Layered Reduction for Modal Specification Theories*

Arpit Sharma and Joost-Pieter Katoen

Software Modeling and Verification Group
RWTH Aachen University, Germany
{arpit.sharma, katoen}@cs.rwth-aachen.de

Abstract. Modal transition systems (MTSs) are a well-known formalism used as an abstraction theory for labeled transition systems (LTSs). MTS specifications support compositionality together with a step-wise refinement methodology, and thus are useful for component-oriented design and analysis of distributed systems. This paper proposes a state-space reduction technique for such systems that are modeled as a network of acyclic MTSs. Our technique is based on the notion of layered transformation. We propose a layered composition operator for acyclic MTSs, and prove the communication closed layer (CCL) laws. Next, we define a partial order (po) equivalence between acyclic MTSs, and show that it enables performing layered transformation within the framework of CCL laws. We also show the preservation of existential ($\exists$) and universal ($\forall$) reachability properties under this transformation.

Keywords: modal transition system, layering, distributed system, existential reachability, universal reachability, CCL laws, partial order equivalence, refinement.

1 Introduction

Modal transition systems (MTSs) [20, 17] are labeled transition systems (LTSs) [23, 1] equipped with two types of transitions: *may* transitions that any implementation (LTS) may (or may not) have and *must* transitions that any implementation must have. An LTS is an MTS where all the transitions are must transitions. MTSs were originally introduced by Larsen and Thomsen almost 25 years ago [20, 17], and have been successfully applied in program analysis [10, 29], model checking [19, 5], equation solving [21], interface theories [28, 31], component-based software development [27] and software product lines [18, 9]. The theory of MTS is equipped with parallel and conjuction operators, and allows comparing two MTSs using a refinement relation. A satisfaction relation is used to check whether an LTS is an implementation of a given MTS. MTS specifications are useful for component-oriented design and analysis of distributed systems. In this setting, a high-level model of the system which abstracts from the implementation details is constructed and used for the verification of interesting properties. A correct implementation can be obtained by applying a series of refinement steps. Model construction involves composing several components in parallel, where each component

* This work is supported by the European Commission SENSATION project.

usually has multiple sub-components that are executed in a sequential manner. Components cooperate through their synchronization over common actions and through their respective action dependencies. Action dependencies between sub-components can be either explicitly stated or derived from the operations performed on data variables that are updated during an action execution (in case of MTS with data [2]). Some example systems that have this structure are distributed algorithms such as the distributed minimum weight spanning tree algorithm [7], the two phase commit protocol [4], Fischer's real-time mutual exclusion protocol, and the randomized mutual exclusion algorithm by Kushilevitz and Rabin [15]. Composing several components using parallel composition naturally leads to the problem of *state-space explosion* [1], where the number of states grows exponentially in the number of parallel components.

In [30], a layered analysis of Kushilevitz and Rabin's randomized mutual exclusion algorithm [15] has been carried out. The underlying model on which layered transformations have been applied is a probabilistic automaton. We modeled and analysed this case study using the PRISM model checker [16]. The obtained results for 3 processes and 5 rounds are summarised in the Table 1. These results clearly indicate that layered reasoning can significantly reduce the state space of system models capturing the behavior of distributed algorithms. In addition, layering has been successfully applied to obtain easier correctness proofs for various distributed systems [11, 13, 14, 12]. Mo-

	Parallel	Layered
Build time (s)	898.70	90.39
# States	198063	71619
# Transitions	351432	128920

Table 1. Parallel vs. layered composition

tivated by this, we propose a state-space reduction technique for a network of acyclic MTSs based on the notion of layering. The main principle is illustrated in Fig. 1. Here two MTS components $\mathcal{M}$ and $\mathcal{N}$ are composed in parallel (left), where each component consists of n sub-components which are executed in a sequential manner. The system obtained after performing layered transformation is shown in Fig. 1 (right). Note that all the sub-components of $\mathcal{M}$ and $\mathcal{N}$ need to be acyclic, but we do allow outermost level of recursion in $\mathcal{M}$ and $\mathcal{N}$. In other words, every component can have multiple rounds of execution, where a new round is started only when the last sub-component of the previous round has been executed. This is important as deadlock states are usually considered to be undesirable for MTSs modeling distributed algorithms. Roughly speaking, layering exploits the independence between sub-components to transform the system under consideration from a distributed representation to a layered representation. A layered composition operator "$\bullet$" is used to denote the layered representation of the system. Informally, $\mathcal{M}_1 \bullet \mathcal{M}_2$ allows synchronization on common actions and interleaving on disjoint actions, except when some action a of $\mathcal{M}_2$ depends on one or more actions of $\mathcal{M}_1$; in this case, a can be executed only after all the actions of $\mathcal{M}_1$

on which it depends have been executed. This new composition operator allows formulating *Communication Closed Layer* (CCL) laws [11], which are required to carry out the structural transformations and establish an equivalence between the two systems. Since the sub-components within a component are executed sequentially, a partial order relation is proposed to relate the $\bullet$ and ; (sequential) operator. The reduced system obtained as a result of applying layered transformation can be used for analysis, provided it preserves a rich class of properties of interest. Reachability is one of the most important properties in the area of model checking. Therefore, we focus on proving that the reduced system preserves existential ($\exists$) and universal ($\forall$) reachability properties to reach its set of final states.

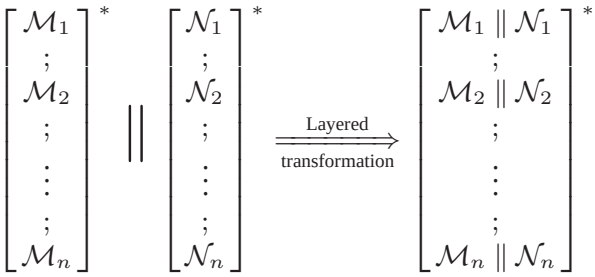

Fig. 1. Layered reduction

Contributions. The main contributions of this paper are as follows:

- We define the notions of *abstract execution* and *realisation*, which are subsequently used to compare the behaviour of acyclic MTSs.
- We define the layered $(\bullet)$ and sequential $(;)$ composition operator, and formulate communication closed layer (CCL) laws for acyclic MTSs.
- We define the partial order (po) equivalence between acyclic MTSs, show that $\bullet$ is po-equivalent to ;, and prove that po-equivalence between $\bullet$ and ; preserves existential ($\exists$) and universal ($\forall$) reachability properties.
- Finally, we show how state space reduction can be achieved by replacing $\bullet$ with ; within the framework of CCL laws.

Related work

Layering The decomposition of a distributed program into communication closed layers to simplify its analysis was originally proposed in [6]. In [13], a layered composition operator and various algebraic transformation rules have been introduced to simplify the analysis of distributed database systems. Some other examples where layering techniques have been applied for the analysis of distributed systems can be found in [14, 11, 12]. An extension of layering operator and CCL laws to the real-time setting has been proposed in [12]. Layered composition for timed automata has been investigated in [25]. In the probabilistic context, layering has been applied to the consensus problem

to prove the lower bounds [24]. A probabilistic Kleene Algebra (pKA), for simplifying the reasoning of randomized distributed algorithms has been recently proposed in [22]. Most recently, the layered composition operator and probabilistic counterparts of the CCL laws have been defined for the PA model [30]. The feasibility of this approach has been demonstrated on a randomized mutual exclusion algorithm.

Organisation of the paper. Section 2 briefly recalls the basic concepts of LTSs and MTSs. Section 3 presents the satisfaction and refinement relations for MTSs. Section 4 discusses the composition operators for MTSs, and introduces CCL laws. Section 5 defines po-equivalence between MTSs, and proves that po-equivalence between $\bullet$ and ; preserves existential ($\exists$) and universal ($\forall$) reachability properties. Finally, Section 6 concludes the paper. All the proofs are contained in the appendix.

2 Preliminaries

This section recalls the basic concepts of labeled transition systems and modal transition systems with a finite state space.

Definition 1. *A labeled transition system (LTS) is a tuple $\mathcal{T} = (S, Act, s_0, S_f, V)$ where:*

- *S is a finite, non-empty set of states,*
- *Act is a finite set of actions,*
- *$s_0 \in S$ is the initial state,*
- *$S_f \subset S$ is the set of final states where $s_0 \notin S_f$,*
- *$V : S \setminus S_f \times Act \times S \to \mathbb{B}_2$ is a two-valued transition function.*

Here $\mathbb{B}_2 = \{\bot, \top\}$, with $\bot < \top$. $V(s, a, s')$ identifies the transition of the transition system: $\top$ indicates its presence and $\bot$ indicates its absence. We write $s \xrightarrow{a} s'$ whenever $V(s, a, s') = \top$. Labeled transition systems are basically directed graphs where nodes represent states, and edges model transitions, i.e., state changes. Transitions specify how the system can evolve from one state to another. In case a state has more than one outgoing transition, the next transition is chosen in a purely non-deterministic fashion. A possible behaviour in an LTS is obtained from the resolution of non-deterministic choices, described in terms of paths. A path π of LTS $\mathcal{T}$ is a (possibly infinite) sequence of the form $\pi = s_0 a_1 s_1 a_2 s_2 a_3 \ldots$ where $\forall n : s_n \xrightarrow{a_{n+1}} s_{n+1}$. Let $last(\pi)$ denote the last state of π (if π is finite). Let $|\pi|$ be the length (number of actions) of a finite path π. For infinite path π and any $i \in \mathbb{N}$, let $\pi[i] = s_i$, the $(i+1)$-st state of π. For finite path π of length n, $\pi[i]$ is only defined for $i \leq n$ and defined as for infinite paths. Let $Paths_{fin}(\mathcal{T})$ be the set of all finite paths in LTS $\mathcal{T}$, and $Paths_{inf}(\mathcal{T})$ the set of all infinite paths of $\mathcal{T}$ that start in state s_0. Let $Paths^{S_f}_{fin}(\mathcal{T})$ be the set of all finite paths of $\mathcal{T}$ that start in state s_0 and end in some state $s \in S_f$.

Definition 2. *LTS $\mathcal{T} = (S, Act, s_0, S_f, V)$ is deterministic, if $\forall s \in S. \forall a \in Act. |\{s' \in S \mid V(s, a, s') \neq \bot\}| \leq 1$*

Example 1. Consider the LTS $\mathcal{T}$ in Fig. 2 (right), where $S = \{s_0, s_1, s_2, s_f\}$, $Act = \{a, b, c, d\}$, s_0 is the initial state, and $S_f = \{s_f\}$. It is easy to check that $\mathcal{T}$ is deterministic. An example finite path π is $s_0 a s_1 c s_2 b s_f$. We have $|\pi| = 3$, and $\pi[1] = s_1$.

Definition 3. *A modal transition system (MTS) is a tuple $\mathcal{M} = (S, Act, s_0, S_f, V)$ where:*

- *S is a finite, non-empty set of states,*
- *Act is a finite set of actions,*
- *$s_0 \in S$ is the initial state,*
- *$S_f \subset S$ is the set of final states where $s_0 \notin S_f$,*
- *$V : S \setminus S_f \times Act \times S \to \mathbb{B}_3$ is a three-valued transition function.*

Here $\mathbb{B}_3 = \{\bot, ?, \top\}$ denotes a complete lattice with the following ordering $\bot < ? < \top$ and meet $\sqcap$ and join $\sqcup$ operators. $V(s, a, s')$ identifies the transition of the automaton: $\top$, $?$ and $\bot$ indicate a *must*, a *may* and absence of transition respectively. For simplicity we write $s \xrightarrow{a}_{\top} s'$ instead of $V(s, a, s') = \top$. Similarly, we write $s \xrightarrow{a}_{?} s'$ instead of $V(s, a, s') =?$. Let $act(s)$ denote the set of enabled actions from state s, i.e., $act(s) = \{a \in Act \mid \exists s' : V(s, a, s') \neq \bot\}$. By definition, it follows $\forall s \in S_f : act(s) = \varnothing$.

Remark 1. An LTS is an MTS in which every transition is a must-transition. Thus, every LTS is an MTS.

Definition 4. *MTS $\mathcal{M} = (S, Act, s_0, S_f, V)$ is deterministic, if* $\forall s \in S. \forall a \in Act. |\{s' \in S \mid V(s, a, s') \neq \bot\}| \leq 1$.

In this paper we only consider deterministic MTSs, as they are sufficient for modeling the behavior of typical distributed algorithms. An abstract execution ρ of an MTS $\mathcal{M}$ is a (possibly infinite) sequence of the form $\rho = s_0 a_1 s_1 a_2 s_2 a_3 \ldots$, where $\forall n : s_n \xrightarrow{a_{n+1}}_{\top} s_{n+1}$ or $s_n \xrightarrow{a_{n+1}}_{?} s_{n+1}$. Let $Exec_{fin}(\mathcal{M})$ be the set of all finite abstract executions, and $Exec_{inf}(\mathcal{M})$ the set of all infinite abstract executions of $\mathcal{M}$ that start in state s_0. Let $Exec_{fin}^{S_f}(\mathcal{M})$ be the set of all finite abstract executions of $\mathcal{M}$ that start in state s_0, and end in some state $s \in S_f$. Let $|\rho|$ be the length (number of actions) of a finite abstract execution ρ. For infinite abstract execution ρ and any $i \in \mathbb{N}$, let $\rho[i] = s_i$, the $(i+1)$-st state of ρ. For finite abstract execution ρ of length n, $\rho[i]$ is only defined for $i \leq n$ and defined as for infinite abstract executions. Let $last(\rho)$ denote the last state of ρ (if ρ is finite). Similarly, let $first(\rho)$ denote the first state of ρ.

Example 2. Consider the MTS $\mathcal{M}$ in Fig. 2 (left), where $S = \{s_0, s_1, s_2, s_f\}$, $Act = \{a, b, c, d\}$, s_0 is the initial state, and $S_f = \{s_f\}$. Here, state s_2 has two outgoing transitions: a must b-transition moving to s_f and a may c-transition moving to s_2. Similarly, state s_1 has two outgoing transitions: a must c-transition moving to s_2 and a may b-transition moving to s_1. Note that $\mathcal{M}$ is deterministic. An example finite abstract execution ρ is $s_0 a s_1 c s_2 b s_f$ with $|\rho| = 3$, and $\rho[1] = s_1$.

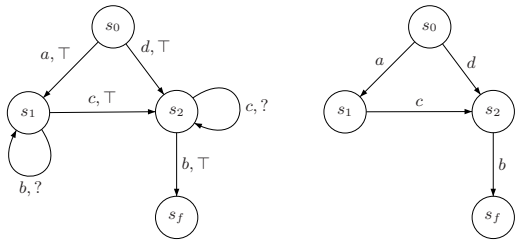

Fig. 2. An MTS $\mathcal{M}$ (left) and an LTS $\mathcal{T}$ (right) that satisfies $\mathcal{M}$

3 Satisfaction and Refinement

This section presents the notions of *satisfaction* and *refinement* originally introduced in [20]. A satisfaction relation allows to relate an LTS (implementation) with an MTS (specification). A refinement relation is used to compare MTSs w.r.t. their sets of implementations. We also define the notions of realisation, existential ($\exists$) and universal ($\forall$) reachability properties of reaching the set of final states computed over the implementations of MTS $\mathcal{M}$.

Definition 5 (Satisfaction). *Let $\mathcal{T} = (S, Act, s_0, S_f, V)$ be an LTS and $\mathcal{M} = (S', Act, s'_0, S'_f, V')$ be an MTS. $R \subseteq S \times S'$ is a* satisfaction relation *iff, for any $(s, s') \in R$, the following conditions hold:*

- $\forall a \in Act, \forall u' \in S' : V'(s', a, u') = \top \Rightarrow (\exists u \in S : V(s, a, u) = \top \wedge uRu')$,
- $\forall a \in Act, \forall u \in S : V(s, a, u) = \top \Rightarrow (\exists u' \in S' : V'(s', a, u') \neq \bot \wedge uRu')$,
- $s \in S_f \Leftrightarrow s' \in S'_f$.

We say that $\mathcal{T}$ satisfies $\mathcal{M}$, denoted $\mathcal{T} \models \mathcal{M}$, iff there exists a satisfaction relation relating s_0 and s'_0. If $\mathcal{T} \models \mathcal{M}$, $\mathcal{T}$ is called an implementation *of $\mathcal{M}$.*

Intuitively, a state s satisfies state s' iff any must transition of s' is matched by a transition of s, s does not contain any transitions that do not have a corresponding transition (may or must) in s', and the final states of two systems are always related. Let $[\![\mathcal{M}]\!] = \{\mathcal{T} \mid \mathcal{T} \models \mathcal{M}\}$, i.e., $[\![\mathcal{M}]\!]$ is the set of all implementations of MTS $\mathcal{M}$. Let $\mathcal{T} \in [\![\mathcal{M}]\!]$, and $\pi = s_0 a_1 s_1 a_2 s_2 \dots s_n$ be a finite path of $\mathcal{T}$, i.e., $\pi \in Paths_{fin}(\mathcal{T})$. π is said to be a realisation of $\rho = s_0 a'_1 s_1 a'_2 s_2 \dots s_n$ where $\rho \in Exec_{fin}(\mathcal{M})$, denoted $\pi \models \rho$, if $\forall i < n : a_{i+1} = a'_{i+1}$.

Example 3. The LTS $\mathcal{T}$ in Fig. 2 (right) is an implementation of the MTS $\mathcal{M}$ in Fig. 2 (left). It is easy to check that there exists a satisfaction relation relating the initial states of $\mathcal{T}$ and $\mathcal{M}$. Note that in this example, for every implementation $\mathcal{T}$ of $\mathcal{M}$, $S_f \neq \varnothing$ (since there exists a finite abstract execution from s_0 to s_f with only must transitions). Finite path $\pi = s_0 a s_1 c s_2 b s_f$ of LTS $\mathcal{T}$ is a realisation of finite abstract execution $\rho = s_0 a s_1 c s_2 b s_f$ of $\mathcal{M}$.

Note that for a deterministic MTS $\mathcal{M}$ and $\mathcal{T} \models \mathcal{M}$, if a path $\pi \in Paths_{fin}(\mathcal{T})$ is a realisation of some finite abstract execution $\rho \in Exec_{fin}(\mathcal{M})$, then it cannot be a realisation of another finite abstract execution of $\mathcal{M}$.

Definition 6 (Refinement). *Let $\mathcal{M} = (S, Act, s_0, S_f, V)$ and $\mathcal{M}' = (S', Act, s'_0, S'_f, V')$ be MTSs. $R \subseteq S \times S'$ is a* strong refinement relation *iff, for all $(s, s') \in R$, the following conditions hold:*

- $\forall a \in Act, \forall u' \in S' : V'(s', a, u') = \top \Rightarrow (\exists u \in S : V(s, a, u) = \top \wedge uRu')$,
- $\forall a \in Act, \forall u \in S : V(s, a, u) \neq \bot \Rightarrow (\exists u' \in S' : V'(s', a, u') \neq \bot \wedge uRu')$,
- $s \in S_f \Leftrightarrow s' \in S'_f$.

$\mathcal{M}$ strongly refines $\mathcal{M}'$, denoted $\mathcal{M} \preceq_S \mathcal{M}'$, iff there exists a strong refinement relation relating s_0 and s'_0.

Intuitively, a state s strongly refines state s' iff any must transition of s' is matched by a must transition of s, s does not contain any transitions (may or must) that do not have a corresponding transition (may or must) in s', and the final states of two systems are always related.

Remark 2. A satisfaction relation is a special type of refinement relation. In simple words, if $\mathcal{T}$ satisfies $\mathcal{M}$, then $\mathcal{T}$ also strongly refines $\mathcal{M}$ (since every LTS is an MTS and all the three conditions of refinement are satisfied).

Definition 7 (Refinement equivalence). *We say that $\mathcal{M}$ and $\mathcal{M}'$ are refinement equivalent, denoted $\mathcal{M} \equiv \mathcal{M}'$, iff $\mathcal{M} \preceq_S \mathcal{M}'$ and $\mathcal{M}' \preceq_S \mathcal{M}$.*

Since strong refinement implies inclusion of sets of implementations, it follows that refinement equivalent MTSs $\mathcal{M}$ and $\mathcal{M}'$ have the same set of implementations, i.e., $[\![\mathcal{M}]\!] = [\![\mathcal{M}']\!]$.

Assumptions. For the rest of the paper we assume the following:

- Every MTS is acyclic.
- Every MTS has a single final state, i.e., $|S_f| = 1$, and all its states (except the final state) have at least one outgoing transition.
- Dependencies between actions of different components are known in advance.

In this paper we focus on reachability properties, i.e., is it possible to reach the set of final states from the initial state in an LTS $\mathcal{T}$. More formally it is defined as follows:

Definition 8 (LTS reachability). *Let $\mathcal{T} = (S, Act, s_0, S_f, V)$ be an LTS. Then $\mathcal{T}$ reaches S_f, denoted $\mathcal{T} \models \Diamond S_f$, iff $\forall \pi \in Paths_{fin}(\mathcal{T}) \exists \pi' \in Paths^{S_f}_{fin}(\mathcal{T}) : \pi$ is a prefix of π'.*

In simple words, all the finite paths starting from the initial state of $\mathcal{T}$ should be extendable s.t. the last state of the new path obtained belongs to S_f.

Example 4. Consider the LTS $\mathcal{T}$ in Fig. 2 (right) where s_0 is the initial state, and s_f is the only final state. Here, $\mathcal{T} \models \Diamond S_f$ since every finite path of $\mathcal{T}$ can be extended s.t. it reaches s_f.

Next, we define two reachability properties of reaching the set of final states determined over the implementations of an MTS $\mathcal{M}$. The first property requires that for an MTS $\mathcal{M}$ there exists at least one implementation $\mathcal{T}$ s.t. $\mathcal{T} \models \Diamond S_f$. The second property requires that all the implementations of $\mathcal{M}$ should be able to reach the set of final states. Formally, these properties are defined as follows:

Definition 9 (Existential reachability). *Let $\mathcal{M} = (S, Act, s_0, S_f, V)$ be an MTS. Then $\mathcal{M}$* possibly *reaches S_f, denoted $\mathcal{M} \models^{\exists} \Diamond S_f$, iff $\exists \mathcal{T} \in [\![\mathcal{M}]\!] : \mathcal{T} \models \Diamond S_f$.*

Definition 10 (Universal reachability). *Let $\mathcal{M} = (S, Act, s_0, S_f, V)$ be an MTS. Then $\mathcal{M}$* inevitably *reaches S_f, denoted $\mathcal{M} \models^{\forall} \Diamond S_f$, iff $\forall \mathcal{T} \in [\![\mathcal{M}]\!] : \mathcal{T} \models \Diamond S_f$.*

Remark 3. The problem of deciding $\mathcal{M} \models^{\exists} \Diamond S_f$ is PSPACE-complete [3]. The same applies to universal reachability.

4 Composition and CCL Laws

In this section we define composition operators for MTSs. We propose sequential, and layered composition operators, and recall parallel composition from [20]. The framework of CCL laws is also formulated, which is required for carrying out the layered transformations.

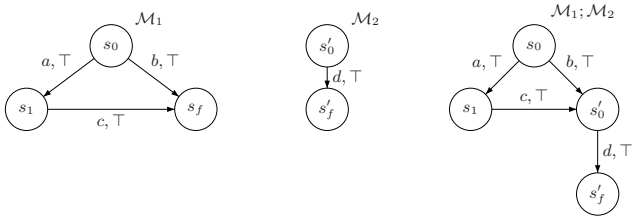

Fig. 3. MTSs $\mathcal{M}_1$ and $\mathcal{M}_2$ (left) and their sequential composition (right)

Definition 11 (Sequential composition). *Given MTSs $\mathcal{M}_i = (S_i, Act_i, s_{0i}, \{s_{fi}\}, V_i)$, where $i \in \{1, 2\}$ with $S_1 \cap S_2 = \varnothing$. The* sequential composition *of $\mathcal{M}_1$ and $\mathcal{M}_2$, denoted $\mathcal{M}_1; \mathcal{M}_2$, is the MTS $(S, Act_1 \cup Act_2, s_{01}, \{s_{f2}\}, V)$, where $S = S_1 \setminus \{s_{f1}\} \cup S_2$ and $V = V'_1 \cup V_2$. Here $V'_1 = V_1[s_{02} \leftarrow s_{f1}]$ is defined by*

$$V'_1(s, a, s') = V_1(s, a, s') \text{ if } s' \neq s_{f1}, \text{ and}$$
$$V'_1(s, a, s_{02}) = V_1(s, a, s_{f1}) \text{ otherwise.}$$

Intuitively, sequential composition of two MTSs $\mathcal{M}_1$ and $\mathcal{M}_2$ requires executing the actions of $\mathcal{M}_1$ followed by actions of $\mathcal{M}_2$. Note that all the incoming transitions to state s_{f1} are redirected to s_{02}. Here, s_{01}, s_{f2} are the new initial and final states in the resulting MTS, respectively.

Example 5. The sequential composition of two MTSs $\mathcal{M}_1, \mathcal{M}_2$ (Fig. 3 (left)) is shown in Fig. 3 (right).

Definition 12 (Parallel composition). *Given MTSs $\mathcal{M}_i = (S_i, Act_i, s_{0i}, \{s_{fi}\}, V_i)$, where $i \in \{1, 2\}$ with $S_1 \cap S_2 = \varnothing$. The* parallel composition *of $\mathcal{M}_1$ and $\mathcal{M}_2$, denoted $\mathcal{M}_1 || \mathcal{M}_2$, is the MTS $(S_1 \times S_2, Act_1 \cup Act_2, (s_{01}, s_{02}), \{(s_{f1}, s_{f2})\}, V)$ where V is defined by:*

– *For all* $(s, s') \in S_1 \times S_2$, $a \in Act_1 \cap Act_2$, *if there exists* $u \in S_1$ *and* $u' \in S_2$, *such that* $V_1(s, a, u) \neq \bot$ *and* $V_2(s', a, u') \neq \bot$, *define* $V((s, s'), a, (u, u')) = V_1(s, a, u) \sqcap V_2(s', a, u')$. *If either* $\forall u \in S_1$, *we have* $V_1(s, a, u) = \bot$, *or* $\forall u' \in S_2$, *we have* $V_2(s', a, u') = \bot$ *then* $\forall (u, u') \in S_1 \times S_2$, $V((s, s'), a, (u, u')) = \bot$.
– *For all* $(s, s'), (u, u') \in S_1 \times S_2$, $a \in Act_1 \setminus Act_2$, *define* $V((s, s'), a, (u, u')) = V_1(s, a, u)$ *if* $s' = u'$, $V((s, s'), a, (u, u')) = \bot$ *otherwise.*
– *For all* $(s, s'), (u, u') \in S_1 \times S_2$, $a \in Act_2 \setminus Act_1$, *define* $V((s, s'), a, (u, u')) = V_2(s', a, u')$ *if* $s = u$, $V((s, s'), a, (u, u')) = \bot$ *otherwise.*

Parallel composition forces synchronization on common actions and interleaving on disjoint actions. Note that the synchronization of two must transitions results in a must transition, and composing may-must, must-may and may-may transitions results in a may transition.

Example 6. The parallel composition of two MTSs $\mathcal{M}_1, \mathcal{M}_2$ (Fig. 3 (left)) is shown in Fig. 4 (left).

Next, we introduce the notion of action independence which is subsequently used to define layered composition. Let $a(s)$ denote the unique state that can be reached from state s in one step (may or must) by performing action $a \in act(s)$ in a deterministic MTS $\mathcal{M}$. The dependency between two actions a and b is denoted $a \dagger b$. Two additional requirements for the dependency relation are that it is reflexive and symmetric. Two distinct actions that are not dependent are said to be independent, where independence is defined as follows:

Definition 13 (Action independence). *For an MTS* $\mathcal{M} = (S, Act, s_0, \{s_f\}, V)$, *actions* $a, b \in Act$ *are said to be independent, denoted by* $a \ddagger b$, *iff for all states* $s \in S$ *with* $a, b \in act(s)$ *we have:*

– $b \in act(a(s))$, $a \in act(b(s))$, *and* $a(b(s)) = b(a(s))$.

The first two conditions assert that a and b should not disable each other. The last condition asserts that the same state should be reached from s by either performing a followed by b, or by performing b followed by a. This notion of action independence originates from the partial-order reduction techniques [26, 8].

Definition 14. *MTSs* $\mathcal{M}_1$ *and* $\mathcal{M}_2$ *are independent, denoted* $\mathcal{M}_1 \ddagger \mathcal{M}_2$, *iff every action of* $\mathcal{M}_1$ *is independent of every action of* $\mathcal{M}_2$ *in* $\mathcal{M}_1 || \mathcal{M}_2$.

Let $s \xrightarrow{*} s'$ denote that state s' is reachable from s through an arbitrary finite sequence of transitions in MTS $\mathcal{M}$. In other words, $s \xrightarrow{*} s'$ means that there exists a finite abstract execution ρ in $\mathcal{M}$ that start in state s s.t. $last(\rho) = s'$.

Definition 15 (Layered composition). *Given MTSs* $\mathcal{M}_i = (S_i, Act_i, s_{0i}, \{s_{fi}\}, V_i)$, *where* $i \in \{1, 2\}$ *with* $S_1 \cap S_2 = \varnothing$. *The* layered composition *of* $\mathcal{M}_1$ *and* $\mathcal{M}_2$, *denoted* $\mathcal{M}_1 \bullet \mathcal{M}_2$, *is the MTS* $(S_1 \times S_2, Act_1 \cup Act_2, (s_{01}, s_{02}), \{(s_{f1}, s_{f2})\}, V)$ *where* V *is defined by:*

– *For all* $(s, s') \in S_1 \times S_2$, $a \in Act_1 \cap Act_2$, *if there exists* $u \in S_1$ *and* $u' \in S_2$, *such that* $V_1(s, a, u) \neq \bot$ *and* $V_2(s', a, u') \neq \bot$, *define* $V((s, s'), a, (u, u')) = V_1(s, a, u) \sqcap V_2(s', a, u')$. *If either* $\forall u \in S_1$, *we have* $V_1(s, a, u) = \bot$, *or* $\forall u' \in S_2$, *we have* $V_2(s', a, u') = \bot$ *then* $\forall (u, u') \in S_1 \times S_2$, $V((s, s'), a, (u, u')) = \bot$.

– *For all* $(s, s'), (u, u') \in S_1 \times S_2$, $a \in Act_1 \setminus Act_2$, *define* $V((s, s'), a, (u, u')) = V_1(s, a, u)$ *if* $s' = u'$, $V((s, s'), a, (u, u')) = \bot$ *otherwise.*
– *For all* $(s, s'), (u, u') \in S_1 \times S_2$, $a \in Act_2 \setminus Act_1$, *define* $V((s, s'), a, (u, u')) = V_2(s', a, u')$ *if* $s = u \wedge \forall s^* : s \xrightarrow{*} s^* : act(s^*) \ddagger a$, $V((s, s'), a, (u, u')) = \bot$ *otherwise.*

Note that the first two clauses of Def. 15 are the same as for Def. 12. Layered composition does not allow an action say d in $\mathcal{M}_2$ to be executed until all the actions in $\mathcal{M}_1$ (on which it is dependent) have been executed. In other words, all finite abstract executions, in which d is executed before any action say a, s.t. $a \dagger d$ will not be part of $Exec_{fin}(\mathcal{M}_1 \bullet \mathcal{M}_2)$ (this is guaranteed by the last clause of Def. 15).

Example 7. The layered composition of two MTSs $\mathcal{M}_1, \mathcal{M}_2$ (Fig. 3 (left)) is shown in Fig. 4 (right). Note that actions a, d are dependent, and therefore d cannot be executed before a in $\mathcal{M}_1 \bullet \mathcal{M}_2$.

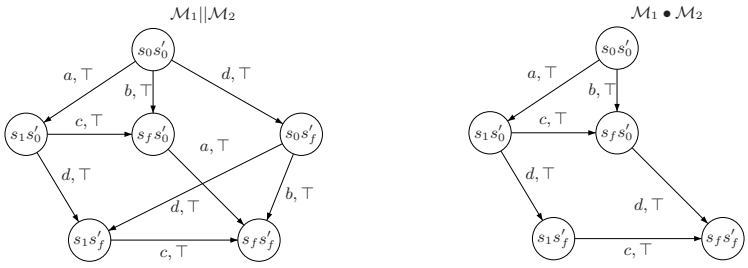

Fig. 4. Parallel composition $\mathcal{M}_1||\mathcal{M}_2$ (left) and layered composition $\mathcal{M}_1 \bullet \mathcal{M}_2$ (right) where $a \dagger d$

Next, we use the above mentioned composition operators for formulating the communication closed layer (CCL) laws as follows:

Theorem 1 (CCL laws). *For MTSs* $\mathcal{N}_1, \mathcal{N}_2, \mathcal{M}_1$, *and* $\mathcal{M}_2$, *with* $\mathcal{N}_1 \ddagger \mathcal{M}_2$ *and* $\mathcal{M}_1 \ddagger \mathcal{N}_2$, *the following communication closed layer (CCL) equivalences hold:*

– $\mathcal{N}_1 \bullet \mathcal{M}_2 \equiv \mathcal{N}_1||\mathcal{M}_2$ *(IND)*
– $(\mathcal{N}_1 \bullet \mathcal{N}_2)||\mathcal{M}_2 \equiv \mathcal{N}_1 \bullet (\mathcal{N}_2||\mathcal{M}_2)$ *(CCL-L)*
– $(\mathcal{N}_1 \bullet \mathcal{N}_2)||\mathcal{M}_1 \equiv (\mathcal{N}_1||\mathcal{M}_1) \bullet \mathcal{N}_2$ *(CCL-R)*
– $(\mathcal{N}_1 \bullet \mathcal{N}_2)||(\mathcal{M}_1 \bullet \mathcal{M}_2) \equiv (\mathcal{N}_1||\mathcal{M}_1) \bullet (\mathcal{N}_2||\mathcal{M}_2)$ *(CCL)*

5 Partial Order Equivalence and Property Preservation

This section defines the notion of partial order equivalence ($\equiv^*_{po}$) between MTSs which is used to prove that sequential and layered composition of MTSs satisfy the same

existential ($\exists$) and universal ($\forall$) reachability properties. For MTSs $\mathcal{M}_1$, and $\mathcal{M}_2$, let $\mathcal{M} = \mathcal{M}_1 \bullet \mathcal{M}_2$. Then we define $\mathcal{M}^{\setminus sync}$ as the MTS obtained from $\mathcal{M}$ s.t. it does not have any synchronized transitions (which are present in $\mathcal{M}$ as a result of synchronization over common actions). Intuitively, this means that abstract executions in $\mathcal{M}$ with synchronized transitions can be rewritten in $\mathcal{M}^{\setminus sync}$ such that for every synchronized transition there is a corresponding sequence of transitions in $\mathcal{M}^{\setminus sync}$ obtained by allowing interleaving on common actions. For example, let $\mathcal{M}$ have a may a-transition which is a result of synchronization of a must a-transition (from $\mathcal{M}_1$), and a may a-transition (from $\mathcal{M}_2$). In this case $\mathcal{M}^{\setminus sync}$ will have a corresponding[1] sequence of transitions, i.e., a must a-transition (from $\mathcal{M}_1$) followed by a may a-transition (from $\mathcal{M}_2$). This transformation is required as we want to establish the result that layered composition is po-equivalent to sequential composition. Note that sequential composition of two MTSs does not have synchronized transitions. This means that for any $\mathcal{M} = \mathcal{M}_1; \mathcal{M}_2$, $\mathcal{M}^{\setminus sync} = \mathcal{M}$.

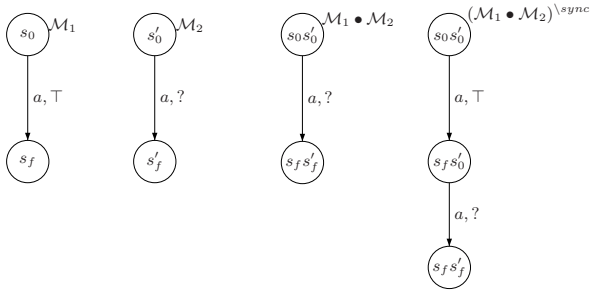

Fig. 5. MTS without synchronized transitions

Example 8. Consider the MTSs $\mathcal{M}_1$ and $\mathcal{M}_2$ shown in Fig. 5 (left). The layered composition of $\mathcal{M}_1, \mathcal{M}_2$, i.e., $\mathcal{M}_1 \bullet \mathcal{M}_2$ is shown in the middle, where $\mathcal{M}_1$ and $\mathcal{M}_2$ synchronize on common action a. A may transition is obtained in $\mathcal{M}_1 \bullet \mathcal{M}_2$ (since composing must-may results in a may transition). The MTS $(\mathcal{M}_1 \bullet \mathcal{M}_2)^{\setminus sync}$ without synchronized transitions is shown in Fig. 5 (right). Here the common action of $\mathcal{M}_1$ is executed before the common action of $\mathcal{M}_2$.

Theorem 2. *For MTSs $\mathcal{M}_1$ and $\mathcal{M}_2$, let $\mathcal{M} = \mathcal{M}_1 \bullet \mathcal{M}_2$, and S_f be the set of final states in $\mathcal{M}$. Then the following holds:*

$$\mathcal{M} \models^{\exists} \Diamond S_f \Leftrightarrow \mathcal{M}^{\setminus sync} \models^{\exists} \Diamond S_f$$

$$\mathcal{M} \models^{\forall} \Diamond S_f \Leftrightarrow \mathcal{M}^{\setminus sync} \models^{\forall} \Diamond S_f$$

[1] In fact, if $\mathcal{M}$ would not be deterministic, then two corresponding transition sequences making a diamond shape would be obtained in $\mathcal{M}^{\setminus sync}$.

In simple words this theorem says that reasoning about $\mathcal{M}^{\setminus sync}$ in place of $\mathcal{M}$ is not a restriction, as the behaviour of $\mathcal{M}$ (w.r.t. reachability properties) is mimicked by $\mathcal{M}^{\setminus sync}$. Next, we define the notion of partial order equivalence between two finite abstract executions.

Definition 16 (po-equivalence). *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two MTSs with transition functions V_1, and V_2 respectively. Let $\rho_1 \in Exec_{fin}(\mathcal{M}_1^{\setminus sync})$ and $\rho_2 \in Exec_{fin}(\mathcal{M}_2^{\setminus sync})$. Then $\rho_1 \equiv_{po} \rho_2$ iff there exist finite abstract executions ρ', ρ'' and $\exists a_1, b_1$ with $a_1 \neq b_1$ s.t. the following holds:*

- *ρ' represents the same sequence of transitions in ρ_1, ρ_2, and ρ'' represents the same sequence of transitions in ρ_1, ρ_2.*
- *$\rho_1 = \rho' a_1 s b_1 \rho'' \wedge \rho_2 = \rho' b_1 s' a_1 \rho''$, where $a_1 \ddagger b_1$.*
- *$V_1(last(\rho'), a_1, s) = V_2(s', a_1, first(\rho'')) \wedge V_1(s, b_1, first(\rho'')) = V_2(last(\rho'), b_1, s')$.*

Let $\equiv_{po}^*$, called po-equivalence*, denote the reflexive, transitive closure of $\equiv_{po}$.*

Stated in words, if two finite abstract executions ρ_1, ρ_2 are po-equivalent, then ρ_1 can be obtained from ρ_2 by repeated permutation of adjacent independent actions. Note that the first condition of Def. 16 is required to ensure that if for example $\rho' = s_0 c_1 s_1 d_1 s_2$ where c_1 is a must transition and d_1 is a may transition in $\mathcal{M}_1^{\setminus sync}$, then c_1, d_1 are also must and may transitions in $\mathcal{M}_2^{\setminus sync}$.

Definition 17 (Layered normal form). *Let S_f be the set of final states in $(\mathcal{M}_1 \bullet \mathcal{M}_2)^{\setminus sync}$. Then $\rho \in Exec_{fin}^{S_f}((\mathcal{M}_1 \bullet \mathcal{M}_2)^{\setminus sync})$ is in* layered normal form *(LNF) iff it involves the consecutive execution of actions of $\mathcal{M}_1$, followed by the consecutive execution of actions of $\mathcal{M}_2$.*

Let $Exec_{fin}^{LNF}((\mathcal{M}_1 \bullet \mathcal{M}_2)^{\setminus sync})$ denote the set of all finite abstract executions in $Exec_{fin}^{S_f}((\mathcal{M}_1 \bullet \mathcal{M}_2)^{\setminus sync})$ that are in LNF.

Next we show that for each finite abstract execution of $Exec_{fin}^{S_f}((\mathcal{M}_1 \bullet \mathcal{M}_2)^{\setminus sync})$, a po-equivalent abstract exceution in LNF does exist.

Lemma 1 (LNF existence). *Let $\mathcal{M}_1, \mathcal{M}_2$ be two MTSs. Then we have $\forall \rho \in Exec_{fin}^{S_f}($ $(\mathcal{M}_1 \bullet \mathcal{M}_2)^{\setminus sync}) \exists \rho' \in Exec_{fin}^{LNF}((\mathcal{M}_1 \bullet \mathcal{M}_2)^{\setminus sync})$ s.t. $\rho \equiv_{po}^* \rho'$.*

Definition 18 (po-equivalence for MTSs). *Two MTSs $\mathcal{M}_1, \mathcal{M}_2$ are said to be po-equivalent, denoted $\mathcal{M}_1 \equiv_{po}^* \mathcal{M}_2$, iff for $i \in \{1, 2\}$: $\forall \rho_i \in Exec_{fin}^{S_{fi}}(\mathcal{M}_i^{\setminus sync}) \exists \rho_{3-i} \in Exec_{fin}^{S_{f3-i}}(\mathcal{M}_{3-i}^{\setminus sync})$ s.t. $\rho_i \equiv_{po}^* \rho_{3-i}$.*

Theorem 3. *For every MTSs $\mathcal{M}_1, \mathcal{M}_2$, we have $\mathcal{M}_1 \bullet \mathcal{M}_2 \equiv_{po}^* \mathcal{M}_1; \mathcal{M}_2$.*

Example 9. It is easy to check that MTS $\mathcal{M}_1; \mathcal{M}_2$ given in Fig. 3 (right) is po-equivalent to MTS $\mathcal{M}_1 \bullet \mathcal{M}_2$ given in Fig. 4 (right).

Theorem 4 (Property preservation). *For MTSs $\mathcal{M}_1, \mathcal{M}_2$, let $\mathcal{M} = \mathcal{M}_1 \bullet \mathcal{M}_2$, $\mathcal{M}' = \mathcal{M}_1; \mathcal{M}_2$ with set of final states S_f, and S'_f respectively. If $\mathcal{M} \equiv^*_{po} \mathcal{M}'$ then we have:*

$$\mathcal{M} \models^{\exists} \Diamond S_f \textit{ iff } \mathcal{M}' \models^{\exists} \Diamond S'_f$$

$$\mathcal{M} \models^{\forall} \Diamond S_f \textit{ iff } \mathcal{M}' \models^{\forall} \Diamond S'_f$$

This theorem asserts that po-equivalence between layered and sequential composition operators satisfies the same reachability properties.

Corollary 1. *Replacing $\bullet$ by ; in the CCL laws yields the po-equivalence which satisfies the same $\exists$ and $\forall$ reachability properties.*

Corollary 1 enable us to replace ; with $\bullet$ and vice versa. This replacement along with CCL laws (Theorem 1) can be used for state space reduction as follows:

State space reduction. Let $\mathcal{N}_1, \mathcal{N}_2$ and $\mathcal{M}_1$ be three MTSs, and $\mathcal{N} = (\mathcal{N}_1; \mathcal{N}_2)||\mathcal{M}_1$. Let us say we want to check whether $\mathcal{N} \models^{\exists} \Diamond S_f$ or $\mathcal{N} \models^{\forall} \Diamond S_f$. Here, S_f is set of final states in $\mathcal{N}$. Assume $\mathcal{M}_1 \ddagger \mathcal{N}_2$, and $\mathcal{N}_1, \mathcal{N}_2, \mathcal{M}_1$ each consist of 20 states. In this case $\mathcal{N}_1; \mathcal{N}_2$ has 39 states which combined with the 20 states of $\mathcal{M}_1$ gives 780 states. We can transform $\mathcal{N}$ in the following way:

$$\begin{array}{ll} (\mathcal{N}_1; \mathcal{N}_2)||\mathcal{M}_1 & \\ \equiv^*_{po} & \text{Corollary 1} \\ (\mathcal{N}_1 \bullet \mathcal{N}_2)||\mathcal{M}_1 & \\ \equiv & \text{CCL-R} \\ (\mathcal{N}_1||\mathcal{M}_1) \bullet \mathcal{N}_2 & \\ \equiv^*_{po} & \text{Corollary 1} \\ (\mathcal{N}_1||\mathcal{M}_1); \mathcal{N}_2 & \end{array}$$

Note that the transformed system, i.e., $(\mathcal{N}_1||\mathcal{M}_1); \mathcal{N}_2$ has 419 states.

6 Conclusion

This paper presented a state-space reduction technique for a network of acyclic MTSs, based on the notion of layering. We proposed a layered composition operator, and formulated communication closed layer (CCL) laws. Next, we define the partial order (po) equivalence between acyclic MTSs, show that layered and sequential compostion operators are po-equivalent and satisfy the same existential ($\exists$) and universal ($\forall$) reachability properties. As implementations of distributed systems typically are in terms of layers, we believe that enabling transforming system MTS specifications into layered form will substantially ease the proof of correct implementation. The theory of layering proposed in this paper can be extended to acyclic MTSs equipped with data variables. An MTS $\mathcal{M}$ can be extended with data variables such that whenever an action is executed its associated data variables are updated according to an arithmetic expression. These data

variables can take values in some finite range D. The definitions of satisfaction, and refinement can be slightly modified by placing an extra condition that ensures that related states have the same valuations. For an MTS with data, two actions are said to be dependent if one of the two writes a variable that is read or written by the other action. More formally, two actions a and b are dependent, denoted $a \dagger b$, if any one of the following holds:

$$Write(b) \cap Read(a) \neq \varnothing,$$
$$Write(a) \cap Read(b) \neq \varnothing,$$
$$Write(a) \cap Write(b) \neq \varnothing.$$

Here, $Write(a)$ denotes the set of data variables written by the action a. Similarly, $Read(a)$ denotes the set of data variables read by the action a. Using this dependency relation, our theory can be applied to MTSs with data. We do not go into details on these matters here, however, refer interested reader to [2, 11, 30]. Future work includes the application of this technique to practical case studies which involve modeling distributed systems using MTSs.

Acknowledgements. The authors thank Ian Larson for modeling the randomized mutual exclusion algorithm case study in PRISM and conducting the experiments.

References

1. C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.
2. S. S. Bauer, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski. A modal specification theory for components with data. In *FACS*, LNCS 7253, pages 61–78, 2011.
3. N. Benes, I. Cerná, and J. Kretínský. Modal transition systems: Composition and LTL model checking. In *ATVA*, LNCS 6996, pages 228–242, 2011.
4. P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1986.
5. G. Bruns. An industrial application of modal process logic. *Sci. Comput. Program.*, 29(1-2):3–22, July 1997.
6. T. Elrad and N. Francez. Decomposition of distributed programs into communication-closed layers. *Sci. Comput. Program.*, 2(3):155–173, 1982.
7. R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, 1983.
8. R. Gerth, R. Kuiper, D. Peled, and W. Penczek. A partial order approach to branching time logic model checking. In *ISTCS*, pages 130–139, 1995.
9. A. Gruler, M. Leucker, and K. D. Scheidemann. Modeling and model checking software product lines. In *FMOODS*, pages 113–131, 2008.
10. M. Huth, R. Jagadeesan, and D. A. Schmidt. Modal transition systems: A foundation for three-valued program analysis. In *ESOP*, pages 155–169, 2001.
11. W. Janssen. *Layered Design of Parallel Systems*. PhD dissertation, Universiteit Twente, 1994.
12. W. Janssen, M. Poel, Q. Xu, and J. Zwiers. Layering of real-time distributed processes. In *FTRTFT*, LNCS 863, pages 393–417, 1994.
13. W. Janssen, M. Poel, and J. Zwiers. Action systems and action refinement in the development of parallel systems - an algebraic approach. In *CONCUR*, LNCS 527, pages 298–316, 1991.

14. W. Janssen and J. Zwiers. From sequential layers to distributed processes: Deriving a distributed minimum weight spanning tree algorithm (extended anstract). In *PODC*, pages 215–227. ACM, 1992.
15. E. Kushilevitz and M. O. Rabin. Randomized mutual exclusion algorithms revisited. In *PODC*, pages 275–283, 1992.
16. M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, LNCS 6806, pages 585–591, 2011.
17. K. G. Larsen. Modal specifications. In *Automatic Verification Methods for Finite State Systems*, LNCS 407, pages 232–246, 1989.
18. K. G. Larsen, U. Nyman, and A. Wasowski. On modal refinement and consistency. In *CONCUR*, LNCS 4703, pages 105–119, 2007.
19. K. G. Larsen, B. Steffen, and C. Weise. A constraint oriented proof methodology based on modal transition systems. In *TACAS*, LNCS 1019, pages 17–40, 1995.
20. K. G. Larsen and B. Thomsen. A modal process logic. In *LICS*, pages 203–210, 1988.
21. K. G. Larsen and L. Xinxin. Equation solving using modal transition systems. In *LICS*, pages 108–117, 1990.
22. A. K. McIver, C. Gonzalia, E. Cohen, and C. C. Morgan. Using probabilistic Kleene algebra pKA for protocol verification. *J. Log. Algebr. Program.*, 76(1):90–111, 2008.
23. R. Milner. *Communication and Concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.
24. Y. Moses and S. Rajsbaum. A layered analysis of consensus. *SIAM J. Comput.*, 31(4):989–1021, 2002.
25. E.-R. Olderog and M. Swaminathan. Layered composition for timed automata. In *FORMATS*, LNCS 6246, pages 228–242, 2010.
26. D. Peled. Combining partial order reductions with on-the-fly model-checking. In *CAV*, pages 377–390, 1994.
27. J.-B. Raclet. Residual for component specifications. *Electr. Notes Theor. Comput. Sci.*, 215:93–110, 2008.
28. J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, and R. Passerone. Why are modalities good for interface theories? In *ACSD*, pages 119–127, 2009.
29. D. A. Schmidt. From trace sets to modal-transition systems by stepwise abstract interpretation. pages 53–71, 2001.
30. M. Swaminathan, J.-P. Katoen, and E.-R. Olderog. Layered reasoning for randomized distributed algorithms. *Formal Asp. Comput.*, 24(4-6):477–496, 2012.
31. S. Uchitel and M. Chechik. Merging partial behavioural models. In *SIGSOFT FSE*, pages 43–52, 2004.

Appendix

Proof of Theorem 1.

Proof. We provide the proof of the law CCL-L. The proofs of the other CCL laws are similar. Let $\mathcal{T} = (\mathcal{N}_1 \bullet \mathcal{N}_2)||\mathcal{M}_2$ and $\mathcal{U} = \mathcal{N}_1 \bullet (\mathcal{N}_2||\mathcal{M}_2)$. In order to prove that $\mathcal{T} \equiv \mathcal{U}$, we have to show $\mathcal{T} \preceq_S \mathcal{U}$ and $\mathcal{U} \preceq_S \mathcal{T}$. Let $((x, y), z)$ and $(x, (y, z))$ denote states of MTSs $\mathcal{T}$ and $\mathcal{U}$, respectively. Here x, y, z represent the state components of $\mathcal{N}_1, \mathcal{N}_2$ and $\mathcal{M}_2$, respectively. We only prove $\mathcal{T} \preceq_S \mathcal{U}$, the proof of $\mathcal{U} \preceq_S \mathcal{T}$ is very similar. To show $\mathcal{T} \preceq_S \mathcal{U}$, we have to prove that $\mathcal{T}$ strongly refines $\mathcal{U}$ according to Def. 6. Let S^* and S be the state space of $\mathcal{T}$ and $\mathcal{U}$, respectively. Let $\mathcal{R} \subseteq S^* \times S$ be a binary relation s.t. $\mathcal{R} = \{\big(((x, y), z), (x, (y, z))\big), \big(((x', y), z), (x', (y, z))\big), \big(((x, y'), z'), (x, (y', z'))\big)$

$\ldots\}$. Intuitively $\mathcal{R}$ relates states of S^* to those states in S that have the same individual state components from $\mathcal{N}_1, \mathcal{N}_2$ and $\mathcal{M}_2$. Let us consider a state $s = (x,(y,z))$ from $\mathcal{U}$ and $s^* = ((x,y),z)$ from $\mathcal{T}$. From the definition of $\mathcal{R}$, we know that $(s^*, s) \in \mathcal{R}$. A transition from s in $\mathcal{U}$ or s^* in $\mathcal{T}$ can be performed by 1) $\mathcal{N}_1$ individually, or 2) $\mathcal{N}_2$ individually, or 3) $\mathcal{M}_2$ individually, or 4) $\mathcal{N}_1$ and $\mathcal{N}_2$ simultaneously, or 5) $\mathcal{N}_2$ and $\mathcal{M}_2$ simultaneously. We show that for every case[2] the conditions of Def. 6 are satisfied, and thus $\mathcal{R}$ is a strong refinement relation between $\mathcal{T}$ and $\mathcal{U}$. It is easy to check that condition 3 which requires final states to be related is satisfied. This is because $\mathcal{R}$ relates states that have the same individual state components from $\mathcal{N}_1, \mathcal{N}_2$ and $\mathcal{M}_2$.
1) $\mathcal{N}_1$ individually: Let $s \xrightarrow{a}_{\top} s'$ (resp. $s \xrightarrow{a}_{?} s'$) be a transition of $\mathcal{N}_1$ taken in $\mathcal{U}$, where $s' = (x',(y,z))$. Since $\mathcal{N}_1$ is the left operand of layering in $\mathcal{U}$ such a transition is also possible from the related state $s^* = ((x,y),z)$, i.e., $s^* \xrightarrow{a}_{\top} s^{**}$ (resp. $s^* \xrightarrow{a}_{?} s^{**}$), where $s^{**} = ((x',y),z)$. From the definition of $\mathcal{R}$ we know that $(s^{**}, s') \in \mathcal{R}$. Similarly, it can be shown that for every transition $s^* \xrightarrow{a}_{\top} s^{***}$ (resp. $s^* \xrightarrow{a}_{?} s^{***}$), there exists a corresponding transition, $s \xrightarrow{a}_{\top} s''$ (resp. $s \xrightarrow{a}_{?} s''$) s.t. $(s^{***}, s'') \in \mathcal{R}$.
2) $\mathcal{N}_2$ individually: Similar to case 1 above.
3) $\mathcal{M}_2$ individually: Let $s \xrightarrow{a}_{\top} s'$ (resp. $s \xrightarrow{a}_{?} s'$) be a transition of $\mathcal{M}_2$ taken in $\mathcal{U}$, where $s' = (x,(y,z'))$. Since this transition is possible after layering operator in $\mathcal{U}$, it is also possible in $\mathcal{T}$ from the state related to s, i.e., s^* as this action is not waiting for some action of $\mathcal{N}_1$ to be executed, and $\mathcal{U}$ induces fewer interleavings due to dominance of layered composition operator. Next, we consider a transition of $\mathcal{M}_2$ taken in $\mathcal{T}$ from state s^*. Since $\mathcal{N}_1 \ddagger \mathcal{M}_2$, a similar transition exists in $\mathcal{U}$ from the state related to s^*, i.e., s as putting $\mathcal{M}_2$ after the layering operator is not a problem as it is independent from $\mathcal{N}_1$.
4) $\mathcal{N}_1$ and $\mathcal{N}_2$ simultaneously: Let $s \xrightarrow{a}_{\top} s'$ (resp. $s \xrightarrow{a}_{?} s'$) be a transition in $\mathcal{U}$ as a result of a synchronization of $\mathcal{N}_1$ and $\mathcal{N}_2$ on action a, where $s' = (x',(y',z))$. Again as $\mathcal{U}$ induces fewer interleavings due to dominance of layered operator, a similar transition is possible in $\mathcal{T}$ from state related to s, i.e., s^*. Similarly, for any transition in $\mathcal{T}$ from s^*, a corresponding transition is enabled in $\mathcal{U}$ as $\mathcal{N}_2 || \mathcal{M}_2$ will not block it (due to the fact that the synchronizing action is not waiting for an action from $\mathcal{M}_2$ to be executed since parallel composition does not respect dependencies).
5) $\mathcal{N}_2$ and $\mathcal{M}_2$ simultaneously: Let $s \xrightarrow{a}_{\top} s'$ (resp. $s \xrightarrow{a}_{?} s'$) be a transition in $\mathcal{U}$ as a result of a synchronization of $\mathcal{N}_2$ and $\mathcal{M}_2$ on action a, where $s' = (x,(y',z'))$. As this action is possible in $\mathcal{N}_2 || \mathcal{M}_2$, it means that this action is not waiting for the execution of some actions from $\mathcal{N}_1$. It is therefore possible to take the same transition in $\mathcal{T}$ from s^* as $\mathcal{N}_1 \bullet \mathcal{N}_2$ will not block it. Similarly, for any transition in $\mathcal{T}$ from s^*, there will be a corresponding transition in $\mathcal{U}$. This is due to the fact this action is not waiting for actions of $\mathcal{N}_1$ to be executed as $\mathcal{N}_1 \ddagger \mathcal{M}_2$. □

Proof of Theorem 2.

Proof. ($\exists$ reachability): We provide the proof of $\mathcal{M} \models^{\exists} \Diamond S_f \Rightarrow \mathcal{M}^{\setminus sync} \models^{\exists} \Diamond S_f$. The proof of $\mathcal{M}^{\setminus sync} \models^{\exists} \Diamond S_f \Rightarrow \mathcal{M} \models^{\exists} \Diamond S_f$ is similar. Let $\mathcal{T} \in [\![\mathcal{M}]\!]$ be an LTS

[2] Note that we do not consider the case where $\mathcal{N}_1$ and $\mathcal{M}_2$ move simultaneously. This is due to the fact that $\mathcal{N}_1 \ddagger \mathcal{M}_2$, and therefore they cannot have common actions (since the dependency relation is reflexive).

s.t. $\mathcal{T} \models \Diamond S_f$. We know that every finite path $\pi \in Paths^{S_f}_{fin}(\mathcal{T})$ is a realisation of some finite abstract execution $\rho \in Exec^{S_f}_{fin}(\mathcal{M})$, and no finite path can be a realisation of more than one finite abstract execution in $\mathcal{M}$ (since $\mathcal{M}$ is deterministic). Let η be the set of all such finite abstract executions where $\eta \subseteq Exec^{S_f}_{fin}(\mathcal{M})$. From the definition of $\mathcal{M}^{\setminus sync}$ we know that for each finite abstract execution $\rho \in \eta$ there exists a corresponding finite abstract execution in $Exec^{S_f}_{fin}(\mathcal{M}^{\setminus sync})$ obtained by allowing interleaving on common actions s.t. the common action of $\mathcal{M}_1$ is executed followed by execution of the common action of $\mathcal{M}_2$. Let $\eta' \subseteq Exec^{S_f}_{fin}(\mathcal{M}^{\setminus sync})$ be the set of all such finite abstract executions. Let $\mathcal{T}' \in [\![\mathcal{M}^{\setminus sync}]\!]$ be an LTS s.t. for every finite abstract execution $\rho' \in \eta'$ there exists a finite path π' in $\mathcal{T}'$ that is a realisation of ρ' and $\mathcal{T}'$ does not contain any path $\pi' \in Paths^{S_f}_{fin}(\mathcal{T}')$ which is not a realisation of some finite abstract execution $\rho' \in \eta'$. In other words we have constructed an implementation $\mathcal{T}'$ corresponding to $\mathcal{T}$ s.t. $\mathcal{T}' \models \Diamond S_f$.
($\forall$ reachability): We provide the proof of $\mathcal{M} \models^{\forall} \Diamond S_f \Rightarrow \mathcal{M}^{\setminus sync} \models^{\forall} \Diamond S_f$. The proof of $\mathcal{M}^{\setminus sync} \models^{\forall} \Diamond S_f \Rightarrow \mathcal{M} \models^{\forall} \Diamond S_f$ is similar. From the definition of $\forall$ reachability (Def. 10) we know that $\mathcal{M}$ reaches S_f if and only if all the implementations of $\mathcal{M}$ are able to reach S_f. This intuitively means that $\mathcal{M}$ does not have those may transitions that block any of its implementations from reaching the final state. Since $\mathcal{M}^{\setminus sync}$ is obtained from $\mathcal{M}$ by allowing interleaving on common actions, such may transitions (or equivalent transition sequences) are also absent in $\mathcal{M}^{\setminus sync}$. In other words, every implementation $\mathcal{T}' \in [\![\mathcal{M}^{\setminus sync}]\!]$ reaches S_f, i.e., $\mathcal{T}' \models \Diamond S_f$. □

Proof of Lemma 1.

Proof. From the definition of layered normal form (LNF) we know that all finite abstract executions that are in LNF consists of the consecutive execution of actions of $\mathcal{M}_1$, followed by the consecutive execution of actions of $\mathcal{M}_2$. We know that in $((\mathcal{M}_1 \bullet \mathcal{M}_2)^{\setminus sync})$, an action of $\mathcal{M}_2$ occurs only when all the actions in $\mathcal{M}_1$ on which it is dependent have been executed. This intuitively means that all the actions of $\mathcal{M}_2$ that occur in a finite abstract execution before any action of $\mathcal{M}_1$ are independent of this action and thus by repeated permutation of these actions any finite abstract execution $\rho \in Exec^{S_f}_{fin}((\mathcal{M}_1 \bullet \mathcal{M}_2)^{\setminus sync})$ can be converted to a finite abstract execution that is in LNF. □

Proof of Theorem 3.

Proof. Let $\mathcal{M}_1, \mathcal{M}_2$ be MTSs. From the definition of LNF (Def. 17), we know that a finite abstract execution in LNF involves the consecutive execution of actions of $\mathcal{M}_1$, followed by the consecutive execution of actions of $\mathcal{M}_2$. This means that for every finite abstract execution $\rho \in Exec^{LNF}_{fin}((\mathcal{M}_1 \bullet \mathcal{M}_2)^{\setminus sync})$ there exists a finite abstract execution ρ' in $((\mathcal{M}_1; \mathcal{M}_2)^{\setminus sync})$ that ends in the final state and where: $\forall n \geq 0 : \rho[n] \approx \rho'[n] \wedge \rho[n] \xrightarrow{a_{n+1}} {}_{\top}\rho[n+1] \Rightarrow \rho'[n] \xrightarrow{a_{n+1}} {}_{\top}\rho'[n+1]$(resp. $\rho[n] \xrightarrow{a_{n+1}} {}_{?}\rho[n+1] \Rightarrow \rho'[n] \xrightarrow{a_{n+1}} {}_{?}\rho'[n+1]$). The relation $\approx$ between states of $((\mathcal{M}_1 \bullet \mathcal{M}_2)^{\setminus sync})$ and $((\mathcal{M}_1; \mathcal{M}_2)^{\setminus sync})$ is defined as follows: $S_1 \times S_2$ is the state space of $((\mathcal{M}_1 \bullet \mathcal{M}_2)^{\setminus sync})$ and $(S_1 \setminus \{s_{f1}\} \cup S_2)$ is the state space of $((\mathcal{M}_1; \mathcal{M}_2)^{\setminus sync})$ then $\forall s_1 \in S_1$ where $s_1 \neq s_{f1} : (s_1, s_{02}) \approx s_1$ and $\forall s_2 \in S_2 : (s_{f1}, s_2) \approx s_2$.

In other words we have related every finite abstract execution of $((\mathcal{M}_1 \bullet \mathcal{M}_2)^{\setminus sync})$ that is in LNF to some finite abstract execution in $((\mathcal{M}_1; \mathcal{M}_2)^{\setminus sync})$ that ends in the final state and vice-versa. It is also clear from Lemma 1 that for every finite abstract execution ρ in $((\mathcal{M}_1 \bullet \mathcal{M}_2)^{\setminus sync})$ that ends in the final state, there exists a finite abstract execution ρ' in $((\mathcal{M}_1 \bullet \mathcal{M}_2)^{\setminus sync})$ that is LNF s.t. $\rho \equiv^*_{po} \rho'$. □

Proof of Theorem 4.

Proof. (∃ reachability): We provide the proof of $\mathcal{M} \models^{\exists} \Diamond S_f \Rightarrow \mathcal{M}' \models^{\exists} \Diamond S'_f$. The proof of $\mathcal{M}' \models^{\exists} \Diamond S'_f \Rightarrow \mathcal{M} \models^{\exists} \Diamond S_f$ is similar. Let $\mathcal{T} \in [\![\mathcal{M}^{\setminus sync}]\!]$ be an LTS s.t. $\mathcal{T} \models \Diamond S_f$. We know that every finite path $\pi \in Paths^{S_f}_{fin}(\mathcal{T})$ is a realisation of some finite abstract execution $\rho \in Exec^{S_f}_{fin}(\mathcal{M}^{\setminus sync})$, and no finite path can be a realisation of more than one finite abstract execution in $\mathcal{M}^{\setminus sync}$ (since $\mathcal{M}^{\setminus sync}$ is deterministic). Let η be the set of all such finite abstract executions where $\eta \subseteq Exec^{S_f}_{fin}(\mathcal{M}^{\setminus sync})$. From the definition of po-equivalence for MTSs (Def. 18) we know that for set η there exists a set $\eta' \subseteq Exec^{S'_f}_{fin}(\mathcal{M}'^{\setminus sync}) : \forall \rho \in \eta \exists \rho' \in \eta' : \rho \equiv^*_{po} \rho'$ and vice versa. Since both $\mathcal{M}^{\setminus sync}$ and $\mathcal{M}'^{\setminus sync}$ are deterministic, this intuitively means that $|\eta| = |\eta'|$. Let $\mathcal{T}' \in [\![\mathcal{M}'^{\setminus sync}]\!]$ be an LTS s.t. for every finite abstract execution $\rho' \in \eta'$ there exists a finite path in $\mathcal{T}'$ that is a realisation of ρ' and $\mathcal{T}'$ does not contain any path $\pi' \in Paths^{S'_f}_{fin}(\mathcal{T}')$ which is not a realisation of some finite abstract execution $\rho' \in \eta'$. In other words we have constructed an implementation $\mathcal{T}'$ corresponding to $\mathcal{T}$ s.t. $\mathcal{T}' \models \Diamond S'_f$.
(∀ reachability): We provide the proof of $\mathcal{M} \models^{\forall} \Diamond S_f \Rightarrow \mathcal{M}' \models^{\forall} \Diamond S'_f$. The proof of $\mathcal{M}' \models^{\forall} \Diamond S'_f \Rightarrow \mathcal{M} \models^{\forall} \Diamond S_f$ is similar. From the definition of ∀ reachability (Def. 10) we know that $\mathcal{M}$ reaches S_f if and only if all the implementations of $\mathcal{M}$ are able to reach S_f. This intuitively means that $\mathcal{M}^{\setminus sync}$ does not have those may transitions that block any of its implementations from reaching the final state. Since $\mathcal{M}'$ is po-equivalent to $\mathcal{M}$ and $\mathcal{M}'$ involves the consecutive execution of actions of $\mathcal{M}_1$, followed by the consecutive execution of actions of $\mathcal{M}_2$ before reaching S'_f, such may transitions are also absent in $\mathcal{M}'^{\setminus sync}$. In other words, every implementation $\mathcal{T}' \in [\![\mathcal{M}'^{\setminus sync}]\!]$ reaches S'_f, i.e.$\mathcal{T}' \models \Diamond S'_f$. □