

# On the Expressive Power of Live Sequence Charts<sup>\*</sup>

Tobe Toben and Bernd Westphal

Carl von Ossietzky Universität Oldenburg, Oldenburg, Germany,  
{toben,westphal}@informatik.uni-oldenburg.de

**Abstract.** The Live Sequence Charts (LSC) language is an extension of the well-known Message Sequence Charts by means to specify liveness and to distinguish possible runs of a system from protocols that all runs should adhere to. This paper studies the expressive power of the automaton-based LSC semantics [11] in terms of temporal logic. The main result is that bonded core LSCs have an exact characterisation in terms of DCCTL, a proper sublanguage of first-order prenex CTL\*. That is, for each bonded core LSC specification there is an equivalent DCCTL formula and vice versa. Both directions of the proof are constructive.

## 1 Introduction and Related Work

The well-known Message Sequence Charts (MSC) language [10] is a visual formalism used to specify *scenarios*, i.e. sequences of communication between modules of a distributed system. Live Sequence Charts (LSCs) have been introduced in [6] in order to overcome the serious lack of expressive power of MSCs. To require liveness in a scenario, to provide so-called legal exits, and to distinguish exemplary from necessary behaviour, LSCs introduce the distinction between possible and mandatory locations, elements, and whole charts. In addition, there are means to specify the activation, i.e. the point in time from which on the scenario should be observed. Scenario-based approaches in general [18, 1] and LSCs in particular [13, 5, 4, 3, 2] have shown adequate for the intended purpose.

From the original proposal of Damm and Harel [6], two different dialects emerged that are specifically tailored for the usage of a given LSC specification. On the one hand, the PlayIn/PlayOut [9] approach employs LSCs for specifying and executing behavioural requirements of reactive systems. Given a prototypical GUI of a system and a set of LSCs, the engine is able to “play out” the LSCs, i.e. when the GUI is operated the play-engine reacts according to the specification. On the other hand, Klose [11] introduces an LSC dialect that is adopted to the more classical usage of specification languages, namely to complement the model-based development of the intra-object behaviour of a system using, e.g.,

---

<sup>\*</sup> This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Centre SFB/TR 14 AVACS and within project USE (DA 206/7-3) as part of the priority program SPP 1064.

Statestate state-charts or UML state-machines. Whether the intra-object description adheres to the LSC specification can then automatically be established by model-checking as demonstrated in [2, 15]. In the following, we only refer to the LSCs of [11] if not otherwise specified. To recall the intuition of LSCs, we briefly discuss the LSC given in Fig. 1(a). Due to space restrictions the reader is referred to [6, 11, 9] for a more thorough introduction.

LSCs are all about communication between modules of a distributed system. Each module is represented by an *instance line* on which arrows representing communication can begin or end, possibly restricted by conditions. In Fig. 1(a), we have the four modules  $Inst_1, \dots, Inst_4$ . The places where LSC elements like messages and conditions are connected to the instance line are called *locations* and partition the instance lines into segments. A solid segment of an instance line indicates that the element(s) at the end of the segment must finally be observed in order to satisfy the LSC, i.e. progress is enforced. The location at the beginning of the segment is called *hot*. For example, Fig. 1(a) requires that the *instantaneous messages*  $msg_1$  and  $msg_2$  are *finally* observed in order to satisfy  $L$  since the topmost segment of  $Inst_2$  is solid. A dashed segment in contrast doesn't enforce progress, the location at the beginning of it is called *cold*. For example, a system that behaves according to  $L$  up to message  $msg_5$  but never sends  $msg_5$  satisfies  $L$ . In contrast to MSCs are conditions semantically relevant in LSCs. They can be *possible* or *mandatory*, the former indicated by a dashed and the latter by a solid outline. A possible condition like  $c_1$  in Fig. 1(a) is a so-called *legal exit*, that is, if  $c_1$  *doesn't* hold at the point in time when (the co-located) message  $msg_5$  is observed, then  $L$  is immediately satisfied. If  $c_1$  holds, then  $L$  is only satisfied by runs where  $msg_6$  is finally observed. If a mandatory condition doesn't hold, the LSC is violated. Local invariants [11] extend conditions from single points in time to the span between a beginning and end point, both *inclusive* or *exclusive*. In Fig. 1(a), the local invariant  $c_2$  begins inclusively, i.e.  $c_2$  is required to hold already in the snapshot where the LSC is activated, and ends exclusively, i.e. has to hold up to but not including the point in time where message  $msg_4$  occurs. Elements of the LSC have to be observed as ordered on the instance lines from top to bottom unless this order is explicitly released using a *coregion*. In Fig. 1(a), the sending of  $msg_3$  and the reception of  $msg_4$  are in a coregion on  $Inst_1$  as indicated by the dotted line in parallel to the instance line, thus they may occur in any order and even simultaneously.

In addition to the *main-chart* discussed up to now, an LSC comprises a header and a (possibly empty) pre-chart. The header of an LSC gives the name, an *activation condition* that determines the set of snapshots within a system run to which the LSC applies, and an *activation mode* that further restricts the activations. An *invariant* LSC is considered in all snapshots where the activation condition holds, an *initial* LSC only in the first one. A solid frame around the LSC body as in Fig. 1(a) indicates the LSC to be *universal*, i.e. it is satisfied by a system iff *each* suffix of any system run starting with a snapshot satisfying the activation prerequisites adheres to the scenario. With a dashed frame, the LSC is *existential* and satisfied iff there *is* a suffix of a system run satisfying the acti-

vation prerequisites and adhering to the scenario. Note that we actually restrict the discussion of LSCs to *core* LSCs. They differ from general LSCs [11] in that they don't have a pre-chart, don't specify quantitative timing restrictions, and only use mandatory messages. Furthermore we focus on the weak interpretation of LSCs that doesn't exclude duplicate occurrences of messages in contrast to the strict interpretation.

The expressive power of LSCs in terms of temporal logic, i.e. a translation from LSCs to temporal logic, has independently been studied in [14]. We study the LSC dialect of [11] and for the first time consider the opposite direction for the sublanguage of bonded LSCs, i.e. discuss the translation of temporal logic formulae back to LSCs. Thereby we establish the equivalence between bonded LSC specifications and a fragment of first-order CTL\*. Given our work, both dialects of LSCs can be formally compared once the full version of [14] appears. There is a vast amount of literature on the general relation between automata and temporal logic [17]. The particular relation between *deterministic partial-order* Symbolic Automata has been studied by [16] in the context of another visual formalism, Symbolic Timing Diagrams, but without considering the translation back, from logic to the visual formalism. This result is a building block of our embedding of LSCs into temporal logic as Symbolic Automata are the basis of the LSC semantics [11].

The paper is structured as follows. In Sec. 2 and 3 we provide a new closed and concise formalisation of the syntax and semantics of LSCs that is equivalent to the algorithmic formalisation of [11]. The algorithmic description of the underlying Symbolic Automaton significantly hinders formal reasoning about LSCs as in this article. Section 4 presents the main contributions, the translation from general core LSCs to first-order prenex CTL\* and the translation from a proper sublanguage thereof back to bonded LSCs specifications. Section 5 concludes.

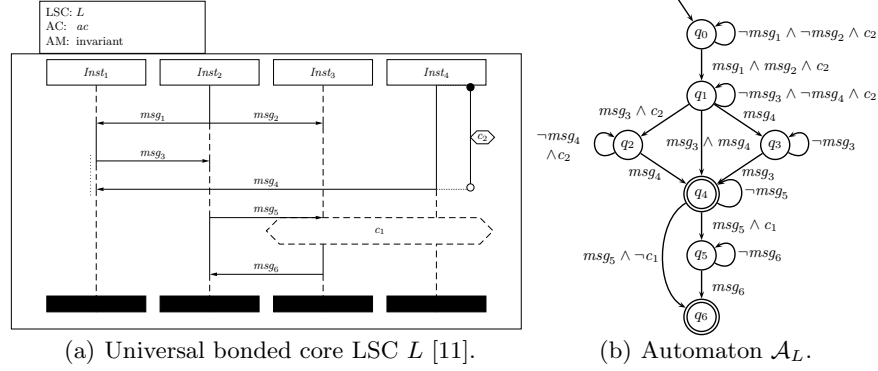
## 2 Core Life Sequence Charts

Annotations of messages, conditions, and local invariants in an LSC are boolean expressions, so we define core LSCs over a signature  $\mathcal{S} = (\mathcal{V}, \mathcal{P}, \chi)$  comprising a set  $\mathcal{V}$  of variables, a set  $\mathcal{P}$  of predicates, and – in addition to the standard definition – a partial function  $\chi : \mathcal{P} \rightarrow \mathcal{P}$  with  $\text{dom } \chi \cap \text{ran } \chi = \emptyset$  that partitions  $\mathcal{P}$  into three sets

- *message send predicates*  $\mathcal{P}_{snd} := \text{dom } \chi$ ,
- *message receive predicates*  $\mathcal{P}_{rcv} := \text{ran } \chi$ , and
- *non-message predicates*  $\mathcal{P}_{cnd} := \mathcal{P} \setminus \mathcal{P}_{msg}$ ,

where  $\mathcal{P}_{msg} := \mathcal{P}_{snd} \dot{\cup} \mathcal{P}_{rcv}$ . This separation of predicates is the key to identify elements in the formula when considering the translation back to LSCs. The *boolean expressions* over  $\mathcal{S}$ , denoted by  $\text{Expr}_{\mathcal{S}}$ , are defined by the grammar

$$\psi ::= \text{true} \mid p(x_1, \dots, x_n) \mid \neg\psi_1 \mid \psi_1 \wedge \psi_2$$



$$\mathcal{I} \models_{LSC} L \Leftrightarrow \forall \iota \in \mathcal{I} \forall \sigma \in \text{Val}_{\mathcal{U}}(\mathcal{S}) \forall k \in \mathbb{N}_0 : (\mathcal{U}, \iota_k), \sigma \models ac \Rightarrow \iota/k \in \mathcal{L}_{\sigma}(\mathcal{A}_L)$$

(c) Full Semantics of  $L$  (cf. Sec. 3).

Fig. 1.

with  $p \in \mathcal{P}$ . In  $\text{Expr}_{\mathcal{S}}^{msg}$  only predicates  $p \in \mathcal{P}_{msg}$  occur. A tuple  $\mathcal{M} = (\mathcal{U}, \mathcal{I})$  is called *structure* of  $\mathcal{S}$  if  $\mathcal{U}$  is a non-empty set called *universe* and  $\mathcal{I}$  is an interpretation of the predicates in  $\mathcal{P}$ . A function  $\sigma : \mathcal{V} \rightarrow \mathcal{U}$  is called *valuation* of  $\mathcal{V}$ . The semantics of  $\psi \in \text{Expr}_{\mathcal{S}}$  is standard given a structure and a valuation from  $\text{Val}_{\mathcal{U}}(\mathcal{S})$ , the set of all valuations of  $\mathcal{V}$ .

A central information in the concrete syntax of an LSC, e.g. as given in Fig. 1(a), is the order of elements along a single instance line. To capture this information, we introduce the *LSC instance line* as a triple  $(\mathbb{A}, \prec, \vartheta)$  where  $\mathbb{A}$  is a finite non-empty set,  $\vartheta : \mathbb{A} \rightarrow \text{Temp} := \{\text{hot}, \text{cold}\}$ , and  $\prec \subseteq \mathbb{A} \times \mathbb{A}$  is a *scenario order*. That is,

1.  $\exists^1 a^\perp \in \mathbb{A} \forall a \in \mathbb{A} : a^\perp \prec^* a$  (Unique Minimum)  
where  $\prec^*$  denotes the reflexive transitive closure of  $\prec$ ,
2.  $\forall a, a_1, a_2 \in \mathbb{A} : a \prec a_1 \wedge a \prec a_2 \implies a_1 \not\prec^* a_2$  (Direct Successors are Unordered),  
where  $a_1 \not\prec^* a_2$  iff  $a_1 \not\prec^* a_2$  and  $a_2 \not\prec^* a_1$ , and
3.  $\forall a_1, a_2 \in \mathbb{A} :$  (Diamond Property)  
 $(\exists a_0 \in \mathbb{A} : a_0 \prec a_1 \wedge a_0 \prec a_2) \implies (\forall a_3 \in \mathbb{A} : a_1 \prec a_3 \implies a_2 \prec a_3).$

Acyclicity of  $\prec$  is implied by (1)–(3). Two *atoms*  $a_1, a_2 \in \mathbb{A}$  are called *instance co-located*, denoted by  $a_1 \bowtie a_2$ .

A *core LSC* over a signature  $\mathcal{S}$  is a tuple  $L = (\ell, ac, am, quant)$  comprising the information from the head, *activation condition*  $ac \in \text{Expr}_{\mathcal{S}}$ , *activation mode*  $am \in \{\text{initial}, \text{invariant}\}$ , and *quantification*  $quant \in \{\text{existential}, \text{universal}\}$  and the *body*  $\ell = (\{(\mathbb{A}_1, \prec_1, \vartheta_1), \dots, (\mathbb{A}_n, \prec_n, \vartheta_n)\}, \text{Msg}, \text{Cond}, \text{LocInv})$ ,  $n \geq 1$ . It comprises a set of pairwise disjoint LSC instance lines,  $\{(\mathbb{A}_1, \prec_1, \vartheta_1), \dots, (\mathbb{A}_n, \prec_n, \vartheta_n)\}$ , and the elements distributed over the sets of *messages*,  $(m \in) \text{Msg} =: \text{Msg}(L)$ , *conditions*,  $(c \in) \text{Cond} =: \text{Cond}(L)$ , and *local invariants*,  $(l \in) \text{LocInv} =:$

$LocInv(L)$  with

$$\begin{aligned} m &= (a_s, a_r, \varsigma, \kappa, \psi_s, \psi_r) \in \mathbb{A}_L \times \mathbb{A}_L \times Sync \times Obl \times Expr_S^{msg} \times Expr_S^{msg} \\ c &= (A_c, \kappa, \psi_c) \in \mathfrak{P}(\mathbb{A}_L) \times Obl \times Expr_S \\ l &= ((a_s, \gamma_s), (a_e, \gamma_e), \kappa, \psi) \in (\mathbb{A}_L) \times (\mathbb{A}_L) \times Obl \times Expr_S. \end{aligned}$$

where  $\mathbb{A}_L := \bigcup_{i \in Inst(L)} \mathbb{A}_i$ ,  $\prec_L := \bigcup_{i \in Inst(L)} \prec_i$ , and  $Inst(L) := \{1, \dots, n\}$ . Each element has an *obligation mode* from  $\{mand, poss\} =: Obl$ , messages have a *synchronicity* from  $\{inst, asyn\} =: Sync$ , and each local invariant start- and end-atom has a *containedness* from  $\{incl, excl\} =: Cont$ . The instantaneous and asynchronous messages in  $L$  are denoted by  $Msg_{inst}(L)$  and  $Msg_{asyn}(L)$ . Note that all expressions in an LSC may use variables from  $\mathcal{V}$  thus dynamic binding of LSCs [12, 7] is covered.

For  $i \in Inst(L)$  we use  $a_i^\perp$  to denote the minimum of  $(\mathbb{A}_i, \prec_i, \vartheta_i)$  and  $A|_i := A \cap \mathbb{A}_i$  to denote the projection of  $A \subseteq \mathbb{A}_L$  onto instance  $i$ . The elements of  $\mathbb{A}_L^\perp := \{a_i^\perp \mid i \in Inst(L)\}$  are also called *instance heads*. The set  $elems(L) := Msg \cup Cond \cup LocInv$  is the set of *elements of  $L$* . For any  $e \in elems(L)$  let  $atoms(e)$  yield the set of atoms of  $e$ . We set  $atoms(L) := atoms(elems(L))$ . A finite non-empty set of core LSCs  $Lsc = \{L_1, \dots, L_n\}$  is called *core LSC specification*.

### 3 The Semantics of Core LSCs

The central concept of the LSC semantics [11] is the cut, a set of atoms used to express how far the individual instance lines, and hence the whole LSC, have been observed. Formally, a set of atoms is a *cut* iff (i) it is empty or comprises at least one atom for each instance, i.e.  $\alpha \neq \emptyset \implies \forall i \in Inst(L) : \alpha|_i \neq \emptyset$  and (ii) all instance co-located atoms in  $\alpha$  are pairwise unordered, i.e.  $\forall a_1, a_2 \in \alpha : a_1 \bowtie a_2 \implies a_1 \not\prec^* a_2$ . Effectively, (ii) requires that all atoms on a single instance line belong to the same core region. The empty cut  $\alpha_0$  is called *initial cut of  $L$*  and the maximal cut  $\alpha$  with  $\forall a \in \alpha \forall a' \in atoms(L) : a \prec^* a' \implies a' = a$  is called *final cut of  $L$*  and denoted by  $\alpha_{fin}(L)$ . The set of all cuts of  $L$  is denoted by  $Cuts(L)$ . The *temperature of  $\alpha$* , denoted by  $\vartheta(\alpha)$ , is ‘cold’ if  $\alpha = \alpha_{fin}(L)$  or  $\vartheta(a) = cold$  for all  $a \in \alpha$ , and ‘hot’ otherwise.

The unit by which a cut can be advanced is the simultaneous class (simclass for short). Two atoms of a synchronous message are supposed to be observed simultaneously as well as all atoms of a condition. Formally, two atoms  $a_1, a_2 \in atoms(L)$  are called *simultaneous*, denoted by  $a_1 \sim a_2$ , iff (i)  $a_1 = a_2$ , or (ii)  $\{a_1, a_2\} \subseteq \mathbb{A}_L^\perp$ , or (iii)  $\exists e \in Cond(L) \cup Msg_{inst}(L) : \{a_1, a_2\} \subseteq atoms(e)$ , or (iv)  $\exists a_3 \in atoms(L) : a_1 \sim a_3 \wedge a_3 \sim a_2$ . For each  $a \in atoms(L)$  we use  $[a]$  to denote the equivalence class of  $a$  wrt.  $\sim$ , i.e. the set  $\{a' \in atoms(L) \mid a' \sim a\}$ . The set  $atoms(L)/\sim$  of all equivalence classes of atoms from  $atoms(L)$  is also denoted by  $Simclass(L)$ , its elements are called *simclass*. The set  $elems(scl) := \{e \in elems(L) \mid atoms(e) \cap scl \neq \emptyset\}$  comprises all LSC elements sharing an atom from the simclass  $scl \in Simclass(L)$ . From the intuition of the meaning of an LSC there are two cases justifying advancement of a cut by a simclass.

Firstly, if a coregion on an instance line has partially been observed, i.e. if the current cut does not comprise all atoms of the coregion, then the cut may be advanced by adding any non-empty subset of the remaining atoms from the coregion. Secondly, if a whole coregion or a single atom not belonging to a coregion are in the cut, then the cut may be advanced by all atoms that are direct successors of this instance line's atoms in the cut. Additionally, the reception of an asynchronous message has to be observed strictly after its sending.

Putting it together, a cut  $\alpha$  is said to *enable* a simclass ' $scl$ ', denoted by  $\alpha \triangleright scl$ , iff

$$\begin{aligned} & (\forall a' \in scl : prereq(a') \subseteq \alpha \vee \exists a \in \alpha : a \bowtie a' \wedge a \not\prec^* a') \\ & \wedge (\forall m \in Msg_{asyn}(L) \cap elems(scl) : a_r(m) \in scl \implies \exists a \in \alpha : a_s(m) \prec^* a) \end{aligned}$$

where  $prereq(a) := \{a' \in atoms(L) \mid a' \prec a\}$  is the *prerequisite* of  $a$ . The set of non-empty sets of simclasses  $Ready_L(\alpha) := \{\emptyset \neq Scl \subseteq Simclass(L) \mid \forall scl \in Scl : \alpha \triangleright scl\}$  is called the *readysset* of  $\alpha$ . A set  $Scl \in Ready_L(\alpha)$  is called *firedset*. The step function formalises the advancement of a cut by a set of simclasses as  $Step_L(\alpha, \{scl_1, \dots, scl_n\}) := Max(\alpha \cup scl_1 \cup \dots \cup scl_n)$  where  $Max(A) := A \setminus \{a \in A \mid \exists a' \in A : a \prec^+ a'\}$ .

In [11], the semantics of LSCs is defined in terms of a variant of Büchi automata with transitions labelled by expressions over a signature instead of by an alphabet's elements. Each cut becomes a state of the automaton and each state has as many outgoing transitions as there are firedsets in the readysset. The destination of these transitions is given by the step function. In addition, each state gets a self-loop that characterises the conditions under which a cut is not advanced and a so-called exit transition to the final node triggered by the violation of cold conditions or local invariants.

A *Symbolic Automaton* (SA) over signature  $\mathcal{S}$  is a tuple  $\mathcal{A} = (Q, q_s, \rightsquigarrow, F)$  comprising a finite set of states  $Q$ , the initial state  $q_s \in Q$ , the transition relation  $\rightsquigarrow \subseteq Q \times Expr_{\mathcal{S}} \times Q$ , and the set of accepting states  $F \subseteq Q$ . We write  $q_i \rightarrow q_j$  iff  $(q_i, \psi, q_j) \in \rightsquigarrow$  for some  $\psi \in Expr_{\mathcal{S}}$  and  $q_i \hat{\rightarrow} q_j$  iff  $q_i \rightarrow q_j$  and  $q_i \neq q_j$ . An SA is called *partially ordered*, or POSA, if the reflexive transitive closure of  $\rightarrow$  is antisymmetric. It is called *deterministic* if for all  $(q, \psi_1, q_1) \in \rightsquigarrow$  and  $(q, \psi_2, q_2) \in \rightsquigarrow$ ,  $q_1 \neq q_2$  are disjoint, i.e.  $\mathcal{M}, \sigma \models \neg(\psi_1 \wedge \psi_2)$  for any  $\mathcal{M}$  and  $\sigma$ . Given a universe  $\mathcal{U}$ , a sequence  $\iota = \iota_0 \iota_1 \iota_2 \dots$  of interpretations  $\iota_i$  of the predicates in  $\mathcal{S}$  is called interpretation sequence. By  $\iota/k := \iota_k \iota_{k+1} \dots$  we denote its suffix starting at position  $k$ . The set of all interpretation sequences is denoted by  $\overrightarrow{Int}_{\mathcal{U}}(\mathcal{S})$ . An infinite sequence  $\pi = q_0 q_1 q_2 \dots$  of states  $q_i \in Q$  is called a *run* of the SA  $\mathcal{A}$  over  $\iota$  under  $\sigma$  iff  $q_0 = q_s$  and  $(q_i, \psi, q_{i+1}) \in \rightsquigarrow$  for some  $\psi \in Expr_{\mathcal{S}}$  s.t.  $(\mathcal{U}, \iota_i), \sigma \models \psi$ ,  $i \in \mathbb{N}_0$ , i.e. the transition predicate holds in the structure  $(\mathcal{U}, \iota_i)$ . The set of runs of  $\mathcal{A}$  over  $\iota$  under  $\sigma$  is denoted by  $\Pi_{\sigma}^{\iota}(\mathcal{A})$ . The *language* accepted by  $\mathcal{A}$  is  $\mathcal{L}_{\sigma}(\mathcal{A}) := \{\iota \in \overrightarrow{Int}_{\mathcal{U}}(\mathcal{S}) \mid \exists \pi \in \Pi_{\sigma}^{\iota}(\mathcal{A}) : inf(\pi) \cap F \neq \emptyset\}$  where  $inf(\pi)$  denotes the set of states occurring infinitely often in  $\pi$ .

The *Symbolic Automaton of L*, denoted by  $\mathcal{A}_L$ , is the tuple  $(Q, q_s, \rightsquigarrow, F)$  with  $Q = Cuts(L)$ ,  $q_s = \alpha_0$ ,  $F = \{\alpha \in Cuts(L) \mid \vartheta(\alpha) = cold\}$ , and transitions

$$\begin{aligned} \rightsquigarrow := & \{(\alpha, Hold_L(\alpha), \alpha) \mid \alpha \in Cuts(L) \setminus \{\alpha_0\}\} \\ & \cup \{(\alpha, Exit_L(\alpha), \alpha_{fin}(L)) \mid \alpha \in Cuts(L) \setminus \{\alpha_{fin}(L)\}\} \\ & \cup \{(\alpha, Trans_L(\alpha, Scl), Step_L(\alpha, Scl)) \mid \alpha \in Cuts(L), Scl \in Ready_L(\alpha)\}. \end{aligned}$$

In the definition of the transition predicates for the self-loop, the exit transition, and the regular transitions we use the helper functions  $Cond(scl) := Cond(L) \cap elems(scl)$  and  $Msg_{snd}(scl) := \{m \in Msg(L) \cap elems(scl) \mid a_s(m) \in scl\}$ ; analogously for  $Cond_{poss}(scl)$  and  $Msg_{rcv}(scl)$ . Furthermore, we must be able to identify the subset of local invariants to be considered in the transition expressions. A local invariant  $l \in LocInv(L)$  is called *active beyond*  $\alpha$  if there are  $a, a' \in \alpha$  s.t.  $a_s(l) \prec^* a \wedge a' \prec^+ a_e(l)$  and *active at*  $\alpha$  if it is active beyond  $\alpha$  but not starting exclusively at  $\alpha$  or if it ends inclusively at  $\alpha$ . By  $ali_{L,=}(\alpha)$  ( $ali_{L,>}(\alpha)$ ) we denote all local invariants *active at (beyond)*  $\alpha$  and by  $ali_{L,=}^{poss}(\alpha)$  ( $ali_{L,>}^{poss}(\alpha)$ ) those  $l \in ali_{L,=}(\alpha)$  ( $ali_{L,>}(\alpha)$ ) with  $\kappa(l) = poss$ . We define  $\bigvee\{\psi_1, \dots, \psi_n\} := (\psi_1 \vee \dots \vee \psi_n)$  and  $\bigvee \emptyset := false$ ; analogously for conjunction with  $\bigwedge \emptyset := true$ . For a firedset  $S$  we set

$$\begin{aligned} A_S &:= \bigwedge(\psi_s(Msg_{snd}(S)) \cup \psi_r(Msg_{rcv}(S))) \quad \text{“all messages of } S\text{”} \\ N_S &:= \neg \bigvee(\psi_s(Msg_{snd}(S)) \cup \psi_r(Msg_{rcv}(S))) \quad \text{“no message of } S\text{”} \end{aligned}$$

The *hold predicate*  $Hold_L(\alpha)$  for a cut  $\alpha$  is  $N_{Ready_L(\alpha)} \wedge \bigwedge \psi(ali_{L,>}(\alpha))$ . It captures the intuition that the automaton remains in the state  $\alpha$  as long as none of the awaited messages are observed while the local invariants that are active beyond the cut are not violated. The *transition predicate*  $Trans_L(\alpha, Scl)$  for a cut  $\alpha$  and firedset  $Scl$  is

$$A_{Scl} \wedge N_{Ready_L(\alpha) \setminus \{Scl\}} \wedge \bigwedge \psi_c(Cond(Scl)) \wedge \bigwedge \psi(ali_{L,=}(Step_L(\alpha, Scl)))$$

and observes the messages required by  $Scl$  while excluding the messages from all the other firedsets in the readysset. Furthermore, the conditions bound to the “fired messages” must hold, as well as the local invariants that are active in the follow-up cut. The *exit predicate*  $Exit_L(\alpha)$  for a cut  $\alpha$  is

$$\bigvee_{\substack{scl \in \\ Ready_L(\alpha)}} \left[ N_{scl} \wedge \neg \bigvee \psi(ali_{L,>}^{poss}(\alpha)) \vee A_{scl} \wedge \left( \neg \bigwedge \psi_c(Cond_{poss}(scl)) \vee \neg \bigwedge \psi(ali_{L,=}^{poss}(Step_L(\alpha, scl))) \right) \right]$$

and allows to take the exit transition if either a possible local invariant is violated while the awaited messages are not yet observed or if a possible condition or local invariant at the target cut is violated when observing the desired messages. For example, the SA of the LSC in Fig. 1(a) is depicted in Fig. 1(b).

The complete semantics of a core LSC is defined using the SA of its body. The LSC quantification *quant* determines the quantification over a set of interpretation sequences and the activation mode *am* the quantification over snapshots in

the interpretation sequence. Figure 1(c) exemplarily gives the complete semantics of a universal, invariant core LSC. For an initial, universal core LSC, only the case  $k = 0$  is considered. For existential LSCs, the two quantifiers become existential. A set  $I$  of interpretation sequences satisfies a core LSC specification  $Lsc$  if it satisfies all core LSCs in  $Lsc$ .

Klose [11] already observed that conditions not synchronised with a message have undesirable semantical effects. Intuitively, they lead to non-determinism in the automaton as the point in time for evaluating the condition is unclear. He gives the advice that “each condition should be bound to at least one message”. We call LSC following this advice *bonded*, formally characterised by

$$\begin{aligned} \forall e \in \text{Cond} \cup \text{LocInv} \forall a \in \text{atoms}(e) \exists m \in \text{Msg} : \\ \kappa(m) = \text{mand} \wedge [a] \cap \text{atoms}(m) \neq \emptyset. \end{aligned}$$

Note that most practically used LSCs are bonded since authors want to avoid the effects named above [8, 11].

Our following observation is central to the translation from LSCs to temporal logic as discussed in the Sect 4.

**Lemma 1.** *Let  $L$  be a core LSC over signature  $\mathcal{S}$  and  $\mathcal{A}_L$  its Symbolic Automaton. Then  $\mathcal{A}_L$  is a POSA. If  $L$  is bonded, then  $\mathcal{A}_L$  is deterministic.*

## 4 From LSCs to Temporal Logic and Back

LSC specifications can be characterised in terms of first-order prenex CTL\* (FOP-CTL\*) formulae, defined for signature  $\mathcal{S}$  by the grammar

$$\begin{aligned} \phi &::= \psi \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \phi_1 \mathbf{U} \phi_2 \mid \mathbf{X}\phi & \varphi^{EA} &::= \phi \mid \mathbf{E}\phi \mid \mathbf{A}\phi \\ \varphi^{\exists\forall} &::= \varphi^{EA} \mid \exists x. \varphi^{\exists\forall} \mid \forall x. \varphi^{\exists\forall} & \varphi &::= \varphi^{\exists\forall} \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \end{aligned}$$

where  $\psi \in \text{Expr}_{\mathcal{S}}$ . We shall use the abbreviations  $\wedge$ ,  $\rightarrow$ ,  $\leftrightarrow$ ,  $\mathbf{F}$ ,  $\mathbf{G}$ , and  $\mathbf{W}$ . The expressive power of FOP-CTL\* is sufficient for our purposes since LSCs only need top-level path quantifiers. It is convenient since the (standard) semantics can be explained using a set of (system) runs, instead of a tree needed for general CTL\*.

**Lemma 2 (Schlör [16]).** *Let  $\mathcal{A}$  be a POSA over signature  $\mathcal{S}$ ,  $\mathcal{U}$  a universe, and  $\sigma \in \text{Val}_{\mathcal{U}}(\mathcal{S})$  a valuation. Then there is a CTL\* formula  $\phi_{\mathcal{A}}$  over  $\mathcal{S}$  with  $\iota \in \mathcal{L}_{\sigma}(\mathcal{A}) \iff \iota, \sigma \models \phi_{\mathcal{A}}$ .*

The formula constructed in the course of the proof of Lemma 2 is recursively defined as  $\phi_{\mathcal{A}} := \phi_{q_s}$  with  $\phi_q := \psi(q, q) \mathbf{U}_q \bigvee_{q \rightarrow q'} (\psi(q, q') \wedge \mathbf{X}\phi_{q'})$  for  $q \in Q$  where  $\psi(q, q')$  denotes the (well-defined) transition predicate of  $q \rightarrow q'$ . The temporal operator  $\mathbf{U}_q$  is  $\mathbf{W}$  if  $q \in F$  and  $\mathbf{U}$  otherwise. Thereby we obtain a formula for each LSC body; the formula for the whole LSC expresses the path and variable quantification by the corresponding operators in FOP-CTL\*. The formula for an LSC specification is the conjunction of the formulae for its LSCs.

In the constructed formulae, further structure can be identified. They are actually communication sequence CTL\* (CSCTL) formulae over  $\mathcal{S}$  defined as

$$\begin{aligned}
 \zeta &::= \xi_E \mid \xi_A \mid \zeta \wedge \zeta & \xi_E &::= \mathbf{E}(\psi \rightarrow \pi) \mid \mathbf{E}\mathbf{G}(\psi \rightarrow \pi) \mid \exists x. \xi_E \\
 \pi &::= \eta \mathbf{U} \hat{\pi} \mid \eta \mathbf{W} \hat{\pi} & \xi_A &::= \mathbf{A}(\psi \rightarrow \pi) \mid \mathbf{A}\mathbf{G}(\psi \rightarrow \pi) \mid \forall x. \xi_A \\
 \eta &::= \neg p_{msg}(x_1, \dots, x_n) \wedge \eta \mid \psi & \hat{\pi} &::= \hat{\pi}_1 \vee \hat{\pi}_2 \mid \tau \wedge \mathbf{X} \pi \mid false \\
 \tau &::= \neg p_{msg}(x_1, \dots, x_n) \wedge \tau \mid p_{msg}(x_1, \dots, x_n) \wedge \tau \mid \psi
 \end{aligned}$$

where  $\psi \in Expr_{\mathcal{S}}$ ,  $x_i \in \mathcal{V}$  for  $1 \leq i \leq n$ , and  $p_{msg} \in \mathcal{P}_{msg}$ . Deterministic CSCTL (DCSCTL) formulae are obtained by replacing the production for  $\hat{\pi}$  by  $\hat{\pi} ::= \tau \vee \phi \mid (\tau \wedge \pi) \vee \phi$ , adding the production  $\phi ::= false \mid \tau \mid \phi_1 \vee \phi_2$ , and requiring that (i) there is an injection between the occurrences of  $p \in \mathcal{P}_{snd}$  and  $\chi^{-1}(p)$  in  $\xi_E$  and  $\xi_A$ , (ii) if  $\neg p_{msg}$  and  $p'_{msg}$  occur on both sides of an  $\mathbf{U}$  or  $\mathbf{W}$ , then  $p_{msg} = p'_{msg}$ , and (iii)  $\tau$  and  $\phi$  are disjoint in all occurrences of  $\tau \vee \phi$ .

**Theorem 1.** *Given a core LSC  $L$  over signature  $\mathcal{S}$ , there is a CSCTL formula  $\phi_L$  over  $\mathcal{S}$  with  $\mathcal{L}(L) = \mathcal{L}(\phi_L)$ . If  $L$  is bonded, then there is an equivalent DC-SCTL formula.*

The proof is based on the addition of path- and variable quantifiers to the formula obtained by Lemma 2, depending on the LSC activation mode and quantification, e.g.  $\phi_L ::= \forall x_1 \dots \forall x_n. \mathbf{A}\mathbf{G}(ac \rightarrow \phi_{\mathcal{A}_L})$  for an LSC  $L = (\ell, ac, invariant, universal)$  over a signature  $\mathcal{S} = (\{x_1, \dots, x_n\}, \mathcal{P}, \chi)$ .

The second claim is established by a result of [16] that allows to transform the CSCTL formula to the desired DCSCTL form if the transition expressions, here  $\psi(q, q')$ , are mutually disjoint which is the case if the LSC is bonded (cf. Lemma 1). The finer structure (i)–(iii) of DCSCTL formulae is obtained by close examination of the construction of the translation relation  $\rightsquigarrow$  for  $\mathcal{A}_L$ . We note, omitting the proof, that the containment is proper, i.e. there are formulae not expressible by any LSC, e.g.  $(\mathbf{X}\mathbf{X}p) \mathbf{U} q$ , where the formula before the ‘ $\mathbf{U}$ ’ uses temporal modalities in addition to predicate logic.

**Theorem 2.** *Let  $\zeta$  be a DCSCTL formula over signature  $\mathcal{S}$ . There exists a core LSC specification  $Lsc$  over  $\mathcal{S}$  such that  $\mathcal{L}(Lsc) = \mathcal{L}(\zeta)$ .*

The proof is constructive and exploits that  $\zeta$  is a DCSCTL formula with the properties as defined in Sec. 4. The formula  $\zeta$  is basically a conjunction  $\pi_1 \wedge \dots \wedge \pi_n$  of formulae of the form  $\pi_i = \eta \mathbf{U} ((\tau \wedge \mathbf{X} \pi_1) \vee \phi) = (\neg \mu \wedge \psi_1) \mathbf{U} (((\mu \wedge \psi_2) \wedge \mathbf{X} \pi_1) \vee \phi)$  that describe a particular one of the orderings of messages allowed by the LSC and is satisfied by each run with this order. For runs with a different but valid message order,  $\phi$  holds as soon as the difference in orders is visible and another sub-formula is satisfied. Invalid runs don’t satisfy any sub-formula  $\pi_i$ . This intuition can be transferred to LSCs. We inductively construct an LSC specification that comprises only bonded LSCs that accept a particular message order each. To this end, the expression  $\psi_1$  in  $\pi$  becomes a mandatory local-invariant that is required to hold until  $\tau$ , one or more messages co-located with a condition  $\psi_2$ , is observed, unless a cold local-invariant with expression  $\neg \phi$  is violated, indicating that this LSC can not accept the run (cf. Fig. 2).



3. Y. Bontemps, P. Heymans, and H. Kugler. Applying LSCs to the specification of an air traffic control system. In *Proc. SCESM'03*, 2003.
4. Annette Bunker, Ganesh Gopalakrishnan, and Konrad Slink. Live sequence charts applied to hardware requirements specification and verification: A VCI bus interface model. *Software Tools for Technology Transfer*, 7(4):341–350, August 2004.
5. Pierre Combes, David Harel, and Hillel Kugler. Modeling and verification of a telecommunication application using live sequence charts and the play-engine tool. In *Proc. ATVA 2005*, number 3707 in LNCS, 2005.
6. W. Damm and D. Harel. LSCs: Breathing Life into Message Sequence Charts. *Formal Methods in System Design*, 19(1):45–80, July 2001.
7. W. Damm and B. Westphal. Live and let die: LSC-based verification of UML-models. *Science of Computer Programming*, 55(1–3):117–159, March 2005.
8. Werner Damm and Jochen Klose. Verification of a radio-based signaling system using the statemate verification environment. *Formal Methods in System Design*, 19(2):121–141, September 2000.
9. D. Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer-Verlag, 2003.
10. ITU-T. *ITU-T Rec. Z.120: Message Sequence Chart (MSC)*. ITU-T, Geneva, 1999.
11. J. Klose. *Live Sequence Charts: A Graphical Formalism for the Specification of Communication Behavior*. PhD thesis, C. v. O. Universität Oldenburg, 2003.
12. Jochen Klose and Bernd Westphal. Relating lsc specifications to uml models. In Hartmut Ehrig and Martin Grosse-Rhode, editors, *INT*, April 2002.
13. Christoph Knieke, Michaela Huhn, and Ursula Goltz. Modelling and simulation of an automotive system using lscs. In Siv Hilde Houmb and Jan Jürjens, editors, *Proc. CSDUML'2005*, pages 0–0. TUM, September 2005. TUM-TR.
14. H. Kugler, D. Harel, A. Pnueli, Y. Lu, and Y. Bontemps. Temporal logic for scenario-based specifications. In N. Halbwachs and L. D. Zuck, editors, *TACAS*, volume 3440 of LNCS. Springer-Verlag, 2005.
15. Ingo Schinz, Tobe Toben, Christian Mrugalla, and Bernd Westphal. The Rhapsody UML Verification Environment. In Jorge R. Cuellar and Zhiming Liu, editors, *Proc. SEFM 2004*, pages 174–183, September 2004.
16. R. C. Schlör. *Symbolic Timing Diagrams: A Visual Formalism for Model Verification*. PhD thesis, C. v. O. Universität Oldenburg, January 2000.
17. W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of theoretical computer science (vol. B): formal models and semantics*, pages 133–191. MIT Press, 1990.
18. Klaus Weidenhaupt, Klaus Pohl, Matthias Jarke, and Peter Haumer. Scenarios in system development: Current practice. *IEEE Software*, 15(2):34–45, March 1998.
19. Hartmut Wittke. *A Framework for Specification Verification for Complex Embedded Systems*. PhD thesis, C. v. O. Universität Oldenburg, 2005. To appear.