Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science 145 (2006) 95–111
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Concurrent LSC Verification

On Decomposition Properties of Partially Ordered Symbolic Automata

Tobe Toben [1,3] Bernd Westphal [2,4]

Carl von Ossietzky Universität Oldenburg, 26111 Oldenburg, Germany

Abstract

Partially Ordered Symbolic Automata (POSAs) are used as the semantical foundation of visual formalisms like the scenario based language of Live Sequence Charts (LSCs). To check whether a model satisfies an LSC requirement, the LSC's POSA can be composed in parallel to the model as an *observer automaton* or it can be translated to a CTL or LTL formula. Thus by the well-known complexity properties of CTL and LTL model-checking, the size of an LSC's POSA directly contributes to the runtime of the model-checking task. The size grows with the concurrency allowed by the LSC, e.g. when the observation order of LSC elements is relaxed by enclosing the elements in a coregion. We investigate decomposition properties of POSAs with deterministic states, i.e. states with disjointly annotated outgoing transitions. We devise a procedure to decompose a POSA with deterministic states into a set of POSAs whose intersection language is equal to the language of the original POSA. When decomposing at dominating states, the obtained POSAs are strictly smaller. As the majority of states in POSAs obtained for LSCs are deterministic and dominating, model-checking of LSCs can effectively be distributed.

Keywords: Distributed Model-Checking, Live Sequence Charts, Automata Decomposition.

[1] This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Centre SFB/TR 14 AVACS.

[2] This work was partly supported by the German Research Council (DFG) within project USE (DA 206/7-3) as part of the priority program SPP 1064 .

[3] Email: toben@informatik.uni-oldenburg.de

[4] Email: westphal@informatik.uni-oldenburg.de

doi:10.1016/j.entcs.2005.10.007

1 Introduction

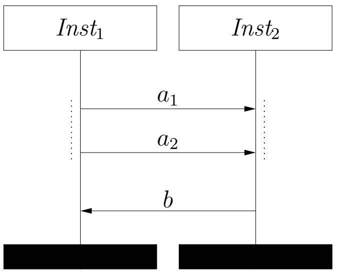
Functional requirements on a modular system under design comprise requirements on the single modules, the *intra-object* behaviour, as well as on the communication between the modules, the *inter-object* behaviour.

The Live Sequence Chart (LSC) language is a visual formalism for *scenarios*, i.e. sequences of inter-object communication [4,9,7]. It is a conservative extension of the well-known Message Sequence Chart (MSC) language [8] but of significantly enhanced expressive power, basically by providing modalities for the whole chart, for elements like messages and conditions, and for locations in the chart. For brevity, we don't discuss the LSC language and its intuition in detail but now only briefly recall the modalities and will elaborate on the examples used throughout the paper. For a more thorough introduction the reader is referred to the literature [4,9,7].

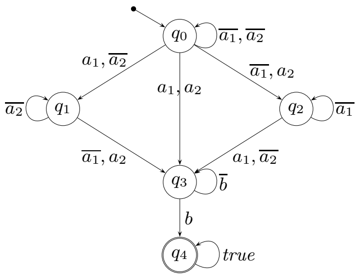
The mode of a chart is one of *existential* and *universal* and allows the specifier to distinguish *examples* from *protocols*. An example is satisfied by a system if there *is* a system run adhering to the scenario (default interpretation of MSCs), a protocol is satisfied if each run of the system adheres to the scenario. The mode of a location, i.e. a graphical position where an LSC element is connected to an instance line, is *hot* or *cold*. The former requires progress, the latter does not. The mode of an LSC element, e.g. a condition or message, is *mandatory* or *possible*. If a mandatory condition is violated, then the run does not satisfy the LSC. If a possible condition is violated, then the scenario is immediately exited and the run is considered to satisfy the LSC.

Just like for MSCs, the observation order of LSC elements is determined by their relative order per instance line. Elements on different instance lines are unordered unless they are synchronised. Synchronising elements are instantaneous messages and conditions, whose locations have to be observed simultaneously, and asynchronous messages, whose reception has to be observed strictly after its sending. But the order of observations can not only be restricted but also explicitly relaxed by enclosing multiple elements into a coregion, graphically indicated by a dotted line in parallel to the instance line. For example consider the LSC body[5] shown in Fig. 1(a). The sendings and receptions of the instantaneous messages a_1 and a_2 are enclosed in a coregion as indicated by the dotted lines, hence they may occur in any order or even simultaneously. As message 'b' is not enclosed in a coregion, it is supposed to be observed strictly after both a_1 and a_2. In other words, the LSC shown in Fig. 1(a) requires module $Inst_2$ to reply with a 'b' after a_1 and a_2 have been

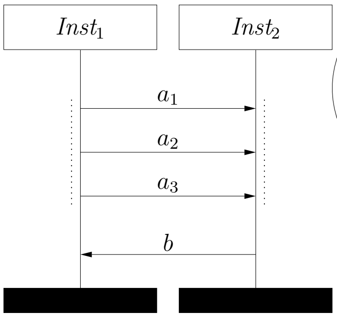
[5] an LSC comprises an LSC body, a name, a chart mode, an activation condition, and an activation mode (cf. Sec. 4 for an introduction of the two latter concepts)



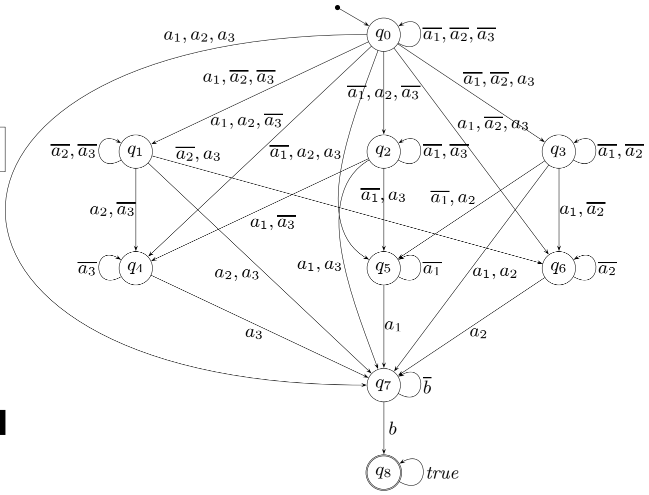
(a) Two messages coregion.



(b) Symbolic Automaton of Fig. 1(a).



(c) Three messages coregion.



q_0
$\overline{a_1}, \overline{a_2}, \overline{a_3}$
a_1, a_2, a_3
$a_1, \overline{a_2}, \overline{a_3}$
$\overline{a_1}, \overline{a_2}, a_3$
$\overline{a_1}, a_2, \overline{a_3}$
$a_1, a_2, \overline{a_3}$
$a_1, \overline{a_2}, a_3$
$\overline{a_1}, a_2, a_3$
q_1
$\overline{a_2}, \overline{a_3}$
$\overline{a_2}, a_3$
q_2
$\overline{a_1}, \overline{a_3}$
q_3
$\overline{a_1}, \overline{a_2}$
$a_2, \overline{a_3}$
$\overline{a_1}, a_3$
$\overline{a_1}, a_2$
$a_1, \overline{a_2}$
$a_1, \overline{a_3}$
q_4
$\overline{a_3}$
q_5
$\overline{a_1}$
q_6
$\overline{a_2}$
a_2, a_3
a_1, a_3
a_1, a_2
a_3
a_1
a_2
q_7
$\overline{b}$
b
q_8
true

(d) Symbolic Automaton of Fig. 1(c).

Fig. 1. **Concurrent LSC examples.** A comma (',') in the transition annotation reads as a conjunction and an overline as a negation. Each message name refers to both sending and reception of the message since the examples use only instantaneous messages. For example, the transition from q_0 to q_1 in Fig. 1(b) can only be taken if message a_1 is sent and received while a_2 is neither sent nor received.

received in any order.

The semantics of LSCs [9] is explained in terms of Symbolic Automata, a variant of Büchi Automata where transitions are annotated by boolean expressions instead of by elements of an alphabet. The Symbolic Automata obtained for LSCs are actually Partially Ordered Symbolic Automata (POSA) where the only cycles in the automaton are self-loops [12]. Fig. 1(b) shows the POSA for the LSC in Fig. 1(a) as defined by [9]. Each automaton state corresponds directly to a partial observation or *cut* of the LSC. For example, state q_1 corresponds to the cut where message a_1 has been observed but a_2 not yet. The only legal continuation is to first observe a_2 and then 'b'.

The instance lines in Fig. 1(a) being solid (not dashed) indicates that all locations have mode *hot* and thus progress is enforced. Consequently there is only a single *accepting state* in the POSA in Fig. 1(b). Recall that a Büchi automaton accepts a word if the word can trigger a sequence of transitions s.t. an accepting state is visited infinitely often.

Fig. 1(c) shows basically the same LSC as Fig. 1(a) but adds a third message a_3 to be sent from module $Inst_1$ to $Inst_2$ before $Inst_2$ replies with a 'b' message. It is not very surprising that the corresponding POSA shown in Fig. 1(d) explodes, both in number of states and in number of transitions, since it now encodes all interleavings of three messages.

Although being synthetic examples, the LSCs shown in Fig. 1 are not pathological since it is common practice to leave the particular order of actions open in the requirements, in particular in higher level specifications and in earlier stages of the development of requirements. Note that coregions are particularly well-suited to demonstrate the impact of concurrency in an LSC to the POSA in small examples like in Fig. 1, but in practice the far more frequently occurring source for unrestricted orders is that elements in unrelated parts of an LSC are not synchronised by synchronising elements. If the modules referred to in the specification are supposed to run concurrently on different resources, the interleaving is inherently unrestricted which has to be reflected in the requirements specification.

For model-checking LSCs against system models there are two approaches in use. Firstly, the POSA can be transformed into an *observer automaton* that is parallel composed to the model [6]. For example in case of universal LSCs, the model satisfies the LSC if the parallel composition satisfies globally that finally an accepting state is reached. Secondly, the POSA can be translated into a CTL formula [12,10] and directly be checked, a subset of LSCs can be translated to an LTL formula [10].

As the size of the model in terms of states and transitions and the size of the specification contribute at least linearly to the worst-case time complexity of CTL and LTL model-checking, model-checking of LSCs faces an explosion problem on the side of the *specification* in the presence of massive concurrency in the LSC. When using the observer automaton, the checked formula is constant but the size of the parallel composition of model and POSA in terms of states and transitions depends linearly on the size of the POSA.

In the course of this paper we discuss an approach to this explosion problem by distributed model-checking. We observe that in the particular POSAs constructed for LSCs many nodes can syntactically be identified to be deterministic, i.e. that the annotations of outgoing transitions are disjoint. Additionally, some of their successor nodes are dominated, i.e. each path to these nodes goes through their dominating node. We show that the POSA then can be split into (strictly smaller) POSAs whose intersection language is equal to the original POSA's language. Model-checking the smaller POSAs can be performed distributedly, yielding a speedup of the model-checking process and rendering some too resource-intensive tasks feasible.

1.1 Related Work

The employability of decomposition of the negative claim automaton, i.e. the negation of the property automaton, in order to obtain a distributed LTL model-checking procedure has been studied by [2,1]. They exploit strongly connected components in the negative claim automaton to obtain a better distribution of the state space in their distributed LTL model-checking algorithm. Our approach is different in that we don't devise a new model-checking algorithm but exploit properties of the automata used in LSC model-checking to decompose only the property. Thereby, LSC model-checking can be distributed using *any* model-checker and does not suffer from the communication overhead needed for synchronisation in [2,1]. Note that [1] independently observed that their approach is well applicable to formulae that are a chain of right-nested "Until" operators without noticing that this is exactly the structure of CSCTL [12] formulae into which POSAs can be translated.

This paper owes much to the work of Schlör [11] who studied POSAs as the basis of Symbolic Timing Diagrams, another visual formalism. He devised a translation of POSAs to LTL and obtained a lemma by which the formula of a deterministic POSA can be translated into a large conjunction with disjunctive terms indicating that another part of the formula is "in charge". Our approach follows this same idea but applies it already on the level of the automaton.

The objective of partial order reduction is to remove some concurrency from a *model* that is invisible for a particular specification. In contrast, our approach targets the specification, where all concurrency has to be preseved.

1.2 Structure

The rest of the paper is structured as follows. In Sec. 2 we introduce POSAs over signatures. Sec. 3, the main contribution of the paper, gives a decomposition procedure for deterministic and dominating POSA states with out-degree larger than 1 that yields strictly smaller automata whose intersection (or union) language is equal to the original automaton's language. Sec. 4 motivates how these results can be applied to obtain distributed LSC model-checking and why they are expected to apply to typical LSCs. Sec. 5 provides empirical results from an evaluation of our approach and Sec. 6 concludes.

2 Symbolic Automata

A Symbolic Automaton [9] is basically a Büchi Automaton accepting infinite words, namely those who have a run visiting an accepting state infinitely often. The difference is that the transitions of a Symbolic Automaton are labelled by expressions over a signature, not only by elements of an alphabet as in the general case of Büchi automata. A transition in the Symbolic Automaton can be taken if an interpretation of the predicates in the signature (together with a fixed valuation of the variables) satisfies the labelling expression of the transition. Thus, the words accepted by a Symbolic Automaton over a signature are sequences of interpretations of the predicates in the signature.

2.1 Preliminaries

Firstly, we introduce the (standard) notions of signature, structure, and valuation. The boolean expressions that can be used in an LSC and as transition annotations in Symbolic Automata are defined over such a signature.

Definition 2.1 A *signature* $\mathcal{S} = (\mathcal{V}, \mathcal{P})$ comprises a set of variables $\mathcal{V}$ and a set $\mathcal{P}$ of predicates. A tuple $\mathcal{M} = (\mathcal{U}, \mathcal{I})$, where $\mathcal{U}$ is a non-empty set of concrete values, called the *universe*, and $\mathcal{I}$ is an interpretation of the predicate symbols in $\mathcal{P}$ is called a *structure* of $\mathcal{S}$. The set of all interpretations of the predicates $\mathcal{P}$ over $\mathcal{U}$ is denoted by $\mathit{Int}_{\mathcal{U}}(\mathcal{S})$. A function σ that maps variables to universe values, i.e. $\sigma : \mathcal{V} \to \mathcal{U}$, is called *valuation* of $\mathcal{V}$ in $\mathcal{U}$. The set of all valuations of variables $\mathcal{V}$ over $\mathcal{U}$ is denoted by $\mathit{Val}_{\mathcal{U}}(\mathcal{S})$. $\diamondsuit$

Definition 2.2 Let $\mathcal{S} = (\mathcal{V}, \mathcal{P})$ be a signature. The *boolean expressions* over $\mathcal{S}$, denoted by $\mathit{Expr}_{\mathcal{S}}$, are defined by the grammar

$$\psi ::= \mathit{true} \mid p_0 \mid p(x_1, \ldots, x_n) \mid \neg\psi_1 \mid \psi_1 \wedge \psi_2$$

where $p_0 \in \mathcal{P}$ is a 0-ary predicate, $p \in \mathcal{P}$ of positive arity, and $x_1, \ldots, x_n \in \mathcal{V}$ variables. We shall use the abbreviations *false*, $\vee$, $\rightarrow$, and $\leftrightarrow$. $\diamondsuit$

The structure $\mathcal{M}$ is said to *satisfy* an expression $\psi \in \mathit{Expr}_{\mathcal{S}}$ under a valuation $\sigma \in \mathit{Val}_{\mathcal{U}}(\mathcal{S})$ iff $\mathcal{M}, \sigma \models \psi$. The "$\models$" relation is defined inductively over the structure of the formula as usual. A boolean expression $\psi \in \mathit{Expr}_{\mathcal{S}}$ is called *tautology*, denoted by $\models \psi$, iff $\mathcal{M}, \sigma \models \psi$ for each structure $\mathcal{M} = (\mathcal{U}, \mathcal{I})$ of $\mathcal{S}$ and each valuation $\sigma \in \mathit{Val}_{\mathcal{U}}(\mathcal{S})$.

The system models under consideration can readily be abstracted to a set of system runs, i.e. to sequences of predicate interpretations. Note that it is in our case not sufficient to use sequences of atomic propositions as a core LSC may use specification variables and predicates of positive arity.

Definition 2.3 Let $\mathcal{S}$ be a signature and $\mathcal{U}$ a universe. An infinite sequence $\boldsymbol{\iota} = \iota_0\, \iota_1\, \iota_2\, \ldots$ with $\iota_i \in \mathit{Int}_{\mathcal{U}}(\mathcal{S})$ for $i \in \mathbb{N}_0$ of interpretations of the predicates of $\mathcal{S}$ is called *interpretation sequence* of $\mathcal{S}$ over $\mathcal{U}$. The set of all interpretation sequences of $\mathcal{S}$ over $\mathcal{U}$ is denoted by $\overrightarrow{\mathit{Int}_{\mathcal{U}}}(\mathcal{S})$.

We denote by $\boldsymbol{\iota}(i)$ the i-th interpretation ι_i of $\boldsymbol{\iota}$ and by $\boldsymbol{\iota}/i$ the suffix $\boldsymbol{\iota}(i)\ \boldsymbol{\iota}(i+1)\ \ldots$ of $\boldsymbol{\iota}$ starting at the i-th interpretation. $\diamondsuit$

2.2 (Partially Ordered) Symbolic Automata

This section introduces the syntax and semantics of Symbolic Automata. A Symbolic Automaton is defined over a signature. Its accepted language is a set of sequences of interpretations as defined above.

Definition 2.4 Let $\mathcal{S}$ be a signature. A *Symbolic Automaton* over $\mathcal{S}$ is a tuple $\mathcal{A} = (Q, q_s, \rightsquigarrow, F)$ where Q is a finite set of *states*, $q_s \in Q$ is the *initial state*, $\rightsquigarrow \subseteq Q \times \mathit{Expr}_{\mathcal{S}} \times Q$ is the *transition relation*, and $F \subseteq Q$ is the set of *accepting states*. $\diamondsuit$

For a Symbolic Automaton $\mathcal{A} = (Q, q_s, \rightsquigarrow, F)$ over a signature $\mathcal{S}$, we define the binary relations $\rightarrow, \hat{\rightarrow} \subseteq Q \times Q$ as

$$\rightarrow := \{(q, q') \in Q \times Q \mid \exists\, \psi \in \mathit{Expr}_{\mathcal{S}} : (q, \psi, q') \in \rightsquigarrow\}, \hat{\rightarrow} := \rightarrow \setminus \{(q, q) \mid q \in Q\}.$$

The state- and transition-size of $\mathcal{A}$ is $|\mathcal{A}|_Q := |Q|$ and $|\mathcal{A}|_{\rightsquigarrow} := |\rightsquigarrow|$, resp., and the indegree and outdegree of a state $q \in Q$ is $\mathit{indeg}(q) := |\{(q', q) \in \hat{\rightarrow}\}|$ and $\mathit{outdeg}(q) := |\{(q, q') \in \hat{\rightarrow}\}|$.

An interpretation sequence $\boldsymbol{\iota}$ is accepted by a Symbolic Automaton if it has an accepting run in the automaton, i.e. a sequence of states starting at the initial state where the predicate interpretation at each position of $\boldsymbol{\iota}$ satisfies the corresponding transition annotation in the automaton.

Definition 2.5 Let $\mathcal{A} = (Q, q_s, \rightsquigarrow, F)$ be a Symbolic Automaton over a signature $\mathcal{S}$ and $\mathcal{U}$ a universe. Let $\boldsymbol{\iota} = \iota_0\, \iota_1\, \iota_2\, \ldots \in \overrightarrow{\mathit{Int}_{\mathcal{U}}}(\mathcal{S})$ be an interpretation sequence of $\mathcal{S}$ over $\mathcal{U}$ and $\sigma \in \mathit{Val}_{\mathcal{U}}(\mathcal{S})$ a valuation.

An infinite sequence $\pi = q_0\, q_1\, q_2\, \ldots$, where $q_i \in Q$ for $i \in \mathbb{N}_0$, is called a *run of* $\mathcal{A}$ *over* $\boldsymbol{\iota}$ *under* σ iff $q_0 = q_s$ and $\forall\, i \in \mathbb{N}_0\ \exists\, (q_i, \psi, q_{i+1}) \in \rightsquigarrow : (\mathcal{U}, \iota_i), \sigma \models \psi$. We use $\Pi^{\boldsymbol{\iota}}_{\sigma}(\mathcal{A})$ to denote the set of runs of $\mathcal{A}$ over $\boldsymbol{\iota}$ under σ.

By $\mathit{inf}(\pi) \subseteq Q$ we denote the set of states that occur infinitely often in π, i.e. $\mathit{inf}(\pi) := \{q \in Q \mid q = q_i \text{ for infinitely many } i \geq 0\}$. A run $\pi \in \Pi^{\boldsymbol{\iota}}_{\sigma}(\mathcal{A})$ with $\mathit{inf}(\pi) \cap F \neq \emptyset$ is called *accepting*.

The *language* accepted by a Symbolic Automaton $\mathcal{A}$ over $\mathcal{S}$ under σ is defined as $\mathcal{L}_{\sigma}(\mathcal{A}) := \{\boldsymbol{\iota} \in \overrightarrow{\mathit{Int}_{\mathcal{U}}}(\mathcal{P}) \mid \exists\, \pi \in \Pi^{\boldsymbol{\iota}}_{\sigma}(\mathcal{A}) : \pi \text{ is accepting}\}$. $\diamondsuit$

Definition 2.6 A Symbolic Automaton $\mathcal{A} = (Q, q_s, \rightsquigarrow, F)$ over a signature $\mathcal{S}$ is called *Partially Ordered Symbolic Automaton* (*POSA*) iff the reflexive, transitive closure of $\rightarrow$ is antisymmetric, i.e. if $\rightarrow^*$ is a partial order on Q. $\diamondsuit$

As already pointed out in the introduction, the Symbolic Automata obtained for LSCs comprise at most self-loops but no cycles where different states are involved, thus are POSAs.

3 POSA Decomposition

In this section, we investigate the decomposition properties of (Partially Ordered) Symbolic Automata. We give a constructive definition that yields a proper decomposition at deterministic states and we identify the dominance property as a sufficient criterion for obtaining strictly state-smaller automata.

Definition 3.1 Let $\mathcal{A} = (Q, q_s, \rightsquigarrow, F)$ be a Symbolic Automaton over signature $\mathcal{S}$. A state $q \in Q$ is called *deterministic* iff $\forall\, (q, \psi_1, q'), (q, \psi_2, q'') \in \rightsquigarrow$: $q' \neq q'' \implies\ \models \neg(\psi_1 \wedge \psi_2)$. A state $q \in Q$ is called *reaching-deterministic* iff all states $q' \in Q$ with $(q', q) \in \rightarrow^*$ are deterministic. $\diamondsuit$

For distributed model-checking, two kinds of decomposition properties are of interest. Corresponding to the chart mode *existential*, one wants to check for a given system, i.e. a set of interpretation sequences, whether at least one interpretation sequence is accepted by the automaton. In this case, the *union* of the languages of the automata in the decomposition should be equal to the original language. Then the query can be distributed since there is an accepting run of the original automaton if one automaton from the decomposition accepts an interpretation sequence. Corresponding to the chart mode *universal*, one wants to check whether the *whole set of interpretation sequences* is in the language. If the *intersection* language of the automata in the decomposition is equal to the original language, then we can distributedly check whether all automata in the decomposition accept all interpretation sequences.

Definition 3.2 Let $\mathcal{A}$ be a Symbolic Automaton over a signature $\mathcal{S}$. The set $\{\mathcal{A}_1, \ldots, \mathcal{A}_N\}$, $N > 0$, of Symbolic Automata over $\mathcal{S}$ is called

(i) an ***A**-decomposition* of $\mathcal{A}$ iff $\mathcal{L}(\mathcal{A}) = \bigcap_{1 \leq i \leq N} \mathcal{L}(\mathcal{A}_i)$ and

(ii) an ***E**-decomposition* of $\mathcal{A}$ iff $\mathcal{L}(\mathcal{A}) = \bigcup_{1 \leq i \leq N} \mathcal{L}(\mathcal{A}_i)$. $\diamondsuit$

The following definition constructs the universal q-decomposition at a reaching-deterministic state q. In each automaton in the decomposition, all but one outgoing transitions of q are removed and a new transition is added that is labelled with the disjunction of the removed transition expressions and leads to an accepting sink state. See Fig. 2 for a small example.

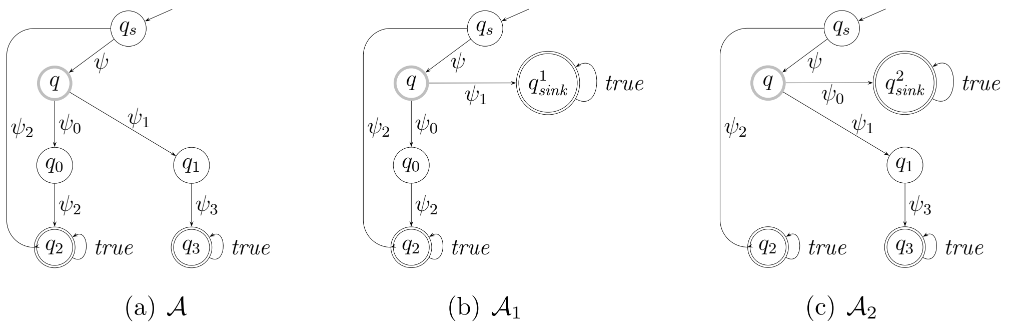


Fig. 2. Universal q-decomposition of $\mathcal{A}$ into $\{\mathcal{A}_1, \mathcal{A}_2\}$

The following lemma establishes the desired property that a universal q-decomposition is an **A**-decomposition.

The procedure is inspired by a lemma of Schlör [11] which turns a disjunction $\bigvee_i \psi_i$ of disjoint expressions into an equivalent conjunction $\bigwedge_i(\psi_i \vee \phi_i)$ where ϕ_i is the disjunction of all expressions ψ_j with $j \neq i$ (see below).

Definition 3.3 Let $\mathcal{A} = (Q, q_s, \rightsquigarrow, F)$ be a *POSA* over signature $\mathcal{S}$ and $q \in Q$ a reaching-deterministic state with $1 < \mathit{outdeg}(q) =: N$.
Let $O = \{(q, \psi_1, \hat{q}_1), \ldots, (q, \psi_N, \hat{q}_N) \mid (q, \hat{q}_i) \in \hat{\rightarrow}, 1 \leq i \leq N\} \subseteq \rightsquigarrow$ be the set of outgoing transitions of q, and $\rightsquigarrow_O := \rightsquigarrow \setminus O$. We define $\rightarrow_O$ analogously to the definition of $\rightarrow$ for $\rightsquigarrow$.

The set $\{\mathcal{A}_1, \ldots, \mathcal{A}_N\}$ of Symbolic Automata $\mathcal{A}_i = (Q_i, q_s, \rightsquigarrow_i, F_i)$ with

$$\begin{aligned}
Q_i &:= \{q' \in Q \mid (q_s, q') \in \rightarrow^*_O \vee (\hat{q}_i, q') \in \rightarrow^*_O\} \cup \{q^i_{sink}\}\\
\rightsquigarrow_i &:= \{(q', \psi, q'') \in \rightsquigarrow_O \mid q', q'' \in Q_i\} \cup \{(q, \psi_i, \hat{q}_i), (q, \phi_i, q^i_{sink}), (q^i_{sink}, \mathit{true}, q^i_{sink})\}\\
F_i &:= \{q^i_{sink}\} \cup (F \cap Q_i)
\end{aligned}$$

where $q^i_{sink} \notin Q$ are fresh automaton states and $\phi_i := \bigvee_{j=1\ldots N,\ j\neq i} \psi_j$, is called the *universal q-decomposition of* $\mathcal{A}$. $\diamondsuit$

Lemma 3.4 (POSA Decomposition) *Let $\mathcal{A} = (Q, q_s, \rightsquigarrow, F)$ be a POSA over signature $\mathcal{S}$ and $q \in Q$ a reaching-deterministic state with $1 < \mathit{outdeg}(q) =:$ N. The universal q-decomposition $\{\mathcal{A}_1, \ldots, \mathcal{A}_N\}$ is an* ***A****-decomposition of $\mathcal{A}$.*

Proof. "$\bigcap_{1\leq i\leq N} \mathcal{L}_\sigma(\mathcal{A}_i) \subseteq \mathcal{L}_\sigma(\mathcal{A})$": Let $\boldsymbol{\iota} \in \bigcap_{1\leq i\leq N} \mathcal{L}_\sigma(\mathcal{A}_i)$, i.e. there exist N accepting runs $\pi_i = q^i_0\ q^i_1 \ldots \in \Pi^{\boldsymbol{\iota}}_\sigma(\mathcal{A}_i)$ for all $1 \leq i \leq N$. We distinguish two cases. Firstly, there is a run π_i that visits q and leaves it again, and secondly the converse, that each run π_i never visits q or never leaves it.

Case 1: $\exists\, i \in \{1, \ldots, N\} : \pi_i = q^i_0\ q^i_1 \ldots q^i_{k-1}\ q^i_k \ldots \in \Pi^{\boldsymbol{\iota}}_\sigma(\mathcal{A}_i), q^i_{k-1} = q, q^i_k \neq q$.
The POSA property ensures that $q^i_l \neq q$ for all $l \geq k$. As q is reaching-deterministic, we have that $q^i_l = q^j_l$ for all $0 \leq l < k$ and $1 \leq j \leq N$. As

q is deterministic, there exists $i \in \{1, \ldots, N\}$ such that $q^i_k \neq q^i_{sink}$, i.e. an automaton $\mathcal{A}_i$ with $q^i_k \in Q$. As in case 1, we have by $\rightsquigarrow_i \setminus \{(q', \psi, q^i_{sink}) \mid q' \in Q\} \subseteq \rightsquigarrow$ that $\pi_i \in \Pi^{\boldsymbol{\iota}}_\sigma(\mathcal{A})$, thus $\boldsymbol{\iota} \in \mathcal{L}_\sigma(\mathcal{A})$.

Case 2: for all $1 \leq i \leq N$ and for all $k \in \mathbb{N}_0$, $q^i_k = q \implies q^i_{k+1} = q$.
As q^i_{sink} is only reachable by leaving q, this implies $q^i_{sink} \neq q^i_k$ for all $k \in \mathbb{N}_0$, thus we have by $\rightsquigarrow_i \setminus \{(q', \psi, q^i_{sink}) \mid q' \in Q_i\} \subseteq \rightsquigarrow$ that $\pi_i \in \Pi^{\boldsymbol{\iota}}_\sigma(\mathcal{A})$ for all $1 \leq i \leq N$, thus $\boldsymbol{\iota} \in \mathcal{L}_\sigma(\mathcal{A})$.

"$\mathcal{L}_\sigma(\mathcal{A}) \subseteq \bigcap_{1 \leq i \leq N} \mathcal{L}_\sigma(\mathcal{A}_i)$": Let $\boldsymbol{\iota} \in \mathcal{L}_\sigma(\mathcal{A})$, i.e. there exist an accepting run $\pi = q_0\ q_1 \ldots \in \Pi^{\boldsymbol{\iota}}_\sigma(\mathcal{A})$. We distinguish the same two cases as above.

Case 1: for all $k \in \mathbb{N}_0$, $q_k = q \implies q_{k+1} = q$.
Then $\pi \in \Pi^{\boldsymbol{\iota}}_\sigma(\mathcal{A}_i)$ for all $1 \leq i \leq N$ by construction of $\mathcal{A}_i$, i.e. $\boldsymbol{\iota} \in \bigcap_{1 \leq i \leq N} \mathcal{L}(\mathcal{A}_i)$.

Case 2: $\pi = q_0\ q_1 \ldots q_{k-1}\ q_k \ldots \in \Pi^{\boldsymbol{\iota}}_\sigma(\mathcal{A}), q_{k-1} = q, q_k \neq q$.
The POSA property ensures that $q_l \neq q$ for all $l \geq k$. As q is reaching-deterministic, we can construct N sequences $q^i_0\ q^i_1 \ldots q^i_{k-1}$ with $q^i_l \in Q_i$ and $q^i_l = q_l$ for all $0 \leq l < k$ and $1 \leq i \leq N$. Since $(q, q_k) \in \hat{\rightarrow}$, there exists $i \in \{1, \ldots, N\}$ s.t. $q_k = \hat{q}_i$ and $(\mathcal{U}, \boldsymbol{\iota}(k)), \sigma \models \psi_i$. Thus the automaton $\mathcal{A}_i$ accepts the run just as $\mathcal{A}$ does. Regarding $\mathcal{A}_j$, $1 \leq j \neq i \leq N$, we have $(\mathcal{U}, \boldsymbol{\iota}(k)), \sigma \models \phi_j$ by construction of ϕ_j. As reaching q^j_{sink} has the effect that every interpretation sequence suffix is accepted due to the self-loop, we can conclude that $\pi \in \Pi^{\boldsymbol{\iota}}_\sigma(\mathcal{A}_j)$ for all $\mathcal{A}_j$. Thus $\boldsymbol{\iota} \in \bigcap_{1 \leq i \leq N} \mathcal{L}(\mathcal{A}_i)$. $\square$

The universal q-decomposition of a Symbolic Automaton as defined in 3.3 yields *outdeg*(q) many automata, each of them comprising at most one state and one transition more than $\mathcal{A}$, i.e. $|\mathcal{A}_i|_Q \leq |\mathcal{A}|_Q + 1$ and $|\mathcal{A}_i|_\rightsquigarrow \leq |\mathcal{A}|_\rightsquigarrow + 1$.

The decomposed automata are strictly state-smaller if q and its successors $\hat{q}_i$ dominate their reachable states and there is at least one reachable state below one of the successors $\hat{q}_i$. A state q dominates a state q', written $q \vdash q'$, if all runs from the initial state q_s to q' visit q. Intuitively, the q-decomposition procedure then eliminates complete subtrees of the original automaton in each decomposed automaton.

Lemma 3.5 (Dominant POSA Decomposition) *Let $\mathcal{A} = (Q, q_s, \rightsquigarrow, F)$ be a POSA over signature $\mathcal{S}$. Let $q \in Q$ and $O \subseteq \rightsquigarrow$ as in Def. 3.3. The universal q-decomposition $\{\mathcal{A}_1, \ldots, \mathcal{A}_N\}$ yields strictly state-smaller and at most as transition-large automata, i.e. $|\mathcal{A}_i|_Q < |\mathcal{A}|_Q$ and $|\mathcal{A}_i|_\rightsquigarrow \leq |\mathcal{A}|_\rightsquigarrow$ for all $1 \leq i \leq N$, if $\forall\, \hat{q} \in \{\hat{q}_i, \ldots, \hat{q}_N\} : q \vdash \hat{q} \wedge (\exists\, q' \in Q : (\hat{q}, q') \in \hat{\rightarrow} \wedge \hat{q} \vdash q')$.*

Proof. *By construction, each Q_i doesn't contain $\hat{q}_j$, $j \neq i$, if it is dominated by q. Also it doesn't contain the states dominated by $\hat{q}_j$, $j \neq i$, i.e. at least two states from Q are not in Q_i. Adding the sink state to Q_i yields the desired* *property. With the same argument, at least two transitions from $\rightsquigarrow$ are not in $\rightsquigarrow_i$. Adding the two transitions into the sink state concludes the proof.* $\square$

By skipping the addition of the designated sink state in Def. 3.3, we obtain the *existential* q-decomposition. This has the effect that for a given sequence $\boldsymbol{\iota}$ all automata in the decomposition who do not have a matching outgoing transition of q get stuck and do not accept $\boldsymbol{\iota}$. We state omitting the proof that this yields an **E**-decomposition. Regarding the size, we have a result similar to Lemma 3.5.

Note that the same decomposition procedure can be applied to completely deterministic, not necessarily partially ordered, Symbolic Automata. We only study the POSA case as the Symbolic Automata obtained for LSCs are POSAs but may exhibit non-determinism.

4 Distributed LSC Model-checking

The semantics of core LSCs is explained in terms of Symbolic Automata in [9]. There is an *unwinding procedure* which translates each LSC body L into a Symbolic Automaton $\mathcal{A}_L$. The semantics of the full LSC, i.e. with activation condition, quantification (one of universal or existential), and activation mode (one of initial or invariant[6]) is then defined using $\mathcal{A}_L$ as follows.

Definition 4.1 Let L be a core LSC over signature $\mathcal{S}$ with activation condition $ac \in \mathit{Expr}_{\mathcal{S}}$, activation mode $am \in \{\mathit{initial}, \mathit{invariant}\}$, and quantification $\mathit{quant} \in \{\mathit{existential}, \mathit{universal}\}$. Let $\mathcal{U}$ be a universe and $\mathcal{A}_L$ the Symbolic Automaton constructed according to [9]. A set of interpretation sequences $\boldsymbol{I} \subseteq \overrightarrow{\mathit{Int}}_{\mathcal{U}}(\mathcal{S})$ is said to *satisfy* the LSC L, denoted $\boldsymbol{I} \models_{\scriptscriptstyle LSC} L$, iff

$$\begin{aligned}\exists\, \boldsymbol{\iota} \in \boldsymbol{I}\; \exists \sigma \in \mathit{Val}_{\mathcal{U}}(\mathcal{S}) : am = \mathit{initial} \wedge ((\mathcal{U},\, \boldsymbol{\iota}(0)), \sigma \models ac \wedge \boldsymbol{\iota}/0 \in \mathcal{L}_\sigma(\mathcal{A}_L))\\ \vee am = \mathit{invariant} \wedge (\exists\, k \in \mathbb{N}_0 : (\mathcal{U},\, \boldsymbol{\iota}(k)), \sigma \models ac \wedge \boldsymbol{\iota}/k \in \mathcal{L}_\sigma(\mathcal{A}_L))\end{aligned}$$

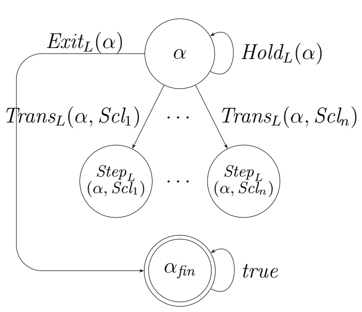
if $\mathit{quant} = \mathit{existential}$ and, if $\mathit{quant} = \mathit{universal}$,

$$\begin{aligned}\forall\, \boldsymbol{\iota} \in \boldsymbol{I}\; \forall \sigma \in \mathit{Val}_{\mathcal{U}}(\mathcal{S}) : am = \mathit{initial} \wedge ((\mathcal{U},\, \boldsymbol{\iota}(0)), \sigma \models ac \implies \boldsymbol{\iota}/0 \in \mathcal{L}_\sigma(\mathcal{A}_L))\\ \vee am = \mathit{invariant} \wedge (\forall\, k \in \mathbb{N}_0 : (\mathcal{U},\, \boldsymbol{\iota}(k)), \sigma \models ac \implies \boldsymbol{\iota}/k \in \mathcal{L}_\sigma(\mathcal{A}_L))\,.\end{aligned}$$

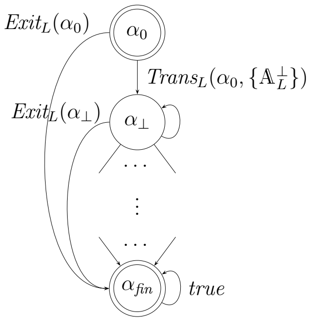
The *accepted language* of L is $\mathcal{L}(L) := \{\boldsymbol{I} \subseteq \overrightarrow{\mathit{Int}}_{\mathcal{U}}(\mathcal{P}) \mid \boldsymbol{I} \models_{\scriptscriptstyle LSC} L\}$. $\diamondsuit$

In [12] we have established that the Symbolic Automata obtained for LSCs are actually POSAs. According to Sec. 3, the prerequisite for decomposition

[6] The third mode, iterative, lies out of the scope of this paper.



(a) Outgoing transitions.



(b) Structure of $\mathcal{A}_L$.

Fig. 3. The LSC body automaton

is the presence of deterministic states. To see that most states in an LSC's POSA are deterministic consider Fig. 3(a) and 3(b). The former shows the general form of a state corresponding to a cut α (cf. Sec. 1 for a brief introduction of cuts). There is always a self-loop annotated with $Hold_L(\alpha)$, the *hold condition* of the cut. It is the negation of the annotations of all other outgoing transitions.

Secondly, there is always a transition to the final node annotated with $Exit_L(\alpha)$, the *exit condition*. The annotation comprises the expressions from cold conditions and cold local invariants active at α and the negation of message predicates from the other outgoing transitions. If no legal exit is possible at a particular cut, the annotation is '*false*' and the transition is omitted in visual representations of the automaton.

Thirdly, there are *regular transitions* which advance the cut by all combinations of *enabled simclasses*, denoted by $Scl_1, \ldots, Scl_n$ in Fig. 3(a), where a simclass is a set of elements that have to be observed simultaneously. A simclass is enabled by a cut if its elements are located directly below the cut.

Each regular transition annotation explicitly names the combination of messages it applies to. Consider for example the POSA in Fig. 1(b) on page 3. The regular transitions starting at q_0 name explicitly to which combination of a_1 and a_2 they apply, referencing a_1, a_2 both positively and negated. Consequently, if all enabled simclasses comprise at least one message s.t. the expressions characterising the relevant sends and receives are disjoint, then the transitions annotations are disjoint, too. Whether send or receive expressions are disjoint can easily be decided since they are, by definition [12], simple predicates without logical connectives. Note that this directly provides a *sufficient* criterion for determinism that can easily be evaluated within the unwinding algorithm [9] that translates an LSC body into the POSA.

Non-deterministic states are, for example, introduced if so called *floating* *conditions* are used, i.e. conditions that are not co-located with messages. A q-decomposition can only be performed until the unwinding algorithm encounters for the first time a non-deterministic state. All predecessors of this state are reaching-deterministic, all successor states are not. In practice, LSCs are typically deterministic to a large amount and hence are decomposable (cf. [5]; Klose [9] explicitly recommends not to use LSCs with floating conditions in practice since their meaning is highly counter-intuitive).

Additionally we can see that a majority of states have dominated successor states. They correspond to locations within the LSC where the partial order is not relaxed by e.g. coregions. Elements at these locations have to be observed before any subsequent elements are enabled, and the corresponding POSA reflects this order in terms of dominating states.

Now let L be a universal LSC, $\mathcal{A}_L = (Q, q_s, \rightsquigarrow, F)$ its POSA, and $Q_{dec} \subseteq Q$ a set of states annotated to be reaching-deterministic and of out-degree greater than one. Each $\{\mathcal{A}_1, \ldots, \mathcal{A}_N\}$ obtained by applying q-decomposition to states $q \in Q_{dec}$ is an $\mathbf{A}$-decomposition by Lemma 3.4. Formally, the distribution of model-checking L is justified as follows:

$$\begin{aligned}
&\boldsymbol{I} \models_{LSC} L \\
\iff\ & \forall\, \boldsymbol{\iota} \in \boldsymbol{I}, \sigma \in Val_{\mathcal{U}}(\mathcal{S}), k \in \mathbb{N}_0 : (\mathcal{U},\, \boldsymbol{\iota}(k)), \sigma \models ac \implies\ \boldsymbol{\iota}/k \in \mathcal{L}_\sigma(\mathcal{A}_L) \\
\iff\ & \forall\, \boldsymbol{\iota} \in \boldsymbol{I}, \sigma \in Val_{\mathcal{U}}(\mathcal{S}), k \in \mathbb{N}_0 : (\mathcal{U},\, \boldsymbol{\iota}(k)), \sigma \models ac \implies \bigwedge_{1 \le i \le N} \boldsymbol{\iota}/k \in \mathcal{L}_\sigma(\mathcal{A}_i) \\
\iff & \bigwedge_{1 \le i \le N} \forall\, \boldsymbol{\iota} \in \boldsymbol{I}, \sigma \in Val_{\mathcal{U}}(\mathcal{S}), k \in \mathbb{N}_0 : (\mathcal{U},\, \boldsymbol{\iota}(k)), \sigma \models ac \implies\ \boldsymbol{\iota}/k \in \mathcal{L}_\sigma(\mathcal{A}_i)
\end{aligned}$$

The last formula corresponds to N independent model-checking tasks. For existential LSCs, applying existential q-decomposition yields an $\mathbf{E}$-decomposition and a disjunction of independent (existential) model-checking tasks.

As we can apply q-decomposition to any state $q \in Q_{dec}$, we maximally obtain a decomposition comprising as many smaller POSAs as there are outgoing transitions on all states in Q_{dec}; in case $Q_{dec} = Q$, the maximal number corresponds to the number of pathes permitted by the scenario. For practical distributed model-checking with a given number of N processing nodes (CPUs) one will try to either obtain N POSAs of roughly the same size by choosing the decomposition states accordingly or at least distribute the obtained POSAs to the nodes such that the workload for each node is similar.

There are two ways for model-checking LSCs (cf. Sec.1). Firstly, the parallel composition of the system and an observer automaton corresponding to the POSA can be checked for globally finally reaching an accepting state. Since the model-checking time is (worst-case) linear in the number of states and the number of transitions in the finally checked transition system, in our case the parallel composition of model and observer, we expect a corresponding effect of decomposition. Namely, if the POSA is split into strictly smaller automata, checking the parallel composition of the model and one of the smaller automata at a time should be faster. There is no expected speedup in the pure model-checking time if the automaton encodes a *safety property*. That is, in the special case where the automaton doesn't make use of Büchi acceptance because a dedicated state can be identified that is entered iff the specification is violated. Then reachability of this violation state can principally be annotated to the original transition system whose size remains unchanged. But decomposition of the automaton should affect the timed needed to establish the annotation.

Secondly, a normalised POSA, i.e. a POSA with at most one transition between two states, can be translated into a CTL formula[7] following the recursive definition $\phi_q = \psi(q,q)\,\mathrm{U} \bigvee_{q \hat{\to} q'} \psi(q,q') \wedge \mathrm{X}\,\phi_{q'}$ from [12] where $\psi(q,q')$ is the annotation of the transition from q to q' and 'U' denotes CTL's weak until or until operator depending on whether the node q is accepting or not. In case of LSCs, $\psi(q,q)$ is a hold condition, and $\psi(q,q')$ is an exit condition or a transition condition as introduced above.

Without a proof we note that the size of the formula ϕ_{q_s} for a given $\mathcal{A}_L$, i.e. the number of nodes in the parse tree not counting leaf nodes (transition annotations), computes as

$$|\phi_{q_s}| = \textstyle\sum_{q \in Q} \mathit{in\text{-}pathes}(q) \cdot 3 \cdot \mathit{outdeg}(q) + \sum_{q \to q'} \mathit{in\text{-}pathes}(q)$$

where $\mathit{in\text{-}pathes}(q)$ is the number of pathes from q_s to q (with $\mathit{in\text{-}pathes}(q_s) = 1$). Intuitively, the first sum accounts for the temporal and logical operators at each state q and the second sum counts the transition annotations. The annotation of a transition $t \in \rightsquigarrow$ occurs as many times as there are pathes to the source state of t. Note that the size of the formula depends directly on the number of transitions in a POSA, but the number of in-pathes is of course determined by the combination of states and transitions. For distributed model-checking we generate the (smaller) formulae of the POSAs in the decomposition and check them independently.

In both cases, the sum of all runtimes will typically be larger than the runtime of direct, sequential model-checking, provided direct model-checking is possible within the given resources. The runtime of parallel model-checking including the generation of the decomposition can be expected to be smaller since except for the initial distribution of the tasks and the final collection

[7] the CTL formula of a universal LSC lies in the intersection of CTL and LTL [10]

LSC		original	distance 1	distance 2
a3w		409/27/98	(7) 107/20/64	(49) 43/15/47
	CTL	202.9s 198.7s	67.0s 62.4s	31.5s 27.0s
	obs	27.0s 21.6s	25.8s 21.4s	26.4s 22.0s
a4nw		5,625/31/130	(15) 989/25/99	(65) 245/22/92
	CTL	7528.2s 7524.1s	1625.8s 1621.4s	448.6s 444.1s
	obs	13.7s 7.8s	12.5s 7.4s	27.6s 23.4s
a4		$7.39 \cdot 10^6$/146/1023	(15) $1.34 \cdot 10^6$/121/815	-
	obs	210.2s 50.9s	130.0s 46.4s	- -

Fig. 4. **Experiments.** The first column denotes the checked LSC. 'a3w' is an LSC with two pairs of three concurrent synchronous messages, 'a4nw' has two pairs and 'a4' has three pairs of four concurrent synchronous messages. For all rows, the third column refers to the original POSA and the second and third row to the results of the q-decomposition applied to all nodes with distance 1 and 2 from the initial node.
In the LSC name's row, the third column gives the number of pathes in the scenario and the numbers of states and transitions in the original POSA, separated by slashes ('/'). The fourth and last columns give the numbers of pathes, states, and transitions in the largest POSA from the decomposition. The number in parentheses is the size of the decomposition, i.e. the number of obtained smaller POSAs.
The 'CTL' and 'obs' rows give the runtime of a complete model-checking procedure including all pre- and post-processing. In the fourth and last columns this is the time for the largest POSA. The rows directly below give model-checking time only.
The 'CTL' rows refer to the formula-based and the 'obs' rows to the observer-based approach. For 'a4' there is no 'CTL' row since the formula-based approach does not terminate within 3 hours. Distance 2 has not been performed since the naive decomposition yields over 200 POSAs.

of the results, there is no communication overhead. The generation of the decomposition itself is an easy syntactical task.

We conjecture that LSCs with pre-charts [9] can be decomposed because by definition each path of the pre-chart implies the main-chart thus it can be verified by checking that each decomposition of the pre-chart implies the main-chart. In contrast, assumption LSCs [9] can generally not be decomposed, since all pathes of an assumption are significant at once.

5 Empirical Evaluation

In order to obtain experimental results we have prototypically implemented the algorithm from Def. 3.3. The decomposition nodes are heuristically chosen by the distance from the initial node, i.e. we currently don't try to balance the POSA sizes. The numbers shown in Fig. 4 have been obtained using the LSC verification toolchain of [9] that employs the VIS [3] as model-checking engine. As the model to be checked, we chose an existing model of reasonable complexity and added specific behaviour s.t. the checked LSCs are satisfied. The same model has been used for all runs. The three LSCs referred to in Fig. 4 are of moderate concurrency, specifying systems of three or four modules communicating concurrently using 12 messages in the largest LSC 'a4'.

For small decompositions, the measurements for the formula-based way support the expected linear influence of the POSA (and thus formula) size to the runtime. For example, for 'a3w', checking the complete POSA takes 202.9s while checking a POSA of roughly 1/4 the size takes 67.0s. Since there are three large and four small POSAs, the whole decomposition could be checked using 4 processing nodes within 67.0s, thus it is worthwhile to devise more sophisticated heuristics that would, in the optimal case, yield 4 POSAs of similar size for this example.

It is not completely understood why the expected effect is hardly visible for the observer based way. For example, in the last row the model-checking time only goes down from about 50s to about 46s from the original to the split automaton, although the smaller automaton is clearly smaller. The used LSCs have a completely deterministic automaton, but they are clearly not of the special kind discussed in Sec. 4, i.e. they don't reduce to a safety property. Experience shows that the observer way is not generally faster than the formula way; it is to be investigated whether it is generally better suited for the kind of LSCs considered in the experiments. To this end, it has to be taken into account that the runtime of highly optimising model-checkers like the VIS [3] we used is very discontinuous, that is, minimal changes to the model or the automaton cause the underlying BDDs to be differently ordered and to vary widely in size. And the size of a BDD is only a weak indication for the expected model-checking time.

6 Conclusion and Further Work

We have devised and evaluated an effective procedure for distributed LSC model-checking based on a decomposition of the POSAs that provide the semantical foundation of LSCs. The POSAs in the decomposition can still be checked by the formula and the observer based approach employing standard model-checking engines. Further work comprises the development of better heuristics to obtain decompositions of similarly sized POSAs and an investigation of the deviating times for the formula and observer way.

Acknowledgement

The authors want to express their gratitude to Hartmut Wittke for clarifying discussions on the intricacies of Symbolic Automata and the anonymous referees for helpful comments.

References

[1] Barnat, J., *How to distribute LTL model-checking using decomposition of negative claim automaton*, in: *Stud. Research Forum Proc.*, Milovy, Czech Rep., 2002.

[2] Barnat, J., "Distributed Memory LTL Model Checking," Ph.D. thesis, Faculty of Informatics, Masaryk University Brno (2004).

[3] Brayton, R. K. et al., *VIS: A system for verification and synthesis.*, in: R. Alur and T. A. Henzinger, editors, *CAV*, LNCS **1102** (1996), pp. 428–432.

[4] Damm, W. and D. Harel, *LSCs: Breathing Life into Message Sequence Charts*, Formal Methods in System Design **19** (2001), pp. 45–80.

[5] Damm, W. and J. Klose, *Verification of a radio-based signaling system using the Statemate Verification Environment*, Formal Methods in System Design **19** (2000), pp. 121–141.

[6] Grégoire, B., "Automata Oriented Program Verification," Master's thesis, Facultés Universitaires Notre-Dame de la Paix, Namur (2002).

[7] Harel, D. and R. Marelly, "Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine," Springer-Verlag, 2003.

[8] ITU-T, "Rec. Z.120: Message Sequence Chart (MSC)," ITU-T, Geneva, 1999.

[9] Klose, J., "Live Sequence Charts: A Graphical Formalism for the Specification of Communication Behavior," Ph.D. thesis, C. v. O. Universität Oldenburg (2003).

[10] Kugler, H., D. Harel, A. Pnueli, Y. Lu and Y. Bontemps, *Temporal logic for scenario-based specifications*, in: N. Halbwachs and L. D. Zuck, editors, *TACAS*, LNCS **3440** (2005), pp. 445–460.

[11] Schlör, R. C., "Symbolic Timing Diagrams: A Visual Formalism for Model Verification," Ph.D. thesis, C. v. O. Universität Oldenburg (2000).

[12] Toben, T. and B. Westphal, *On the expressive power of Live Sequence Charts*, in: *Proceedings of the SofSem 2006 Poster Session* (2006), to appear.