An Equivalence Based Method for Compositional Verification of the Linear Temporal Logic of Constraint Automata

Mohammad Izadi 1,2
Department of Computer Engineering
Sharif University of Technology and
IPM School of Computer Science
Tehran, IRAN

Ali Movaghar Rahimabadi3
Department of Computer Engineering
Sharif University of Technology and
IPM School of Computer Science
Tehran, IRAN

Abstract

Constraint automaton is a formalism to capture the operational semantics of the channel based coordination language Reo. In general constraint automaton can be used as a formalism for modeling coordination of some components. In this paper we introduce a standard linear temporal logic and two fragments of it for expressing the properties of the systems modeled by constraint automata and show that the equivalence relation defined by Valmari et al. is the minimal compositional equivalence preserving that fragment of linear time temporal logic which has no next-time operator and has an extra operator distinguishing deadlocks and a slight modification of this equivalence is the minimal equivalence preserving linear time temporal logic without next-time operator. We present a compositional model checking method based on these equivalences for component-based systems modeled by labeled transition systems and constraint automata and a simplification of it for model checking the coordinating subsystems modeled by constraint automata.

Keywords: formal verification, compositional verification, Constraint automata, component based systems.

1 This work was supported by IPM under Grant No. CS-1383-2-01
2 Email: izadi@ipm.ir
3 Email: movaghar@sharif.ir
1 Introduction

Constraint automaton is a formalism to capture the operational semantics of Reo [2]. Reo is a channel based coordination language in which complex coordinators are compositionally built out of simpler ones [1]. In a more fundamental view, constraint automaton by itself can be used as a formalism for modeling coordination of some components. Such as any other modeling formalism, it needs ways for expressing desired properties of the actual modeled system and then verifying them. If the correctness requirements of a formally modeled computing system are given in a mathematical notion, such as linear temporal logic [10], branching time temporal logic [17] or automata on infinite objects [14], an algorithmic model theoretic process called model checking [4] 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 computer systems verification. 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 state explosion problem. The main goal of this paper is to show how theories of behavioral equivalences with a compositional state space generation help us to analyze large constraint automata models in the context of model checking temporal properties by alleviating the state space explosion.

Compositional verification is one of the main proposed methods for dealing with the problem of state explosion [4,5]. In the compositional verification of a system, one seeks to deduce properties of the system from properties of its constituent modules. An obvious strategy is to check local properties of each component of a compositional system and then present a way for deducing that a desired property is satisfied by the complete system. Because of their compositionality in their nature, component-based systems [13] and their formal specification formalisms, such as Reo or constraint automata, are very natural for applying the methods of compositional verification. An especial case of compositional verification is the method of equivalence based compositional reduction [15,16,12]. In this method components of a system are reduced with respect to an equivalence relation before building the complete system from them. If the modeling formalism saves the property of compositionality in all levels of hierarchal construction of a large scale system, this method can be applied in all levels and modules of the system. Fortunately Reo and its operational semantics, i.e. constraint automata, completely save this compositionality in all steps of the process of modeling coordinating systems.

A component-based system has two main parts: a set of components and a coordinating subsystem. By Reo specifications or constraint automata you
can specify or model the coordinating subsystem in a compositional and hier-
archal way. In other words, if a component based system is modeled by Reo
or constraint automata, both the whole system and the coordinating part of it
are compositional and hierarchal. Thus the method of compositional reason-
ning or verification can be applied both for desired properties of the complete
component system and for desired properties of the coordinating subsystem.

In this paper first we introduce a standard linear temporal logic and two
fragments of it for expressing the properties of the systems modeled by con-
straint automata and show that the equivalence relation defined by initial
stability, traces and stable failures in \([n15,n16]\) is the minimal compositional
equivalence preserving that fragment of linear time temporal logic which has
no next-time operator and has an extra operator distinguishing deadlocks. In
addition, a slight modification of this equivalence \([8]\) is the minimal equiva-
ience preserving linear time temporal logic without next-time operator. There
are reduction algorithms for reducing a constraint automaton to an equiva-
alent one which is smaller in its size and preserves temporal properties of
the modeled system with respect to the above mentioned equivalence rela-
tions. Thus in the last part of this work we use these equivalences and re-
spect reduction algorithms in the context of compositional model checking of
large scale component-based systems and their coordinating subsystems. We
present a compositional model checking algorithm based on these equivalences
for component-based systems modeled by labeled transition systems and con-
straint automata and a simplification of it for model checking the coordinating
subsystems modeled by constraint automata.

The paper proceeds as follows: in section 2 we briefly define constraint
automaton and introduce a way for modifying its definition such that the labels
of transitions be propositional formulas. In section 3 we recall some basic
concepts of process algebras and give the definitions two kinds of equivalences
based on the set of all traces or behaviors of labeled transition systems. In
section 4 we introduce a standard linear temporal logic and two fragments of it
for expressing the properties of the systems modeled by constraint automata.
This section contains a way for interpreting temporal operators over labeled
transitions instead of labeled states (labeled transition systems versus Kripke
structures). In section 5 we show that the above mentioned equivalences
preserve properties specified in the two fragments of linear temporal logic.
It can be shown that these equivalences are the weakest equivalence relations
possible which preserve temporal properties and there are reduction algorithms
for reducing constraint automata to equivalent ones with respect to these
equivalence relations. In chapter 6 we present a compositional model checking
algorithm based on these equivalences for component-based systems modeled
by labeled transition systems and constraint automata and a simplification of it for model checking the coordinating subsystems modeled by constraint automata.

2 Constraint Automata

Constraint automata were introduced by Arbab et al. in [2] as a formalism to capture the operational semantics of Reo. Timed data streams, which constitute the foundation of the coalgebraic semantics of Reo, are also the referents in the language of constraint automata. In this section we introduce the notion of constraint automata.

Let \( V \) be any set. We define the sets \( V^* \) and \( V^\omega \) as the sets of all finite and infinite sequences over \( V \) respectively. We denote individual streams as \( a = (a_0, a_1, a_2, ...) \). We call \( a_0 \) the initial value of \( a \). The (stream) derivative \( a' \) of a stream \( a \) is defined as \( a' = (a_1, a_2, ...) \). We recall the definition of timed data streams from [4]:

\[
TDS = \{ < \alpha, a > \in Data^\omega \times \mathbb{R}_+^\omega \mid \forall n \geq 0 : a_n < a_{n+1} \text{ and } \lim_{n \to \infty} a_n = \infty \}
\]

A timed data stream \( A = < \alpha, a > \) represents occurrence of events at a port \( A \) and consists of a data stream \( \alpha \in Data^\omega \) and a time stream \( a \in \mathbb{R}_+^\omega \) consisting of increasing positive real numbers. The time stream \( a \) indicates for each data item \( \alpha_n \) the moment \( a_n \) at which it occurs at a port \( A \).

Constraint automata can be viewed as acceptors for tuples of timed data streams that are observed at certain ports \( A_1, ..., A_n \). The rough idea is that such an automaton observes the data occurring at \( A_1, ..., A_n \) and either changes its state according to the observed data or rejects the data if there is no corresponding transition in the automaton. Further, constraint automata are augmented with the names of their ports \( A_1, ..., A_n \), where \( A_i \) stands for the \( i \)th TDS. Each transition in a constraint automata is labeled with a pair \( n, g \) such that \( n \) is a non-empty subset of \( N = \{ A_1, ..., A_n \} \), and a guard \( g \) that constrains data in the TDS of ports referenced in \( n \). Data constraints are defined by the following grammar:

\[
g ::= \text{false} | \text{true} | \text{data}(A) = d | g_1 \lor g_2 | g_1 \land g_2
\]

We use DC as the set of all data constraints defined by the above grammar. We recall the definition of a constraint automaton from [2] as a quadruple \( C = (Q, N, T, q_0) \) where

- \( Q \) is a finite set of states,
- \( N \) is a finite set of names,
- \( T \subseteq Q \times 2^N \times DC \times Q \) is a finite set of transitions of \( C \),
- \( q_0 \) is the initial state.
We write \( p \overset{n,g}{\rightarrow} q \) instead of \((p, n, g, q) \in T\) and call \( n \) the name set and \( g \) the guard of the transition.

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 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 \( d_1 \) and \( d_2 \) resulting in state \( p \). If there is no \( \{A_1, A_2\} \)-transition from \( q \) whose data constraint is fulfilled then \( C \) rejects.

For the simplicity of our discussion in the rest of this paper we present any constraint automata in a new way. Our purpose is to present constraint automata such that the transitions are labeled with (atomic or compound) propositions. For this purpose we can define the transition relation as: \( T \subseteq Q \times PS \times Q \) in which \( PS \) is the set of all propositions of the form \( \psi \land g \). In other words, each \( \phi \in PS \) is of the form \( \phi \equiv \psi \land g \) in which \( g \) is a data constraint as defined above and \( \psi \) is of the form \( \psi \equiv ((\pm p_1) \land (\pm p_2) \land \ldots \land (\pm p_n)) \). Each proposition states that the port \( A_i \) belongs to the set \( n \) which is a subset of \( N \). For example suppose that \( N = \{A_1, A_2, A_3\} \), the transition \((p, \{A_1, A_2\}, g, q)\) of a constraint automaton can be presented as \((p, (p_1 \land p_2 \land (\neg p_3) \land g), q)\). In the case of nondeterminism, \( \psi \) is not a full conjunctive formula and it contains only the positive clauses. We call \( PS \) as Port-Constraint Propositions.

### 3 The Equivalence Theory from Process Algebra

In this section we recall some basic concepts of process algebras and give the definitions of CFFD and NDFD-equivalences. For a more detailed discussion of these equivalences and the intuitions behind them please see [15,16,8]. Note that constraint automaton with our simplification in the last paragraph of the previous section is a particular case of the notion of lts, such that we will define bellow.

**Definition 3.1** A transition alphabet is a countable infinite set \( \Sigma \) not containing the empty transition label \( \varepsilon \). We write \( \Sigma_\varepsilon \) for \( \Sigma \cup \{\varepsilon\} \), and \( \Sigma^* (\Sigma^\omega) \) for the set of all finite (infinite) strings consisting of elements of \( \Sigma \). The symbol \( \varepsilon \) is used to denote the empty string. If \( \sigma \in (\Sigma^* \cup \Sigma^\omega) \) and \( n \geq 1 \) we write \( \sigma_n \) for the \( n \):th element of \( \sigma \) and \( \sigma^{(n)} \) for the string obtained by leaving the first \( n \) elements out of \( \sigma \). If \( \sigma, \pi \in (\Sigma^* \cup \Sigma^\omega) \), \( \sigma.\pi \) is used to denote the concatenation of \( \sigma \) and \( \pi \) and \( \sigma \prec \pi \) denote that \( \sigma \) is a prefix of \( \pi \), and \(|\sigma|\) to denote the length of \( \sigma \). If \( \sigma \in (\Sigma^*_\varepsilon \cup \Sigma^\omega_\varepsilon) \), \( \text{vis}(\sigma) \) is used to denote the
string obtained by removing all $\varepsilon$-symbols from $\sigma$ and $\Sigma(\sigma)$ denote the set of elements of $\sigma$.

**Definition 3.2** 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 $\Delta \subseteq S \times \Sigma \times S$ is the transition relation. The alphabet of $L$, $\Sigma(L)$ is the bellow set:

$$\Sigma(L) = \{L \in \Sigma \mid \exists s, s' : (s, l, s') \in \Delta\}$$

The alphabet of any LTS is required to be finite. If $\rho \in \Sigma^*$, we write $s_0 \xrightarrow{\rho} s_n$ iff there are $s_1, \ldots, s_{n-1}$ 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 $\rho \in \Sigma^\omega$, we write $s_0 \xrightarrow{\rho} \exists s_1, s_2, \ldots$ such that for all $i > 0$, $(s_i, \rho_i, s_i) \in \Delta$. If $\sigma \in (\Sigma^* \cup \Sigma^\omega)$, we write $s_0 \xrightarrow{\sigma} s_n (s_0 \xrightarrow{\sigma})$ iff there is a $\rho \in (\Sigma^* \cup \Sigma^\omega)$ such that $s_0 \xrightarrow{\rho} s_n$, $(s_0 \xrightarrow{\rho})$ and $\sigma = \text{vis}(\rho)$.

**Definition 3.3** Let $L = (S, s, \Delta)$ be a labeled transition system.

- $\sigma \in \Sigma^*$ is a trace of $L$ iff $s \xrightarrow{\rho} \cdot \text{tr}(L)$ is the set of all traces of $L$.
- $\sigma \in \Sigma^\omega$ is an infinite trace of $L$ iff $s \xrightarrow{\rho} \cdot \text{inftr}(L)$ is the set of all infinite traces of $L$.
- $\sigma \in \Sigma^*$ is a divergence trace of $L$ iff there is a $\rho \in \Sigma^\omega$ such that $s \xrightarrow{\rho}$ and $\sigma = \text{vis}(\rho)$. $\text{divtr}(L)$ is the set of all divergence traces of $L$.
- $s' \in S$ is stable, if not $s' \xrightarrow{\varepsilon}$ . $\text{Lts} L$ is stable if the initial state $s$ is stable. We write $\text{stable}(L)$ if $L$ is stable, and $\neg \text{stable}(L)$ if it is not.

- $(\sigma, A) \in \Sigma^* \times P(\Sigma)$ where $P(\Sigma)$ denotes the power set of $\Sigma$, is a failure of $L$ iff there is an $s' \in S$ such that $s \xrightarrow{\sigma} s'$ and $s' \xrightarrow{\sigma}$ for no $a \in A$.
- $(\sigma, A) \in \Sigma^* \times P(\Sigma)$ is a stable failure of $L$ iff there is a stable $s' \in S$ such that $s \xrightarrow{\sigma} s'$ and $s' \xrightarrow{\sigma}$ for no $a \in A$. $\text{sfail}(L)$ is the set of all stable failures of $L$.
- $(\sigma, A) \in \Sigma^* \times P(\Sigma)$ is a nondivergent failure of $L$ iff $(\sigma, A)$ is a failure and $\sigma$ is not a divergence trace. $\text{ndfail}(L)$ is the set of all nondivergent failures of $L$.
- $\sigma \in \Sigma^*$ is a deadlock trace of $L$ iff $(\sigma, A)$ is a stable failure of $L$. $\text{dtr}(L)$ is the set of deadlock traces of $L$.
- $\sigma \in \Sigma^*$ is a nondivergent deadlock trace of $L$ iff $(\sigma, A)$ is a nondivergent failure of $L$. $\text{nddtr}(L)$ is the set of nondivergent deadlock traces of $L$. Note that $\text{nddtr}(L) = \text{dtr}(L) - \text{divtr}(L)$.

In addition to the preceding concepts we need some notation which does not ignore the $\varepsilon$ transition labels.

**Definition 3.4** Let $L = (S, s, \Delta)$ be a labeled transition system.

- $\rho \in \Sigma^*$ is a path of $L$ iff $s \xrightarrow{\rho}$.
- \( \rho \in \Sigma^\omega \) is an infinite path of \( L \) iff \( s \xrightarrow{\rho} \). \( \text{infpath}(L) \) is the set of all infinite paths of \( L \).
- \( \rho \in \Sigma^* \) is a deadlock path of \( L \) iff there is a \( s' \in S \) such that \( s \xrightarrow{\rho} S' \) and for no \( \rho' \), \( s' \xrightarrow{\rho'} \) holds. \( \text{dpath}(L) \) is the set of all deadlock paths of \( L \).

The following proposition lists some consequences of the definitions for later use.

**Proposition 3.5** Let \( L \) be an lts.

a) \( \text{tr}(L) = \text{divtr}(L) \cup \{\sigma\vert(\sigma, \phi) \in \text{sfail}(L)\} = \text{divtr}(L) \cup \{\sigma\vert(\sigma, \phi) \in \text{ndfail}(L)\} \).

b) If \( \rho \in \text{dpath}(L) \) then \( \text{vis}(\rho) \in \text{dtr}(L) \).

c) If \( \rho \in \text{infpath}(L) \) and \( \text{vis}(\rho) \in \Sigma^\omega \) then \( \text{vis}(\rho) \in \text{inftr}(L) \).

d) If \( \rho \in \text{infpath}(L) \) and \( \text{vis}(\rho) \in \Sigma^* \) then \( \text{vis}(\rho) \in \text{divtr}(L) \).

e) If \( \rho \in \text{dpath}(L) \cup \text{infpath}(L) \) then
\[ \text{vis}(\rho) \in \text{inddtr}(L) \cup \text{divtr}(L) \cup \text{inftr}(L). \]

f) If \( \sigma \in \text{dtr}(L) \) there is a \( \rho \in \text{dpath}(L) \) such that \( \text{vis}(\rho) = \sigma \).

g) If \( \sigma \in \text{divtr}(L) \) there is a \( \rho \in \text{infpath}(L) \) such that \( \text{vis}(\rho) = \sigma \).

h) If \( \sigma \in \text{inftr}(L) \) there is a \( \rho \in \text{infpath}(L) \) such that \( \text{vis}(\rho) = \sigma \).

i) If \( \sigma \in \text{nddtr}(L) \cup \text{divtr}(L) \cup \text{inftr}(L) \) there is a
\[ \rho \in \text{dpath}(L) \cup \text{infpath}(L) \) such that \( \text{vis}(\rho) = \sigma \).

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

**Definition 3.6** Let \( L \) and \( L' \) be ltss. We say that \( L \) and \( L' \) are CFFD(NDFD) equivalent and write \( L \overset{\text{cfd}}{\approx} L'(L \overset{\text{nfd}}{\approx} L') \) iff \( \text{stable}(L) \Leftrightarrow \text{stable}(L'), \text{divtr}(L) = \text{divtr}(L'), \text{inftr}(L) = \text{inftr}(L'), \) and \( \text{sfail}(L) = \text{sfail}(L') \) \( (\text{ndfail}(L) = \text{ndfail}(L')). \)

If the labeled transition systems examined are finite, the component \( \text{inftr} \) in the definition of CFFD-equivalence is superfluous. This corresponds to the original definition of CFFD-equivalence in [15], where only finite ltss were considered.

**Proposition 3.7** Let \( L \) and \( L' \) be finite ltss. Then \( L \overset{\text{cfd}}{\approx} L'(L \overset{\text{nfd}}{\approx} L') \) iff \( \text{stable}(L) \Leftrightarrow \text{stable}(L'), \text{divtr}(L) = \text{divtr}(L'), \) and \( \text{sfail}(L) = \text{sfail}(L') \) \( (\text{ndfail}(L) = \text{ndfail}(L')). \)

The following proposition is an immediate consequence of the definitions 3.3 and 3.6 and is essential for the preservation of linear temporal logic.

**Proposition 3.8** If \( L \overset{\text{cfd}}{\approx} L'(L \overset{\text{nfd}}{\approx} L') \), then \( \text{inftr}(L) = \text{inftr}(L), \)
divtr(L) = divtr(L'), and dtr(L) = dtr(L') (nddtr(L) = nddt(L')).

Next we introduce some operators that can be used to combine labeled transition systems and state that CFFD and NDFD-equivalences are congruences with respect to these operators. The operators used are parallel composition |] and hiding and renaming.

**Definition 3.9** Let $L_1 = (S_1, s_1, \Delta_1)$ and $L_2 = (S_2, s_2, \Delta - 2)$ be ltss,

$$G = \{g_1, \ldots, g_n\} \subset \Sigma \text{ and } H = \{h_1, \ldots, h_n\} \subset \Sigma \text{ then:}$$

$L_1[[g_1, \ldots, g_n]]L_2$ (parallel composition) is the lts $(S_1 \times S_2, (s_1, s_2), \Delta)$, where

- $((t, u), g_i, (t', u')) \in \Delta$, where $g_i \in G$, iff $(t, g_i, t') \in \Delta_1$ and $(u, g_i, u') \in \Delta_2$, and
- $((t, u), g_i, (t', u')) \in \Delta$ where $l$ is not in $G$, iff either $(t, l, t') \in \Delta_1$ and $u = u'$ or $(u, l, u') \in \Delta_2$ and $t = t'$.

$L_1|L_2$ is the lts $(s \times \{0\} \cup S_1 \times \{1\} \cup S_2 \times \{2\}, (s, 0), \Delta)$, 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 $(s, l, t) \in \Delta_i$.

Hide $g_1, \ldots, g_n$ in $L_1$ is the lts $(S_1, s_1, \Delta)$ where

- $(t, l, t') \in \Delta$, iff either $l$ is not in $G$ and $(t, l, t') \in \Delta_1$ or $l = \varepsilon$ and there is a $g_i \in G$ such that $(t, g_i, t') \in \Delta_1$.

$L_1[h_1/g_1, \ldots, h_n/g_n]$ (renaming) is the lts $(S_1, s_1, \Delta)$ where

- $(t, l, t') \in \Delta$ iff either $l$ is not in $G$ and $(t, l, t') \in \Delta_1$ or $l = h_i$ and $(t, g_i, t') \in \Delta_1$.

**Definition 3.10** 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: $f(L_1, \ldots, L_n) \approx f(L_1', \ldots, L_n')$.

**Proposition 3.11** CFFD and NDFD equivalences are congruences with respect to all the operators defined in 3.9.

**Proof.** For the finite case CFFD see [15], for the general case [16,8]. \(\square\)

4. **The Linear Temporal Logic of Constraint Automata**

In this section we recall the definitions of linear models and linear temporal logic, and discuss some aspects of the relation between process algebras and temporal logic. In this section we work on constraint automata as restricted form of the general notion of labeled transition system. Thus the general results will be about labeled transition systems but some particular results will be about constraint automata.
Definition 4.1 A Linear Model is a finite or infinite sequence $\sigma = (\sigma_1, \sigma_2, \ldots)$ of sets of atomic propositions. Let the set of all atomic propositions be $AP$. We call any $\sigma_i \subseteq AP$ a state of (in) the linear model $\sigma$.

Definition 4.2 The set of all well-formed formulas (wffs) of linear temporal logic (LTL) is defined by the bellow rules:
1- If $\phi \in AP$ then $\phi$ is a wff.
2- If $\phi_1$ and $\phi_2$ are wffs, then $(-\phi_1)$, $(\phi_1 \lor \phi_2)$ and $(\phi_1 U \phi_2)$ are wffs.
3- If $\phi$ is a wff then $O\phi$ is a wff.
4- There are no other wffs.

We use the abbreviations $\top \equiv df (p \lor (\neg p))$ for some fixed proposition $p$, $(\phi_1 \land \phi_2) \equiv df ((\neg \phi_1) \lor (\neg \phi_2))$, $(F\phi) \equiv df (\top U \phi)$ and $(G\phi) \equiv df ((F(\neg \phi)))$.

Definition 4.3 The set of all well-formed formulas (wffs) of Nexttime-less linear temporal logic ($LTL_{\neq}X$) is defined by the above mentioned rules 1, 2, and 4.

Definition 4.4 The set of all well-formed formulas (wffs) of Restricted linear temporal logic ($LTL_\omega$) is defined by the above mentioned rules 1, 2, 4 and the bellow rule:
3'- If $\phi$ is a wff then $\omega F \phi$ is a wff.

Definition 4.5 A temporal formula $\phi$ of the above defined syntactic structures is true in a linear model $\sigma = (\sigma_1, \sigma_2, \ldots)$ (namely $\sigma \models \phi$) according to the following rules:
1- If $\phi \in AP$, then $\sigma \models \phi$ iff $\phi \in \sigma_1$.
2- $\sigma \models \neg \phi$ iff not $\sigma \models \phi$.
3- $\sigma \models (\phi_1 \lor \phi_2)$ iff $\sigma \models \phi_1$ or $\sigma \models \phi_2$.
4- $\sigma \models (\phi_1 U \phi_2)$ iff $\exists i : 0 \leq i < |\sigma|, \sigma^{(i)} \models \phi_2$ and $\forall j : 0 \leq j < i, \sigma^{(j)} \models \phi_1$.
5- $\sigma \models O\phi$ iff $\sigma^{(2)} \neq \emptyset$ and $\sigma^{(2)} \models \phi$.
6- $\sigma \models \omega F \phi$ iff there are infinitely many $i \geq 0$ such that $\sigma^{(i)} \models \phi$.

In LTL there is $\omega F \phi \equiv GOF\phi$. Thus $LTL_\omega$ is a restricted version of LTL. From the expressiveness power, it can be shown that $LTL_{\neq}X \subset LTL_\omega \subset LTL$.

In all infinite linear models $\omega F \phi \equiv G\phi$. Therefore, the temporal operator $\omega F$ is an operator for distinguishing a finite linear model from an infinite one, i.e. distinguishing a deadlock from a divergence. The same expressive power could be obtained by the less general operator $\omega F \top$, the future is infinite, as well.

Definition 4.6 Let $\sigma = (\sigma_1, \sigma_2, \ldots)$ be a linear model. The finitely reduced form of $\sigma (\text{fred}(\sigma))$ is constructed by collapsing all finite continuous sequences $\sigma_i, \sigma_{i+1}, \ldots, \sigma_j$ of identical elements $\sigma_i = \sigma_{i+1} = \ldots = \sigma_j$ to one element $\sigma_i$. 
The reduced form of $\sigma$ ($\text{red}(\sigma)$) 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 $\sigma_i$. If $\sigma_1$ and $\sigma_2$ be two linear models, we say that $\sigma_1$ and $\sigma_2$ are equivalent under stuttering iff $\text{red}(\sigma_1) = \text{red}(\sigma_2)$.

**Proposition 4.7** Let $\sigma = (\sigma_1, \sigma_2, \ldots)$ be a linear model. If $\phi$ is an LTL$_\omega$-formula, then $\sigma \models \phi$ iff $\text{red}(\sigma' \models \phi)$. If $\phi$ is an LTL$_X$-formula, then $\sigma \models \phi$ iff $\text{red}(\sigma' \models \phi)$.

**Proof.** It is a straightforward result of the stuttering free result of [9] based on an induction on the structure of the formula.

4.1 From states to transitions

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 defined in previous section are simplifications of Kripke structures.) On the other hand 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 present a way of interpreting the transition labels as functional state transformers: an initial state description and a sequence of transformations induce a sequence of state descriptions on which temporal logic formulas may be interpreted.

**Definition 4.8** 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.

**Definition 4.9** A temporal semantics for an lts $L$ is a mapping $f : \Sigma(L) \cup \{\varepsilon\} \rightarrow TS$ such that $f(\varepsilon) = I$. If $\rho = a_1a_2\ldots$ is a path of $L$, we write $f(\rho)$ for the sequence $(f(a_1), f(a_2), \ldots)$. In particular, A temporal semantics for constraint automaton $L$ with Port-Constraint Propositions set $PS$ ($\Sigma = PS$), is a mapping $f : PS \cup \{\varepsilon\} \rightarrow TS$ such that $f(\varepsilon) = I$. (In the case of determinism there are no $\varepsilon$-transitions. Thus a temporal semantics will be of the form $f : PS \rightarrow TS$). A temporal semantics for a path $\rho$ is a mapping $f : \Sigma(\rho) \cup \{\varepsilon\} \rightarrow TS$ such that $f(\varepsilon) = I$.

**Definition 4.10** The linear model induced by a state $\nu \subseteq AP$ and a state modifier sequence $sms$, denoted $\text{Model}(\nu, sms)$, is a sequence of states such that:

1- $\text{Model}(\nu, sms)_1 = \nu$
2- Model($\nu, sms$)$_{i+1} = sms_i(Model(\nu,sms)_i)$.

If $sms$ is finite then $|Model(\nu, sms)| = |sms| + 1$.

**Definition 4.11** Let $\sigma \in (\sigma^* \cup \Sigma^\omega_{\epsilon})$ be a path of lts $L$, $f$ a temporal semantics for $\sigma$, $\nu_0$ a state and $\phi$ an LTL formula. We say $\phi$ is true of $\sigma$ with respect to temporal semantics $f$ and initial state $\nu_0$ and write $\sigma, f, \nu_0 \vDash \phi$ iff $Model(\nu_0, f(\sigma)) \vDash \phi$. (If $L$ is a deterministic constraint automaton, $\sigma \in (PS^* \cup PS^\omega)$ is a path of it).

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 4.12** Let $L$ be an lts (in particular a constraint automaton), $f$ a temporal semantics for $L$, $\nu_0$ a state and $\phi$ an LTL formula. We say $\phi$ is true of $L$ with respect to temporal semantics $f$ and initial state $\nu_0$ and write $L, f, \nu_0 \vDash \phi$ if $\phi$ for all $\sigma \in dpath(L) \cup infpath(L)$.

Now a module of a coordinating system can be modeled by a constraint automata and a temporal interpretation expressing the changes in the state information of that module caused by the transition. These modules can then be combined to larger units of coordination system by syntactic operators such as parallel composition, hiding and renaming.

## 5 Property Preservation, Minimality and Reduction

In this section we show that CFFD and NDFD-equivalences preserve properties specified in and respectively. In [15] it was shown that a CFFD is the minimal equivalence relation in which some temporal logic properties are preserved. With a straightforward and highly similar proof it can be shown that NDFD is the minimal preserving equivalence relation for $LT_{\omega}$ temporal logic. Also in [15,16] a reduction algorithm for CFFD-equivalence was presented. By such reduction algorithm, we can reduce the size of an lts or in particular an constraint automata such that those properties of the modeled system which can be expressed by $LT_{\omega}$ temporal logic formulas are preserved. Thus the process of verification or model checking can be simplified. A modification on the above mentioned reduction algorithm can be applied for NDFD-equivalence relation (see [8]).

**Definition 5.1** Let $L_1$ and $L_2$ be ltss and $\phi$ an LTL-formula. We say that $L_1$ and $L_2$ agree on $\phi$ iff for every temporal semantics $f$ and for every initial state $\nu_0$ it is the case that $L_1, f, \nu_0 \vDash \phi$ iff $L_2, f, \nu_0 \vDash \phi$. 
Definition 5.2 An equivalence \( \approx \) between ltss is \( \text{LTL-preserving} \) iff for any \( L_1, L_2 \) such that \( L_1 \approx L_2 \), \( L_1 \) and \( L_2 \) agree on every LTL formula. Similarly, an equivalence \( \approx \) between ltss is \( \text{LTL}_\omega\text{-preserving} \) iff for any \( L_1, L_2 \) such that \( L_1 \approx L_2 \), \( L_1 \) and \( L_2 \) agree on every \( \text{LTL}_\omega \) formula.

Now we are in the situation in which we can prove that CFFD and NDFD-equivalences are \( \text{LTL}_\omega \)-preserving and \( \text{LTL}_X \)-preserving respectively.

Proposition 5.3 Let \( L \) and \( L' \) be ltss and \( \inftr(L) = \inftr(L') \), \( \divtr(L) = \divtr(L') \) and \( \dtr(L) = \dtr(L') \). Then \( L \) and \( L' \) agree on every \( \text{LTL}_\omega \)-formula.

Proof. Let \( \phi \) be an \( \text{LTL}_\omega \)-formula and \( f, \nu_0 \) arbitrary temporal semantics and initial set respectively. Now,

\[
L, f, \nu_0 \models \phi \iff \rho, f, \nu_0 \models \phi \text{ for all } \rho \in \text{dpath}(L) \cup \text{infpath}(L)
\]

iff \( \text{vis}(\rho), f, \nu_0 \models \phi \) for all \( \rho \in \text{dpath}(L) \) and for all \( \text{infpath}(L) \) such that \( \text{vis}(\rho) \in \Sigma^\omega \) and \( \text{vis}(\rho).\varepsilon^\omega, f, \nu_0 \models \phi \) for all \( \rho \in \text{infpath}(L) \) such that \( \text{vis}(\rho) \in \Sigma^* \)

iff \( \sigma, f, \nu_0 \models \phi \) for all \( \sigma \in \text{dtr}(L) \) and for all \( \sigma \in \text{inftr}(L) \) and \( \sigma.\varepsilon^\omega, f, \nu_0 \models \phi \) for all \( \sigma \in \text{divtr}(L) \) (see 3.5)

iff \( \sigma, f, \nu_0 \models \phi \) for all \( \sigma \in \text{dtr}(L') \) and for all \( \sigma \in \text{inftr}(L') \) and \( \sigma.\varepsilon^\omega, f, \nu_0 \models \phi \) for all \( \sigma \in \text{divtr}(L') \) (by assumption) iff \( \sigma, f, \nu_0 \models \phi \). \( \square \)

Proposition 5.4 CFFD-equivalence is \( \text{LTL}_\omega \)-preserving.

Proof. This proposition is a direct consequence of 3.8 and 5.3. \( \square \)

Proposition 5.5 Let \( L \) and \( L' \) be ltss and \( \inftr(L) = \inftr(L') \), \( \divtr(L) = \divtr(L') \) and \( \text{nddtr}(L) = \text{nddtr}(L') \). Then \( L \) and \( L' \) agree on every \( \text{LTL}_X \)-formula.

Proof. is highly similar to the proof of proposition 5.3 (see [n16,n8]). \( \square \)

Proposition 5.6 NDFD-equivalence is \( \text{LTL}_X \)-preserving.

Proof. This proposition is a direct consequence of 3.8 and 5.5. \( \square \)

6 Compositional Verification of Component-Based Systems

With the rapid growth of the power of computing systems, from both hardware and software points of view, the demand of large and complex computing systems has increased dramatically. The concept of component-based systems especially component-based software is a new 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. Thus coordination is a way for composing components and building large scale computing systems. Reo is a channel based coordination language in which complex coordinators are compositionally built out of simpler ones [1]. Constraint automaton is a formalism to capture the operational semantics of Reo [2]. Thus in general constraint automaton is a fundamental modeling formalism for coordination. In this section we present a method for compositional model checking of a component-based system and its coordinating subsystem by using the above mentioned equivalences for minimizing formal models.

A component-based system has two main parts: a set of components and a coordinating subsystem. By Reo specifications or constraint automata you can specify or model the coordinating subsystems in a compositional and hierarchical way. In other words, if the coordinating subsystem of a component-based system is modeled by Reo or constraint automaton, both the whole system and the coordinating part of it are compositional and hierarchical. Thus the methods of compositional reasoning can be applied both for desired properties of the complete component-based system and for desired properties of the coordinating subsystem. Fortunately, our above process algebraic discussions enable us to use equivalence based compositional reduction method in both cases:

Verification of Coordinating Subsystem

In this case we want to verify desired properties of the coordinating subsystem of a component-based system. If we consider the coordinating subsystem (for example a Reo circuit or a compositional constraint automata) as a complete system, the set of the components of the component-based system is the environment of it. 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 work on its boundary nodes.) Rest of the actions within the coordinating subsystem, and its internal states are not interesting if only the correct functionality of coordinating subsystem, that is correct coordinating, is concerned. The main steps of model checking of desired properties of coordinating subsystem will be:

1- Expressing the desired property by an $LTL_X$ or $LTL_\omega$ formula.

2- Modeling the coordination subsystem by a compositional constraint automaton.
3- According to the type of the property which we want to verify, using an equivalence relation for minimizing the size of the constraint automaton.

4- Using one of ordinary LTL model checking algorithms on the minimized model.

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

**Verification of Coordinating Subsystem**

In this case we want to verify desired properties of the whole component-based system. Fortunately, we can simply model any component by a labeled transition system (lts) such that we defined in section 3 and the coordinating system by a compositional constraint automaton. The equivalence relations defined in section 3 work both for lts in general and constraint automata. Thus the main steps of model checking of desired properties of a complete component based system will be:

1- Expressing the desired property by an $LTL_\omega$ or $LTL_\omega X$ formula.

2- Modeling every component by a labeled transition system.

3- According to the type of the desired property formula, using an equivalence relation for minimizing the size of all lts models.

4- Modeling the coordination subsystem by a constraint automaton.

5- According to the property which we want to verify, using an equivalence relation for minimizing the size of constraint automaton model of coordinating subsystem.

6- Combining the minimized lts and the constraint automata by using composition operator and minimizing it.

7- Using standard LTL model checking algorithm for the minimized model.

Note that there are some other compositional reasoning methods, such as assumption-guarantee method [12], in which the reasoning is done separately on the component of the model by decomposing the desired property formula. we can consider using such techniques of compositional reasoning jointed to our minimization method. If we consider such techniques, the above 6 and 7 steps should be replaced by proper steps based on the selected algorithm of verification.
7 Conclusions

In this paper we introduced a standard linear temporal logic and two fragments of it for expressing the properties of the systems modeled by constraint automata and show that the equivalence relation defined by initial stability, traces and stable failures in [15,16] is the minimal compositional equivalence preserving that fragment of linear time temporal logic which has no next-time operator and has an extra operator distinguishing deadlocks. In addition, a slight modification of this equivalence is the minimal equivalence preserving linear time temporal logic without next-time operator. There are reduction algorithms for reducing a constraint automaton to an equivalent one which is smaller in its size and preserves temporal properties of the modeled system with respect to the above mentioned equivalence relations. Thus we used these equivalences and respect reduction algorithms in the context of compositional verification and model checking of large scale component based systems and their coordinating subsystems. We presented a compositional model checking algorithm based on these equivalences for component based systems modeled by labeled transition systems and constraint automata and a simplification of it for the coordinating subsystems modeled by constraint automata.

In comparison with other techniques for dealing with state explosion problem such as the partial order reduction by representatives [11], the preorder reduction [7], abstraction [3] and symmetry [6], the main advantages of our method are:

1- Its ability in joining with other above called techniques for dealing with state explosion problem.

2- Because of the minimizations, the efficiency of our method is better than applying algorithms of LTL model checking directly. However, any improvement in the ordinary algorithms of LTL model checking or any improvement in the other techniques for dealing with state explosion problem jointed to our method, improves the efficiency of our method.

References


