Model checking of component connectors

Izadi, M.

8 Compositional Reduction

In the previous chapters, we introduced constraint automata, Büchi automata of records and their augmented versions as operational models for Reo connectors. We have shown that they have increasing expressiveness. We also introduced methods for model checking of Reo nets using both global and on-the-fly translations of linear temporal logic formulas into automata. Now, we deal with the problem of *state explosion*, namely that the model of the systems tend to be extremely large. In this chapter we investigate the method of *compositional reduction* to deal with the problem of state explosion for the case of large scale Reo nets. We concentrate on the most basic semantic model of Reo, namely constraint automata, and we leave for future work the investigation of similar compositional reduction techniques for ABAR models. In Section 8.1, we introduce the method and overview the way in which we are able to minimize the models of Reo nets. In the subsequent sections, we present the technical details with some examples.

8.1 Introduction

Equivalence based *compositional reduction* is a way to deal with the problem of state explosion [50, 139]. In this method, the models of the components of a system are reduced with respect to an *equivalence relation* before building the model of the whole system [60, 50, 79, 82]. In order to be useful, the equivalence relation should satisfy two properties: *preservation* of all properties to be verified and being a *congruence* relation with respect to all operators that are used for composing the models. By a congruence relation we mean that the replacement of a component of a model by an equivalent one should always yield a model that is equivalent with the original one.

When transition systems are used as the semantics of specification formalisms, one of the key questions is whether two models are equivalent. In the case of labeled transition systems with simple alphabets, numerous equivalence relations have been presented in the literature. *Trace equivalence*, *visible-trace equivalence* (automata-theoretic equivalence), weak and strong *bisimilarity* presented by Milner [112], *failure-based equivalences*, and CSP-like equivalences presented by Hoare [62] are examples of these equivalences. (For a survey on several equivalence relations see [143, 144].) From a theoretical point of view, the investigation of these equivalences in the case of labeled transition systems with compound alphabets such as constraint automata and record-based labeled transition systems are interesting.

Fortunately, in the context of failure based semantic models of the process description language LOTOS, there are two equivalence relations, called *chaos-free failures divergences (CFFD)* and *non-divergent failure divergences (NDFD)*, which satisfy the preservation property for two fragments of linear temporal logic. NDFD preserves linear time temporal logic without next-time operator (called LTL_{-X}) [86]. CFFD preserves linear temporal logic without the next-time operator but with an extra operator that distinguishes deadlocks from divergences (called LTL^{ω}) [141, 142]. Also, it has been shown that CFFD and NDFD are the weakest equivalence relations that preserve the above mentioned fragments of linear temporal logic [86, 141]. In addition, it has been shown that in the case of labeled transition systems with simple alphabets, CFFD and NDFD are congruences with respect to all composition op-

erators defined in LOTOS [142].

Now, we investigate the above mentioned results for the case of constraint automata. In other words, instead of their TDS-based semantics, we consider the failure based semantics for constraint automata as labeled transition systems with compound labels. Thus, first we define CFFD and NDFD equivalences for constraint automata. Then, we show that the temporal logic preservation results also will hold in these cases. Next, we consider the congruency results. Obviously, if we consider constraint automata as labeled transition systems with the composition operators defined in LOTOS, then the previously established congruency results for CFFD and NDFD also hold for constraint automata. In this chapter, we consider two other composition operators that refer to the internal structures of the transition labels. These two composition operators are the operators of join and hiding a port name, as we introduced them in Chapter 3. We prove that failure-based equivalence relations CFFD and NDFD are congruences with respect to both join and hiding operators of constraint automata. Therefore, based on the congruency results, and because of the linear time temporal logic preservation properties of CFFD and NDFD equivalences and their minimality properties, CFFD and NDFD can be used for compositional reduction of constraint automata models in the field of model checking.

8.2 Failure based equivalence of constraint automata

Now, we define the notions of CFFD and NDFD-equivalence relations. We define these equivalences for *labeled transition systems* in general and for constraint automata in particular. First, recall the notion of labeled transition systems:

Definition 8.1
- A *transition alphabet* is a countable set of symbols 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 $L = \langle S, s, \Delta \rangle$, 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 set: $\Sigma(L) = \{l \in \Sigma |\ \exists s, s' : (s, l, s') \in \Delta\}$. The alphabet of any *LTS* is required to be finite. In addition, an LTS is finite if its set of states is finite.

Now we introduce some operators that can be used to compose labeled transition systems. These operators are parallel composition with the possibility of synchronization on some transition labels, nondeterministic choice, simple hiding, and renaming.

Definition 8.2 Let $L_1 = \langle S_1, s_1, \Delta_1 \rangle$ and $L_2 = \langle S_2, s_2, \Delta_2 \rangle$ be two LTSs.
(i) The *parallel composition* of L_1 and L_2 with respect to $G = \{g_1, \ldots, g_n\} \subseteq \Sigma$, denoted by $L_1 |[g_1, \ldots, g_n]| L_2$, is the LTS $\langle S_1 \times S_2, (s_1, s_2), \Delta \rangle$, where
- $((t, u), g_i, (t', u')) \in \Delta$, for $g_i \in G$, iff $(t, g_i, t') \in \Delta_1$ and $(u, g_i, u') \in \Delta_2$, and

- $((t,u),l,(t',u')) \in \Delta$ for $l \notin G$, iff either $(t,l,t') \in \Delta_1$ and $u = u'$ or $(u,l,u') \in \Delta_2$ and $t = t'$.
(ii) The *nondeterministic choice* composition of L_1 and L_2, denoted by $L_1[\,]L_2$, is the LTS $\langle S_1 \times \{1\} \cup S_2 \times \{2\} \cup \{(s,0)\}, (s,0), \Delta\rangle$, where
- $((t,i),l,(t',i)) \in \Delta$, where $i \in \{1,2\}$, iff $(t,l,t') \in \Delta_i$, and
- $((s,0),l,(t,i)) \in \Delta$, where $i \in \{1,2\}$, iff $(s_i,l,t) \in \Delta_i$.

Definition 8.3 Let $L_1 = \langle S_1, s_1, \Delta_1\rangle$ be an LTS and $G = \{g_1,\ldots,g_n\} \subset \Sigma$ and $H = \{h_1,\ldots,h_n\} \subset \Sigma$.
(i) The *simple hiding* of G in L_1, denoted by $Hide\ g_1,\ldots,g_n\ in\ L_1$, is the LTS $\langle S_1, s_1, \Delta\rangle$ where
- $(t,l,t') \in \Delta$, iff either $l \notin G$ and $(t,l,t') \in \Delta_1$ or $l = \tau$ and there is a $g_i \in G$ such that $(t,g_i,t') \in \Delta_1$.
(ii) The *renaming* of L_1 with respect to G and H, denoted by $L_1[h_1/g_1,\ldots,h_n/g_n]$, is the LTS $\langle S_1, s_1, \Delta\rangle$ where
- $(t,l,t') \in \Delta$ iff either $l \notin G$ and $(t,l,t') \in \Delta_1$ or $l = h_i$ and $(t,g_i,t') \in \Delta_1$.

Now, we recall some basic concepts of process algebra and give the definitions of CFFD and NDFD-equivalences [141, 142, 86].

Definition 8.4 Let $L = \langle S, s, \Delta\rangle$ be a labeled transition system.
- 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 $\sigma = vis(\rho)$.

Now, we can define the notions of traces, divergence, stability and failures for labeled transition systems in general, based on [86]:

Definition 8.5 Let $L = \langle S, s, \Delta)\rangle$ be a labeled transition system.
- $\sigma \in \Sigma^*$ 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* of L 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 $s' \xrightarrow{\tau}$.
- An LTS L is stable if its 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 $\forall a \in A.\neg(s' \stackrel{a}{\Longrightarrow})$.
- $fail(L)$ is the set of all failures of L.
- $(\sigma, A) \in \Sigma^* \times 2^\Sigma$ 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 *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.

The following lemma lists some direct consequences of the above definition for later use.

Lemma 8.1 Let L be a labeled transition system,
a) $tr(L) = divtr(L) \cup \{\sigma | (\sigma, \emptyset) \in sfail(L)\}$.
b) $tr(L) = \{\sigma | (\sigma, \emptyset) \in fail(L)\} = \{\sigma | (\sigma, \emptyset) \in dfail(L)\}$.
c) $dfail(L) = sfail(L) \cup (divtr(L) \times 2^\Sigma)$.
d) If L is a finite labeled transition system,
$$inftr(L) = \{\omega \in \Sigma^\omega | \forall \sigma \in \Sigma^* : (\sigma \textit{ is a proper prefix of } \omega \rightarrow \sigma \in tr(L))\}.$$

Now, we introduce two failure based equivalences for labeled transition systems that were originally introduced in [141, 142]:

Definition 8.6 Let L and L' be two labeled transition systems.
(i) We say that L and L' are CFFD equivalent and write $L \overset{cffd}{\approx} L'$ if and only if $stable(L) \Leftrightarrow stable(L')$, $divtr(L) = divtr(L')$, $inftr(L) = inftr(L')$ and $sfail(L) = sfail(L')$.
(ii) We say that L and L' are NDFD equivalent and write $L \overset{ndfd}{\approx} L'$ if and only if $stable(L) \Leftrightarrow stable(L')$, $divtr(L) = divtr(L')$, $inftr(L) = inftr(L')$ and $dfail(L) = dfail(L')$.

The NDFD-equivalence is strictly weaker than CFFD-equivalence in the sense of the following lemma:

Lemma 8.2 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 definitions is superfluous. Now, we define the notion of being *congruence* for equivalence relations with respect to a composition operator:

Definition 8.7 Let $\approx$ be an equivalence relation and f be a composition operator over a set of labeled transition systems. We say that $\approx$ is a *congruence* with respect to 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: $f(L_1, \ldots, L_n) \approx f(L'_1, \ldots, L'_n)$.

Obviously, each constraint automaton $C = \langle Q, \mathcal{N}, \rightarrow, q_0 \rangle$ over data set $\mathcal{D}$ can be considered as a labeled transition system with alphabet

$$\Sigma = \{(N, g) | N \subseteq \mathcal{N} \wedge g \in DC(\mathcal{N}, \mathcal{D}) \wedge N \neq \emptyset\}:$$

Lemma 8.3 For a constraint automaton $C = \langle Q, \mathcal{N}, \rightarrow, q_0 \rangle$ over a data set $\mathcal{D}$, let $L(C) = (S, s, \Delta)$ be the labeled transition system over the alphabet $\Sigma = \{(N, g) | N \subseteq \mathcal{N} \wedge g \in DC(\mathcal{N}, \mathcal{D}) \wedge N \neq \emptyset\}$, where, $S = Q$, $s = q_0$ and $(q_i, (N, g), q_j) \in \Delta$ if and only if $(q_i, N, g, q_j) \in \rightarrow$. Then, the constraint automata C and C' are (TDS-based) equivalent if and only if they are infinite-trace-based equivalent. In other words $L_{TDS}(C) = L_{TDS}(C')$ if and only if $inftr(L(C)) = inftr(L(C'))$.

Proof. This lemma is a direct consequence of Definitions 3.7 and 8.5 □

Based on the above lemma, if we consider the elements of the alphabet of every constraint automaton as simple elements and do not refer to their internal structures then, constraint automata can be composed using every well defined operator for composing labeled transition systems, such as parallel composition with synchronization, nondeterministic choice, and renaming. In addition, in the Chapter 3 we introduced two composition operators, join and hiding with respect to a port name, whose definitions depend on the internal structures of the elements of the alphabet sets of constraint automata.

In [142] it has been proved that CFFD and NDFD (without the need to check for the stability predicates) are congruences with respect to all basic composition operators of LOTOS, except for the operator of nondeterministic choice. For the case of nondeterministic choice operator, it is also necessary to check the stability predicate. For composing constraint automata, not only we can use these operators, but we also have the two extra operators (join and hiding a port name) which refer to the internal structure of the elements of alphabet sets. In the following sections, we show that equivalence relations CFFD and NDFD are also congruences with respect to both join and hiding operators of constraint automata.

8.3 Congruency Results for Joining of Constraint Automata

In this section, we prove that the equivalence relation CFFD is a congruence with respect to the join of constraint automata and it is also the case for the equivalence relation NDFD. Our method of proof is a modification and extension of the proof of that CFFD and NDFD relations are congruences for the case of parallel composition of LTSs presented in [142].

First, we define a predicate **Join**$(\sigma;\pi,\rho)$, which intuitively means that words π and ρ can be considered as traces of two constraint automata while σ is a trace in the join constraint automaton resulting from the join of ρ and π.

Definition 8.8 Let $Data$ be a set of data, Nam_1 and Nam_2 be 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 finite or infinite word over the alphabet Σ. We define the predicate **Join**$(\sigma;\pi,\rho)$ to hold (to be true) if and only if there is a function *moved* from $\{1,2,\ldots\}$ to $\{first, second, both\}$ such that:

1-

$$moved(i) = \begin{cases} first & \text{if } N_i \cap Nam_2 = \emptyset \text{ and } g_i \in DC(Nam_1, Data), \\ second & \text{if } N_i \cap Nam_1 = \emptyset \text{ and } g_i \in DC(Nam_2, Data), \\ both & \text{otherwise.} \end{cases}$$

2- π is obtained from σ by:

2-1- for all $i \geq 1$ where, $moved(i) = both$, change (N_i, g_i) to

$(N_i \cap Nam_1, g_i[Nam_1])$,
2-2- remove all (N_i, g_i) where, $moved(i) = second$.
3- ρ is obtained from σ by:
3-1- for all $i \geq 1$ where, $moved(i) = both$, change (N_i, g_i) to
$(N_i \cap Nam_2, g_i[Nam_2])$,
3-2- remove all (N_i, g_i) where, $moved(i) = first$.

By $g[Nam_i]$ we mean the restriction of data constraint g to the name set Nam_i: in the conjunctive normal form of g, the restricted $g[Nam_i]$ can be obtained by replacing all terms containing $d_A = d$ where $A \notin Nam_i$ with $true$. Obviously, the obtained word π is a word over alphabet Σ_1 and ρ is a word over alphabet Σ_2.

Now, we show that the sets of finite or infinite traces, stable failures, divergent traces and divergence-masked failures of the join automaton can be characterized by their counterparts in the two constraint automata. Based on these characterizations, we prove our congruency results.

Proposition 8.4 Let $C_1 = \langle Q_1, Nam_1, T_1, q_{01}\rangle$ and $C_2 = \langle Q_2, Nam_2, T_2, q_{02}\rangle$ be two constraint automata. Then,
(i) $tr(C_1 \bowtie_C C_2) = \{\sigma \mid \exists \pi \in tr(C_1),\ \exists \rho \in tr(C_2),\ Join(\sigma;\pi,\rho)\}$.
(ii) $sfail(C_1 \bowtie_C C_2) = \{(\sigma, A) \mid \exists (\pi, B) \in sfail(C_1),\ \exists (\rho, D) \in sfail(C_2),$
$Join(\sigma;\pi,\rho)\ \ and\ A \cap G \subseteq B \cap D\ \wedge\ A \cap G' \subseteq B \cup D\}$,
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) $stable(C_1 \bowtie_C C_2) = stable(C_1) \wedge stable(C_2)$.
(iv) $divtr(C_1 \bowtie_C C_2) = \{\sigma \mid \exists \pi \in tr(C_1),\ \exists \rho \in tr(C_2), Join(\sigma;\pi,\rho)\ and$
$(\pi \in divtr(C_1) \vee \rho \in divtr(C_2))\}$.
(v) $dfail(C_1 \bowtie_C C_2) = \{(\sigma, A) \mid \exists (\pi, B) \in dfail(C_1),\ \exists (\rho, D) \in dfail(C_2),$
$Join(\sigma;\pi,\rho)\ \ and\ A \cap G \subseteq B \cap D\ \wedge\ A \cap G' \subseteq B \cup D\}\ \cup\ (divtr(C_1 \bowtie_C C_2) \times 2^{\Sigma})$,
where, Σ is the same as defined in Definition 8.8 and G and G' are the same as defined in (ii).
(vi) $inftr(C_1 \bowtie_C C_2) = \{\omega \mid \exists \pi \in tr(C_1) \cup inftr(C_1),\ \exists \rho \in tr(C_2) \cup inftr(C_2),$
$Join(\omega;\pi,\rho) \wedge (\pi \in inftr(C_1) \vee \rho \in inftr(C_2))\}$.

Proof.
First note that in general constraint automata can be nondeterministic, i.e. there are transitions with the same source states and the same labels but with different target states. Thus, the last state after a finite trace can be more than one and for a trace σ in the join of two constraint automata the predicate $Join(\sigma;\pi,\rho)$ can be satisfied by more than one pair of traces (π,ρ). Now we prove the proposition:

(i) This proposition is a direct consequence of Definitions 8.5, 3.8 and 8.8.

(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 $(\sigma, A) \in sfail(C_1 \bowtie_C C_2)$.

First note that, $\pi \in tr(C_1)$, $\rho \in tr(C_2)$ and $Join(\sigma;\pi,\rho)$, thus based on Proposition 8.4(i), $\sigma \in tr(C_1 \bowtie_C C_2)$ and because (π, B) and (ρ, D) are stable failures, there is no outgoing transition with label τ from the last state in $C_1 \bowtie_C C_2$ 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 member of 2^Σ such that $A \cap G \subseteq B \cap D \ \wedge \ A \cap G' \subseteq B \cup D$. (Since Σ is finite, such a set exists). Now using proof by contradiction, suppose that there is an outgoing transition from state q_F in $C_1 \bowtie_C C_2$ with label $(N, g) \in A$. Based on Definition 3.8, we have three cases, based on N: (1) $N \subseteq Nam_1$ 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 fail runs in their corresponding automata. Thus, it is impossible for (N, g) to be the label of an outgoing transition from q_F in the product automaton. (2) $N \subseteq Nam_2$ 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, either $(N, g) \in B$ or $(N, g) \in D$. In either case it is impossible for (N, g) to be the label of an outgoing transition from q_F in the product automaton, because at least one of the states q_B and q_D does not have an outgoing transition with label (N, g) in its 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 $(\sigma, A) \in sfail(C_1 \bowtie_C C_2)$. Thus $\sigma \in tr(C_1 \bowtie_C C_2)$ and based on Proposition 8.4(i), there are $\pi \in tr(C_1)$ and $\rho \in tr(C_2)$ such that $Join(\sigma;\pi,\rho)$. 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 $C_1 \bowtie_C C_2$ after tracing σ by q_F. Because q_F is stable, based on Definition 3.8, 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 3.8 it cannot 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 $N \cap Nam_2 \neq \emptyset$. Using proof by contradiction, let $(N, g) \notin B \cup D$. Thus, there is 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 3.8, there is an outgoing transition with label (N, g) from q_F in $C_1 \bowtie_C C_2$. But this contradicts that (σ, A) is a failure.

(iii),(iv) These propositions are direct consequences of Definitions 8.5 and 3.8.

(v) By Lemma 8.1(c), $dfail(C_1 \bowtie_C C_2) = sfail(C_1 \bowtie_C C_2) \cup (divtr(C_1 \bowtie_C C_2) \times 2^\Sigma)$. Using 8.4(ii),

$$dfail(C_1 \bowtie_C C_2) = \{(\sigma, A) \mid \exists(\pi, B) \in sfail(C_1),\ \exists(\rho, D) \in sfail(C_2), \\ Join(\sigma;\pi,\rho) \ \wedge \ A \cap G \subseteq B \cap D \ \wedge \ A \cap G' \subseteq B \cup D\} \\ \cup \ (divtr(C_1 \bowtie_C C_2) \times 2^\Sigma). \qquad (**)$$

Equation (**) contains two instances 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 $(divtr(C_1 \bowtie_C C_2) \times 2^{\Sigma})$. Thus, the union set (the righthand side of the equation) does not change. For this purpose, 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 Lemma 8.1(c)), the only effect of this 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 $Join(\sigma;\pi, \rho)$ holds. 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 Equation (**).

(vi) This item is a direct consequence of Definitions 8.5, 3.8 and 8.8. □

Now, we can prove that CFFD is a congruence with respect to join of constraint automata:

Proposition 8.5 Let C and C' be constraint automata over the same set of names and D and D' be constraint automata over the same set of names, such that $C \overset{cffd}{\approx} C'$ and $D \overset{cffd}{\approx} D'$. Then, $C \bowtie_C D \overset{cffd}{\approx} C' \bowtie_C D'$.

Proof. According to Definition 8.6 we need to prove four items:

(i) $stable(C \bowtie_C D) = stable(C) \wedge stable(D)$, based on Proposition 8.4(iii),
$= stable(C') \wedge stable(D')$, because $C \overset{cffd}{\approx} C'$ and $D \overset{cffd}{\approx} D'$,
$= stable(C' \bowtie_C D')$.

(ii) Based on Proposition 8.4(ii),
$sfail(C \bowtie_C D) = \{(\sigma, A) \mid \exists(\pi, B) \in sfail(C), \exists(\rho, E) \in sfail(D), Join(\sigma;\pi, \rho)$
$\wedge A \cap G \subseteq B \cap E \wedge A \cap G' \subseteq B \cup E\}$ 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-equivalence $sfail(C) = sfail(C')$ and $sfail(D) = sfail(D')$. Because of the equality of the names sets, G and G' in the case of $C \bowtie_C D$ are, respectively, equal to G and G' in the case of $C' \bowtie_C D'$, respectively. Thus, $sfail(C \bowtie_C D) = sfail(C' \bowtie_C D')$.

(iii) Based on Proposition 8.4(iv),
$divtr(C \bowtie_C D) = \{\sigma \mid \exists \pi \in tr(C),\ \exists \rho \in tr(D),\ Join(\sigma;\pi, \rho)\ and\ (\pi \in divtr(C) \vee \rho \in divtr(D))\}$. Based on Lemma 8.1(a), $tr(C) = divtr(C) \cup \{\sigma | (\sigma, \emptyset) \in sfail(C)\}$ and this fact holds also for C', D and D'. For CFFD equivalence, it holds that $divtr(C) = divtr(C')$, $divtr(D) = divtr(D')$, $sfail(C) = sfail(C')$, and $sfail(D) = sfail(D')$. Thus, $tr(C) = tr(C')$ and $tr(D) = tr(D')$. Therefore, $divtr(C \bowtie_C D) = divtr(C' \bowtie_C D')$.

(vi) In part (iii) above we proved that $tr(C) = tr(C')$ and $tr(D) = tr(D')$. Also, using the definition of CFFD-equivalence relation, we know that $inftr(C) = inftr(C')$ and

$inftr(D) = inftr(D')$. Thus, using Proposition 8.4(vi), it is the case that $inftr(C \bowtie_C D) = inftr(C' \bowtie_C D')$. □

Thus, CFFD-equivalence is a congruence with respect to the join of constraint automata. A similar result holds also for NDFD-equivalence:

Proposition 8.6 Let C and C' be constraint automata over the same set of names, D and D' be constraint automata over the same set of names, $C \overset{ndfd}{\approx} C'$ and $D \overset{ndfd}{\approx} D'$. Then, $C \bowtie_C D \overset{ndfd}{\approx} C' \bowtie_C D'$.

Proof.
The proofs for the claims $stable(C \bowtie_C D) = stable(C' \bowtie_C D')$, $divtr(C \bowtie_C D) = divtr(C' \bowtie_C D')$ and $inftr(C \bowtie_C D) = inftr(C' \bowtie_C D')$ are similar to the proofs of their counterparts in Proposition 8.5. (We use $dfail$ sets instead of $sfail$ sets and part (b) of Lemma 8.1 instead of part (a) to show the trace equivalences.) Now we prove that, $dfail(C \bowtie_C D) = dfail(C' \bowtie_C D')$. By Proposition 8.4(v),
$dfail(C \bowtie_C D) = \{(\sigma, A) \mid \exists(\pi, B) \in dfail(C),\ \exists(\rho, E) \in dfail(D),\ Join(\sigma;\pi, \rho)$
and $A \cap G \subseteq B \cap E \ \wedge\ A \cap G' \subseteq B \cup E\} \ \cup\ (divtr(C_1 \bowtie_C C_2) \times 2^{\Sigma})$.
Because $C \overset{ndfd}{\approx} C'$ and $D \overset{ndfd}{\approx} D'$, $dfail(C) = dfail(C')$, $dfail(D) = dfail(D')$ and $divtr(C \bowtie_C D) = divtr(C' \bowtie_C D')$. Because of the equality of the names sets, G and G' in the case of $C \bowtie_C D$ are, respectively, equal to G and G' in the case of $C' \bowtie_C D'$. Thus, $dfail(C \bowtie_C D) = dfail(C' \bowtie_C D')$. □

Therefore, NDFD-equivalence is a congruence with respect to the join of constraint automata.

8.4 Congruency Results for Hiding Names

In this section we prove that the equivalence relation CFFD is a congruence with respect to hiding of port names in constraint automata (with τ-transitions) and that is also the case for the equivalence relation NDFD. Our method of proof is a modification and extension of the proof of that CFFD and NDFD relations are congruences for the case of hiding of an alphabet member in all transitions of an LTS presented in [142].

First, we show that the sets of finite or infinite traces, stable failures, divergent traces and divergence-masked failures of the automaton after hiding of a port name can be characterized by their counterparts in the original constraint automaton. Based on these characterizations, we prove our congruency results.

Definition 8.9

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 every set $\Sigma_1 \subseteq \Sigma$ such that:

$$\textbf{hide } B \textbf{ in } \Sigma_1 \ = \ \{(N \backslash \{B\}, \exists B[g]) |\ (N, g) \in \Sigma_1\} \setminus \{\tau\},$$

where for data constraint g, we define $\exists B[g] = \vee_{d \in \mathcal{D}} g[d_B/d]$ (see Definition 3.9).

Also, for every finite or infinite string $\sigma = (N_1, g_1)(N_2, g_2)\ldots$ we define the string **hide** B **in** σ as the string that is obtained by removing all pairs of the form $(\emptyset, g)$ from the word $(N_1 \backslash \{B\}, \exists B[g_1])(N_2 \backslash \{B\}, \exists B[g_2])\ldots$.

The following proposition lists some basic results which we need in the proof of other theorems:

Proposition 8.7
Let $C = \langle Q, Nam, T, q_0 \rangle$ be a constraint automaton, $B \in Nam$ be a port name, and $\exists B[C]$ be the constraint automaton resulting from hiding of B in C (see Definition 3.10). Then,
(i) $tr(\exists B[C]) = \{$**hide** B **in** $\sigma |\ \sigma \in tr(C)\}$.
(ii) $sfail(\exists B[C]) = \{(\textbf{hide}\ B\ \textbf{in}\ \sigma, A) |\ (\sigma, A \cup A' \cup \widehat{B}) \in sfail(C)\}\}$, where
$A' = \{(N \cup \{B\}, g) |\ \exists g' \in DC(N, data){:}(N, g') \in A\}$,
$\widehat{B} = \{(\{B\}, g) |\ g \in DC(\{B\}, data)\}$.
(iii) $stable(\exists B[C]) = stable(C) \wedge \forall g \in DC(\{B\}, Data){:}(\{B\}, g) \notin tr(C)$.
(iv) $divtr(\exists B[C]) = \{$**hide** B **in** $\sigma |\ \sigma \in divtr(C)\} \cup$
$\{$**hide** B **in** $\sigma | \sigma \in inftr(C)\ \wedge |$**hide** B **in** $\sigma| < \infty\}$.
(v) $dfail(\exists B[C]) = \{(\textbf{hide}\ B\ \textbf{in}\ \sigma, A) |\ (\sigma, A \cup A' \cup \widehat{B}) \in dfail(C)\}\ \cup$
$(divtr(\exists B[C]) \times 2^{\Sigma})$, where, Σ is so defined in Definition 8.9.
(vi) $inftr(\exists B[C]) = \{$**hide** B **in** $\omega |\ \omega \in inftr(C) \wedge |$**hide** B **in** $\omega| = \infty\}$.

Proof.
(i) This is a direct consequence of Definitions 8.5 and 8.9.

(ii) If $(\rho, A) \in sfail(\exists B[C])$, then for the automaton $(\exists B[C])$, we know that there is a state $q \in Q$ where $q_{0,B} \stackrel{\rho}{\Longrightarrow} q$ 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 the automaton C, $q_0 \stackrel{\sigma}{\Longrightarrow} q$. Because, q is stable in $\exists B[C]$, there is no transition of the form $q \stackrel{\tau}{\rightarrow}_B q'$, and 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 all $(N, g) \in A$, $B \notin A$. 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 the 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. It follows that, $sfail(\exists B[C]) \subseteq \{(\textbf{hide}\ B\ \textbf{in}\ \sigma, A) |\ (\sigma, A \cup A' \cup \widehat{B})\}$.
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 $q \in Q$ where $q_0 \stackrel{\sigma}{\Longrightarrow} q$ 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 $q \stackrel{a}{\rightarrow} q'$,

$a \in \{(\{B\}, g) | g \in DC(\{B\}, data)\}$, and q is stable in $\exists B[C]$. Thus, (ρ, A) is a stable failure of $\exists B[C]$. Therefore, $\{(\textbf{hide } B \textbf{ in } \sigma, A) | \ (\sigma, A \cup A' \cup \widehat{B})\} \subseteq \ sfail(\exists B[C])$.

(iii),(iv) These are direct consequences of Definitions 8.5 and 3.10.

(v) By Lemma 8.1(c), $dfail(\exists B[C]) = sfail(\exists B[C]) \cup (divtr(\exists B[C]) \times 2^{\Sigma})$. Thus, using 8.7(ii),

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

The effect of the replacement of $sfail$ by $dfail$ in Equation (*) is that new pair $(\textbf{hide } B \textbf{ in } \sigma, A)$ may be introduced where $\sigma \in divtr(C)$. But by Definition 8.9, 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 its righthand side.
(vi) This is a direct consequence of Definitions 8.5 and 8.9. □

Now, we can prove that CFFD is a congruence with respect to hiding of port names of constraint automata:

Proposition 8.8 Let C and C' be 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 8.7(iii),
$stable(\exists B[C]) = stable(C) \wedge \forall g \in DC(\{B\}, Data)\text{:}(\{B\}, g) \notin tr(C)$.
Because $C \overset{cffd}{\approx} C'$, $stable(C) = stable(C')$, $divtr(C) = divtr(C')$ and $sfail(C) = sfail(C')$. By Lemma 8.1(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 8.7(ii),

- $sfail(\exists B[C]) = \{(\textbf{hide } B \textbf{ in } \sigma, A) | \ (\sigma, A \cup A' \cup \widehat{B}) \in sfail(C)\}\}$,
- $A' = \{(N \cup \{B\}, g) | \ \exists g' \in DC(N, data)\text{:}(N, g') \in A\}$,
- $\widehat{B} = \{(\{B\}, g) | \ g \in DC(\{B\}, data)\}$.

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 8.7(iv), $divtr(\exists B[C])$ is equal to $\{\textbf{hide } B \textbf{ in } \sigma | \ \sigma \in divtr(C)\} \cup \{\textbf{hide } B \textbf{ in } \sigma | \sigma \in inftr(C) \wedge |\textbf{hide } B \textbf{ in } \sigma| \langle \infty\}$. Because $C \overset{cffd}{\approx} C'$, $inftr(C) = inftr(C')$ and $divtr(C) = divtr(C')$. Therefore, $divtr(\exists B[C]) = divtr(\exists B[C'])$.

(iv) Because $C \overset{cffd}{\approx} C'$, $inftr(C) = inftr(C')$. Thus, using Proposition 8.7(vi), we know that $inftr(\exists B[C]) = inftr(\exists B[C'])$. □

Therefore, CFFD-equivalence is a congruence with respect to the hiding of port names in constraint automata. Similarly, we prove that NDFD is a congruence with respect to the hiding operator for constraint automata:

Proposition 8.9 Let C and C' be 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:
$stable(\exists B[C]) = stable(\exists B[C'])$, $divtr(\exists B[C]) = divtr(\exists B[C'])$ and
$inftr(\exists B[C]) = inftr(\exists B[C'])$ are similar to the proofs of their counterparts in Proposition 8.8. Further, by Proposition 8.7(v),
$dfail(\exists B[C]) = \{(\textbf{hide } B \textbf{ in } \sigma, A) | \ (\sigma, A \cup A' \cup \widehat{B}) \in dfail(C)\} \cup (divtr(\exists B[C]) \times 2^{\Sigma})$.
Because $C \overset{ndfd}{\approx} C'$, $dfail(C) = dfail(C')$. As we showed, $divtr(\exists B[C]) = divtr(\exists B[C'])$.
Thus, $dfail(\exists B[C]) = dfail(\exists B[C'])$. □

Thus, NDFD-equivalence is a congruence with respect to the hiding of port names in constraint automata.

8.5 Linear Temporal Logic of Constraint Automata

Traditionally temporal logics are logical systems for specification and verification of the properties that are based on the truth values of propositions in the states of a transition system. Such transition systems are called Kripke structures. Linear models (see Definition 8.10) are simplifications or runs of Kripke structures. On the other hand, labeled transition systems and constraint automata are transition systems with labels on their transitions. Also, process algebraic equivalences and composition operators usually work purely on information that is based on transition labels. In this section, we augment the definitions of labeled transition systems and constrain automata by introducing functions that assign to each of their states a set of propositions. Then, we introduce linear temporal logic and two of its fragments interpreted over linear models as executions of augmented labeled transition systems or augmented constraint automata.

Definition 8.10
(i) Let AP be a set of atomic propositions. A *Linear Model* is a finite or infinite sequence $\sigma = \sigma_1, \sigma_2, \ldots$ of subsets of AP. We call any $\sigma_i \subseteq AP$ a state of (in) the linear model σ.
(ii) An *augmented labeled transition system (aLTS)* is a 5-tuple $A = \langle S, s, \Delta, AP, L \rangle$, where, $\langle S, s, \Delta \rangle$ is an LTS, AP is a set of propositions, and $L{:}S \rightarrow 2^{AP}$ is a labeling function. Let $\sigma \in \Sigma^{\omega}$ be an infinite trace of the LTS $\langle S, s, \Delta \rangle$. Because σ is an infinite trace, there is an infinite (or deadlocking) sequence of LTS (S, s, Δ), of the form $r = (s, \sigma_1, s_1), (s_1, \sigma_2, s_2), \ldots$. The *linear model* defined by r in A is $M_r = L(s), L(s_1), L(s_2), \ldots$.

(iii) A tuple $C = \langle Q, Nam, T, q_0, AP, L\rangle$ is called as an *augmented constraint automaton* (aCA) where $\langle Q, Nam, T, q_0\rangle$ is a constraint automaton, AP is a set of propositions, and $L{:}Q \rightarrow 2^{AP}$ is a labeling function. Let C be an aCA and $r = (q_0, \phi_1, q_1), (q_1, \phi_2, q_2), \ldots$ be an infinite or deadlocking run of C. The *linear model* defined by r in C is $M_r = L(q_0), L(q_1), L(q_2), \ldots$.

Now, we present the syntax and semantics of linear temporal logic and two of its fragments:

Syntax of LTL and its fragments

(i) The set of all well-formed formulas of *linear temporal logic (LTL)* is defined by the following abstract syntax:

$$\phi ::= \ P \mid \neg\phi \mid \phi \vee \phi \mid \phi U \phi \mid X\phi \qquad P \in AP$$

(ii) The set of all well-formed formulas of *Next-time-less linear temporal logic (LTL_{-X})* is defined by the following abstract syntax:

$$\phi ::= \ P \mid \neg\phi \mid \phi \vee \phi \mid \phi U \phi \qquad P \in AP$$

(iii) The set of all well-formed formulas of *restricted linear temporal logic (LTL_{ω})* is defined by the following abstract syntax:

$$\phi ::= \ P \mid \neg\phi \mid \phi \vee \phi \mid \phi U \phi \mid \overset{\omega}{F}\ \phi \qquad P \in AP$$

We also use the following abbreviations:

$$\top \equiv_{df} (p \vee (\neg p))$$

where p is a fixed proposition,

$$\phi_1 \wedge \phi_2 \equiv_{df} \neg(\neg\phi_1 \vee \neg\phi_2),$$

$$F\phi \equiv_{df} \top U \phi,$$

and

$$G\phi \equiv_{df} \neg F(\neg\phi).$$

Semantics of LTL and its fragments

A temporal formula ϕ of the above defined syntactic structures holds in a linear model σ (denoted by $\sigma \vDash \phi$) according to the following rules:

1- If $\phi \in AP$, then $\sigma \vDash \phi$ iff $\phi \in \sigma_1$.
2- $\sigma \vDash \neg\phi$ iff not $\sigma \vDash \phi$.
3- $\sigma \vDash (\phi_1 \vee \phi_2)$ iff $\sigma \vDash \phi_1$ or $\sigma \vDash \phi_2$
4- $\sigma \vDash (\phi_1 U \phi_2)$ iff $\exists i{:}0 \leq i < |\sigma|, \sigma^i \vDash \phi_2$ *and* $\forall j{:}0 \leq j < i, \sigma^j \vDash \phi_1$.
5- $\sigma \vDash X\phi$ iff $\sigma^1 \neq \emptyset$ and $\sigma^1 \vDash \phi$.
6- $\sigma \vDash \overset{\omega}{F}\ \phi$ iff there are infinitely many $i \geq 0$ such that $\sigma^i \vDash \phi$.

In terms of expressiveness power, it can be shown that $LTL_{-X} \subset LTL_{\omega} \subset LTL$. In general, in LTL we have, $\overset{\omega}{F}\ \phi \equiv GXF\phi$, but if we restrict to infinite linear models only then $\overset{\omega}{F}\ \phi \equiv GF\phi$. Therefore, the temporal operator $\overset{\omega}{F}$ is an operator for distinguishing a finite linear model from an infinite one, i.e., distinguishing a *deadlock* from a *divergence*.

Definition 8.11 Let $\sigma = \sigma_1, \sigma_2, \ldots$ be a linear model.
(i) The *finitely reduced form* of σ (denoted by $fred(\sigma)$) is constructed by collapsing all finite continuous sequences $\sigma_i, \sigma_{i+1}, \ldots, \sigma_{i+m}$ of identical elements $\sigma_i = \sigma_{i+1} = \ldots = \sigma_{i+m}$ to one element σ_i.
(ii) The *reduced form* of σ (denoted by $red(\sigma)$) is constructed by collapsing all finite and infinite continuous sequences $\sigma_i, \sigma_{i+1}, \ldots$ of identical elements $\sigma_i = \sigma_{i+1} = \ldots$ to one element σ_i.
(iii) If σ_1 and σ_2 are two linear models, we say that σ_1 and σ_2 are *equivalent under stuttering* iff $red(\sigma_1) = red(\sigma_2)$.

By induction on the syntactic structure of formulas, we obtain the following proposition.

Proposition 8.10 Let $\sigma = \sigma_1, \sigma_2, \ldots$ be a linear model.
(i) If ϕ is an LTL_{ω}-formula, then $\sigma \vDash \phi$ iff $fred(\sigma) \vDash \phi$.
(ii) If ϕ is an LTL_{-X}-formula, then $\sigma \vDash \phi$ iff $red(\sigma) \vDash \phi$.

In the context of model checking, we use aLTSs and aCAs as the models of our systems and also as the semantic domain of our temporal logic. On the other hand, we want to use of the equivalence relations to reduce the models' sizes. This equivalence based reduction will be useful in model checking if the reduction process preserves the truth values of each temporal logic formula. Now we intend to formally define the concept of preservation of the truth values of temporal formulas according to each equivalence relation. For this, we can use a way of interpreting the transition labels as functional state transformers [86]. In this section, we use this transformation only for defining the concept of truth preservation, but in the next section we will use a modified version of it in our reduction algorithm.

Definition 8.12
(i) A *state modifier* sm is a mapping $sm{:}2^{AP} \rightarrow 2^{AP}$. The set of all state modifiers is denoted by TS. The identity state modifier I is the identity function. A *state modifier sequence* is a finite or infinite sequence of state modifiers.
(ii) A *temporal semantics* for an LTS or constraint automaton L is a mapping $f{:}\Sigma(L) \cup \{\tau\} \rightarrow TS$ such that $f(\tau) = I$. If $\rho = a_1 a_2 \ldots$ is a path of L, we write $f(\rho)$ for the sequence $(f(a_1), f(a_2), \ldots)$. A temporal semantics for a path ρ is a mapping $f{:}\Sigma(\rho) \cup \{\tau\} \rightarrow TS$ such that $f(\tau) = I$.
(iii) The linear model induced by a state $\nu \subseteq AP$ and a state modifier sequence sms, denoted as $Model(\nu, sms)$, is a sequence of states such that:
1- $Model(\nu, sms)_0 = \nu$
2- $Model(\nu, sms)_{i+1} = sms_i(Model(\nu, sms)_i)$.
If sms is finite then $|Model(v, sms)| = |sms| + 1$.
(ix) Let $\sigma \in (\Sigma_{\tau}^{*} \cup \Sigma_{\tau}^{\omega})$ be a path of an LTS L, f a temporal semantics for σ, ν_0 a state, and ϕ an LTL formula. We say ϕ is true of σ with respect to temporal semantics f and initial state ν_0 and write $\sigma, f, \nu_0 \vDash \phi$ iff $Model(\nu_0, f(\sigma)) \vDash \phi$.

Usually, linear temporal logic formulas are interpreted over the complete paths generated by a transition system. These correspond to the infinite and deadlocking paths of an LTS.

Definition 8.13 (i) Let L be an LTS, f a temporal semantics for L, ν_0 a state, and ϕ an LTL formula. We say ϕ is true of L with respect to temporal semantics f and initial state ν_0, and write $L, f, \nu_0 \models \phi$ iff $\sigma, f, \nu_0 \models \phi$ for all $\sigma \in dpath(L) \cup infpath(L)$.
(ii) Let L_1 and L_2 be LTSs and ϕ an LTL-formula. We say that L_1 and L_2 agree on ϕ iff for every temporal semantics f and for every initial state ν_0 it is the case that $L_1, f, \nu_0 \models \phi$ iff $L_2, f, \nu_0 \models \phi$.
(iii) An equivalence $\approx$ between LTSs is *LTL-preserving* iff for any pair L_1 , L_2 such that $L_1 \approx L_2$, L_1 and L_2 agree on every LTL formula. Similarly, An equivalence $\approx$ between LTSs is LTL_{-X} (LTL_ω)-preserving iff for any L_1 , L_2 such that $L_1 \approx L_2$, L_1 and L_2 agree on every LTL_{-X} (LTL_ω) formula.

Let L be a labeled transition system. Intuitively, a temporal semantics for L expresses the changes caused by the transitions in the information contained in each state of L. But, L can be composed with other labeled transition systems using composition operators defined in Definitions 8.2 and 8.3, and in addition, in the case of constraint automata, using join and hiding operators defined in Chapter 3. Thus, we need to define how a composition operator affects the temporal semantics of the original labeled transition systems, which will be composed using these operators.

For the composition operators defined in in Definitions 8.2 and 8.3, all temporal semantics for compositional labeled transition systems have been defined in [86]. Also, it was shown in [86] that:

Proposition 8.11 For each labeled transition system and with respect to all composition operators that have well defined temporal semantics:
(i) CFFD-equivalence is LTL_ω-preserving and NDFD-equivalence is LTL_{-X}-preserving.
(ii) If $\approx$ is an equivalence between LTSs and it is congruence with respect to $||[\cdots]||$ and $[]$ (defined in Definition 8.2) and is LTL_ω-preserving, then $L \approx L'$ implies $L \overset{cffd}{\approx} L'$. Thus, CFFD is the weakest compositional equivalence preserving LTL_ω.
(iii) If $\approx$ is an equivalence between LTSs and it is congruence with respect to $||[\cdots]||$ and $[]$ and is LTL_{-X}-preserving, then $L \approx L'$ implies $L \overset{ndfd}{\approx} L'$. Thus, NDFD is the weakest compositional equivalence preserving LTL_{-X}.

The proof of Proposition 8.11(i) depends only on the definitions of the equivalences, temporal semantics, and the notion of temporal logic preservation (see [86]). According to Proposition 8.11(ii),(iii), the minimality property holds whenever an arbitrary equivalence $\approx$ is a congruence with respect to the parallel composition $||[\cdots]||$ and non-deterministic choice $[]$ operators. We have shown that every constraint automaton $C = \langle Q, Nam, T, q_0 \rangle$ can be considered as a labeled transition system with alphabet $\Sigma = \{(N, g) | N \subseteq Nam \wedge g \in DC(N, Data) \wedge N \neq \emptyset\}$ (Proposition 8.3) and proved that CFFD and NDFD-equivalences are congruences with respect to our defined join and hiding operators for constraint automata. Thus, if we can define the temporal semantics of the composed constraint automaton by means of the temporal semantics of the original automata, then all parts of Proposition 8.11

will hold not only for constraint automata composed by operators defined in Definitions 8.2 and 8.3, but also when they are composed by join and hiding defined in Chapter 3.

Now we investigate the effects of join and hiding composition operators for constraint automata on their temporal semantics. First, we need to make precise the meaning of *effects* of a composition operator on a temporal semantics:

Definition 8.14
(i) A state modifier sm affects an atomic proposition a iff there is a $\nu \subseteq AP$ such that either $a \in \nu$ and $a \notin sm(\nu)$, or $a \notin \nu$ and $a \in sm(\nu)$. We denote by $af(sm)$ and $\overline{af}(sm)$, the set of all atomic propositions affected by sm and the set $AP \backslash af(sm)$.
(ii) State modifiers sm and sm' are *compatible* iff for all atomic proposition $a \in af(sm) \cap af(sm')$ and all $\nu \subseteq AP$, $a \in sm(\nu)$ iff $a \in sm'(\nu)$. If this is the case, the *combination* of sm and sm', denoted by $sm \oplus sm'$, is the function $c{:}2^{AP} \rightarrow 2^{AP}$ where $c(\nu) = (sm(\nu) \cap af(sm)) \cup (sm'(\nu) \cap af(sm')) \cup (\nu \cap \overline{af}(sm) \cap \overline{af}(sm'))$.

Definition 8.15 Let $C_1 = \langle Q_1, Nam_1, T_1, q_{01} \rangle$ and $C_2 = \langle Q_2, Nam_2, T_2, q_{02} \rangle$ be two constraint automata, and f_1 and f_2 be temporal semantics for C_1 and C_2, respectively. f_1 and f_2 are *compatible with respect to synchronization set* $G = \{(N, g) | \ (N = N_1 \cup N_2) \wedge (g = g_1 \wedge g_2) \wedge (N_1 \neq \emptyset) \wedge (N_2 \neq \emptyset) \wedge (N_1 \subseteq Nam_1) \wedge (N_2 \subseteq Nam_2) \wedge (g_1 \in DC(Nam_1, Data)) \wedge (g_2 \in DC(Nam_2, Data)) \wedge (N_1 \cap Nam_2 = N_2 \cap Nam_1)\}$, iff for all $g \in G$, $f_1(g)$ and $f_2(g)$ are compatible and for all $l \in (\Sigma(C_1) \cap \Sigma(C_2)) \backslash G$, $f_1(l) = f_2(l)$.

Let $C = C_1 \bowtie_C C_2$ and f_1 and f_2 be temporal semantics for C_1 and C_2, respectively. The state information of C consists of the state information of both C_1 and C_2. The temporal semantics f_1 expresses changes in the state information of C_1 and f_2 that of C_2. These changes are made by transitions. Thus, if a transition in C corresponds to a transition of C_1 alone, the change in the state information of C is the same as in C_1. Also, if a transition in C corresponds to a transition of C_2 alone, the change in the state information in C is the same as in C_2. But, if a transition in C corresponds to synchronized transitions of C_1 and C_2, the change in the state information of C should consist of both changes in C_1 and C_2. The set G defined in Definition 8.15 identifies the set of all synchronization alphabets. If the temporal semantics f_1 and f_2 are mutually conflicting, it is impossible to define a joint temporal semantics in the case of synchronization. In Definition 8.15, the compatibility requirement guarantees the possibility of joining two temporal semantics. Thus, based on the above definition, the temporal semantics for $C_1 \bowtie_C C_2$ can be characterized as:

Proposition 8.12 Let $C_1 = \langle Q_1, Nam_1, T_1, q_{01} \rangle$ and $C_2 = \langle Q_2, Nam_2, T_2, q_{02} \rangle$ be two constraint automata, f_1 and f_2 be temporal semantics for C_1 and C_2, respectively, and let f_1 and f_2 are compatible with respect to set G (defined in Definition 8.15). The temporal semantics for $C = C_1 \bowtie_C C_2$ is the function f such that:
$\forall l \in G{:}f(g) = f_1(l) \oplus f_2(l)$, $\forall l \in \Sigma(C_1) \backslash G{:}f(l) = f_1(l)$ and $\forall l \in \Sigma(C_2) \backslash G{:}f(l) = f_2(l)$.

Let $\exists B[C]$ be the constraint automaton resulting from hiding of $B \in Nam$ in constraint automaton $C = (Q, Nam, T, q_0)$, and f_1 be a temporal semantics for C. To characterize the temporal semantics of $\exists B[C]$, first note that every transition label of the form $(\{B\}, g)$ in C is a transition with the label τ in $\exists B[C]$. Because τ-transitions do not affect information of states (see Definition 8.12(ii)), it must be that $f_1(l) = I$, for all $l \in \{(\{B\}, g) | g \in$

$DC(Nam, Data)\}$, where I is the identity function. If this condition holds, the temporal semantics of $\exists B[C]$ must do the same changes to the information of states as the temporal semantics of C does. This can be expressed by: $\forall(N, g) \in \Sigma(C)$:$f_1((N \backslash \{B\}, \exists B[g])) = f((N, g))$ and $f_1(\tau) = I$. Thus:

Proposition 8.13 Let $C = \langle Q, Nam, T, q_0 \rangle$ be a constraint automaton, $B \in Nam$, and f_1 be a temporal semantics for C. The temporal semantics for $\exists B[C]$ can be defined iff $\forall l \in \{(\{B\}, g) | g \in DC(Nam, Data)\}$:$f_1(l) = I$. If this is the case, then, $\forall(N, g) \in \Sigma(C)$:$f_1((N \backslash \{B\}, \exists B[g])) = f((N, g))$ and $f_1(\tau) = I$.

Based on Propositions 8.12 and 8.13, we have a well-defined temporal semantics for the join and hiding operators of constraint automata. Because of our translation of constraint automata to labeled transition systems, we have the followings:

Proposition 8.14 For each constraint automaton and with respect to all composition operators defined in Definitions 8.2 and 8.3 extended with the join and hiding operators defined in Chapter 3:
(i) CFFD-equivalence is LTL_ω-preserving and NDFD-equivalence is LTL_{-X}-preserving.
(ii) If $\approx$ is an equivalence between constraint automata and it is congruence with respect to $||[\cdots]||$ and $[]$, and it is LTL_ω-preserving, then $C \approx C'$ implies $C \overset{cffd}{\approx} C'$. Thus, CFFD is the weakest compositional equivalence over the set of constraint automata preserving LTL_ω.
(iii) If $\approx$ is an equivalence between constraint automata and it is congruence with respect to $||[\cdots]||$ and $[]$, and it is LTL_{-X}-preserving, then $C \approx C'$ implies $C \overset{ndfd}{\approx} C'$. Thus, NDFD is the weakest compositional equivalence over the set of constraint automata preserving LTL_{-X}.

8.6 Reduction Algorithms

The process of model checking contains three main steps: 1- Modeling of the actual system using a formal system such as aLTS or aCA. 2- Expressing the requirement or property that we want to verify by using a formula of a temporal logic. 3- Using a model checking algorithm for deciding if the formula is true in the model or not. In our method for model checking of an aLTS or aCA, before doing the third step, we need to reduce the size of the model by using an equivalence relation. This reduction process can be done before or after defining the property or formula that we need to verify. Thus, the reduction can be done before the second or the third step of the model checking process. In this section we present some algorithms for reducing the sizes of aLTSs and aCAs, while preserving NDFD and CFFD-equivalences. The method that we present here is a modification of the algorithm introduced in [141, 86]. Then, we consider the case where we reduce the model after defining a property or a set of properties that we need to verify.

The algorithms for minimizing an aLTS $A = \langle S, s, \Delta, AP, L \rangle$ (or an aCA) with respect to CFFD and NDFD-equivalences have three main steps:

1- Converting the aLTS or aCA A into an *acceptance graph* AG, which relies on the process of converting a finite automaton to its deterministic counterpart. Each node of the graph AG contains a set of states $D \subseteq S$. For each node of the graph AG all states in D are reachable from an initial state by using the same finite divergence trace.

2- Labeling of the nodes of the acceptance graph (deterministic automaton) with the information about stability, divergences, stable failures and non-divergent failures (see [141] for the detail of this part of the labeling process). We also label each node of the acceptance graph with a set of propositions that can be true in it. To determine this set of propositions, let n be a node of the acceptance graph AG that contains the set of states $D \subseteq S$. Let P be the union set of all $L(d)$ where $d \in D$. Obviously, for every $p \in P$, there is a finite trace in aLTS A in the last state of which the proposition p is true. Thus, we label the node n with the set P.

3- Minimizing the acceptance graph (labeled deterministic automaton) by using traditional algorithms for minimizing finite automata. In this step, we must partition the set of all states (nodes). This first level partitioning is done by considering both the propositions that hold in states and the requirements of the intended equivalence. Two states are in the same class, if the sets of propositions that hold in them are compatible. Also, in the case of the CFFD-equivalence, two states with the same stability, divergent traces, and stable failures belong to the same class. In the case of the NDFD-equivalence, two states with the same stability, divergent traces, and divergence-masked failures belong to the same class.

In practice, the main advantage of reducing models with respect to an equivalence, without considering any property to be verified, is that we can run the reduction algorithm once and use the minimized models whenever we need to model check a property. The property must be expressible in a temporal logic that the equivalence preserves it. But suppose that we need to model check a formula or a set of formulas whose set of atomic propositional constituents is $A \subset AP$ and $|A| \ll |AP|$ (the size of A is very much smaller than the size of AP). In cases like this, the above method is not efficient in practice, because in the first phase of partitioning (in step 3), several states that agree on the truth values of the members of A may be allocated in different classes based on their different truth values of the other members of AP. This implies that the size of the model is not reduced much. Thus, in such cases, we first define the property or the set of properties that we need to verify, and then filter all sets of propositions assigned to the states of the model such that they contain only subsets of A. We call the resulting model a *filtered model*. Then, we run the above reduction method on the filtered model.

From the worst case complexity analysis point of view, it can be shown that all of the above reduction algorithms are exponential in the size of the input model. This is true not only for reductions based on CFFD and NDFD-equivalences, but also for a wide range of equivalences and simulation relations defined in automata theory, graph theory, Petri Nets, and process algebras [35]. Experience in all of these fields indicates that, in practice, the worst case rarely happens (for more references to these experiences and a detailed discussion about the complexity of failure based equivalences see [140]).

8.7 Compositional Model Checking

In this section we present a method for compositional model checking of a component-based system and its coordinating subsystem using the above mentioned equivalences to minimize their formal models. A component-based system has two main parts: a set of components and a coordinating subsystem (glue code). Using Reo specifications, one can specify or model the coordinating subsystems in a compositional and hierarchal way. Using constraint automata, not only the coordinating subsystem, but also all components can be modeled as constraint automata in a compositional way. Thus, the methods of compositional reasoning not only can be applied on the coordinating subsystem , but also on the whole component-based system. Fortunately, our above process algebraic discussions enable us to use the equivalence based compositional reduction method in both cases.

Verification of coordinating subsystem. In this case we need to verify the desired properties of the coordinating subsystem of a component-based system. If we consider the coordinating subsystem (for example a Reo circuit or a constraint automaton) as a complete system, the set of the components of the component-based system is its environment. Externally visible actions of this coordinating subsystem are the *read (input or get)* and *write (output or put)* operations it uses to communicate with the environment. (In Reo these operations involve only the boundary nodes of the circuite.) The rest of the actions within the coordinating subsystem, and its internal states are not interesting, if only the correct functionality of the coordinating subsystem is of concern. Thus, the main steps of model checking of the desired properties of the coordinating subsystem consist of:

1- Modeling the behavior of connectors and the observable behavior of components by augmented constraint automata. Because in this case all actions are considered as visible, none of the constraint automata models have τ-transitions.

2- Expressing the desired property by an LTL_{-X} or LTL_{ω} formula.

3- According to the type of the property to verify, using an equivalence relation for reducing the constraint automata models.

4- Composing the reduced constraint automata models using join and hiding operators. Because we proved that CFFD and NDFD are congruences for all composition operators defined in this paper, the composed model will be reduced by itself.

5- Use one of the ordinary LTL model checking algorithms on the minimized model (for the algorithms of LTL model checking see [50]).

Note that because of the minimization, the efficiency of our method is better than applying LTL model checking algorithms directly. Moreover, according to step 4 above, any improvement in the ordinary LTL model checking algorithms, improves the efficiency of our method.

Example 8.1 (Dining Philosophers) The classical dining philosophers problem can be described as a coordination system in Reo [14]. This system can be designed as a set of pairs of instances of two components: philosopher and chopstick. As illustrated in Figures 8.1(a) and (b), the interface of philosopher i has four output ports: lt_i, rt_i, lf_i and rf_i, which serve to take and return the chopsticks on the left- and right-hand sides of the philosopher.

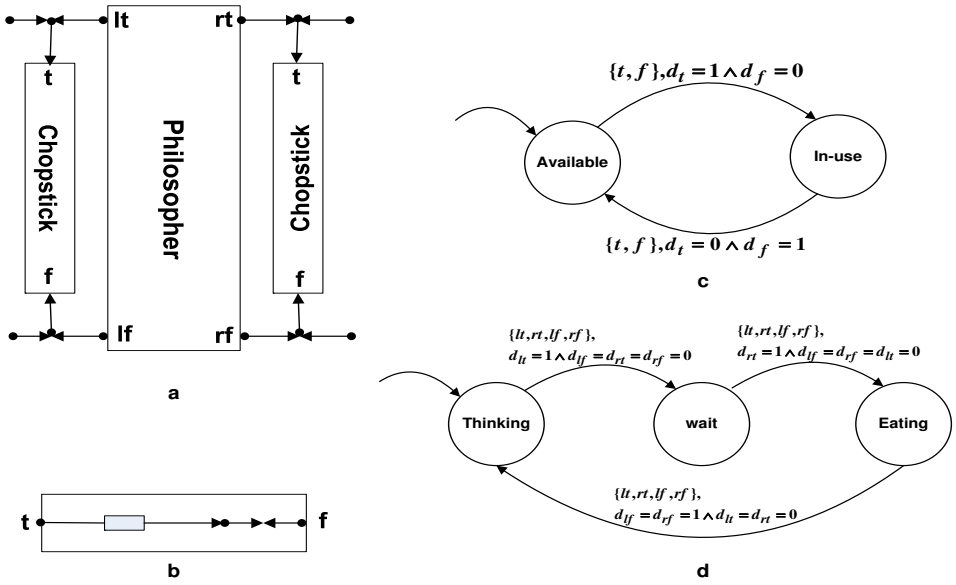

Figure 8.1: (a) Dining philosophers scenario in Reo and (b) a chopstick, (c) minimized constraint automaton for a chopstick and (d) a philosopher

The externally observable behavior of a philosopher component is as follows. After some period of thinking, it decides to eat, attempts to obtain its two chopsticks by issuing requests on its lt_i and rt_i ports. We assume that it always issues a request through its left chopstick before requesting the one on its right. Once both of its take requests are granted, it proceeds to eat for some time, at the end of which it then issues requests to free its left and right chopsticks by writing tokens on its lf_i and rf_i ports. A chopstick component has two input ports: t_i for take and f_i for free requests. Every chopstick is modeled by a FIFO1 channel and a synchronous drain [14]. The constraint automata for the interfaces of the philosophers and the chopsticks are shown in Figures 8.1(c) and (d). Note that the constraint automaton of a chopstick is obtained as a minimized product automaton of the constraint automata for FIFO1 and SyncDrain connectors.

Verification of the whole component-based system. The dining philosophers is an example of a system in which all components can be modeled by constraint automata without any internal action. It is a very restrictive assumption that all components can be modeled so. We need to consider a more general case: components are transition systems that have both internal and external actions and connectors are constraint automata all of whose actions are visible. In such cases, we can simply model any component by a labeled transition system and the coordinating system by a compositional constraint automaton. Labeled transition systems can be embedded in constraint automata as was shown in Lemma 8.3. The equivalence relations CFFD and NDFD are used to reduce the sizes of all constraint automata models and

then they are composed. Thus, the main steps of model checking of a complete component-based system, except the first, will be the same as we described for the coordination system. The first step is replaced with the following step and the other steps remain the same as the model checking algorithm for coordination subsystems:

1'- Model each component by an augmented labeled transition system and translate it to an augmented constraint automaton; and model the set of connectors directly by some augmented constraint automata.

Example 8.2 (A Resource Allocation System) As an example of a component based system, consider a resource allocation system with the requirement of mutual exclusion in using a resource as illustrated in Figure 8.2(a). The system consists of n processes which sometimes need to have access to a limited resource. The resource can be used by only one process at a time and it must be guaranteed that all requests for the resource are eventually granted. Also, if a process P has requested to use the resource, no other process is granted access to the resource more than once before the request of P has been granted. We suppose that each process has two ports: an output port rq through which it announces its request for using the resource and an input port gr through which the coordinator allows the process to access the resource and enter its critical section. The constraint automaton model of each process is shown in Figure 8.2(b). In this figure, state q_0 is the initial state of the process. In state q_1 the process announces its request for using the resource and waits for permission to access it. After receiving a signal gr, the process enters its critical section modeled by the state q_2. Once the process has finished using the resource, it turns the signal rq off and waits until the coordinator notices this and turns the signal qr off.

The coordination scenario performed by the coordinator to manage the resource is a pooling based allocation. For processes $P1$ to Pn sequentially, if rq_i is turned on, the coordinator turns the signal gr_i on and waits to observe the turning off of the signal rq_i. Then, it turns signal gr_i off. Figure 8.2(c) shows the constraint automaton model of the coordinator when there are two processes in the system. In this case, the last state P_4 is the same as the initial state P_0. This model can easily be generalized for the case of n processes.

This resource allocation system is an example of a component based system in which components (processes) are modeled by labeled transition systems, which can be embedded in constraint automata. In this case we model all internal actions by $\tau-$transitions. The coordinating subsystem is modeled directly by constraint automata without the need for $\tau-$transitions. In the next section, we report our results in compositional minimization of the resource allocation system using CFFD and NDFD equivalences.

Note that there are other compositional reasoning methods, such as the assumption-guarantee method [123], in which the reasoning is done separately on the components of the model by decomposing the desired property formula. Such techniques of compositional reasoning can be used in conjunction with our proposed minimization. We have not considered these methods here.

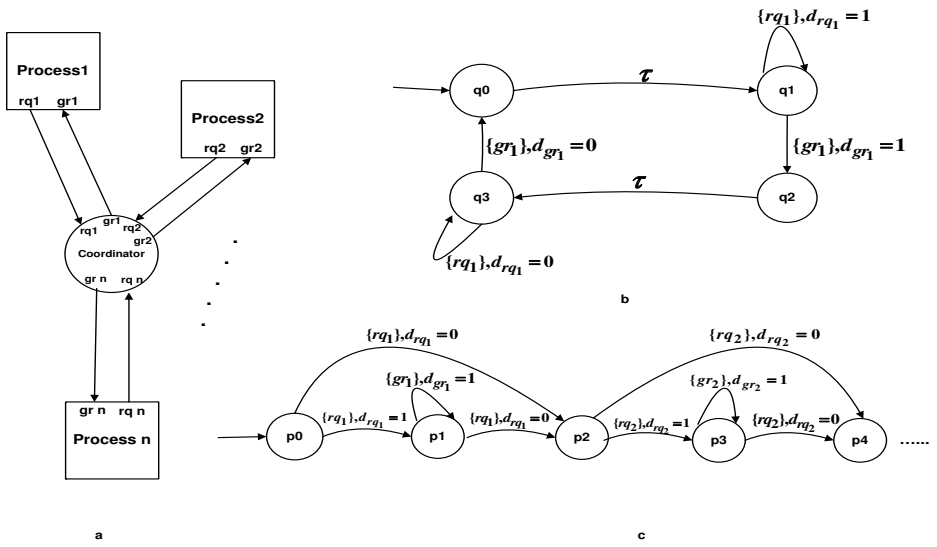

Figure 8.2: (a) A resource allocation system, (b) constraint automaton model of a process, (c) constraint automaton model of the coordinator

8.8 Case studies

We designed and implemented a tool for modeling and verification of systems modeled as constraint automata. One of the main goals of this tool is preparing an environment for specification of software architectures using constraint automata and verification of their properties, especially non-functional and qualitative properties of software architectures. Because of this aim the tool is called *ArQuVer* (Architecture Quality Verification tool). *ArQuVer* was implemented using Java. It receives the descriptions of a constraint automaton in XML format and can perform all composition operators defined in this thesis on them. It contains also components for minimizing constraint automata using (bi)simulation and CFFD equivalences. We can use its component for CFFD-minimization for NDFD-minimizing as well, through an intermediate software component.

Example 8.3 (Inres Protocol) As a case study, we considered the Inres protocol based system [63], as a component-based system whose coordinating subsystem can be modeled directly by a constraint automaton and its components can be modeled by labeled transition systems that can be transformed into constraint automata. The Inres protocol implements a reliable, connection oriented data transfer service, the *Inres service*, between two users. The main architecture of the protocol is shown in Figure 8.3. The Inres service is not symmetrical: it offers only a one way transition from an initiating process to a responding process. The

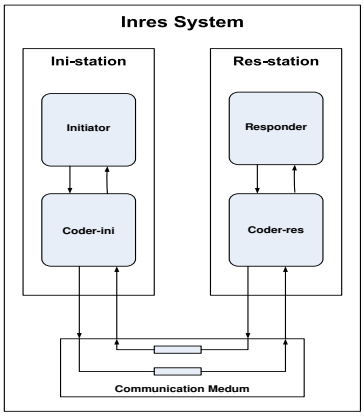

Figure 8.3: Inres protocol architecture (the connectors are Reo primitive channels)

protocol itself operates on top of a medium that offers a data transfer service. A description of the Inres protocol using the SDL language is presented in [54]. This description consists of four main processes: *Initiator* and *Responder* which implement the service by exchanging the protocol data units between themselves and *Coder-ini* and *Coder-res* which are used to hide the interface to the medium.

We modeled each of these four main processes by a labeled transition system, transformed into a constraint automaton, and considered the connectors between *Initiator* and *Coder-ini* and the connectors between *Responder* and *Coder-res* as two pairs of parallel Reo Sync channels. Also, we assumed that the low level medium of communication is a pair of parallel Reo $FIFOn$ channels, in which, n is a natural number constant. In the simplest case, $n = 1$. In an Inres system, components and connectors work and communicate in a sequential manner. First, *Initiator* activates the communication process, then its request is sent to *Coder-ini* through the Sync channel between them, and so on. We compose the models of all components and connectors with the join operator in the same order as the components and connectors are activated in the protocol. We hide the names of ports that can be considered as internal and invisible (in a more abstract view of the system). In the case study, we applied the compositional minimization method, namely, we minimized all constraint automata models before composing them. The minimizations were done using bisimulation, CFFD, and NDFD equivalences.

The results of our attempt to minimize the components of the Inres system are summarized in the columns A to E of Table 8.1 (more details in [115]). In Table 8.1, column A consists of the names of the components, column B contains the number of reachable states of the constraint automata models of the components without any minimization, column C reports the results after minimizing column B's models by bi-simulation relation, column D contains minimization of column B's models by CFFD, and column E contains minimization of column B's models by by NDFD. As we expect, the order of the sizes of the models in the columns B, C, D, and E is decreasing (except than for the size of the model of the commu-

A	B	C	D	E	F	G
Component name	CA, Not-reduced	Bisimulation	CFFD	NDFD	LTS, Not-reduced	Bisimulation
Initiator	145	34	30	24	131	30
Coder-ini	97	22	15	13	92	10
Responder	44	27	24	19	35	14
Coder-res	112	13	10	8	101	10
Ini-station	3451	1506	1145	1023	3217	1424
Res-station	1129	403	332	305	947	391
Comm. Medium	9	9	9	9	–	–
Inres System	164	54	36	30	135	28

Table 8.1: Number of reachable states for the Inres protocol system.

nication medium that is the same in all columns). It means that NDFD reduces models more than CFFD, and CFFD reduces more than bi-simulation relation. This fact is exactly because the bi-simulation equivalence relation preserves a bigger set of properties than CFFD. Also, CFFD preserves more properties than NDFD. See Proposition 8.11. For the case of the communication medium we have used the minimized model in all columns.

The main importance of our work is that it is the first attempt to model the Inres system by constraint automata and minimizing them using bi-simulation, CFFD, and NDFD equivalences and that (as a case study) it shows the applicability of failure based equivalences for model checking of constraint automata models. However, to be a more realistic and comparable case study, in columns F and G of Table 8.1, we summarize the results of another attempt to minimize the Inres system as reported in [105].

In the work reported in [105], all components have been modeled by labeled transition systems, the communication medium is considered as a set of peer-to-peer channels without any buffer (not modeled) and the minimization was done using only the bi-simulation equivalence relation. In Table 8.1, column F contains of the number of reachable states of the LTS models of the components without any minimization and column G reports the results after minimizing column F's models by bi-simulation relation, as reported in [105].

To compare the two works, first note that, we have modeled all components by constraint automata and minimized them using bi-simulation, CFFD, and NDFD equivalences. However, in the work reported in [105], the models are LTSs and only bi-simulation based minimization is considered. In our work, the order of minimizing and composing models of components is the same as reported in [105]. The sizes of our basic models of components (before minimization) are slightly bigger than the sizes reported in [105] because our models are constraint automata while in [105] simple labeled transition systems are used. Also, our models contain more details (less abstracted) and we modeled all connectors and channels while in [105] these are ignored. Because we don't have access to the exact models of the components used in [105], we can not give an indication of how many extra states,

No. of Processes	No Minimization	CFFD Minimization	NDFD Minimization
2	36	24	16
5	2430	60	40
10	1180980	120	80
20	5165606520	240	160
100	$>10^{49}$	1200	800

Table 8.2: Number of reachable states for the resource allocation system.

for instance, they would have if they modeled the system at the same level of detail as we do. Conversely, considering connectors (channels) as stateless models (as it is considered in [105]), is more abstract than we need to show the applicability and usefulness of our above mentioned method of model checking of the whole component-based systems including their coordination subsystems and components.

There is no well-established numerical relation between the sizes of models before and after minimization using one of the equivalence relations. For example, while the bi-simulation reduction from 135 to 28 in [105] produces a model 21% of its original size (see last row, columns F and G in Table 8.1), our bi-simulation reduction produces a model 33% of its original size. It is, of course, because that bi-simulation reduction is not linear. In fact, for another example, the situation can be in a reverse order. More importantly, in our results, the sizes of the reduced models using bi-simulation, CFFD and NDFD relations are about 33% (column C), 22% (column D), and 18% (column E) of the sizes of the original models, respectively. This shows the effectiveness of the reductions. As reported in [105], if one tries to generate the state space of the LTS model of the Inres protocol by the SDT Validator the final LTS model will have 388408 states and over 1880000 transitions. Because our constraint automata models contain more detail than their LTS models, the constraint automaton model of the Inres protocol will be bigger than its LTS model. Thus, our obtained minimizations using all three equivalence relations are highly significant.

Example 8.4 (Reduction of the resource allocation system models)

As a simple case study, we considered the compositional minimization method for the resource allocation system introduced in Example 8.2 using CFFD and NDFD equivalences. The results based on the number of processes in the system have been summarized in Table 8.2. Note that the structure of the model of the coordinating subsystem shown in Figure 8.2(c) is completely symmetric. Adding a new process to the system adds a block of two states with symmetric structures as the previous blocks to the automaton model of the coordinating subsystem. Further, the models of all processes are isomorphic. These symmetry and isomorphic facts give us the opportunity to construct the complete system using minimized basic models by a symmetric and repeating algorithm. These facts also result in a numerical relation between the number of processes and the number of reachable states in the minimized model of the system. In Table 8.2, we see that the numbers of reachable states of each minimized model using CFFD and NDFD equivalences are respectively 12 and 8 times the number of processes. This is a very interesting example that motivates using other kinds

of compositional verification methods, such as the *symmetric techniques* for model checking [56], in conjunction with the reduction techniques we proposed in this chapter, or with the automata theoretic techniques we proposed in the previous chapters.