};

IF $Y \cap$Temp1=Ø **THEN**

IF s=s_{oR_1} **THEN** return "NO SOLUTION"; **STOP**;

ELSE

replace s'- $t \rightarrow s$ by s'- $t \rightarrow Fail_{R_1}$ for each t in $entering(s)$;

R_1:=$CC(R_1)$; Temp:=Ø; Temp2:=Ø;

IF $entering(Fail_{R_1}) \cap In \neq \emptyset$ **THEN**

Temp3 :={(s'', t, $Fail_{R_1}$)$\in T_{R_1}$ | $t \in In$};

WHILE Temp3≠Ø **DO**

(s'', t, $Fail_R$) **:=** an element of Temp3;

IF s''=s_{oR} **THEN** return "NO SOLUTION"; **STOP**;

ELSE

replace s' - $t' \rightarrow s''$ by s' - $t' \rightarrow Fail_{R_1}$ for each t' in $entering(s'')$;

R_1:=$CC(R_1)$;

Temp3 :={(s', t', $Fail_{R_1}$)$\in T_{R_1}$ | $t' \in In$};

ENDWHILE Temp3≠Ø

$ConstrainedStates$:= {$p \in S_R$ | TempConstraint(p)≠Ø};

T_{R_1} := T_{R_1}\{(p, t, $Fail_{R_1}$)$\in T_{R_1}$};

ENDWHILE Temp2≠Ø

IF $s \in S_{R_1}$ **THEN**

$NonSilentStates$={$s' \in S_{CONST(c)}$ | $out(s') \cap O_{IOA_A} \neq \emptyset$};

Temp4=$S_{CONST(c)}$\{c, $Final$};

WHILE Temp4≠Ø **DO**

c' **:=** an element of Temp4;

Temp4 := Temp4\{c'};

IF$Succint(c') \cap NonSilentStates$=Ø **THEN**

$S_{CONST(c)}$:= $S_{CONST(c)} \cup \{Fail_{CONST(c)}\}$;

replace c''- $t \rightarrow c'$ by c''- $t \rightarrow Fail_{CONST(c)}$ for each t in $entering(c')$;

CONST(*c*):=*CC*(*CONST*(*c*)); Temp4:=Temp4∩$S_{CONST(c)}$;

ENDWHILE Temp4≠Ø

IF $Fail_{CONST(c)} \in S_{CONST(c)}$ **THEN**

Temp5=Ø;

FOR each transition c'- $t \rightarrow Fail_{CONST(c)}$ **DO**

σ_t := the shortest trace in $Tr_{CONST(c)}$ such that $c_{\sigma_t} = c'$;

Temp5 := Temp5∪{σ_t };

WHILE Temp5≠Ø **DO**

σ_t **:=** an element of Temp5;

Temp5 := Temp5\{σ_t };

replace s_{σ_t} - $t \rightarrow s'$ by s_{σ_t} - $t \rightarrow Fail_{R_1}$;

ENDWHILE Temp5≠Ø

IF *entering*($Fail_{R_1}$)∩*In* ≠Ø **THEN**

Temp6 :={(s'', t, $Fail_{R_1}$)∈ T_{R_1} | $t \in In$};

WHILE Temp6≠Ø **DO**

(s'', t, $Fail_R$) **:=** an element of Temp6;

IF $s''=s_{oR}$ **THEN** return "NO SOLUTION"; **STOP**;

ELSE

replace s' - $t' \rightarrow s''$ by s' - $t' \rightarrow Fail_{R_1}$ for each t' in *entering*(s'');

R_1:=*CC*(R_1);

Temp6 :={(s', t', $Fail_{R_1}$)∈ T_{R_1} | $t' \in In$};

ENDWHILE Temp6≠Ø

Temp:=Ø;

ConstrainedStates := {$p \in S_R$ | TempConstraint(p)≠Ø};

T_{R_1} := T_{R_1}\{(p, t, $Fail_{R_1}$)∈ T_{R_1}};

ENDWHILE Temp≠Ø

ENDWHILE *ConstrainedStates* ≠Ø

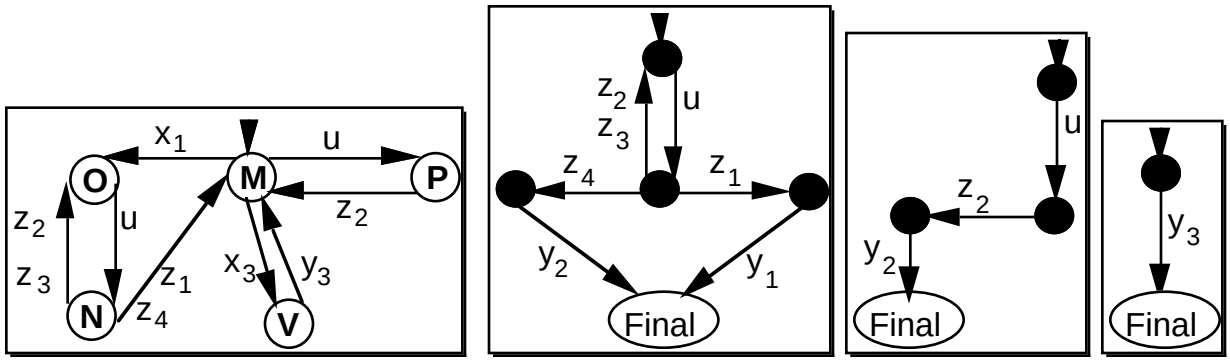

Figure 18 : The *IOA*s R_1, *CONST*(1), *CONST*(5) and *CONST*(15) after Step 5.

For our example, the work done in Step 5 is as follow. We first remove from R_1 the transition **Q**-y_3->$Fail_{R_1}$. After updating the *IOA CONST*(5), the transition labelled by z_3 leads to a state from which no external output can be reached. Then the trace uz_3 is used to replace the state **Q** of R_1 by $Fail_{R_1}$. After removing the transition **P**-z_3->$Fail_{R_1}$, all the constraints are satisfied and the obtained *IOA*s R_1, *CONST*(1), *CONST*(5) and *CONST*(15) are shown in Figure 18.

Step 6. In this step, we construct the generic solution in the form of the *IOAWOCT Sol* . The *IOA* IOA_{Sol} is equal to R_1. To obtain the sets MT_{Sol} and OCT_{Sol} , we initialize the sets $MT_{Sol}(s)$ and $OCT_{Sol}(s)$ with the empty set for each state s in $S_{IOA_{Sol}}$. If TempConstraint(s) is not empty then for each element ($MT_R(c)$, *CONST*(c)), we assign to a temporary set Temp2 the set of traces leading to the state *Final* in *CONST*(c), moreover for each element Y in $MT_R(c)$ if its intersection with the set of external outputs occurring in c is empty then we add to $MT_{Sol}(s)$ the projection over the alphabet of IOA_{Sol} of the subset of Temp2 containing traces which end with an element in Y; we assign to a temporary set Temp3 the projection over the alphabet of IOA_{Sol} of Temp2, and we remove from Temp3 the empty word and the elements which are not in $OCT_{Sol}(s)$ and which are proper prefixes of elements not in $OCT_{Sol}(s)$, and from $OCT_{Sol}(s)$ we remove the elements which are not in Temp3 and which are proper prefixes of elements not in Temp3. Then we assign to $OCT_{Sol}(s)$ its union with Temp3. Due to cycles labelled with internal actions, the sets $OCT_{Sol}(c)$ and $MT_{Sol}(c)$ could be infinite. To represent such infinite sets we represent them as finite sets of regular expressions or as finite sets of *IOA*s. The complexity in the worst case of this step is polynomial in the number of states of R_1.

Construction of the *IOAWOCT Sol*

IOA_{Sol} := R_1;
OCT_{Sol} :=Ø;
MT_{Sol} :=Ø;
L :=$In \cup (I_C \setminus I_{IOA_A}) \cup (O_{IOA_A} \setminus O_C)$;
FOR each state s is in $S_{IOA_{Sol}}$ **DO**
  $OCT_{Sol}(s)$:=Ø;
  $MT_{Sol}(s)$:=Ø;
  WHILE TempConstraint(s)≠Ø **DO**
    ($MT_R(c)$, *CONST*(c)) := an element of TempConstraint(s);
    TempConstraint(s) :=TempConstraint(s)\{($MT_R(c)$, *CONST*(c))};
    Temp1 := $out(c) \cap O_C$;
    Temp2 :={$\sigma \in Tr_{CONST(c)}(c) \mid c_\sigma = Final$};
    WHILE $MT_R(c)$≠Ø **DO**
      Y **:=** an element in $MT_R(c)$;

$MT_R(c) := MT_R(c)\backslash\{Y\}$;

IF $Y \cap$Temp1$=\emptyset$ **THEN**

$MT_{Sol}(s) := MT_{Sol}(s) \cup (Pr_L(\{\sigma \in$ Temp2 $| Pr_{O_{IOA_A}}\{\sigma) \in Y\})$;

ENDWHILE $MT_R(c) \neq \emptyset$

Temp3 $:= Pr_L$(Temp2)$\backslash\{\varepsilon\}$;

$OCT_{Sol}(s) := OCT_{Sol}(s)\backslash\{\sigma \in OCT_{Sol}(s)$ and $\sigma \notin$ Temp3 and σ is a proper prefix of an element in Temp3$\}$;

Temp3 $:=$Temp3$\backslash\{\sigma \in$ Temp3 and $\sigma \notin OCT_{Sol}(s)$ and σ is a proper prefix of an element in $OCT_{Sol}(s)\}$;

$OCT_{Sol}(s) := OCT_{Sol}(s) \cup$Temp3;

ENDWHILE TempConstraint$(s) \neq \emptyset$

$MT_{Sol}(s) := \{Y \cap OCT_{Sol}(s) \mid Y \in MT_{Sol}(s)\}$;

$MT_{Sol}(s) := MT_{Sol}(s)\backslash\{Y \in MT_{Sol}(s) \mid$ there exits $Y' \in MT_{Sol}(s)$ with $Y' \subset Y\}$;

$OCT_{Sol}(s) := OCT_{Sol}(s)\backslash\{\sigma \in OCT_{Sol}(s) \mid$ length of σ is equal to 1$\}$;

$OCT_{Sol} := OCT_{Sol} \cup \{(s, OCT_{Sol}(s))\}$;

$MT_{Sol} := MT_{Sol} \cup \{(s, MT_{Sol}(s))\}$;

ENDFOR

$Sol := (IOA_{Sol}, MT_{Sol}, OCT_{Sol})$;

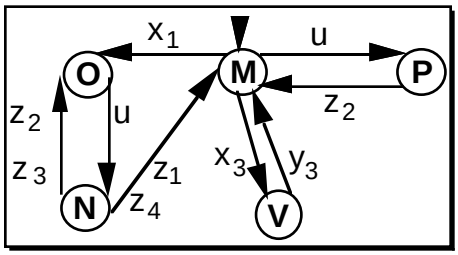

Figure 19 : The IOA IOA_{Sol}.

For the obtained *IOAWOCT Sol*, IOA_{Sol} is shown in Figure 19, and $MT_{Sol} = \{(\mathbf{M}, \{\{uz_2\}\}), (\mathbf{O}, \{\{u(z_2u + z_3u)^*z_1\}\}), (\mathbf{N}, \emptyset), (\mathbf{P}, \emptyset), (\mathbf{V}, \{\{y_3\}\})\}$, and $OCT_{Sol} = \{(\mathbf{M}, \{uz_2\}), (\mathbf{O}, \{u(z_2u + z_3u)^*z_1, u(z_2u + z_3u)^*z_4\}), (\mathbf{N}, \emptyset), (\mathbf{P}, \emptyset), (\mathbf{V}, \emptyset)\}$.

Theorem 3 : Given a deterministic *IOA C*, a deterministic *IOAWO A*, and a given input set *In* such that $(I_{IOA_A} \backslash I_C) \cup (O_C \backslash O_{IOA_A}) \subseteq In \subseteq I_{IOA_A} \cup O_C$, if Algorithm 2 produces an *IOAWOCT Sol* then $(C||IOA_{Sol}) \leq_{conf} A$, else there is no solution for $(C||X) \leq_{conf} A$ with the specified set of inputs *In*.

6.2 The set of solutions

The solution obtained by the algorithm above is a generic one, which means that we can derive from it the set of solutions of the equation $(C||X) \leq_{conf} A$.

Theorem 4 : Given a deterministic *IOA C*, a deterministic *IOAWO A*, and an input set *In* such that $(I_{IOA_A} \setminus I_C) \cup (O_C \setminus O_{IOA_A}) \subseteq In \subseteq I_{IOA_A} \cup O_C$, if Algorithm 2 produces an *IOAWOCT Sol* then for any *IOA B*, with $I_B = In$ and $O_B = O_{Sol_S}$, the following propositions are equivalent :

i - $(C||B) \leq_{conf} A$,

ii - $B \leq_{conf} Sol$.

Example : Figure 20 shows some solution *IOAs* which are subtypes of the generic solution *Sol* shown in Figure 19.

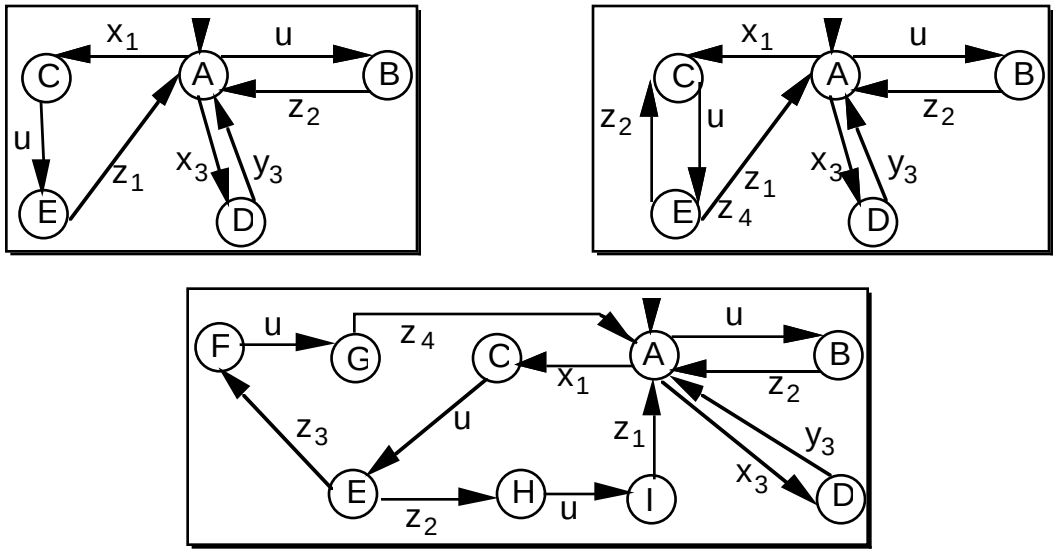

Figure 20 : Some subtypes of *Sol*.

7 The case of input/output Finite State Machines

We turn our attention now to see how the results of this paper can be adapted to deal with the resolution of the equation $(C||X) \Re A$ where the specifications of *C* and *A* are given as non-deterministic partially specified input/output Finite State Machines and $\Re$ represents the quasi-equivalence or the reduction relation. In order to apply our algorithms in this context, we first transform the specifications of *C* and *A* into the corresponding *IOAs*, denoted *IOA*(*C*) and *IOA*(*A*), by unfolding the transitions. In the case where $\Re$ is the quasi-equivalence relation, we denote by *A'* the *IOAWOCT* $IOA(A)^{woct}$, and in the case where $\Re$ is the reduction criterion, we denote by *A'* the *IOAWOCT* $IOA(A)^{red}$, then we resolve the equation $(IOA(C)||X) \Re A'$ under the constraint $I_X = (I_{IOA(A)} \setminus I_{IOA(C)}) \cup (O_{IOA(C)} \setminus O_{IOA(A)})$. If we obtain a generic solution *Sol*, we construct the FSM $FSM_{Sol} = (S, I_{IOA_{Sol}}, O_{IOA_{Sol}}, h, s_0)$. The set of states $S_{IOA_{Sol}}$ enjoys a nice property, it can be devided into two sets : the first set contains the states *s* such that *leaving*(*s*) is included in $I_{IOA_{Sol}}$ and *entering*(*s*) is not empty and included in $O_{IOA_{Sol}}$, this set contains the initial state of IOA_{Sol}; the second set contains the states *s* such that *leaving*(*s*) is not empty and included

in $O_{IOA_{Sol}}$ and *entering*(*s*) are not empty and included in $I_{IOA_{Sol}}$. From this observation, the set of states of FSM_{Sol} is equal to the first set; and for each state s_i of FSM_{Sol}, there is a transition $s_i \rightarrow v/w \rightarrow s_k$ in FSM_{Sol} if there are transitions $s_i \rightarrow v \rightarrow s_j$ and $s_j \rightarrow w \rightarrow s_k$ in IOA_{Sol}. Since for any solution *D* to the equation $(C||X) \mathbin{\Re} A$ the corresponding *IOA IOA*(*D*) is a solution to the equation $(IOA(C)||X) \mathbin{\&} A'$, the set of solutions to the equation $(C||X) \mathbin{\Re} A$ is the set of FSMs quasi-equivalent to an FSM derived from some subtype of *Sol* which is trace included in *Sol*. In the particular case where the FSMs *C* and *A* are completely specified, the quasi equivalence relation reduces to trace equivalence and we obtain the set of solutions to the equation for completely specified FSMs and trace equivalence using the same algorithms.

We note that Theorem 4 of Section 6 provides a new exact characterization of the set of solutions. In previous works it was necessary to check whether there is no livelock, i.e. cycle labelled only with internal actions, when combining a candidate solution with the context *C* [4, 16, 22], or to assume that at least one of the FSMs, context or solution, has to be a Moore FSM [1].

8 Conclusion

We have presented in this paper an approach to solve the problem of submodule construction in the realm of I/O automata. This problem may be formulated mathematically by the equation $(C||X) \mathbin{\Re} A$ under the constraint $I_X = In$, where *C* represents the specification of the known part of the system, *A* represents the specification of the whole system, *X* represents the specification of the submodule to be constructed, || is a composition operator, $\Re$ is a conformance relation and *In* is the required set of inputs for *X*. The conformance relations considered are the safe realization criterion and the subtype relation. For the safe realization criterion the set of solutions to the equation (if they exist) can be represented as the set of safe realizations of an I/O automaton $Sol_{\S}$. An algorithm for finding $Sol_{\S}$ is given. For defining the subtype relation we enhance the I/O automata model to allow the description of mandatory behaviors and we show that the set of solutions to the equation (if they exist) can be represented as the set of subtypes of an I/O automaton with optional complete traces *Sol*. An algorithm for finding *Sol* is given.

We also show that the submodule construction problem for non-deterministic partially specified input/output Finite State Machines is a particular case of this work for various criteria. The algorithms proposed in this paper were implemented in Java in the context of a tool for the construction of submodules [3].

References

[1] A. Aziz, F. Balarin, R. K. Brayton, M. D. DiBenedetto and A. Saldanha, *Supervisory Control of Finite State Machines*, Proceedings of the 7th International Conference, CAV'95, Liège, Belgium., pp. 279-292, July 3-5, 1995.

[2] A. Black, N. Hutchinson, E. Jul, H. Levy and L. Carter, *Distribution and Abstract Types in Emerald*, IEEE Transaction on Software Engineering, vol. SE-13, no. 1, pp. 65-76, January 1987.

[3] J. Drissi and G. v. Bochmann, *Submodule construction tool*, in the proceeding of CIMCA'99, Vienna, Austria, 1999.

[4] J. Drissi, N. Yevtushenko, A. Petrenko and G. v. Bochmann, *On the design of a submodule based on the input/output FSM model*, Technical Report no. 1120, DIRO, University of Montreal, 1998.

[5] A. Gill, *Introduction to the theory of Finite-State Machines*, Mc Graw-Hill Book Company, Inc, 1962.

[6] R. J. v. Glabbeek, *The Linear Time-Branching Time Spectrum*, Proceedings of CONCUR'90, Theories of Consurrency : Unification and Extension, pp. 278-297, Amsterdam, The Netherlands, August 27-30, 1990.

[7] E. Haghverdi and H. Ural, *An Algorithm for Submodule Construcyion*, Technical report of the Department of computer Science, University of Ottawa, 1996.

[8] S. G. H. Kelekar George W., *Synthesis of protocols and protocol converters using the submodule construction approach*, Proceedings of Protocol Specification, Testing and Verification, XIII, A. Danthine, G. Leduc, P. Wolper (Editors), 1994.

[9] K. G. Larsen, *Modal Specification*, Proceedings of International Workshop, Automatic Verification Methods for Finite State Machines, pp. 232-246, Grenoble, France, June 1989.

[10] B. Lin, G. de Jong and T. Kolks, *Hierarchical Optimization of Asynchronous Circuits*, Proceedings of the 32nd Design Automation Conference, pp 712-717, 1995.

[11] G. Luo, A. Petrenko and G. v. Bochmann, *Selecting test sequences for partially-specified nondeterministic finite state machines*, Technical report #864, University of Montreal, 1993.

[12] N. A. Lynch and M. R. Tuttle, *AN INTRODUCTION TO INPUT/OUTPUT AUTOMATA*, MIT/LCS/TM-373, Laboratory for computer science, Massachusetts Institute of Technology, Nov. 1998.

[13] P. Merlin and G. v. Bochmann, *On the Construction of Submodule Specifications and Communication Protocols*, ACM Trans. on Programming Languages and Systems, Vol. 5, No. 1, pp. 1-25, Jan. 1983.

[14] R. Negulescu and J. A. Brzozowski, *Relative Liveness : from intuition to automated verification*, Research report CS-95-32, University of Waterloo, Canada, 1995.

[15] A. Petrenko, N. Yevtushenko and G. v. Bochmann, *Experiments on Nondeterministic Systems for the Reduction Relation*, IWTCS'96.

[16] A. Petrenko and N. Yevtushenko, *Solving asynchronous equations*, in the proceeding of FORTE/PSTV'98, Paris, 1998.

[17] M. Phalippou, *Relations d'implantation et hypothèses de test sur des automates à entrées et sorties*, Thèse de Doctorat, Bordeaux, France, 1994.

[18] H. Qin and P. Lewis, *Factorisation of Finite State Machines under Strong and Observational Equivalences*, Journal of Formal Aspects of Computing, Vol. 3, pp 284-307, July-Sept. 1991.

[19] M. W. Shields, *Implicit System Specification and the Interface Equation*, Computer Journal, Vol. 32, 5, pp. 399-412, Oct. 1989.

[20] P. H. Starke, *Abstract automata*, American Elsevier Publishing Company, Inc-New York, 1972.

[21] S. H. Unger, *Asynchronous Sequential Switching Circuits*, New York, Wiley-Interscience, 1969.

[22] Y. Watanabe and R. K. Brayton, *The maximal set of permissible behaviors for fsm networks*, Proc. of the IEEE/ACM International Conference on Computer-Aided Design, pp 316-320, 1993.

[23] D. Wood, *Theory of Computation*, John Wiley & Sons, Inc, 1987.

Annex : Proofs of Theorems and some Lemmas

Lemma 1: For an *IOA A* and a composite *IOA* $B=B_1||B_2||...||B_n$, with $I_A=I_B$, the following propositions are equivalent :

i - $\mathbb{S}(\tilde{A}||B_1||B_2||...||B_n)$,

ii - for every *IOA E*, with $I_E=O_A$ and $O_E=I_A$, $\mathbb{S}(E||A) \Rightarrow \mathbb{S}(E||B_1||B_2||...||B_n)$.

Proof of Lemma 1 :

First part : (i) $\Rightarrow$ (ii)

Let E be an *IOA*, with $I_E=O_A$ and $O_E=I_A$, such that $\mathbb{S}(E||A)$. We have to prove that $\mathbb{S}(E||B_1||B_2||...||B_n)$. For a trace σ, we note $\sigma_{[k]}$ the prefix of σ of length k.

Suppose that there exists $\sigma \in (I_{(E||B_1||B_2||...||B_n)} \cup O_{(E||B_1||B_2||...||B_n)})^*$ with $|\sigma|=m$ such that :

$$Pr_E(\sigma) \in Tr_E.(I_E \cup \{\varepsilon\}) \wedge Pr_{B_i}(\sigma) \in Tr_{B_i}.(I_{B_i} \cup \{\varepsilon\}) \text{ for } 1 \le i \le n$$

First, we show by induction that $Pr_E(\sigma) \in Tr_A$.

If $Pr_E(\sigma_{[1]})=\varepsilon$ then $Pr_E(\sigma_{[1]}) \in Tr_A$

If $Pr_E(\sigma_{[1]}) \in I_E$, since $\mathbb{S}(\tilde{A}||B_1||B_2||...||B_n)$ then

$$Pr_E(\sigma_{[1]}) \in Tr_{\tilde{A}}.I_{\tilde{A}} \wedge Pr_{B_i}(\sigma_{[1]}) \in Tr_{B_i}.(I_{B_i} \cup \{\varepsilon\}) \text{ for } 1 \le i \le n \Rightarrow Pr_E(\sigma_{[1]}) \in Tr_A$$

If $Pr_E(\sigma_{[1]}) \in O_E$, since $\mathbb{S}(E||A)$ then $Pr_E(\sigma_{[1]}) \in Tr_E \wedge Pr_A(\sigma_{[1]}) \in Tr_A.I_A \Rightarrow Pr_E(\sigma_{[1]}) \in Tr_A$

Assume that $Pr_E(\sigma_{[k]}) \in Tr_A$ for $1 \le k < m$, we put $Pr_E(\sigma_{[k+1]})=Pr_E(\sigma_{[k]}).t$

If $t=\varepsilon$ then $Pr_E(\sigma_{[k+1]}) \in Tr_A$

If $t \in I_E$, since $\mathbb{S}(\tilde{A}||B_1||B_2||...||B_n)$ then

$$Pr_E(\sigma_{[k+1]}) \in Tr_{\tilde{A}}.I_{\tilde{A}} \wedge Pr_{B_i}(\sigma_{[k+1]}) \in Tr_{B_i}.(I_{B_i} \cup \{\varepsilon\}) \text{ for } 1 \le i \le n \Rightarrow Pr_E(\sigma_{[k+1]}) \in Tr_A$$

If $t \in O_E$, since $\mathbb{S}(E||A)$ then $Pr_E(\sigma_{[k+1]}) \in Tr_E \wedge Pr_A(\sigma_{[k+1]}) \in Tr_A.I_A \Rightarrow Pr_E(\sigma_{[k+1]}) \in Tr_A$

By the principle of induction $Pr_E(\sigma) \in Tr_A$.

Since $\mathbb{S}(\tilde{A}||B_1||B_2||...||B_n)$ then

$$Pr_E(\sigma) \in Tr_A \wedge Pr_{B_i}(\sigma) \in Tr_{B_i}.(I_{B_i} \cup \{\varepsilon\}) \text{ for } 1 \le i \le n \Rightarrow Pr_{B_i}(\sigma) \in Tr_{B_i} \text{ for } 1 \le i \le n$$

Since $\mathbb{S}(E||A)$ then $Pr_E(\sigma) \in Tr_E.(I_E \cup \{\varepsilon\}) \wedge Pr_E(\sigma) \in Tr_A \Rightarrow Pr_E(\sigma) \in Tr_E$

Therefore $\sigma \in Tr_{(E||B_1||B_2||...||B_n)}$.

We conclude that $\mathbb{S}(E||B_1||B_2||...||B_n)$.

Second part : (ii) $\Rightarrow$ (i)

This part is obvious since $\tilde{A}$ is an *IAO* with $I_{\tilde{A}}=O_A$ and $O_{\tilde{A}}=I_A$, and we have $\mathbb{S}(E||A)$. □

The proofs of Lemma 2, Lemma 3, Lemma 4, Lemma 5 and Lemma 6 are obvious.

Theorem 1 : Given two deterministic *IOAs* A and C, and given a set In such that $(I_A \backslash I_C) \cup (O_C \backslash O_A) \subseteq In \subseteq I_A \cup O_C$, if Algorithm 1 produces an *IOA* $Sol_{\S}$ then $(C \parallel Sol_{\S}) \leq_{\S} A$ and $In = I_{Sol_{\S}}$, else there is no solution for $(C \parallel X) \leq_{\S} A$ with the specified input alphabet In.

Proof of Theorem 1 :

First part :

If the Algorithm 1 produce an *IOA* $Sol_{\S}$ then we have to prove that $(C \| Sol_{\S}) \leq_{\S} A$. Suppose that there exists $\sigma \in (I_{(C\|Sol_{\S}\|\tilde{A})} \cup O_{(C\|Sol_{\S}\|\tilde{A})})^*$ such that :

$Pr_C(\sigma) \in Tr_C.(I_C \cup \{\varepsilon\}) \wedge Pr_{Sol_{\S}}(\sigma) \in Tr_{Sol_{\S}}.(I_{Sol_{\S}} \cup \{\varepsilon\}) \wedge Pr_A(\sigma) \in Tr_A.(O_A \cup \{\varepsilon\}) \wedge \sigma \notin Tr_{(C\|Sol_{\S}\|\tilde{A})}$

we consider a σ such that for any proper prefix the previous property don't holds.

Since : $Pr_C(\sigma) \in Tr_{Ief(C)} \wedge Pr_{Sol_{\S}}(\sigma) \in Tr_{Ch} \wedge Pr_A(\sigma) \in Tr_{Ief(\tilde{A})}$ THEN $\sigma \in Tr_R$

This imply that σ leads to the *Fail* state at some step of algorithm 1 and then was removed.

Let $\sigma = \sigma'.t$,

IF $t \in O_{Sol_{\S}}$ THEN $Pr_{Sol_{\S}}(\sigma) \notin Tr_{Sol_{\S}}(I_{Sol_{\S}} \cup \{\varepsilon\})$, contradiction.

IF $t \in (I_A \cup O_C)$ THEN after removing t, σ' leads to the *Fail* state and $\sigma' \notin Tr_{(C\|Sol_{\S}\|\tilde{A})}$, contradiction.

We conclude that $(C \| Sol_{\S}) \leq_{\S} A$.

Second part :

If the algorithm returns "NO SOLUTION", then there exists $\sigma \in (I_A \cup O_C)^*$, with $\sigma = \sigma'.t$ and $Pr_C(\sigma') \in Tr_C$ and $Pr_A(\sigma') \in Tr_A$ and $t \in (I_A \cap I_C) \cup (O_A \cap O_C)$, such that $(s_{0R})_{\sigma} = Fail_R$.

Suppose that there exists an *IOA* B with $I_B = In$ and $(C \| B) \leq_{\S} A$. If $\sigma' = \varepsilon$ then $\sigma' \in Tr_{(C\|B\|\tilde{A})}$, else by induction we show that $\sigma' \in Tr_{(C\|B\|\tilde{A})}$, let $|\sigma'| = n$ and $\sigma'_{[k]}$ the prefix of σ' of length k for $0 \leq k \leq n$,

- Since $(C \| B) \leq_{\S} A$, then $Pr_B(\sigma'_{[1]}) \in Tr_B.(I_B \cup \{\varepsilon\}) \Rightarrow \sigma'_{[1]} \in Tr_{(C\|B\|\tilde{A})}$,
- Assume that $\sigma'_{[k]} \in Tr_{(C\|B\|\tilde{A})}$ for $1 \leq k < n$, we will show that $\sigma'_{[k+1]} \in Tr_{(C\|B\|\tilde{A})}$

Since $\sigma'_{[k]} \in Tr_{(C\|B\|A)}$ then $Pr_B(\sigma'_{[k]}) \in Tr_B$, which imply $Pr_B(\sigma'_{[k+1]}) \in Tr_B.(I_B \cup \{\varepsilon\})$

Since $(C \| B) \leq_{\S} A$, then $Pr_B(\sigma'_{[k+1]}) \in Tr_B.(I_B \cup \{\varepsilon\}) \Rightarrow \sigma'_{[k+1]} \in Tr_{(C\|B\|\tilde{A})}$.

By the principle of induction $\sigma' \in Tr_{(C\|B\|\tilde{A})}$.

Now, if $t \in (I_A \cap I_C)$ then $Pr_C(\sigma) \in Tr_C.(I_C \cup \{\varepsilon\}) \wedge Pr_B(\sigma) \in Tr_B.(I_B \cup \{\varepsilon\}) \wedge Pr_A(\sigma) \in Tr_A$ but $\sigma \notin Tr_{(C\|B\|\tilde{A})}$ since $Pr_C(\sigma) \notin Tr_C$, contradiction.

If $t \in (O_A \cap O_C)$ then $Pr_C(\sigma) \in Tr_C \wedge Pr_B(\sigma) \in Tr_B.(I_B \cup \{\varepsilon\}) \wedge Pr_A(\sigma) \in Tr_A.(O_A \cup \{\varepsilon\})$ but $\sigma \notin Tr_{(C\|B\|\tilde{A})}$ since $Pr_A(\sigma) \notin Tr_A$, contradiction.

We conclude that there is no *IOA* B with $I_B = In$ such that $(C \| B) \leq_{\S} A$. □

Theorem 2 : Given two deterministic *IOAs* A and C, and given a set In such that $(I_A \backslash I_C) \cup (O_C \backslash O_A) \subseteq In \subseteq I_A \cup O_C$, an *IOA* B, with $I_B = In$ and $O_B = O_{Sol_{\S}}$, is a solution of the equation $(C \parallel X) \leq_{\S} A$ iff $B \leq_{\S} Sol_{\S}$.

Proof of Theorem 2 :

We will use in this proof the fact $(C||Sol_{\S})\leq_{\S} A$, i.e. for each $\sigma\in (I_{(C||Sol_{\S}||\tilde{A})}\cup O_{(C||Sol_{\S}||\tilde{A})})^*$

$Pr_C(\sigma)\in Tr_C.(I_C\cup\{\varepsilon\})\wedge Pr_{Sol_{\S}}(\sigma)\in Tr_{Sol_{\S}}.(I_{Sol_{\S}}\cup\{\varepsilon\})\wedge Pr_A(\sigma)\in Tr_A.(O_A\cup\{\varepsilon\}) \Rightarrow$

$\sigma\in Tr_{(C||Sol_{\S}||\tilde{A})}$

First part : $(C||B)\leq_{\S} A \Rightarrow B\leq_{\S} Sol_{\S}$

Consider $\sigma\in (I_{Sol_{\S}}\cup O_{Sol_{\S}})^*$, we have to prove that :

$$\sigma\in Tr_{Sol_{\S}}.(O_{Sol_{\S}}\cup\{\varepsilon\}) \wedge \sigma\in Tr_B.(I_B\cup\{\varepsilon\}) \Rightarrow \sigma\in Tr_{(B||\widetilde{Sol_{\S}})}$$

We put $\sigma=\sigma_1.t$.

Case 1 : $t\in I_{Sol_{\S}}$,

we have $\sigma\in Tr_{Sol_{\S}} \wedge \sigma\in Tr_B.I_B$,

by construction of $Sol_{\S}$, there exists $\sigma_2.t\in Tr_{(C||\tilde{A})}$ such that $Pr_{Sol_{\S}}(\sigma_2.t)=\sigma$

therefore $Pr_C(\sigma_2.t)\in Tr_C \wedge Pr_A(\sigma_2.t)\in Tr_A \wedge Pr_B(\sigma_2.t)\in Tr_B.I_B$

since $(C||B)\leq_{\S} A$ then $\sigma\in Tr_B$

therefore $\sigma\in Tr_{(B||\widetilde{Sol_{\S}})}$.

Case 2 : $t\in O_{Sol_{\S}}$

we have $\sigma\in Tr_{Sol_{\S}}.O_{Sol_{\S}} \wedge \sigma\in Tr_B$,

If $\sigma_1\neq\varepsilon$, by construction of $Sol_{\S}$, there exists $\sigma_3\in Tr_{(C||\tilde{A})}$ such that $Pr_{Sol_{\S}}(\sigma_3)=\sigma_1$ and $Pr_{Sol_{\S}}(\sigma_{3[k]})\neq\sigma_1$ for $0\leq k<|\sigma_3|$, else we put $\sigma_3=\varepsilon$

therefore $Pr_C(\sigma_3.t)\in Tr_C.(I_C\cup\{\varepsilon\})\wedge Pr_A(\sigma_3.t)\in Tr_A.(O_A\cup\{\varepsilon\}) \wedge Pr_B(\sigma_3.t)\in Tr_B$

since $(C||B)\leq_{\S} A$ then $\sigma_3.t\in Tr_{(C||\tilde{A})}$

If $Pr_B(\sigma_3.t)\notin Tr_{Sol_{\S}}$, by construction of $Sol_{\S}$ there exists $\sigma_3.\sigma_4.t.\sigma_5.t'\in Tr_{(Ief(C)||Ief(\tilde{A}))}$

which leads to *Fail* with $\sigma_4\in((I_A\cup O_C)\backslash In)^*$, $\sigma_5\in(I_A\cup O_C)^*$ and $t'\in((I_A\cap I_C)\cup(O_A\cap O_C))$

since $(C||B)\leq_{\S} A$ then $Pr_B(\sigma_3.\sigma_4.t.\sigma_5)\in Tr_B$

Now, if $t'\in(I_A\cap I_C)$ then

$$Pr_C(\sigma_3.\sigma_4.t.\sigma_5.t')\in Tr_C.(I_C\cup\{\varepsilon\})\wedge Pr_B(\sigma_3.\sigma_4.t.\sigma_5.t')\in Tr_B.(I_B\cup\{\varepsilon\})$$
$$\wedge Pr_A(\sigma_3.\sigma_4.t.\sigma_5.t')\in Tr_A$$

but $\sigma_3.\sigma_4.t.\sigma_5.t'\notin Tr_{(C||B||\tilde{A})}$ since $Pr_C(\sigma_3.\sigma_4.t.\sigma_5.t')\notin Tr_C$, contradiction.

If $t'\in(O_A\cap O_C)$ then

$$Pr_C(\sigma_3.\sigma_4.t.\sigma_5.t')\in Tr_C\wedge Pr_B(\sigma_3.\sigma_4.t.\sigma_5.t')\in Tr_B.(I_B\cup\{\varepsilon\})$$
$$\wedge Pr_A(\sigma_3.\sigma_4.t.\sigma_5.t')\in Tr_A.(O_A\cup\{\varepsilon\})$$

but $\sigma_3.\sigma_4.t.\sigma_5.t'\notin Tr_{(C||B||\tilde{A})}$ since $Pr_A(\sigma_3.\sigma_4.t.\sigma_5.t')\notin Tr_A$, contradiction.

therefore $Pr_B(\sigma_3.t)\in Tr_{Sol_{\S}}$ and $\sigma\in Tr_{(B||\widetilde{Sol_{\S}})}$

We conclude that $B\leq_{\S} Sol_{\S}$

Second part : $B\leq_{\S} Sol_{\S} \Rightarrow (C||B)\leq_{\S} A$

Consider $\sigma\in(I_{(C||\tilde{A})}\cup O_{(C||\tilde{A})})^*$, we have to prove that :

$Pr_C(\sigma)\in Tr_C.(I_C\cup\{\varepsilon\})\wedge Pr_B(\sigma)\in Tr_B.(I_B\cup\{\varepsilon\})\wedge Pr_A(\sigma)\in Tr_A.(O_A\cup\{\varepsilon\}) \Rightarrow \sigma\in Tr_{(C||B||\tilde{A})}$

We put $\sigma=\sigma_1.t$, with $\sigma_1 \in Tr_{(C||B||\tilde{A})}$

Let $\sigma'=Pr_B(\sigma_1)$ and $|\sigma'|$=n, we show by induction that $\sigma' \in Tr_{Sol_{\mathbb{S}}}$

If $\sigma'_{[1]} \in O_{Sol_{\mathbb{S}}}$, since $B \leq_{\mathbb{S}} Sol_{\mathbb{S}}$ then $\sigma'_{[1]} \in Tr_{Sol_{\mathbb{S}}}.O_{Sol_{\mathbb{S}}} \wedge \sigma'_{[1]} \in Tr_B \Rightarrow \sigma'_{[1]} \in Tr_{Sol_{\mathbb{S}}}$

If $\sigma'_{[1]} \in I_{Sol_{\mathbb{S}}}$, then there exists a prefix σ_2 of σ_1 such that $\sigma_2=\sigma_3.\sigma'_{[1]}$ and $Pr_{Sol_{\mathbb{S}}}(\sigma_2)=\sigma'_{[1]}$, since $(C||Sol_{\mathbb{S}}) \leq_{\mathbb{S}} A$ then

$Pr_C(\sigma_2) \in Tr_C \wedge Pr_{Sol_{\mathbb{S}}}(\sigma_2) \in Tr_{Sol_{\mathbb{S}}}.I_{Sol_{\mathbb{S}}} \wedge Pr_A(\sigma_2) \in Tr_A \Rightarrow \sigma'_{[1]}=Pr_{Sol_{\mathbb{S}}}(\sigma_2) \in Tr_{Sol_{\mathbb{S}}}$

Assume that $\sigma'_{[k]} \in Tr_{Sol_{\mathbb{S}}}$ for $1 \leq k < n$, we put $\sigma'_{[k+1]}=\sigma'_{[k]}.t'$

If $t' \in O_{Sol_{\mathbb{S}}}$, since $B \leq_{\mathbb{S}} Sol_{\mathbb{S}}$ then $\sigma'_{[k+1]} \in Tr_{Sol_{\mathbb{S}}}.O_{Sol_{\mathbb{S}}} \wedge \sigma'_{[k+1]} \in Tr_B \Rightarrow \sigma'_{[k+1]} \in Tr_{Sol_{\mathbb{S}}}$

If $t' \in I_{Sol_{\mathbb{S}}}$, then there exists a prefix σ_4 of σ_1 such that $\sigma_4=\sigma_5.t'$ and $Pr_{Sol_{\mathbb{S}}}(\sigma_4)=\sigma'_{[k+1]}$, since $(C||Sol_{\mathbb{S}}) \leq_{\mathbb{S}} A$ then

$Pr_C(\sigma_4) \in Tr_C \wedge Pr_{Sol_{\mathbb{S}}}(\sigma_4) \in Tr_{Sol_{\mathbb{S}}}.I_{Sol_{\mathbb{S}}} \wedge Pr_A(\sigma_4) \in Tr_A \Rightarrow \sigma'_{[k+1]}=Pr_{Sol_{\mathbb{S}}}(\sigma_4) \in Tr_{Sol_{\mathbb{S}}}$

By the principle of induction $\sigma' \in Tr_{Sol_{\mathbb{S}}}$, therefore $\sigma_1 \in Tr_{(C||Sol_{\mathbb{S}}||\tilde{A})}$.

Case 1 : $t \in O_{Sol_{\mathbb{S}}}$,

We have $Pr_{Sol_{\mathbb{S}}}(\sigma) \in Tr_B$, and since $B \leq_{\mathbb{S}} Sol_{\mathbb{S}}$ then $Pr_{Sol_{\mathbb{S}}}(\sigma) \in Tr_{Sol_{\mathbb{S}}}$

since $(C||Sol_{\mathbb{S}}) \leq_{\mathbb{S}} A$ then

$Pr_C(\sigma) \in Tr_C.(I_C \cup \{\varepsilon\}) \wedge Pr_{Sol_{\mathbb{S}}}(\sigma) \in Tr_{Sol_{\mathbb{S}}} \wedge Pr_A(\sigma) \in Tr_A.(O_A \cup \{\varepsilon\}) \Rightarrow \sigma \in Tr_{(C||Sol_{\mathbb{S}}||\tilde{A})}$

therefore $\sigma \in Tr_{(C||A)}$ and then $\sigma \in Tr_{(C||B||\tilde{A})}$

Case 2 : $t \notin O_{Sol_{\mathbb{S}}}$,

since $(C||Sol_{\mathbb{S}}) \leq_{\mathbb{S}} A$

$Pr_C(\sigma) \in Tr_C.(I_C \cup \{\varepsilon\}) \wedge Pr_{Sol_{\mathbb{S}}}(\sigma) \in Tr_{Sol_{\mathbb{S}}}.(I_{Sol_{\mathbb{S}}} \cup \{\varepsilon\}) \wedge Pr_A(\sigma) \in Tr_A.(O_A \cup \{\varepsilon\}) \Rightarrow \sigma \in Tr_{(C||Sol_{\mathbb{S}}||\tilde{A})}$

If $t \in I_{Sol_{\mathbb{S}}}$ then $Pr_{Sol_{\mathbb{S}}}(\sigma)=\sigma'.t \in Tr_{Sol_{\mathbb{S}}}$

since $B \leq_{\mathbb{S}} Sol_{\mathbb{S}}$ then $\sigma'.t \in Tr_B$ therefore $\sigma \in Tr_{(C||B||\tilde{A})}$

If $t \notin I_{Sol_{\mathbb{S}}}$ then $Pr_B(\sigma)=\sigma' \in Tr_S$ therefore $\sigma \in Tr_{(C||B||\tilde{A})}$

We conclude that $(C||B) \leq_{\mathbb{S}} A$. □

Theorem 3 : Given a deterministic *IOA C*, a deterministic *IOAWO A*, and a given input set *In* such that $(I_{IOA_A} \setminus I_C) \cup (O_C \setminus O_{IOA_A}) \subseteq In \subseteq I_{IOA_A} \cup O_C$, if Algorithm 2 produces an *IOAWOCT Sol* then $(C||IOA_{Sol}) \leq_{conf} A$, else there is no solution for $(C||X) \leq_{conf} A$ with the specified set of inputs *In*.

Proof of Theorem 3 :

First part : $(C||IOA_{Sol}) \leq_{conf} A$

If Algorithm 2 produces an *IOAWCT Sol* then we have to prove that $(C||IOA_{Sol}) \leq_{conf} A$. By definition of the relation $\leq_{conf}$, this is equivalent to $(C||IOA_{Sol}) \leq_{\mathbb{S}} IOA_A$ and $(C||IOA_{Sol}) \leq_{\mathbb{P}} A$.

- $(C||IOA_{Sol}) \leq_{\mathbb{S}} IOA_A$ is equivalent to $IOA_{Sol} \leq_{\mathbb{S}} Sol_{\mathbb{S}}$

Consider $\sigma \in (I_{Sol_{\mathbb{S}}} \cup O_{Sol_{\mathbb{S}}})^*$, by construction IOA_{Sol} is trace included in $Sol_{\mathbb{S}}$

Then $\sigma \in Tr_{Sol_{\mathbb{S}}}.O_{Sol_{\mathbb{S}}} \wedge \sigma \in Tr_{IOA_{Sol}} \Rightarrow \sigma \in Tr_{Sol_{\mathbb{S}}}$

We put $\sigma=\sigma_1.t$, with $t\in I_{Sol_{\mathbb{S}}}$ and suppose that

$$\sigma\in Tr_{Sol_{\mathbb{S}}} \wedge \sigma\in Tr_{IOA_{Sol}}.I_{Sol_{\mathbb{S}}} \wedge \sigma\notin Tr_{IOA_{Sol}}$$

Then t was removed from $Tr_{(C||Sol_{\mathbb{S}}||\widetilde{IOA_A})}$, but $t\in I_{Sol_{\mathbb{S}}}$ implies $\sigma_1\notin Tr_{IOA_{Sol}}$, contradiction.

Therefore $IOA_{Sol}\leq_{\mathbb{S}} Sol_{\mathbb{S}}$

- To prove $(C||IOA_{Sol})\leq_{\mathbb{P}} A$, we have two cases :

case 1 : If $MT_A(s_{0IOA_A})\neq\emptyset$ then

i - suppose that there exists $Y\in MT_A((s_{0IOA_A}))$ such that

$$Pr_{IOA_A}(Tr_{(C||IOA_{Sol})}((s_{0(C||IOA_{Sol})})))\cap Y=\emptyset$$

Then in Step 2 or Step 5 of Algorithm 2, "NO SOLUTION" is returned, contradiction.

ii - suppose that there exists $\sigma_1\in Tr_{(C||IOA_{Sol})}((s_{0(C||IOA_{Sol})}))$ such that

$Pr_{IOA_A}(\sigma_1)=\varepsilon$ and $Pr_{IOA_A}(Tr_{(C||IOA_{Sol})}((s_{0(C||IOA_{Sol})})_{\sigma_2}))\cap out((s_{0IOA_A}))=\emptyset$

Then in Step 2 or Step 5 of Algorithm 2 σ_1 will be removed from $Tr_{(C||Sol_{\mathbb{S}}||\widetilde{IOA_A})}$,

Then $\sigma_1\notin Tr_{(C||IOA_{Sol}||\widetilde{IOA_A})}$ and therefore $\sigma_1\notin Tr_{(C||IOA_{Sol})}$, contradiction.

case 2 : consider $\sigma=\sigma_1.t\in Tr_{(C||IOA_{Sol})}$ such that

$$t\in (I_{IOA_A}\cup O_{IOA_A}),\ \sigma'=Pr_{IOA_A}(\sigma)\in Tr_{IOA_A} \text{ and } MT_A((s_{0IOA_A})_{\sigma'})\neq\emptyset$$

We have $\sigma\in Tr_{(C||IOA_{Sol}||\widetilde{IOA_A})}$,

i - suppose that there exists $Y\in MT_A((s_{0IOA_A})_{\sigma'})$ such that

$$Pr_{IOA_A}(Tr_{(C||IOA_{Sol})}((s_{0(C||IOA_{Sol})})_{\sigma}))\cap Y=\emptyset$$

Then in Step 2 or Step 5 of Algorithm 2 σ will lead to $Fail_{R_1}$ and t will be removed

Therefore $\sigma\notin Tr_{(C||IOA_{Sol}||\widetilde{IOA_A})}$, contradiction.

ii - suppose that there exists $\sigma_2\in Tr_{(C||IOA_{Sol})}((s_{0(C||IOA_{Sol})})_{\sigma})$ such that

$Pr_{IOA_A}(\sigma_2)=\varepsilon$ and $Pr_{IOA_A}(Tr_{(C||IOA_{Sol})}((s_{0(C||IOA_{Sol})})_{\sigma\sigma_2}))\cap out((s_{0IOA_A})_{\sigma'})=\emptyset$

Then in Step 2 or Step 5 of Algorithm 2 $\sigma\sigma_2$ will be removed from $Tr_{(C||Sol_{\mathbb{S}}||\widetilde{IOA_A})}$,

Then $\sigma\sigma_2\notin Tr_{(C||IOA_{Sol}||\widetilde{IOA_A})}$ and therefore $\sigma\sigma_2\notin Tr_{(C||IOA_{Sol})}$, contradiction.

Therefore $(C||IOA_{Sol})\leq_{\mathbb{P}} A$.

We conclude that $(C||IOA_{Sol})\leq_{conf} A$.

Second part :

If the generic solution $Sol_{\mathbb{S}}$ does not exists, we have shown in Theorem 1 that there is no solution for $(C||X)\leq_{\mathbb{S}} IOA_A$ with the specified set of input In and therefore there is no soution for $(C||X)\leq_{conf} A$ with the specified set of input In.

Now, suppose that the generic solution $Sol_{\mathbb{S}}$ exists, and Algorithm 2 return "NO SOLUTION", and there exists an IOA B with $(C||B)\leq_{conf} A$,

Since $(C||Sol_{\mathbb{S}})\leq_{\mathbb{S}} IOA_A$ and $(C||B)\leq_{\mathbb{S}} A$ then by Theorem 2 $B\leq_{\mathbb{S}} Sol_{\mathbb{S}}$,

Therefore $Tr_{(C||B||\widetilde{IOA_A})}\subseteq Tr_{(C||Sol_{\mathbb{S}}||\widetilde{IOA_A})}$

Since $(C||B)\leq_{conf}A$, any time we remove a trace from $Tr_{(C||Sol_{\S}||\widetilde{IOA_A})}$ in Step 2 or Step 5, this trace can not be in $Tr_{(C||B||\widetilde{IOA_A})}$,

Algorithm 2 return "NO SOLUTION" if the initial state of $C||Sol_{\S}||\widetilde{IOA_A}$ must be removed due to the elimination of a trace in step 2 or Step 5,

Therefore the initial state of $C||B||\widetilde{IOA_A}$ can not satisfy a constraint, contradiction.

We conclude that there is no *IOA B* such that $(C||B)\leq_{conf}A$. □

Theorem 4 : Given a deterministic *IOA C*, a deterministic *IOAWO A*, and an input set *In* such that $(I_{IOA_A}\backslash I_C)\cup(O_C\backslash O_{IOA_A})\subseteq In\subseteq I_{IOA_A}\cup O_C$, if Algorithm 2 produces an *IOAWOCT Sol* then for any *IOA B*, with $I_B=In$ and $O_B=O_{Sol_{\S}}$, the following propositions are equivalent :

i - $(C||B)\leq_{conf}A$,

ii - $B\leq_{conf}Sol$.

Proof of Theorem 4 :

First part : $(C||B)\leq_{conf}A \Rightarrow B\leq_{conf}Sol$

1 - The proof of $B\leq_{\S}IOA_{Sol}$

consider $\sigma\in(I_B\cup O_B)^*$ and let $|\sigma|=n$, we will prove that :

$\sigma\in Tr_{IOA_{Sol}}.(O_B\cup\{\varepsilon\})\wedge\ \sigma\in Tr_B\ .(I_B\cup\{\varepsilon\}) \Rightarrow \sigma\in Tr_{(B||IOA_{Sol})}$

Since $(C||B)\leq_{\S}A$ then by Theorem 2 $B\leq_{\S}Sol_{\S}$

Therefore $Tr_{(C||B||\widetilde{IOA_A})}\subseteq Tr_{(C||Sol_{\S}||\widetilde{IOA_A})}$

Case 1 : $\sigma\in Tr_{IOA_{Sol}}\wedge\ \sigma\in Tr_B\ .I_B$

Since $Tr_{IOA_{Sol}}\subseteq Tr_{Sol_{\S}}$ then $\sigma\in Tr_{Sol_{\S}}\wedge\ \sigma\in Tr_B\ .I_B \Rightarrow \sigma\in Tr_B$

Therefore $\sigma\in Tr_{(B||IOA_{Sol})}$

Case 2 : $\sigma\in Tr_{IOA_{Sol}}.O_B\wedge\ \sigma\in Tr_B$

Since $Tr_{IOA_{Sol}}\subseteq Tr_{Sol_{\S}}$ then $\sigma\in Tr_{Sol_{\S}}.O_{Sol_{\S}}\wedge\ \sigma\in Tr_B\ \Rightarrow \sigma\in Tr_{Sol_{\S}}$

We put $\sigma=\sigma_1.t$ white $t\in O_B$,

Suppose that $\sigma\notin Tr_{IOA_{Sol}}$, since $\sigma_1\in Tr_{IOA_{Sol}}$ then σ was removed from $Tr_{Sol_{\S}}$ in Step 2 or in Step 5

- σ was removed from $Tr_{Sol_{\S}}$ in Step 2

Then there exists $\sigma_2.t\in Tr_{(C||Sol_{\S}||IOA_A)}$ such that $Pr_{Sol_{\S}}(\sigma_2.t)=\sigma$

Moreover $\sigma_2.t\in Tr_{(C||B)}$

Let $\sigma_3=\sigma_4.t_1=Pr_A(\sigma_2)$ and $\sigma_2=\sigma_5.t_1.\sigma_6$ such that $Pr_A(\sigma_5.t_1)=\sigma_3$,

There exists $Y\in MT_A((s_{0IOA_A})_{\sigma_3})$ such that

$$Y\cap Tr_{(C||Sol_{\S}||IOA_A)}(s_{0(C||Sol_{\S}||IOA_A)})_{\sigma_5.t_1})=\emptyset$$

Since $(C||B)\leq_p A$, then there exists $\sigma_7.t_2\in Tr_{(C||B)}(s_{0(C||B)})_{\sigma_5.t_1})$ such that

$$Pr_A(\sigma_7.t_2)=t_2\in Y$$

Therefore $\sigma_8=(\sigma_5.t_1.\sigma_7.t_2)\in Tr_{(C||B||IOA_A)}$

Since $B \leq_{\mathbb{S}} Sol_{\mathbb{S}}$ and $(C||Sol_{\mathbb{S}}) \leq_{\mathbb{S}} A$ then $\sigma_8 \in Tr_{(C||Sol_{\mathbb{S}}||IOA_A)}$

Contradiction with the fact that σ was removed from $Tr_{Sol_{\mathbb{S}}}$ in Step 2.

- σ was removed from $Tr_{Sol_{\mathbb{S}}}$ in Step 5

Then $\sigma_1.t$ leads to $Fail_{R_1}$ at some iteration of Step 5

Since $Tr_{(C||B||\widetilde{IOA_A})} \subseteq Tr_{(C||Sol_{\mathbb{S}}||\widetilde{IOA_A})}$, $C||B$ is not a conforming implementation of A, Contradiction.

Then we have $\sigma \in Tr_{IOA_{Sol}}$ and then $\sigma \in Tr_{(B||IOA_{Sol})}$

We conclude $B \leq_{\mathbb{S}} IOA_{Sol}$

2 - The proof of $B \leq_{\mathrm{P}} Sol$

Consider $\sigma \in Tr_B$ such that $\sigma \in Tr_{IOA_{Sol}}$ and $MT_{Sol}((s_{0IOA_{Sol}})_\sigma) \neq \emptyset$

Let $Y \in MT_{Sol}((s_{0IOA_{Sol}})_\sigma)$, by construction of Sol there exists

$\sigma_1 \in Tr_{(C||IOA_{Sol}||IOA_A)}$, $\sigma_2 \in Tr_{IOA_A}$, $Y_2 \in MT_A((s_{0IOA_A})_{\sigma_2})$,

$Y_1 \subseteq Tr_{(C||IOA_{Sol}||IOA_A)}((s_{0(C||IOA_{Sol}||IOA_A)})_{\sigma_1})$ and

$Y_3 \subseteq Tr_{(C||IOA_{Sol}||IOA_A)}((s_{0(C||IOA_{Sol}||IOA_A)})_{\sigma_1})$

such that

$\sigma = Pr_B(\sigma_1)$, $\sigma_2 = Pr_{IOA_A}(\sigma_1)$, $Y = Pr_B(Y_1)$, $OCT_{Sol}((s_{0IOA_{Sol}})_\sigma) = Pr_B(Y_3)$, $Pr_{IOA_A}(Y_1) \subseteq Y_2$

and for each $\beta.t \in Tr_{(C||IOA_{Sol}||IOA_A)}((s_{0(C||IOA_{Sol}||IOA_A)})_{\sigma_1})$ with $t \in O_{IOA_A}$,

$Pr_A(\beta.t) = t \in Y_2 \Rightarrow \beta.t \in Y_1$

$Pr_A(\beta.t) = t \in OCT_A((s_{0IOA_A})_{\sigma_2}) \Rightarrow \beta.t \in Y_3$

i - Since $\sigma_1 \in Tr_{(C||IOA_{Sol}||IOA_A)}$ and $Pr_B(\sigma_1) \in Tr_B$ then $\sigma_1 \in Tr_{(C||B||IOA_A)}$

Since $(C||B) \leq_{\mathrm{P}} A$ then there exists $\sigma_3 = \sigma_4.t_1 \in Tr_{(C||B)}(s_{0(C||B)})_{\sigma_1})$ such that $Pr_{IOA_A}(\sigma_3) = t_1 \in Y_2$

Since $B \leq_{\mathbb{S}} IOA_{Sol}$ and $(C||IOA_{Sol}) \leq_{\mathbb{S}} IOA_A$ then $\sigma_1.\sigma_3 \in Tr_{(C||IOA_{Sol}||IOA_A)}$

Therefore $\sigma_3 \in Y_1$ and then $Pr_B(\sigma_3) \in Y$

Then $Tr_B((s_{0B})_\sigma) \cap Y \neq \emptyset$

ii - Let $\sigma_5 = \sigma_6.t_2 \in Tr_B((s_{0B})_\sigma)$ such that

$\sigma_5 \in Pref(OCT_{Sol}((s_{0IOA_{Sol}})_\sigma))$ and $\sigma_5 \notin OCT_{Sol}((s_{0IOA_{Sol}})_\sigma)$

There exists $\sigma_7 = \sigma_1.\sigma_8.t_2 \in Tr_{(C||IOA_{Sol}||IOA_A)}$ such that $Pr_B(\sigma_8.t_2) = \sigma_5$

Since $\sigma_7 \in Tr_{(C||IOA_{Sol}||IOA_A)}$ and $Pr_B(\sigma_7) \in Tr_B$ then $\sigma_7 \in Tr_{(C||B||IOA_A)}$

Since $(C||B) \leq_{\mathrm{P}} A$ then there exists $\sigma_9 = \sigma_{10}.t_3 \in Tr_{(C||B)}(s_{0(C||B)})_{\sigma_7})$ such that

$Pr_{IOA_A}(\sigma_9) = t_3 \in OCT_A((s_{0IOA_A})_{\sigma_2})$

Since $B \leq_{\mathbb{S}} IOA_{Sol}$ and $(C||IOA_{Sol}) \leq_{\mathbb{S}} IOA_A$ then $\sigma_7.\sigma_9 \in Tr_{(C||IOA_{Sol}||IOA_A)}$

Therefore $\sigma_{10} = \sigma_8.t_2.\sigma_9 \in Y_3$ and then $Pr_B(\sigma_{10}) \in OCT_{Sol}((s_{0IOA_{Sol}})_\sigma)$

We conclude $B \leq_{\mathrm{P}} Sol$

Second part : $B \leq_{\mathrm{conf}} Sol \Rightarrow (C||B) \leq_{\mathrm{conf}} A$

1 - The proof of $(C||B) \leq_{\mathbb{S}} IOA_A$

We have $B \leq_{\mathbb{S}} IOA_{Sol}$ and $IOA_{Sol} \leq_{\mathbb{S}} Sol_{\mathbb{S}}$, consider $\sigma \in (I_B \cup O_B)^*$ and let $|\sigma|=n$, we will prove that : $\sigma \in Tr_{Sol_{\mathbb{S}}}.(O_B \cup \{\varepsilon\}) \wedge \sigma \in Tr_B .(I_B \cup \{\varepsilon\}) \Rightarrow \sigma \in Tr_{(B||Sol_{\mathbb{S}})}$

If n>1, we show by induction that $\sigma_{[n-1]} \in Tr_{IOA_{Sol}}$

If $\sigma_{[1]} \in O_B$, since $B \leq_{\mathbb{S}} IOA_{Sol}$ then $\sigma_{[1]} \in O_B \wedge \sigma_{[1]} \in Tr_B \Rightarrow \sigma_{[1]} \in Tr_{IOA_{Sol}}$

If $\sigma_{[1]} \in I_B$, since $IOA_{Sol} \leq_{\mathbb{S}} Sol_{\mathbb{S}}$ then $\sigma_{[1]} \in I_B \wedge \sigma_{[1]} \in Tr_{Sol_{\mathbb{S}}} \Rightarrow \sigma_{[1]} \in Tr_{IOA_{Sol}}$

Assume that $\sigma_{[k]} \in Tr_{IOA_{Sol}}$ for $1 \leq k < n-1$, we put $\sigma_{[k+1]}=\sigma_{[k]}.t$

If $t \in O_B$, since $B \leq_{\mathbb{S}} IOA_{Sol}$ then $\sigma_{[k+1]} \in Tr_{IOA_{Sol}}.O_B \wedge \sigma_{[k+1]} \in Tr_B \Rightarrow \sigma_{[k+1]} \in Tr_{IOA_{Sol}}$

If $t \in I_B$, since $IOA_{Sol} \leq_{\mathbb{S}} Sol_{\mathbb{S}}$ then $\sigma_{[k+1]} \in Tr_{IOA_{Sol}}.I_B \wedge \sigma_{[k+1]} \in Tr_{Sol_{\mathbb{S}}} \Rightarrow \sigma_{[k+1]} \in Tr_{IOA_{Sol}}$

By the principle of induction we have $\sigma_{[n-1]} \in Tr_{IOA_{Sol}}$

Case 1 : $\sigma \in Tr_{Sol_{\mathbb{S}}}.O_{Sol_{\mathbb{S}}} \wedge \sigma \in Tr_B$

Since $B \leq_{\mathbb{S}} IOA_{Sol}$ then $\sigma \in Tr_{IOA_{Sol}}.O_B \wedge \sigma \in Tr_B \Rightarrow \sigma \in Tr_{IOA_{Sol}}$

And since $IOA_{Sol} \leq_{\mathbb{S}} Sol_{\mathbb{S}}$ then $\sigma \in Tr_{Sol_{\mathbb{S}}}.O_B \wedge \sigma \in Tr_{IOA_{Sol}} \Rightarrow \sigma \in Tr_{Sol_{\mathbb{S}}}$

Therefore $\sigma \in Tr_{(B||Sol_{\mathbb{S}})}$

Case 2 : $\sigma \in Tr_{Sol_{\mathbb{S}}} \wedge \sigma \in Tr_B .I_B$

Since $IOA_{Sol} \leq_{\mathbb{S}} Sol_{\mathbb{S}}$ then $\sigma \in Tr_{Sol_{\mathbb{S}}} \wedge \sigma \in Tr_{IOA_{Sol}} .I_B \Rightarrow \sigma \in Tr_{IOA_{Sol}}$

And since $B \leq_{\mathbb{S}} IOA_{Sol}$ then $\sigma \in Tr_{IOA_{Sol}} \wedge \sigma \in Tr_B .I_B \Rightarrow \sigma \in Tr_B$

Therefore $\sigma \in Tr_{(B||Sol_{\mathbb{S}})}$

Therefore $B \leq_{\mathbb{S}} Sol_{\mathbb{S}}$ which is equivalent by Theorem 2 to $(C||B) \leq_{\mathbb{S}} IOA_A$

2 - The proof of $(C||B) \leq_P A$

suppose that $MT_A(s_{oIOA_A}) \neq \emptyset$ and consider an $Y \in MT_A(s_{oIOA_A})$,

By construction of *Sol* there exists $Y_2 \subseteq Tr_{(C||IOA_{Sol}||IOA_A)}$ such that $Y_2 \neq \emptyset$, and for each element $\beta \in Y_2$ $Pr_{IOA_A}(\beta) \in Y$

a - $Y_2 \cap O_C \neq \emptyset$

There exists $y \in Y_2 \cap O_C$ such that $y \in Tr_{(C||IOA_{Sol}||IOA_A)}$

Since $(C||B) \leq_{\mathbb{S}} IOA_A$ then

$Pr_C(y) \in Tr_C \wedge Pr_B(y) \in Tr_B.I_B \wedge Pr_{IOA_A}(y) \in Tr_{IOA_A} \Rightarrow y \in Tr_{(C||B||IOA_A)}$

Therefore $Pr_{IOA_A}(Tr_{(C||B)}) \cap Y \neq \emptyset$

b - $Y_2 \cap O_C = \emptyset$

$Y_3 = Pr_B(Y_2) \in MT_{Sol}\ (s_{oIOA_{Sol}})$

Since $B \leq_P Sol$, there exists $\sigma_5 \in Y_3$ such that $\sigma_5 \in Tr_B$

Therefore there exists $\sigma_6 \in Y_2$ such that $Pr_B(\sigma_6)=\sigma_5$

Since $\sigma_6 \in Tr_{(C||IOA_{Sol}||IOA_A)}$ then $Pr_C(\sigma_6) \in Tr_C$

Moreover $Pr_B(\sigma_6)=\sigma_5 \in Tr_B$, Then $\sigma_6 \in Tr_{(C||B)}$

Therefore $Pr_{IOA_A}(Tr_{(C||B)}) \cap Y \neq \emptyset$

ii - Suppose that there exists $\sigma'' \in Tr_{(C||B)}$ such that $Pr_{IOA_A}(\sigma'')=\varepsilon$

Let $|\sigma''|$=n, we show by induction that $\sigma'' \in Tr_{IOA_{Sol}}$

If $\sigma''_{[1]} \in I_B$, since $(C||IOA_{Sol}) \leq_{\mathbb{S}} IOA_A$ then

$Pr_C(\sigma''_{[1]}) \in Tr_C \wedge Pr_B(\sigma''_{[1]}) \in Tr_{IOA_{Sol}}.I_B \wedge Pr_{IOA_A}(\sigma''_{[1]}) \in Tr_{IOA_A} \Rightarrow \sigma''_{[1]} \in Tr_{IOA_{Sol}}$

If $\sigma''_{[1]} \in O_B$, since $B \leq_{\mathbb{S}} IOA_{Sol}$ then

$$\sigma''_{[1]} \in Tr_{IOA_{Sol}}.O_B \wedge \sigma''_{[1]} \in Tr_B \Rightarrow \sigma''_{[1]} \in Tr_{IOA_{Sol}}$$

Assume that $\sigma''_{[k]} \in Tr_{IOA_{Sol}}$ for $1 \leq k < n$, we put $\sigma''_{[k+1]} = \sigma''_{[k]}.u$

If $u \in I_B$, since $(C||IOA_{Sol}) \leq_{\mathbb{S}} IOA_A$ then

$$Pr_C(\sigma''_{[k+1]}) \in Tr_C \wedge Pr_B(\sigma''_{[k+1]}) \in Tr_{IOA_{Sol}}.I_B \wedge Pr_{IOA_A}(\sigma''_{[k+1]}) \in Tr_{IOA_A} \Rightarrow \sigma''_{[k+1]} \in Tr_{IOA_{Sol}}$$

If $u \in O_B$, since $B \leq_{\mathbb{S}} IOA_{Sol}$ then

$$\sigma''_{[k+1]} \in Tr_{IOA_{Sol}}.O_B \wedge \sigma''_{[k+1]} \in Tr_B \Rightarrow \sigma''_{[k+1]} \in Tr_{IOA_{Sol}}$$

By the principle of induction we have $\sigma'' \in Tr_{IOA_{Sol}}$

Therefore $\sigma'' \in Tr_{(C||IOA_{Sol}||IOA_A)}$

Since $(C||IOA_{Sol}) \leq_P A$, there exists $Y_4 \subseteq Tr_{(C||IOA_{Sol}||IOA_A)}$ such that $Y_4 \neq \emptyset$, $Y_5 = Pr_B(Y_4) \subseteq OCT_{Sol}$ $(s_{0IOA_{Sol}})$ and for each element $\beta \in Y_4$, σ'' is a prefix of β and $Pr_{IOA_A}(\beta) \in OCT_A(s_{0IOA_A})$

Since $B \leq_P Sol$, there exists $\sigma_9 \in Tr_B(s_{0B})$ such that $\sigma_9 \in Y_5$

Moreover there exists $\sigma_{10} \in Y_4$ such that $\sigma_{10} \in Tr_{(C||IOA_{Sol}||IOA_A)}$ and $Pr_B(\sigma_{10}) = \sigma_9$

Therefore $\sigma_{10} \in Tr_{(C||B||IOA_A)}$ and $Pr_{IOA_A}(\sigma_{10}) \in OCT_A(s_{0A})$.

Consider $\sigma = \sigma_1.t \in Tr_{(C||B)}$ such that $t \in (I_{IOA_A} \cup O_{IOA_A})$ and $\sigma_2 = Pr_{IOA_A}(\sigma) \in Tr_{IOA_A}$,

We put $\sigma' = Pr_B(\sigma)$ and let $|\sigma'|$=n, we show by induction that $\sigma' \in Tr_{IOA_{Sol}}$

If $\sigma'_{[1]} \in I_B$, then there exists a prefix σ_3 of σ such that $Pr_B(\sigma_3) = \sigma'_{[1]}$

Since $(C||IOA_{Sol}) \leq_{\mathbb{S}} IOA_A$ Then

$Pr_C(\sigma_3) \in Tr_C \wedge Pr_B(\sigma_3) \in Tr_{IOA_{Sol}}.I_B \wedge Pr_{IOA_A}(\sigma_3) \in Tr_{IOA_A} \Rightarrow \sigma'_{[1]} \in Tr_{IOA_{Sol}}$

If $\sigma'_{[1]} \in O_B$, since $B \leq_{\mathbb{S}} IOA_{Sol}$ then $\sigma'_{[1]} \in O_B \wedge \sigma'_{[1]} \in Tr_B \Rightarrow \sigma'_{[1]} \in Tr_{IOA_{Sol}}$

Assume that $\sigma'_{[k]} \in Tr_{IOA_{Sol}}$ for $1 \leq k < n$, we put $\sigma'_{[k+1]} = \sigma'_{[k]}.u$

If $u \in I_B$, then there exists a prefix σ_4 of σ such that $Pr_B(\sigma_4) = \sigma'_{[k+1]}$

Since $(C||IOA_{Sol}) \leq_{\mathbb{S}} IOA_A$ Then

$Pr_C(\sigma_4) \in Tr_C \wedge Pr_B(\sigma_4) \in Tr_{IOA_{Sol}}.I_B \wedge Pr_A(\sigma_4) \in Tr_A \Rightarrow \sigma'_{[k+1]} \in Tr_{IOA_{Sol}}$

If $u \in O_B$, since $B \leq_{\mathbb{S}} IOA_{Sol}$ then $\sigma'_{[k+1]} \in Tr_{IOA_{Sol}}.O_B \wedge \sigma'_{[k+1]} \in Tr_B \Rightarrow \sigma'_{[k+1]} \in Tr_{IOA_{Sol}}$

By the principle of induction we have $\sigma' \in Tr_{IOA_{Sol}}$

Therefore $\sigma \in Tr_{(C||IOA_{Sol}||IOA_A)}$

iii - Now suppose that $MT_A((s_{0IOA_A})_{\sigma_2}) \neq \emptyset$ and consider an $Y \in MT_A((s_{0IOA_A})_{\sigma_2})$,

By construction of *Sol* there exists $Y_2 \subseteq Tr_{(C||IOA_{Sol}||IOA_A)}((s_{0(C||IOA_{Sol}||IOA_A)})_{\sigma})$ such that $Y_2 \neq \emptyset$, and for each element $\beta \in Y_2$ $Pr_{IOA_A}(\beta) \in Y$

a - $Y_2 \cap O_C \neq \emptyset$

There exists $y \in Y_2 \cap O_C$ such that $\sigma y \in Tr_{(C||IOA_{Sol}||IOA_A)}$

Since $(C||B) \leq_{\S} IOA_A$ then

$Pr_C(\sigma y) \in Tr_C \wedge Pr_B(\sigma y) \in Tr_B.I_B \wedge Pr_{IOA_A}(\sigma y) \in Tr_{IOA_A} \Rightarrow \sigma y \in Tr_{(C||B||IOA_A)}$

Therefore $Pr_{IOA_A}(Tr_{(C||B)}((s_{0(C||B)})_\sigma)) \cap Y \neq \emptyset$

b - $Y_2 \cap O_C = \emptyset$

$Y_3 = Pr_B(Y_2) \in MT_{Sol}\ ((s_{0IOA_{Sol}})_{\sigma'})$

Since $B \leq_P Sol$, there exists $\sigma_5 \in Y_3$ such that $\sigma_5 \in Tr_B((s_{0B})_{\sigma'})$

Therefore there exists $\sigma_6 \in Y_2$ such that $Pr_B(\sigma_6) = \sigma_5$

Since $\sigma.\sigma_6 \in Tr_{(C||IOA_{Sol}||IOA_A)}$ then $Pr_C(\sigma.\sigma_6) \in Tr_C$

Moreover $Pr_B(\sigma.\sigma_6) = \sigma'.\sigma_5 \in Tr_B$, Then $\sigma.\sigma_6 \in Tr_{(C||B)}$

Therefore $Pr_{IOA_A}(Tr_{(C||B)}((s_{0(C||B)})_\sigma)) \cap Y \neq \emptyset$

iv - Suppose that there exists $\sigma'' \in Tr_{(C||B)}((s_{0(C||B)})_\sigma)$ such that $Pr_{IOA_A}(\sigma'') = \varepsilon$

Let $|\sigma''| = n$, we show by induction that $\sigma'' \in Tr_{IOA_{Sol}}$

If $\sigma''_{[1]} \in I_B$, then there exists a prefix σ_7 of $\sigma.\sigma''$ such that $Pr_B(\sigma_7) = \sigma'.\sigma''_{[1]}$

Since $(C||IOA_{Sol}) \leq_{\S} IOA_A$ then

$Pr_C(\sigma_7) \in Tr_C \wedge Pr_B(\sigma_7) \in Tr_{IOA_{Sol}}.I_B \wedge Pr_{IOA_A}(\sigma_7) \in Tr_{IOA_A} \Rightarrow \sigma'.\sigma''_{[1]} \in Tr_{IOA_{Sol}}$

If $\sigma''_{[1]} \in O_B$, since $B \leq_{\S} IOA_{Sol}$ then

$$\sigma'.\sigma''_{[1]} \in Tr_{IOA_{Sol}}.O_B \wedge \sigma'.\sigma''_{[1]} \in Tr_B \Rightarrow \sigma'.\sigma''_{[1]} \in Tr_{IOA_{Sol}}$$

Assume that $\sigma'.\sigma''_{[k]} \in Tr_{IOA_{Sol}}$ for $1 \leq k < n$, we put $\sigma'.\sigma''_{[k+1]} = \sigma'.\sigma''_{[k]}.u$

If $u \in I_B$, then there exists a prefix σ_8 of $\sigma.\sigma''$ such that $Pr_B(\sigma_8) = \sigma'.\sigma''_{[k+1]}$

Since $(C||IOA_{Sol}) \leq_{\S} IOA_A$ then

$Pr_C(\sigma_8) \in Tr_C \wedge Pr_B(\sigma_8) \in Tr_{IOA_{Sol}}.I_B \wedge Pr_{IOA_A}(\sigma_8) \in Tr_{IOA_A} \Rightarrow \sigma'.\sigma''_{[k+1]} \in Tr_{IOA_{Sol}}$

If $u \in O_B$, since $B \leq_{\S} IOA_{Sol}$ then

$$\sigma'.\sigma''_{[k+1]} \in Tr_{IOA_{Sol}}.O_B \wedge \sigma'.\sigma''_{[k+1]} \in Tr_B \Rightarrow \sigma'.\sigma''_{[k+1]} \in Tr_{IOA_{Sol}}$$

By the principle of induction we have $\sigma'.\sigma'' \in Tr_{IOA_{Sol}}$

Therefore $\sigma.\sigma'' \in Tr_{(C||IOA_{Sol}||IOA_A)}$

Since $(C||IOA_{Sol}) \leq_P A$, there exists $Y_4 \subseteq Tr_{(C||IOA_{Sol}||IOA_A)}((s_{0(C||IOA_{Sol}||IOA_A)})_\sigma)$ such that $Y_4 \neq \emptyset$, $Y_5 = Pr_B(Y_4) \subseteq OCT_{Sol}\ ((s_{0IOA_{Sol}})_{\sigma'})$ and for each element $\beta \in Y_4$, σ'' is a prefix of β and $Pr_{IOA_A}(\beta) \in OCT_A((s_{0IOA_A})_{\sigma_2})$

Since $B \leq_P Sol$, there exists $\sigma_9 \in Tr_B((s_{0B})_{\sigma'})$ such that $\sigma_9 \in Y_5$

Moreover there exists $\sigma_{10} \in Y_4$ such that $\sigma.\sigma_{10} \in Tr_{(C||IOA_{Sol}||IOA_A)}$ and $Pr_B(\sigma_{10}) = \sigma_9$

Therefore $\sigma.\sigma_{10} \in Tr_{(C||B||IOA_A)}$ and $Pr_{IOA_A}(\sigma_{10}) \in OCT_A((s_{0A})_{\sigma_2})$.

We conclude that $(C||B) \leq_P A$. □