Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
Electronic Notes in Theoretical Computer Science 250 (2009) 105–122
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Compositional Failure-based Equivalence of Constraint Automata

Mohammad Izadi[1]

Department of computer Engineering, Sharif University of Technology, Tehran, Iran.
Research Institute for Humanities and Cultural Studies (IHCS), Tehran, Iran.
Leiden Institute of Advanced Computer Science (LIACS), Leiden University, The Netherlands.

Ali Movaghar[2]

Department of computer Engineering, Sharif University of Technology, Tehran, Iran.

Abstract

Reo is a coordination language for modeling component connectors of component-based computing systems. Constraint automaton, as an extension of finite automaton, has been proposed as the operational semantics of Reo. In this paper, we introduce an extended definition of constraint automaton by which, every constraint automaton can be considered as a labeled transition system and each labeled transition system can be translated into a constraint automaton. We show that failure-based equivalences CFFD and NDFD are congruences with respect to composition of constraint automata using their join (production) and hiding operators. Based on these congruency results and by considering the temporal logic preservation properties of CFFD and NDFD equivalences, they can be used for reducing sizes of models before doing model checking based verification.

Keywords: Constraint Automata, Failure-based Equivalences, Component-based Systems, Compositional Model Checking, Equivalence-based Reduction.

1 Introduction

The concept of *component based systems*, especially component based software, is a philosophy or way of thinking to deal with the complexity in designing large scale computing systems. One of the main goals of this approach is to compose reusable components by some glue codes. The model or the way in which these components are composed is called *coordination model*. Sometimes there are some formal or programming languages which are used for specification of coordination models. Such languages are called as *coordination languages*. Reo, as one of the

[1] Email: Izadi@ce.sharif.edu
[2] Email: Movaghar@sharif.edu

doi:10.1016/j.entcs.2009.08.008

most recently proposed coordination languages, is a channel based exogenous coordination language in which complex coordinators are compositionally built out of simpler ones [1,3,2]. By using Reo specifications, complex component connectors can be organized in a network of channels and build in a compositional manner. Reo relies on a very liberal and simple notion of channels and can model any kind of peer-to-peer communication. The channels used in Reo networks can be considered as simple communicating processes and the only requirements for them are that channels should have two ends (or I/O interfaces), declared to be *sink* or *source* ends, and a user-defined semantics. At source ends data items enter the channel by performing corresponding write operations. Data items are received from a channel at sink ends by performing corresponding read operations. Reo allows for an open ended set of channel types with user defined semantics.

If we want to be able to reason about properties of specifications or verify their correctness, Reo, as well as any other process specification languages, should be given abstract semantics. Generally, when the ultimate aim is reasoning about or verification the properties of specifications, the language is modeled by some kind of transition systems. Labeled transition systems and automata are examples of these semantic models. The key question in giving a semantic model to a specification language is: "Whenever can we say that two specifications or two models are equivalent?" Numerous definitions of different equivalence-relations for transition system based models have been presented in the literature of process algebra, automata theory and theories of concurrency. *Trace equivalence* (automata-theoretic equivalence), *weak bisimilarity* presented by Milner [14] and *failure-based equivalences* (CSP-like equivalences) such as the equivalence presented by Hoare [10] are examples of these equivalences. We say that an equivalence relation R_1 is *stronger* than R_2, if whenever two models be equivalent with respect to R_1, they are also equivalent with respect to R_2. Equivalence relations R_1 and R_2 are *incomparable*, if neither R_1 is stronger than R_2, nor R_2 is stronger than R_1.

The main goal of verification methods is trying to ascertain that an actual system or program satisfies its requirements. In formal verification methods one tries to achieve this aim by describing the system using a mathematical model, expressing the requirements as properties of this model and showing through rigorous mathematical reasoning that the model of the system indeed has the required properties [6,13]. If the correctness requirements of a formally modeled computing system are given in a mathematical notion, such as linear or branching time temporal logics or automata on infinite objects, an algorithmic model theoretic process called *model checking* [6] can be used to check if the system respects its correctness requirements. Model checking has shown to be an efficient and easy to use technique in verification of computer systems. However, there is a major drawback in using exhaustive model checking: the model of the system tends to be extremely large. In literature this problem is often referred as the *state explosion problem*.

During the last two decades, many methods have been suggested to reduce the number of states which need to be constructed for answering certain verification or analysis questions. Such enhanced state space methods increase substantially the

size of systems that can be verified, while preserving most of the advantages of the model checking method of verification. Compositional verification and its special case, equivalence based compositional reduction [12,16], partial order reduction by representatives [15], the pre-order reduction [9], abstraction [5] and using the symmetry properties [8] are the main of these methods for dealing with state explosion problem. In the compositional verification of a system, one seeks to verify properties of the system from properties of its constituent modules [6,7,16]. In general, compositional verification may be exploited more effectively when the model is naturally decomposable [17]. In the method of *equivalence based compositional reduction* components of a system are reduced with respect to an equivalence relation before building the model of the complete system [9,6,11].

An equivalence relation should have two properties in order to be useful in the equivalence based compositional reduction method: it should *preserve* the class of properties to be verified and also, it should be a *congruence* with respect to the syntactic operators which are used for composing of the components of the model. By congruence relation we mean that the replacement of a component of a model by an equivalent one should always yield a model which is equivalent with the original one. Fortunately, in the context of compositional failure based semantic models of process description languages such as CCS and LOTOS, there are two equivalence relations, introduced by Valmari et al. and called CFFD and NDFD, which have the preservation property: CFFD-equivalence preserves that fragment of linear time temporal logic which has no next-time operator and has an extra operator distinguishing deadlocks [19,20] and NDFD-equivalence preserves linear time temporal logic without next-time operator [12]. It was also shown that CFFD and NDFD are the minimal equivalences preserving the above mentioned fragments of linear time temporal logic. In [20], it was also shown that if we use labeled transition systems as semantic models, CFFD an NDFD are congruences with respect to all composition operators defined in LOTOS.

Constraint automaton, as an extension of finite or Buchi automaton, is a formalism proposed to capture the operational semantics of Reo [4]. In a constraint automaton, contrary to finite automata and labeled transition systems, the label of a transition is not a simple character or action name. A transition label contains a set of names and a (constraint) proposition. The set of names indicates the names of ports which are participant in doing the transition and the proposition expresses some constraint about the data of the ports. In [4], constraint automaton has been defined as the semantic model of Reo and also, trace-based and weak (bi)simulation-based equivalences of constraint automata have been presented. In this paper, we are interested to investigate failure-based equivalences CFFD and NDFD for constraint automata and their congruency with respect to composition operators which are useful in composing Reo specifications.

The ultimate goal of this paper is to prepare an environment for compositional model checking of Reo specifications modeled by constraint automata using equivalence based reduction method. For this purpose we introduce an extended definition of constraint automaton by which every constraint automaton can be considered as

a labeled transition system and each labeled transition system can be translated into a constraint automaton. Also, we introduce two new composition operators for constraint automata: join (production) of two automata with respect to their common port names and hiding of a port name in all transition labels of an automaton. We show that failure-based equivalences CFFD and NDFD are congruence with respect to join and hiding operators of constraint automata (see Subsections 4.1 and 4.2). Based on these congruency results and because of the linear time temporal logic preservation properties of CFFD and NDFD equivalences and their minimality properties (proved in [12]), they will useful candidates for compositional reduction of models in the field of model checking in future works.

The paper proceeds as follows: in section 2, we define the notion of *labeled transition systems*, some composition operators for them and two equivalence relations CFFD and NDFD. In section 3, we briefly introduce the traditional definition of constraint automata. Then, we present a new definition of constraint automata by which each labeled transition system can be translated to a constraint automaton and vice-versa. Also, in this section we introduce two composition operators for our defined constraint automata: join (production) and hiding. In section 4, we prove that CFFD and NDFD-equivalences are congruences with respect to join and hiding of constraint automata. In section 5, we conclude and discuss some about the results of this work.

2 Preliminaries and Basic Definitions

In this section, we define the notion of *labeled transition systems* and introduce CFFD and NDFD-equivalence relations on them based on the papers introduced them [19,20,12].

Definition 2.1 A *transition alphabet* is a countable infinite set Σ not containing the empty transition label τ. We write Σ_τ for $\Sigma \cup \{\tau\}$, and Σ^* (Σ^ω) for the set of all finite (infinite) words consisting of elements of Σ. The symbol τ is used to denote the empty word. If $\sigma \in (\Sigma_\tau^* \cup \Sigma_\tau^\omega)$, $vis(\sigma)$ is used to denote the word obtained by removing all τ-symbols from σ and $\Sigma(\sigma)$ denote the set of elements of σ. A *labeled transition system (lts)* is a triple $L = (S, s, \Delta)$, where S is the set of states, $s \in S$ is the initial state and $\Delta \subseteq S \times \Sigma_\tau \times S$ is the transition relation. The alphabet of L , $\Sigma(L)$ is the following set: $\Sigma(L) = \{l \in \Sigma |\ \exists s, s' : (s, l, s') \in \Delta\}$. The alphabet of any *lts* is required to be finite.

Now, we recall some basic concepts of process algebra and give the definitions of CFFD and NDFD-equivalences. For a more detailed discussion of these equivalences and the intuitions behind them see [19,20,12].

Definition 2.2 Let $L = (S, s, \Delta)$ be a labeled transition system (lts).
If $\rho \in \Sigma_\tau^*$, we write $s_0 \xrightarrow{\rho} s_n$ for $n = |\rho|$ iff there are $s_1, ..., s_{n-1}$ such that for all $0 < i \leq n$, $(s_{i-1}, \rho_i, s_i) \in \Delta$. If there is an s_n such that $s_0 \xrightarrow{\rho} s_n$ we write $s_0 \xrightarrow{\rho}$. If $\rho \in \Sigma_\tau^\omega$, we write $s_0 \xrightarrow{\rho}$ iff $\exists s_1, s_2, ...$ such that for all $i > 0$, $(s_{i-1}, \rho_i, s_i) \in \Delta$. If $\sigma \in (\Sigma^* \cup \Sigma^\omega)$, we write $s_0 \stackrel{\sigma}{\Longrightarrow} s_n$ ($s_0 \stackrel{\sigma}{\Longrightarrow}$) iff there is a $\rho \in (\Sigma_\tau^* \cup \Sigma_\tau^\omega)$ such that

$s_0 \xrightarrow{\rho} s_n$, $(s_0 \xrightarrow{\rho})$ and $\sigma = vis(\rho)$.
- $\sigma \in \Sigma^*$ is a *trace* of L iff $s \stackrel{\sigma}{\Longrightarrow}$. $tr(L)$ is the set of all traces of L.
- $\sigma \in \Sigma^\omega$ is an *infinite trace* iff $s \stackrel{\sigma}{\Longrightarrow}$. $inftr(L)$ is the set of all infinite traces of L.
- $\sigma \in \Sigma^*$ is a *divergence trace* of L iff there is a $\rho \in \Sigma_\tau^\omega$ such that $s \xrightarrow{\rho}$ and $\sigma = vis(\rho)$. $divtr(L)$ is the set of all divergence traces of L.
- $s' \in S$ is *stable*, if not $s' \xrightarrow{\tau}$. Lts L is stable if the initial state s is stable. We write $stable(L)$ if L is stable, and $\neg stable(L)$ if it is not.
- $(\sigma, A) \in \Sigma^* \times 2^\Sigma$, where 2^Σ denotes the power set of Σ, is a *failure* of L iff there is an $s' \in S$ such that $s \stackrel{\sigma}{\Longrightarrow} s'$ and $\forall a \in A. \neg(s' \stackrel{a}{\Longrightarrow})$.
- $(\sigma, A) \in \Sigma^* \times 2^\Sigma$ is a *stable failure* of L iff there is a stable $s' \in S$ such that $s \stackrel{\sigma}{\Longrightarrow} s' \wedge \forall a \in A. \neg(s' \stackrel{a}{\Longrightarrow})$. $sfail(L)$ is the set of all stable failures of L.
- $(\sigma, A) \in \Sigma^* \times 2^\Sigma$ is a *nondivergent failure* of L iff (σ, A) is a failure and σ is not a divergence trace. $ndfail(L)$ is the set of all nondivergent failures of L.
- $(\sigma, A) \in \Sigma^* \times 2^\Sigma$ is a *divergence-masked failure* of L iff (σ, A) is a failure or σ is a divergence trace. $dfail(L)$ is the set of divergence-masked failures of L.
- $\sigma \in \Sigma^*$ is a *deadlock trace* iff (σ, Σ) is a stable failure of L. $dtr(L)$ is the set of all deadlock traces of L.
- $\sigma \in \Sigma^*$ is a *nondivergent deadlock trace* of L iff (σ, Σ) is a nondivergent failure of L. $nddtr(L)$ is the set of all nondivergent deadlock traces of L.Note that, $nddtr(L) = dtr(L) - divtr(L)$.
- The set of all *nondivergent traces* of L is $ndtr(L) = \{\sigma | (\sigma, \emptyset) \in ndfail(L)\}$.

The following proposition lists some consequences of the definitions for later use (for proofs see [20]).

Proposition 2.3 *Let L be an lts.*
a) $tr(L) = divtr(L) \cup \{\sigma | (\sigma, \emptyset) \in sfail(L)\} = divtr(L) \cup \{\sigma | (\sigma, \emptyset) \in ndfail(L)\}$.
b) $tr(L) = \{\sigma | (\sigma, \emptyset) \in fail(L)\} = \{\sigma | (\sigma, \emptyset) \in dfail(L)\}$.
c) $ndfail(L) = sfail(L) - (divtr(L) \times 2^\Sigma)$.
d) $dfail(L) = sfail(L) \cup (divtr(L) \times 2^\Sigma)$.
e) $tr(L) = divtr(L) \cup ndtr(L)$.
f) $divtr(L) \cap ndtr(L) = \emptyset$.
g) If L be a finite lts, then,
$inftr(L) = \{\omega \in \Sigma^\omega | \forall \sigma \in \Sigma^* : (\sigma \ is \ a \ proper \ prefix \ of \ \omega \rightarrow \sigma \in tr(L))\}$.

On the basis of the above definitions and propositions, the equivalence concepts can be easily defined.

Definition 2.4 (i) Let L and L' be ltss. We say that L and L' are CFFD equivalent and write $L \stackrel{cffd}{\approx} L'$ iff $stable(L) \Leftrightarrow stable(L')$ and $divtr(L) = divtr(L')$ and $inftr(L) = inftr(L')$ and $sfail(L) = sfail(L')$.
(ii) Let L and L' be ltss. We say that L and L' are NDFD equivalent and write $L \stackrel{ndfd}{\approx} L'$ iff $stable(L) \Leftrightarrow stable(L')$ and $divtr(L) = divtr(L')$ and $inftr(L) = inftr(L')$ and $ndfail(L) = ndfail(L')$.

The NDFD-equivalence is strictly weaker than CFFD-equivalence in the sense

of the following theorem [20]:

Proposition 2.5 *If* $L \overset{cffd}{\approx} L'$, *then* $L \overset{ndfd}{\approx} L'$.

If the labeled transition systems examined are finite, the component $inftr$ in the above definition is superfluous. This corresponds to the original definition of CFFD-equivalence in [19], where only finite ltss were considered. Also, it can be shown that it is possible to replace $ndfail(L)$ by $dfail(L)$ in the definition of NDFD-equivalence [20]. Thus, we have:

Proposition 2.6 *Let* L *and* L' *be two finite ltss.*
1- $L \overset{cffd}{\approx} L'$ *iff* $stable(L) \Leftrightarrow stable(L')$, $divtr(L) = divtr(L')$, *and* $sfail(L) = sfail(L')$.
2- $L \overset{ndfd}{\approx} L'$ *iff* $stable(L) \Leftrightarrow stable(L')$, $divtr(L) = divtr(L')$, *and* $dfail(L) = dfail(L')$.

If two systems or processes A and B are CFFD equivalent, then the intuitive meaning is that both have the same computation sequences and furthermore, A can deadlock after a sequence of actions if and only if B can also deadlock after the same sequence of actions. Also, the computation sequences that lead to a divergence (infinite sequence of internal actions) are the same for the two processes.

Definition 2.7 An equivalence $\approx$ between ltss is a *congruence* with respect to a syntactic operator f iff for every $L_1, \ldots, L_n$ and $L'_1, \ldots, L'_n$ such that $L_i \approx L'_i$ the following holds: $f(L_1, \ldots, L_n) \approx f(L'_1, \ldots, L'_n)$.

In [20,12] it has been shown that, CFFD and NDFD equivalences are congruences with respect to the composition operators defined for labeled transition systems based semantics of basic LOTOS and CSP. In the rest of this paper we investigate the congruency of these equivalences with respect to composition operators of constraint automata.

3 Constraint Automata

Constraint automata have been introduced as the operational semantics of Reo specifications [4]. In this section, after a short review on Reo, we present the original definition of constraint automata (as the acceptors of languages of timed data streams, presented in [4]. Then we introduce an extended definition of them such that they can be considered as labeled transition systems and each label transition system can be translated into a constraint automaton. Also, in this section we introduce two composition operators for our defined constraint automata: join (production) of two constraint automata with respect to their common port names and hiding of a name from all transition labels of a constraint automaton. For more about Reo specification language and constraint automata as its semantics model, see [1,3,2,4]

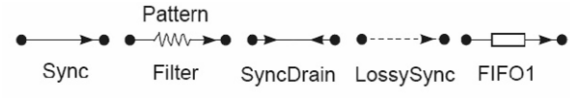

Fig. 1. Basic channel-types in Reo

3.1 Reo and its Operational Semantics

Reo is an exogenous coordination language which is based on a calculus of channels [1,2,4]. By using Reo specifications, complex component connectors can be organized in a network of channels and build in a compositional manner. Reo relies on a very liberal and simple notion of channels and can model any kind of peer-to-peer communication. The only requirements for the channels used in Reo networks are that channels should have two channel ends, declared to be *sink* or *source* ends, and a user-defined semantics. At source ends data items enter the channel by performing corresponding write operations. Data items are received from a channel at its sink end by performing corresponding read operations. It has been proved that the set of channel-types shown in Figure 1 by their graphical representations, is an expressively complete set of channels [1]. A complex connector has a graphical representation, called *Reo circuit or network*. The nodes of a Reo network represent sets of channel ends. They arise through Reo's *join* operator and can be classified into *source, sink* and *mixed* nodes, depending on whether all channel ends that coincide on a node A are source ends (then A is a source node), sink ends (then A is a sink node) or whether A combines sink and source ends (then A is a mixed node). Source and sink nodes represent input and output ports where components might connect to the network. The mixed nodes serves as routers where data items can be transmitted through the network. For more about Reo networks and their examples see [1,2,3,4].

Now, we can introduce the original definition of constraint automata as the semantic model of Reo [4].

Definition 3.1 Let N be a set of port names and $Data$ be a set of data. A *data constraint* g over names set N and data set $Data$ is a proposition, which can be constructed by using the following grammar:

$g \quad ::= \quad true \mid d_A = d \mid g_1 \vee g_2 \mid \neg g \qquad d \in Data, \quad A \in N$

We use $DC(N, Data)$ as the set of all data constraints over names set N and data set $Data$, defined by the above grammar.

Definition 3.2 A *constraint automaton* is a quadruple $C = (Q, Names, T, Q_0)$, where, Q is a finite set of states, $Names$ is a finite set of names, $Q_0 \subseteq Q$ is the set of initial states, $T \subseteq Q \times 2^{Names} \times DC(Names, Data) \times Q$ is a set of transitions of C. If $(p, N, g, q) \in T$, we call N the name set and g the guard of the transition. It is required that, $N \neq \emptyset$ and $g \in DC(N, Data)$. The constraint automaton C is finite iff the sets Q, T and $Data$ be finite.

The intuitive operational behavior of a constraint automaton is as follows. It starts in its initial state q_0. If the current state is q, then C waits until data

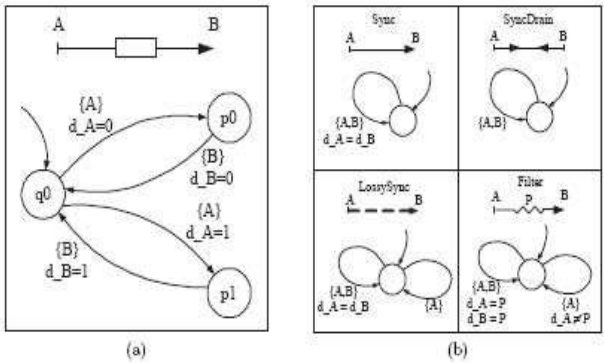

Fig. 2. Constraint automata for some basic channels in Reo

items occur at some of its ports $A_1, ..., A_n$. Suppose data item d_1 occurs at A_1 and data item d_2 at A_2 while (at this moment) no data is observed at the other ports $A_3, ..., A_n$. This triggers the automaton to check the data Constraints of the outgoing transitions of state q with a name set $\{A_1, A_2\}$ to choose a transition t, such that its guard is satisfied by $A_1 \mapsto d_1$ and $A_2 \mapsto d_2$ resulting in state p. If there is no $\{A_1, A_2\}$-transition from q whose data Constraint is fulfilled then C rejects. Figure 2 shows Constraint automata for some basic channels in Reo.

Definition 3.2 is the original definition of constraint automaton (presented in [4]). In this paper, we will use a modified definition of constraint automaton (such as it will be defined in Definition 3.3). Thus, we sometimes will refer to automaton defined in Definition 3.2 as *traditional* constraint automaton.

3.2 Constraint Automata as Labeled Transition Systems

Constraint automata, as were presented in [4] are used as the models of component connectors of a component based system. A component based system contains both components and connectors. If we consider the whole system, we need to model and compose both components and connectors. Basically, the components can be modeled by labeled transition systems. Thus, we need to find a way to compose Constraint automata and labeled transition systems with each other. For this purpose we can generalize the definition of constraint automaton such that one can consider it as a labeled transition system and also translate labeled transition systems to constraint automata. In this section, we investigate a bidirectional translation between constraint automata and labeled transition systems. We introduce a modified definition of constraint automaton where, transitions can be labeled by internal or external actions. The external actions are as defined in traditional Definition 3.2. The internal actions are introduced by τ labels on transitions. Using this definition, each constraint automaton can be considered as a labeled transition system and each labeled transition system can simply be translated to a constraint automaton. Also, in this section we introduce two composition operators for constraint automata using their new definition: production (join) of two constraint automata with respect to their common port names and hiding of a port name from

a constraint automaton. We prove that CFFD and NDFD equivalences, introduced in Section 2, are congruence with respect to these composition operators.

Definition 3.3 Let *Data* be a set of data. A *constraint automaton* is a quadruple $C = (Q, Nam, T, q_0)$ where, Q is a finite set of states, Nam is a finite set of names, $T \subseteq Q \times (2^{Nam} \times DC(Nam, Data)) \times Q$ is the transition relation, and $q_0 \in Q$ is the initial state.
We write $p \xrightarrow{N,g} q$ instead of $(p, N, g, q) \in T$ and call N the name set and g the guard of the transition. For each $(p, N, g, q) \in T$, it is required that $g \in DC(N, Data)$. Note that $DC(\emptyset, Data) = \{true, false\}$. We use τ as a shorthand symbol for the transition label $(\emptyset, true)$. In other words, transition $p \xrightarrow{\tau} q$ is the same as $p \xrightarrow{\emptyset,true} q$.

The main differences of our definition of constraint automaton and its original definition (defined in [4]) are: 1- In the new definition, $\tau - transtion$ is permitted, while in its original definition it is not. We need $\tau - transtion$ because of two reasons: first that $\tau - transtion$ can be used as a symbol for each kind of internal action which is occurred in the actual system but its real type is not important in modeling by a constraint automaton, second that, the *hiding* operators can hide all port-names of a transition. In such cases, we replace the transition label by τ. 2- We supposed that the initial state of a constraint automaton is unique because $\tau - transtions$ are permitted. We can simulate multiple initial states by using a unique initial state and $\tau - transtions$ from it to other possible initial states. 3- Our definition of constraint automaton is departed from the original one by dropping the requirement that all runs have to be infinite. We also deal with finite runs, which are necessary to argue about deadlock configurations.

Obviously, if we consider constraint automata as defined in Definition 3.3, each labeled transition system can be translated to a constraint automaton. For this goal, we should save the internal actions by using $\tau - transitions$ and for external actions we should determine their (input or output) ports names and constraints (if they are necessary). On the other hand, each constraint automaton can be considered as a labeled transition system with alphabet $\Sigma = \{(N, g) | N \subseteq Nam \wedge g \in DC(N, Data) \wedge N \neq \emptyset\}$. Thus,

Proposition 3.4 *Let $C = (Q, Nam, T, q_0)$ be a constraint automaton over data set data, such as defined in Definition 3.3. C can be considered as a lts $L = (S, s, \Delta)$ over alphabet Σ, such as defined in Definition 2.1, where, $S = Q$, $s = q_0$,*
$\Sigma = \{(N, g) | N \subseteq Nam \wedge g \in DC(N, Data) \wedge N \neq \emptyset\}$,
$(q_i, (N, g), q_j) \in \Delta \Leftrightarrow (q_i, N, g, q_j) \in T$ and $(q_i, \tau, q_j) \in \Delta \Leftrightarrow (q_i, \tau, q_j) \in T$.

Based on Proposition 3.4, the definitions of traces and failures for constraint automata will be the same as defined in Definition 2.2.

3.3 Composing Constraint Automata

Because constraint automata are intended to capture the operational semantics of Reo, we need two composition operators for composing or reconstructing them: production (join) of two constraint automata with respect to their common port names

and hiding a port name from a constraint automaton. In this section, we present our definitions of these two composition operators using the new definition of constraint automaton. These definitions are generalizations of their original counterparts and save all properties proved for them in [4].

Definition 3.5 Let $C_1 = (Q_1, Nam_1, T_1, q_{01})$ and $C_2 = (Q_2, Nam_2, T_2, q_{02})$ be two constraint automata. The *product (join) constraint automaton* of C_1 and C_2 is: $C_1 \bowtie C_2 = (Q_1 \times Q_2, Nam_1 \cup Nam_2, T, q_{01} \times q_{02})$ in which, T is defined as follow:
1) If $(q_1, N_1, g_1, p_1) \in T_1$, $(q_2, N_2, g_2, p_2) \in T_2$, $N_1 \neq \emptyset$, $N_2 \neq \emptyset$ and $N_1 \cap Nam_2 = N_2 \cap Nam_1$, then, $(< q_1, q_2 >, N_1 \cup N_2, g_1 \wedge g_2, < p_1, p_2 >) \in T$,
2) If $(q, N, g, p) \in T_1$ and $N \cap Nam_2 = \emptyset$, then, $(< q, q' >, N, g, < p, q' >) \in T$,
3) If $(q, N, g, p) \in T_2$ and $N \cap Nam_1 = \emptyset$, then, $(< q', q >, N, g, < q', p >) \in T$.

Because the above definition of product is a generalization of its original counterpart, if we restrict the alphabet of the language of a constraint automaton to its observable elements (consider all transition labels of the form (N, g) as observable and ignore τ in all words), it saves all properties which have been shown for its counterpart in [4].

Definition 3.6 Let $C = (Q, Nam, T, q_0)$ be a constraint automaton and B be a name, $B \in Nam$. The constraint automaton resulted by *hiding* of B in A is $\exists B[C] = (Q, Nam \backslash \{B\}, T_{\exists B}, q_0)$ in which, T is defined as follow:
(1) If $(q, N, g, p) \in T$ and $N \neq \emptyset$ then $(q, N \backslash \{B\}, \exists B[g], p) \in T_{\exists B}$, where $\exists B[g] = \vee_{d \in Data} g[d_B/d]$.
(2) If $(q, \tau, p) \in T$ then, $(q, \tau, p) \in T_{\exists B}$.

In addition to the above two composition operators, based on Proposition 3.4, we can compose constraint automata using any other well defined composition operators defined for label transition systems.

4 Congruency Results

In this section we investigate the congruency of CFFD and NDFD equivalences with respect to the join (production) and hiding composition of constraint automata. This section contains two parts, in the first we consider the join operator (as defined in Definition 3.5) and in the other we consider the hiding operator (as defined in Definition 3.6).

4.1 CFFD and NDFD are congruences with respect to join of constraint automata

In this section we prove that equivalences CFFD and NDFD are congruences with respect to production (join) of finite constraint automata as defined in definition 3.5. Our method for the proof is very similar to the methods used by the authors of CFFD and NDFD equivalences for proving their congruences with respect to the composition operators defined in [20]. First, We define a predicate $\mathbf{Join}(\sigma; \pi, \rho)$, which intuitively means that words π and ρ can be considered as traces of two constraint automata and word σ as a trace in the product (join) constraint automaton

such that σ is the result of the production of ρ and π. Then, we show that the sets of all traces, all stable failures, all divergent traces and all divergence-masked failures of the product automaton can be characterized by their counterparts in the two constraint automaton (see Proposition 4.2). Based on these characterizations, we will prove the congruences. Because our ultimate goal is using of the equivalences in the context of model checking, we will prove the congruences for finite constraint automata.

Definition 4.1 Let $Data$ be a finite set of data and Nam_1 and Nam_2 be two finite sets of names. Let $\Sigma_1 = \{(N,g)|N \subseteq Nam_1 \wedge N \neq \emptyset \wedge\ g \in DC(N, Data)\}$,
$\Sigma_2 = \{(N,g)|N \subseteq Nam_2 \wedge N \neq \emptyset \wedge\ g \in DC(N, Data)\}$,
$\Sigma = \{(N,g)|N \subseteq Nam_1 \cup Nam_2 \wedge N \neq \emptyset \wedge\ g \in DC(N, Data)\}$ and
$\sigma = (N_1,g_1)(N_2,g_2)\ldots$ be a word over alphabet Σ. The predicate $\mathbf{Join}(\sigma;\pi,\rho)$ holds (is true) if and only if the following procedure can obtain words π and ρ from σ, successfully:

1- Define a function $moved$ from $\{1,2,3,\ldots\}$ to $\{"first","second","both"\}$ such that: $moved(i) = "first"$ iff $N_i \cap Nam_2 = \emptyset$ and $g_i \in DC_1$, $moved(i) = "second"$ iff $N_i \cap Nam_1 = \emptyset$ and $g_i \in DC_2$, otherwise $moved(i) = "both"$.
2- For obtaining π from σ do:
2-1- for all $i \geq 1$ which, $moved(i) = "both"$, change (N_i, g_i) to
$(N_i \cap Nam_1, g_i[Nam_1])$,
2-2- remove all (N_i, g_i) which $moved(i) = "second"$ from σ.
3- For obtaining ρ from σ do:
3-1- for all $i \geq 1$ which, $moved(i) = "both"$, change (N_i, g_i) to
$(N_i \cap Nam_2, g_i[Nam_2])$,
3-2- remove all (N_i, g_i) which $moved(i) = "first"$ from σ.

By $g[Nam_i]$ we mean the restriction of proposition g to the name set Nam_i: it can be obtained by removing all terms containing $d_A = d$ where $A \notin Nam_i$ from the conjunctive normal form of g. Obviously, if the above procedure can obtain words π and ρ successfully, π will be a word over alphabet Σ_1 and ρ will be a word over alphabet Σ_2.

Proposition 4.2 *Let $C_1 = (Q_1, Nam_1, T_1, q_{01})$ and $C_2 = (Q_2, Nam_2, T_2, q_{02})$ be two finite constraint automata. Let $C = C_1 \bowtie C_2$ then,*
(i) $tr(C) = \{\sigma \mid \exists\pi \in tr(C_1),\ \exists\rho \in tr(C_2),\ Join(\sigma;\pi,\rho)\}$.
(ii) $sfail(C) = \{(\sigma,A) \mid \exists(\pi,B) \in sfail(C_1),\ \exists(\rho,D) \in sfail(C_2),\ Join(\sigma;\pi,\rho)\ \wedge A \cap G \subseteq B \cap D\ \wedge\ A \cap G' \subseteq B \cup D\}$, *where,*
$G = \{(N,g)|\ N \subseteq Nam_1 \cup Nam_2 \wedge N \neq \emptyset \wedge (N \cap Nam_1 = \emptyset \vee N \cap Nam_2 = \emptyset)\}$,
$G' = \{(N,g)|\ N \subseteq Nam_1 \cup Nam_2 \wedge N \neq \emptyset \wedge (N \cap Nam_1 \neq \emptyset \wedge N \cap Nam_2 \neq \emptyset)\}$.
(iii) $stable(C) = stable(C_1) \wedge stable(C_2)$.
(ix) $divtr(C) = \{\sigma \mid \exists\pi \in tr(C_1),\ \exists\rho \in tr(C_2), Join(\sigma;\pi,\rho)$ *and* $(\pi \in divtr(C_1) \vee \rho \in divtr(C_2))\}$.
(x) $dfail(C) = \{(\sigma,A) \mid \exists(\pi,B) \in dfail(C_1),\ \exists(\rho,D) \in dfail(C_2), Join(\sigma;\pi,\rho)\ \wedge A \cap G \subseteq B \cap D\ \wedge\ A \cap G' \subseteq B \cup D\}\ \cup\ (divtr(C_1 \bowtie C_2) \times 2^{\Sigma})$, *where,* Σ *is the*

same as defined in Definition 4.1 and G and G′ are the same as defined in (ii).

Proof. See Appendix A □

Proposition 4.3 *Let C and C′ be finite constraint automata over the same set of names, D and D′ be finite constraint automata over the same set of names,* $C \overset{cffd}{\approx} C'$ *and* $D \overset{cffd}{\approx} D'$. *Then,* $C \bowtie D \overset{cffd}{\approx} C' \bowtie D'$.

Proof. (i) Based on Proposition 4.2(iii), $stable(C \bowtie D) = stable(C) \wedge stable(D)$. Because of $C \overset{cffd}{\approx} C'$ and $D \overset{cffd}{\approx} D'$, we have, $stable(C) = stable(C')$ and $stable(D) = stable(D')$. Thus, $stable(C \bowtie D) = stable(C' \bowtie D')$.
(ii) Based on Proposition 4.2(ii),
$sfail(C \bowtie D) = \{(\sigma, A) \mid \exists(\pi, B) \in sfail(C), \exists(\rho, E) \in sfail(D), Join(\sigma; \pi, \rho) \wedge A \cap G \subseteq B \cap E \wedge A \cap G' \subseteq B \cup E\}$ where,
$G = \{(N, g) | N \cap Nam_C = \emptyset \vee \ N \cap Nam_D = \emptyset\}$ and
$G' = \{(N, g) | N \subseteq Nam_C \cup Nam_D \wedge N \neq \emptyset \wedge N \cap Nam_C \neq \emptyset \wedge N \cap Nam_D \neq \emptyset\}$.
Because of the CFFD-equivalences $sfail(C) = sfail(C')$ and $sfail(D) = sfail(D')$. Because of the equality of the names sets, G and G' in the case of $C \bowtie D$ are equal with G and G' in the case of $C' \bowtie D'$, respectively. Thus, $sfail(C \bowtie D) = sfail(C' \bowtie D')$.
(iii) Based on Proposition 4.2(ix),
$divtr(C \bowtie D) = \{\sigma \mid \exists \pi \in tr(C), \ \exists \rho \in tr(D), \ Join(\sigma; \pi, \rho)$ $and (\pi \in divtr(C) \vee \rho \in divtr(D))\}$. Based on Proposition 2.3(a), $tr(C) = divtr(C) \cup \{\sigma | (\sigma, \emptyset) \in sfail(C)\}$ and this fact holds also for C', D and D'. Because of CFFD-equivalences $divtr(C) = divtr(C')$, $divtr(D) = divtr(D')$, $sfail(C) = sfail(C')$, $sfail(D) = sfail(D')$. Thus, $tr(C) = tr(C')$ and $tr(D) = tr(D')$. Thus, $divtr(C \bowtie D) = divtr(C' \bowtie D')$. □

Corollary 4.4 *CFFD-equivalence is a congruence with respect to the product (join) of finite constraint automata.*

Proposition 4.5 *Let C and C′ be finite constraint automata over the same set of names, D and D′ be finite constraint automata over the same set of names,* $C \overset{ndfd}{\approx} C'$ *and* $D \overset{ndfd}{\approx} D'$. *Then,* $C \bowtie D \overset{ndfd}{\approx} C' \bowtie D'$.

Proof.
The proofs for claims $stable(C \bowtie D) = stable(C' \bowtie D')$ and $divtr(C \bowtie D) = divtr(C' \bowtie D')$ are the same as in the proof of Proposition 4.3.
Now we prove that, $dfail(C \bowtie D) = dfail(C' \bowtie D')$.
By Proposition 4.2(x), $dfail(C \bowtie D) = \{(\sigma, A) \mid \exists(\pi, B) \in dfail(C), \ \exists(\rho, E) \in dfail(D), \ Join(\sigma; \pi, \rho)$ and $A \cap G \subseteq B \cap E \ \wedge \ A \cap G' \subseteq B \cup E\} \ \cup \ (divtr(C_1 \bowtie C_2) \times 2^{\Sigma})$. Because of $C \overset{ndfd}{\approx} C'$ and $D \overset{ndfd}{\approx} D'$, $dfail(C) = dfail(C')$, $dfail(D) = dfail(D')$ and $divtr(C \bowtie D) = divtr(C' \bowtie D')$. Because of the equality of the names sets, G and G' in the case of $C \bowtie D$ are equal with G and G' in the case of $C' \bowtie D'$, respectively. Thus, $dfail(C \bowtie D) = dfail(C' \bowtie D')$. □

Corollary 4.6 *NDFD-equivalence is a congruence with respect to the product (join) of finite constraint automata.*

4.2 CFFD and NDFD are congruences with respect to hiding of constraint automata

In this section we prove that equivalences CFFD and NDFD are congruences with respect to hiding of port names in finite constraint automata as defined in definition 3.6. First, we show that the sets of all traces, all stable failures, all divergent traces and all divergence-masked failures of the automaton after hiding of a port name in a constraint automaton can be characterized by their counterparts in the original constraint automaton (see Proposition 4.8). Based on these characterizations, we will prove the congruences. Because our ultimate goal is using of the equivalences in the context of model checking, we will prove the congruences for finite constraint automata.

Definition 4.7 Let Nam be a set of names, $Data$ be a set of data, $\Sigma = \{(N, g) | N \subseteq Nam \ \wedge \ g \in DC(N, Data)\}$ and $B \in Nam$. We define the set **hide** B **in** Σ_1, for each set $\Sigma_1 \subseteq \Sigma$ and the word **hide** B **in** σ, for each word $\sigma = (N_1, g_1)(N_2, g_2)\ldots$ such that:
hide B **in** $\Sigma_1 = \{(N \backslash \{B\}, \exists [B] g) | (N, g) \in \Sigma_1 \wedge N \neq \{B\} \wedge N \neq \emptyset\}$.
hide B **in** σ is the word that is obtained after removing all pairs of the form $(\emptyset, g)$ from word $(N_1 \backslash \{B\}, \exists [B] g_1)(N_2 \backslash \{B\}, \exists [B] g_2)\ldots$.

Proposition 4.8 *Let $C = (Q, Nam, T, q_0)$ be a finite constraint automaton and $A = \exists B[C]$ be the constraint automaton resulted by hiding of B in C (for $B \in Nam$). Then,*
(i) $tr(A) = \{\textbf{\textit{hide}}\ B\ \textbf{\textit{in}}\ \sigma |\ \sigma \in tr(C)\}$.
(ii) $sfail(A) = \{(\textbf{\textit{hide}}\ B\ \textbf{\textit{in}}\ \sigma, A) | (\sigma, A \cup A' \cup \widehat{B}) \in sfail(C)\}\}$, *where*
$A' = \{(N \cup \{B\}, g) |\ \exists g' \in DC(N, data) : (N, g') \in A\}$ *and*
$\widehat{B} = \{(\{B\}, g) |\ g \in DC(\{B\}, data)\}$.
(iii) $stable(A) = stable(C) \wedge \forall g \in DC(\{B\}, Data) : (\{B\}, g) \notin tr(C)$.
(ix) $divtr(A) = \{\textbf{\textit{hide}}\ B\ \textbf{\textit{in}}\ \sigma |\ \sigma \in divtr(C)\} \cup \{\textbf{\textit{hide}}\ B\ \textbf{\textit{in}}\ \sigma | \sigma \in inftr(C) \ \wedge |\textbf{\textit{hide}}\ B\ \textbf{\textit{in}}\ \sigma| < \infty\}$.
(x) $dfail(A) = \{(\textbf{\textit{hide}}\ B\ \textbf{\textit{in}}\ \sigma, A) |\ (\sigma, A \cup A' \cup \widehat{B}) \in dfail(C)\} \cup (divtr(\exists B[C]) \times 2^{\Sigma})$, *where, Σ is so defined in Definition 4.7.*

Proof. See Appendix B □

Proposition 4.9 *Let C and C' be finite constraint automata over the same set of names, $C \overset{cffd}{\approx} C'$ and B be a name in the set of names. Then, $\exists B[C] \overset{cffd}{\approx} \exists B[C']$.*

Proof.
(i) By Proposition 4.8(iii), $stable(\exists B[C]) = stable(C) \wedge \forall g \in DC(\{B\}, Data) : (\{B\}, g) \notin tr(C)$. Because, $C \overset{cffd}{\approx} C'$, $stable(C) = stable(C')$, $divtr(C) = divtr(C')$ and $sfail(C) = sfail(C')$. By Proposition 2.3(a), $tr(C) = divtr(C) \cup \{(\sigma, \emptyset) | \sigma \in sfail(C)\}$. Thus, $tr(C) = tr(C')$. Therefore, $stable(\exists B[C]) = stable(\exists B[C'])$.

(ii) By Proposition 4.8(ii),
$sfail(\exists B[C]) = \{(\textbf{hide } B \textbf{ in } \sigma, A) |\ (\sigma, A \cup A' \cup \widehat{B}) \in sfail(C)\}$, $A' = \{(N \cup \{B\}, g) |\ \exists g' \in DC(N, data) : (N, g') \in A\}$, $\widehat{B} = \{(\{B\}, g) |\ g \in DC(\{B\}, data)\}$. Because, $C \overset{cffd}{\approx} C'$, $sfail(C) = sfail(C')$. Because the name sets of C and C' are equal, the definitions of sets A' and $\widehat{B}$ in the cases of C and C' are the same. Thus, $sfail(\exists B[C]) = sfail(\exists B[C'])$.
(iii) By Proposition 4.8(ix), $divtr(\exists B[C]) = \{\textbf{hide } B \textbf{ in } \sigma |\ \sigma \in divtr(C)\} \cup \{\textbf{hide } B \textbf{ in } \sigma | \sigma \in inftr(C) \wedge |\textbf{hide } B \textbf{ in } \sigma| < \infty\}$. As we showed in (i), $tr(C) = tr(C')$. By Proposition 2.3(g) and because C and C' are finite, $inftr(C) = \{\omega \in \Sigma^{\omega} | \forall \sigma \in \Sigma^* : (\sigma \textit{ is a proper prefix of } \omega \rightarrow \sigma \in tr(C))\}$. Thus, $inftr(C) = inftr(C')$ and $divtr(\exists B[C]) = divtr(\exists B[C'])$. □

Corollary 4.10 *CFFD-equivalence is a congruence with respect to the hiding of port names from finite constraint automata.*

Proposition 4.11 *Let C and C' be finite constraint automata over the same set of names, $C \overset{ndfd}{\approx} C'$ and B be a name in the set of names. Then, $\exists B[C] \overset{ndfd}{\approx} \exists B[C']$.*

Proof. The proofs for claims $stable(\exists B[C]) = stable(\exists B[C'])$ and $divtr(\exists B[C]) = divtr(\exists B[C'])$ are the same as in the proof of Proposition 4.9. By Proposition 4.8(x), $dfail(\exists B[C]) = \{(\textbf{hide } B \textbf{ in } \sigma, A) |\ (\sigma, A \cup A' \cup \widehat{B}) \in dfail(C)\} \cup (divtr(\exists B[C]) \times 2^{\Sigma})$. Because, $C \overset{ndfd}{\approx} C'$, $dfail(C) = dfail(C')$. As we showed, $divtr(\exists B[C]) = divtr(\exists B[C'])$. Thus, $dfail(\exists B[C]) = dfail(\exists B[C'])$. □

Corollary 4.12 *NDFD-equivalence is a congruence with respect to the hiding of port names from finite constraint automata.*

5 Conclusions

The ultimate goal of this paper was to prepare an environment for compositional model checking of Reo specifications modeled by constraint automata using equivalence based reduction method. For this purpose we introduced an extended definition of constraint automaton by which every constraint automaton can be considered as a labeled transition system and each labeled transition system can be translated into a constraint automaton. Also, we introduced two new composition operators for constraint automata: join (production) of two automata with respect to their common port names and hiding of a port name in all transition labels of an automaton. We showed that failure-based equivalences CFFD and NDFD are congruence with respect to join and hiding operators of constraint automata.

Based on these congruency results and because of the linear time temporal logic preservation properties of CFFD and NDFD equivalences and their minimality properties (proved in [12]), they will useful candidates for compositional reduction of models in the field of model checking in future works. For this purpose, we need to have algorithms for reducing sizes of constraint automata with respect to the equivalence relations. Naturally, the algorithms will be generalizations of ordinary

algorithms for converting an automaton into its deterministic counterpart and then minimization of it.

References

[1] Arbab F., *Reo: A Channel-based Coordination Model for Component Composition*, Math. Struc. in Computer Science, **14(3)**, (2004), 329-366.

[2] Arbab F., *Abstract Behaiviour Types: A foundation model for components and their composition*, science of Computer Programming, **55**, (2005), 3-52.

[3] Arbab F., Mavadat F., *Coordination Through Channel Composition*, Proceedings of Coordination Languages and Models 2002, LNCS, **2315**, Springer-Verlag, (2002).

[4] Baier C., Sirjani M., Arbab F., Rutten J., *Modelling Component connectors in Reo by Constraint Automata*, Science of Computer Programming, **61**, (2006), 75-113.

[5] Clarke E., Grumberg O., Long D., *Model Checking and Abstraction*, ACM Transactions on Programming Languages and Systems, **16(5)**, (1994), 1512-1542.

[6] Clarke E., Grumberg O., Peled D., "Model Checking", The MIT Press, (1999).

[7] Clarke E., Long D., McMillan K., *Compositional Model Checking*, Proceeding of the 4th IEEE Symposium on Logic in Computer Science, (1989), 353-362.

[8] Emerson A., Sistla A., *Symmetry and Model Checking*, Proceedings of CAV'93, (1993), 463-478.

[9] Graf S., Steffen B., *Compositional Minimization of Finite-State Systems*, Proceedings of CAV'90, Springer-Verlag, (1991), 186-196.

[10] Hoare C.A.R., "Communicating Sequential Processes", Prentice-hall, (1985).

[11] Izadi M., Movaghar A., *An Equivalence Based Method for Compositional Verification of the Linear Temporal Logic of Constraint Automata*, Proceedings of FSEN05, Electronic Notes in Theoretical Computer Science (ENTCS), Elsevier (2005).

[12] Kaivola R., Valmari, A., *The Weakest Semantic Equivalence Preserving Nexttime-less Linear Temporal Logic*, Proceedings of CONCUR'92, LNCS, **630**, Springer-Verlag, (1992), 207-221.

[13] Manna Z., Pnueli A., "The Temporal Logic of Reactive and Concurrent Systems: Specification", Springer-Verlag, (1991).

[14] Milner R., "Communication and Concurrency", Prentice-Hall, (1989).

[15] Peled D., *Verification for Robust Specification*, Conference on Theorm Proving in Higher Order Logic, Springer-Verlag, (1997), 231-241.

[16] Pnueli A., *In Transition from Global to Modular Temporal Reasoning about Programs*, "Logics and Models of Concurrent Systems", NATO ASI series, **F13**, Springer-Verlag, (1985), 123-146.

[17] de Roever W. P., Langmaack h., Pnueli A., *Compositionality: The Significant Difference*, International Symposium, COMPOS'97, Bad Malente, Germany, September 1997, Revised Lectures, Lecture Notes in Computer Science, **1536**, Springer-Verlag, (1998).

[18] Valmari A., *Failur-based Equivalences are Faster than Many Believe*, Proc. Structures in Concurrency Theory, May 1995, Springer-Verlag (1995), 326-340.

[19] Valmari A., Tienari M., *An Improved Failure Equivalence for Finite State Systems with a Reduction Algorithm*, "Protocol Specification, Testing and Verification", **XI**, (1991), 3-18.

[20] Valmari A., Tienari M., *Compositional Failure Based Semantic Models for Basic LOTOS*, Formal Aspects of Computing **7**, (1995), 440-468.

A Proof of Proposition 4.2

In this section, we present the proofs of all parts of Proposition 4.2.

(i) This proposition is a direct consequence of Definitions 2.2, 3.5 and 4.1.

(ii) Let $(\pi, B) \in sfail(C_1)$, $(\rho, D) \in sfail(C_2)$ and $Join(\sigma; \pi, \rho)$. We prove that for all $A \subseteq \Sigma$, if $A \cap G \subseteq B \cap D \ \wedge \ A \cap G' \subseteq B \cup D$, then, $(\sigma, A) \in sfail(C_1 \bowtie C_2)$. First note that, $\pi \in tr(C_1)$, $\rho \in tr(C_2)$ and $Join(\sigma; \pi, \rho)$, thus based on Proposition 4.2(i), $\sigma \in tr(C_1 \bowtie C_2)$ and because (π, B) and (ρ, D) are stable failures, there is no outgoing transition with label τ from the last state in $C_1 \bowtie C_2$ after tracing σ (We denote this state by q_F, the last state in C_1 after tracing π by q_B and the last state in C_2 after tracing ρ by q_D). Let A be the greatest subset of Σ where $A \cap G \subseteq B \cap D \ \wedge \ A \cap G' \subseteq B \cup D$ and (in the way of proof by contradiction) suppose that there is an outgoing transition from state q_F in $C_1 \bowtie C_2$ with label $(N, g) \in A$. Based on Definition 3.5, N should have one of the three following forms: (1) $N = N_1$ where $N_1 \subseteq Nam_1$ and $N \cap Nam_2 = \emptyset$. In this case, $(N, g) \in A \cap G$. Thus, $(N, g) \in B \cap D$. But, both (ρ, D) and (π, B) are fails in their corresponding automata. Thus, it is impossible that (N, g) be the label of an outgoing transition from q_F in the product automaton. (2) $N = N_2$ where $N_2 \subseteq Nam_2$ and $N \cap Nam_1 = \emptyset$. The proof is symmetric with case (1). (3) $N = N_1 \cup N_2$ where $N_1 \subseteq Nam_1$, $N_2 \subseteq Nam_2$ and $N_1 \cap Nam_2 = N_2 \cap Nam_1$. In this case, $(N, g) \in A \cap G'$. Thus, $(N, g) \in B$ or $(N, g) \in D$. In both cases it is impossible that that (N, g) be the label of an outgoing transition from q_F in the product automaton, because at least one of states q_B and q_D does not have an outgoing transition with label (N, g) in their corresponding automaton. Because we supposed that A is the greatest subset of Σ where $A \cap G \subseteq B \cap D \ \wedge \ A \cap G' \subseteq B \cup D$, our claim holds for the smaller subsets of Σ.

On the other hand, let $(\sigma, A) \in sfail(C_1 \bowtie C_2)$. Thus $\sigma \in tr(C_1 \bowtie C_2)$ and based on Proposition 4.2(i), $\exists \pi \in tr(C_1)$, $\exists \rho \in tr(C_2)$, $Join(\sigma; \pi, \rho)$. Let B be the greatest subset of Σ where $(\pi, B) \in fail(C_1)$ and D be the greatest subset of Σ where $(\rho, D) \in fail(C_2)$. Again, we denote the last state in C_1 after tracing π by q_B, the last state in C_2 after tracing ρ by q_D and the last state in $C_1 \bowtie C_2$ after tracing σ by q_F. Because q_F is stable and based on Definition 3.5, q_B and q_D are stable. Thus, (π, B) and (ρ, D) are stable failures. If $(N, g) \in A \cap G$ then $N \cap Nam_1 = \emptyset$ or $N \cap Nam_2 = \emptyset$ and there is no outgoing transition with label (N, g) from q_F. If $N \cap Nam_1 = \emptyset$ then obviously, $(N, g) \in B$ and based on Definition 3.5 it can not be the label of an outgoing transition from q_D in C_2 . Thus, because of the maximality of D, $(N, g) \in D$. Thus, $(N, g) \in B \cap D$. Similarly, if $N \cap Nam_2 = \emptyset$ then $(N, g) \in B \cap D$. Thus, $A \cap G \subseteq B \cap D$. If $(N, g) \in A \cap G'$ then $N \cap Nam_1 \neq \emptyset$ and $N \cap Nam_2 \neq \emptyset$. Let (in the way of proof by contradiction) that $(N, g) \notin B \cup D$. Thus, there are an outgoing transition with label (N, g) from q_B in C_1 and an outgoing transition with label (N, g) from q_D in C_2 and based on Definition 3.5 there is an outgoing transition with label (N, g) from q_F in $C_1 \bowtie C_2$. But this contradicts that (σ, A) is a failure.

(iii), (ix) These propositions are direct consequences of Definitions 2.2 and 3.5.

(x) By Proposition 2.3(d), $dfail(C_1 \bowtie C_2) = sfail(C_1 \bowtie C_2) \cup (divtr(C_1 \bowtie C_2) \times 2^{\Sigma})$. By using 4.2(ii),
$dfail(C_1 \bowtie C_2) = \{(\sigma, A) \mid \exists(\pi, B) \in sfail(C_1),\ \exists(\rho, D) \in sfail(C_2),$
$Join(\sigma; \pi, \rho) \wedge A \cap G \subseteq B \cap D \wedge A \cap G' \subseteq B \cup D\} \cup (divtr(C_1 \bowtie C_2) \times 2^{\Sigma})$. (**)
The Equation (**) contains two instance of $sfail$ and we need to show that the replacement of both by $dfail$ do not add any new pair (σ, A) to the righthand side of the equation. In fact, we can show that the replacement of instances of $sfail$ by $dfail$ adds some pairs to the set $\{(\sigma, A)\ |...\}$ in the righthand side of the equation, but all of these new pairs are in $(divtr(C_1 \bowtie C_2) \times 2^{\Sigma})$. Thus, the union set (the righthand side of the equation) does not change. For this goal, first suppose that we replace $sfail(C_1)$ by $dfail(C_1)$. Because, $dfail(C_1) = sfail(C_1) \cup (divtr(C_1) \times 2^{\Sigma})$ (see Proposition 2.3(d)), the only effect of replacement is that new pairs (σ, A) may be introduced related to some (π, B) and (ρ, D) such that $\pi \in divtr(C_1)$, $(\rho, D) \in sfail(C_2)$ and $Join(\sigma; \pi, \rho)$ hold. But then $\rho \in tr(C_2)$, and by the replacement of $sfail$ by $dfail$, (σ, A) belongs to $(divtr(C_1) \times 2^{\Sigma})$. By a symmetric argument, we can show that the replacement of the other $sfail$ by $dfail$ does not change the righthand side of the Equation (**).

B Proof of Proposition 4.8

In this section, we present the proofs of all parts of Proposition 4.8.

(i) This is a direct consequence of Definitions 2.2 and 4.7.

(ii) If $(\rho, A) \in sfail(\exists B[C])$, then for the automaton $(\exists B[C])$, we know that there is a state $q \in Q$ which, $q_{0,B} \stackrel{\rho}{\Longrightarrow} q$ and $stable(q)$ and $\forall a \in A\ (\neg\ q \stackrel{a}{\Longrightarrow})$. Because ρ is a trace in $\exists B[C]$, there is a trace $\sigma \in tr(C)$ such that $\rho =$ **hide** B **in** σ, $\Sigma(\rho) =$ **hide** B **in** $\Sigma(\sigma)$ and in automaton C, $q_0 \stackrel{\sigma}{\Longrightarrow} q$. Because, q is stable in $\exists B[C]$, there is no transition of the form $q \stackrel{\tau}{\rightarrow}_B q'$, by using the definition of hiding, there is no transition of the form $q \stackrel{\tau}{\rightarrow} q'$ in C. Thus, q is also stable in C. Now we prove that $(\sigma, A \cup A' \cup \widehat{B})$ is a failure of C. First note that because (ρ, A) is a failure of $\exists B[C]$, for every $(N, g) \in A$, $B \notin A$. Thus A and A' are two disjoint sets. Because (ρ, A) is a failure in $\exists B[C]$ and $\rho =$ **hide** B **in** σ, (σ, A) is a failure of C. For set A', we know that $A =$ **hide** B **in** A'. Thus, (σ, A') is also a failure of C. Because q is stable in $\exists B[C]$, by the definition of hiding, there is no transition of the form $q \stackrel{\{B\},g}{\longrightarrow} q'$ in C. Thus, $(\sigma, \widehat{B})$ is a failure of C. As the overall consequence: $sfail(\exists B[C]) \subseteq \{(\textbf{hide}\ B\ \textbf{in}\ \sigma, A) |\ (\sigma, A \cup A' \cup \widehat{B})\}$.
On the other hand, let $(\sigma, A \cup A' \cup \widehat{B}) \in sfail(C)\}$ and $\rho =$ **hide** B **in** σ. Thus, for the automaton C, we know that there is a state $q \in Q$ which, $q_0 \stackrel{\sigma}{\Longrightarrow} q$ and $stable(q)$ and $\forall a \in A \cup A' \cup \widehat{B}$, $(\neg\ q \stackrel{a}{\Longrightarrow})$. Because, $q_0 \stackrel{\sigma}{\Longrightarrow} q$ is a run of C and $\rho =$ **hide** B **in** σ, $q_{0,B} \stackrel{\rho}{\Longrightarrow} q$ is a run of $\exists B[C]$. Because, in the automaton C there is no transition of the form $q \stackrel{a}{\rightarrow} q'$ in which, $a \in A \cup A'$, by using the definition of hiding, there is no transition of the form $q \stackrel{a}{\rightarrow} q'$ in which, $a \in A$ in the automaton $\exists B[C]$. Thus, (ρ, A) is a failure of $\exists B[C]$. Because, q is stable in C and there is no transition of the form $q \stackrel{a}{\rightarrow} q'$, $a \in \{(\{B\}, g) | g \in DC(\{B\}, data)\}$, q is stable in $\exists B[C]$. Thus, (ρ, A) is a stable failure of $\exists B[C]$. As the overall consequence:

$\{(\textbf{hide } B \textbf{ in } \sigma, A)|\ (\sigma, A \cup A' \cup \widehat{B})\} \subseteq\ sfail(\exists B[C])\}$.

(iii) (ix) These are direct consequences of Definitions 2.2 and 3.6.

(x) By Proposition 2.3(d), $dfail(\exists B[C]) = sfail(\exists B[C]) \cup (divtr(\exists B[C]) \times 2^{\Sigma})$. Thus, using 4.8(ii),

$dfail(\exists B[C]) = \{(\textbf{hide } B \textbf{ in } \sigma, A)|\ (\sigma, A \cup A' \cup \widehat{B}) \in sfail(C)\}\} \cup (divtr(\exists B[C]) \times 2^{\Sigma})$ $\quad (*)$

The only effect of replacement of $sfail$ by $dfail$ in Equation (*) is that new pairs $(\textbf{hide } B \textbf{ in } \sigma, A)$ may be introduced where, $\sigma \in divtr(C)$. But obviously based on Definition 4.7, if $\sigma \in divtr(C)$ then, $\textbf{hide } B \textbf{ in } \sigma \in divtr(\exists B[C]))$. Thus, the replacement of $sfail$ by $dfail$ in Equation (*) does not change the righthand side of it.