

On the Expressive Power of Live Sequence Charts^{*}

Werner Damm, Tobe Toben, and Bernd Westphal

Carl von Ossietzky Universität Oldenburg, 26111 Oldenburg, Germany,
{damm,toben,westphal}@informatik.uni-oldenburg.de

Abstract. The Live Sequence Charts (LSC) language is a formally rigorous variant of the well-known scenario language Message Sequence Charts (MSC). LSCs yield expressive power by means to distinguish mandatory and scenario behaviour, means to characterise by another scenario the context in which a specification applies, and means to distinguish required from possible progress, i.e. to require liveness. From the original proposal by Damm & Harel [1], two slightly different dialects emerged, one in the context of LSC play-in and -out [2] and one for the use of LSCs as formal requirements specification language in formal, model-based approaches to software development [3]. In this paper, we investigate the expressive power of LSCs in the sense of [3]. That is, we first (constructively) show that for each LSC there is an equivalent CTL^{*} formula. Complementing existing work, we show that the containment is strict, that is, not each CTL^{*} formula has an equivalent LSC. To complete the discussion, we present for the first time a way back, from a syntactically characterised fragment of CTL^{*} to the subset of bonded LSC specifications, thereby establishing an equivalence.

1 Introduction

Scenario-based approaches are an adequate approach to the formal specification of requirements on systems that are composed of different components [4, 5]. The common idea of scenario-based specification is to formally yet comprehensibly describe all interactions between system components, and their interaction with the environment, that are necessary to accomplish a certain task. This is, for example, in contrast to temporal logic patterns [6, 7], which also claim comprehensibility but only provide rather atomic templates for action/response pairs of which many have to be used to cover more complex tasks.

The Live Sequence Charts (LSC) language is a particular formalism that supports the scenario-based approach. It has been introduced by Damm & Harel in [1] as a conservative extension of the well-known ITU-standard Message Sequence Charts (MSC) [8] and in the meantime gained wide adoption [9–13].

In the following, we give a brief example to recall the scenario-based approach and the LSC language. Although the LSC language is graphical and intuitive,

^{*} This work was partly supported by the German Research Council (DFG) in SFB/TR 14 AVACS and in project DA 206/7-3 (USE), SPP 1064.

there is no complete introduction possible within a page or two. For a more thorough introduction, the reader is referred to [1, 3, 2].

Consider the design of the software of a level-crossing system. There might be a central controller ‘CrossingCtrl’ and separate controllers ‘LightsCtrl’ and ‘BarrierCtrl’ for the traffic lights and the barriers. One requirement on the system is clearly to secure the crossing on a request ‘*secreq*’ by the environment. Then the central controller shall finally trigger the lights and barrier controller by sending appropriate messages. If the lights controller is operational at that point in time, it shall continue to switch on the red traffic lights and report back success while the barrier controller simultaneously initiates lowering of the barrier and finally reports back success. The barrier moving up in between the latter two events would be an error. After both sub-controllers reported success, it would be kind (but not necessary) if the central controller reported success back to the environment.

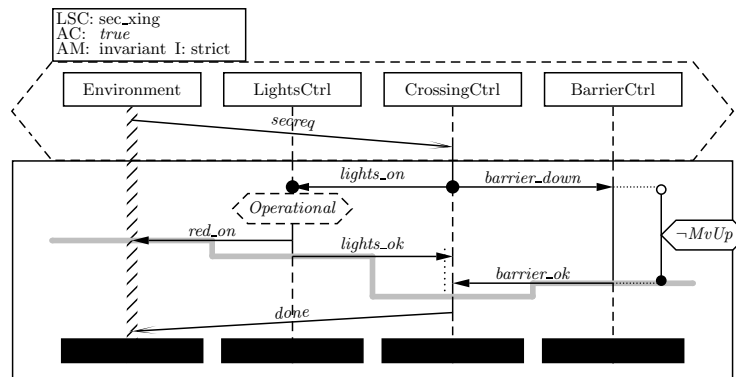


Fig. 1. Live Sequence Chart for a system of level-crossing controllers.

This prosaic specification can be formalised using scenarios by the LSC shown in Figure 1. There is one (vertical) instance line for each sub-controller labelled accordingly and one for the environment, as indicated by the diagonal lines. The large dashed hexagon is called *pre-chart* and characterises the activation of the scenario, that is, the situations in which the remainder (or *main chart*) of the chart is supposed to be observed. In the example, this is the single *asynchronous message*, i.e. receipt occurs strictly after sending, ‘*secreq*’ from the environment, which requests securing. To express that the central controller shall *finally* send the following two *instantaneous messages* ‘*lights_on*’ and ‘*barrier_down*’, we can use the LSC feature that instance line segments have a temperature. Temperature *hot*, graphically indicated by a solid line segment, enforces progress, while *cold*, indicated by a dashed line segment, doesn’t. The segment of the ‘CrossingCtrl’ instance line starting at the top of the main-chart is hence solid.

Black circles on instance lines are *simultaneous regions*, which enforce simultaneity. Thus messages ‘*lights_on*’ and ‘*barrier_down*’ should be sent (and received) simultaneously.

The smaller hexagon on the ‘LightsCtrl’ instance line is a *condition*, which requires that the controller shall be operational at the point in time when ‘*lights_on*’ is received. Note that conditions in LSC are treated differently than MSC conditions. First of all, they are semantically significant in LSCs, while in MSCs they are merely comments. And secondly they may also have a temperature, either hot or cold, which is graphically indicated by solid or dashed outline. The semantics of cold conditions is that if they hold when evaluated, then in order to satisfy the LSC one has to adhere to the remainder of the scenario; and if they don’t hold, the whole scenario is immediately considered to be satisfied. Thus cold conditions can be used to provide *legal exits* to a scenario. In practice, scenarios with legal exits are typically complemented by another scenario which shares the same prefix and continues after the complementary condition with the actions to take in the complementary case. Thus in our example, there should be another chart which specifies the system behaviour in case the lights controller is *not* operational when receiving ‘*lights_on*’.

To indicate that a condition is supposed to hold for a span of time, we can use the LSC feature *local invariant*. In Figure 1, there is a local invariant that starts *exclusively* with the receipt of ‘*barrier_down*’ and ends *inclusively* with the sending of ‘*barrier_ok*’ as denoted by the unfilled and filled circles. It requires that the barrier shall not move up from (strictly after) the point in time where ‘*barrier_down*’ has been received up to (and including) the point in time where ‘*barrier_ok*’ is sent. The solid outline of the condition hexagon indicates that its temperature is hot, that is, if the condition is violated during a system run that matched the scenario up to this point, then the system violates the LSC.

As we don’t care in which order the subsidiary controllers reported back to the central controller, the order on messages ‘*lights_ok*’ and ‘*barrier_ok*’ is explicitly relaxed by enclosing them in a *coregion*, graphically indicated by the dotted line in parallel to the ‘CrossingCtrl’ instance line. Both messages may be observed in either order or even simultaneously.

Note that progress is enforced up to the configuration or cut (cf. 2.2) indicated by the horizontal gray line, as there are hot instance line segments at the participating instances. Below the gray line, all instance line segments are cold, that is, progress is no longer enforced. In other words, the message ‘*done*’ may be sent, but need not.

In addition to locations and conditions, the whole chart also has a temperature, either hot or cold, which is graphically indicated by the box enclosing the *main chart*, which complements the pre-chart. In order to satisfy a hot (or *universal*) chart as in the example, a system has to satisfy the main chart *whenever* the pre-chart is observed. In contrast, a cold (or *existential*) chart is already satisfied if there exists a single system run that adheres to the concatenation of pre- and main-chart.

The last aspect of Figure 1 to discuss is the *header*, the small box on top of the pre-chart. It assigns the chart a name, chooses an interpretation, and may further restrict activation. Namely, the pre-chart is only considered, if the *activation condition* (AC) holds. That is, if the activation holds at some point in time and *from there on* the pre-chart is observed, then the main-chart has to be considered. The *activation mode* (AM) denotes the candidates for evaluation of the activation condition. If it is *invariant*, then any point in time is considered while if it is *initial*, only the initial states of the system are. A third mode, *iterative*, is similar to invariant but excludes overlapping activation. If the LSC is non-reactivating, that is, if the pre-chart is not a sub-sequence of the main-chart, then the iterative mode is equivalent to the invariant mode. If the LSC is reactivating, then the LSC has not even an equivalent in CTL* thus we will exclude this mode from the discussion in the following sections. Finally, the *interpretation* (I) determines the significance of additional occurrences of messages. In the *strict* interpretation, additional occurrences are considered to be violations, thus an implementation of ‘LightsCtrl’ that sends ‘red_on’ twice before reporting back success violates Figure 1. In the *weak* (or *tolerant* [2]) interpretation, additional occurrences of messages are ignored.

Having introduced the most common features of the LSC language, we can summarise that the most significant differences between LSCs and MSCs are

- *modalities* for
 - the whole chart, distinguishing example scenarios from universal ones,
 - locations, possibly indicating liveness,
 - conditions, providing for legal exits and anti-scenarios, and
 - messages (cf. [1, 3, 2]),
- precise characterisation of the activation time by pre-charts, activation condition, and activation mode, and
- semantical significance of conditions.

These additions make LSCs significantly more powerful than MSCs in the sense that more behaviour is distinguishable with LSCs than with MSCs (cf. [1] for details) whereas the graphical appeal and intuitive comprehensibility of MSCs is preserved by indicating the modalities graphically.

In order to understand the relation of this work to [14, 15], note that after the original introduction of LSCs by Damm & Harel [1], two slightly different dialects, motivated by different application domains, emerged.

The LSC language of Harel & Marelly [2] is tailored for the so called play-out approach. They employ a tool called play-engine to execute LSC specifications, i.e. sets of LSCs. Thereby there needn’t be an implementation of the intra-object behaviour of the system under design; the set of LSCs *is* the implementation. To this end, they added elements like actions to modify the state of the system, loops, and sub-charts to [1]. The semantics is given using the linearisation of partial orders [2].

The LSC language of Damm & Klose [3], which is the subject of this paper, is tailored to complement the model-based development of the intra-object behaviour of a system using, e.g., Statemate state-charts or UML state-machines. The model can then formally be checked for whether it adheres to the LSC specification, for example employing model-checking techniques as discussed in [16] and demonstrated in [13] for Statemate and in [17] for Rhapsody/UML.¹ To this end, they added local invariants and the activation mode [1]. The semantics is given by a variant of Büchi Automata [18, 3].

The motivation of this work is also rooted in the latter application domain. As MSCs and LSCs are, in contrast to other graphical formalisms, not simply graphical representations of more fundamental and well-understood formalisms like Temporal Logic but have been designed driven by the needs of the application domain, with an intuition of the semantics in mind, which has then been formalised. In order to understand the potentials and limitations of the LSC language, it is necessary to compare its expressive power to more fundamental formalisms like temporal logic. Pragmatically, a translation from LSCs to temporal logic makes it possible to employ any of the many temporal logic model-checkers for formal verification of LSCs against given system models.

The first result of our paper is similar to [14, 15] where the relation of a small subset of the LSCs of [2] to CTL* has already been established. We consider the different LSC dialect of [3] and obtain the result in a different way. Using the automaton based semantics of LSCs [3], we can employ older results for the translation from particular automata to temporal logic by [19]. In addition, we already convey a first comparison of both dialects on the common level of temporal logic; a full comparison will be possible with the full version of [15] that proposes to discuss a larger subset of the LSCs of [2]. Furthermore, we not only provide an embedding of LSCs into temporal logic but can more precisely characterise a fragment of first-order CTL* that comprises the subset of LSCs we consider via an inductive syntactical definition (similar to the syntactical characterisation of the common fragment of LTL and CTL* [20]).

As a second result, we obtain for the first time a description of an *equivalent* fragment of first-order CTL* for the slightly smaller but most commonly used subset of *bonded LSCs*, i.e. where conditions and local invariants only appear in simultaneous regions with messages. The equivalence is shown constructively by providing a translation back from formulae to LSCs.

A minor original contribution of this paper is the full version of a closed formalisation of the syntax and the semantics of the LSCs of [3], including the interpretations weak and strict. It turned out to be necessary to introduce an alternative formalisation, since [3, 18] give the semantics of LSCs only in form of an imperative unwinding algorithm that iteratively constructs the automaton for a given LSC. Our formalisation allowed to establish the results presented here and in [21, 16]. We expect it to be useful in further research on both dialects of LSCs since we are confident that it is extendable to the LSCs of [2].

¹ Statemate and Rhapsody are trademarks of i-Logix, Inc.

2 Core Live Sequence Charts

For a (slightly clearer) presentation, in the following we introduce a subset of the LSCs of [3] which we call *core* LSCs. Core LSCs are missing three features that are out of the scope of this paper. Firstly, we discuss activation only by an *activation condition* and not the general case of pre-charts. An activation condition can be used if the activation only depends on properties of a single system state and not on a whole scenario. For the translation to temporal logic, pre-charts can be treated similar to [15] as an implication from the formula of the pre-chart to the formula of concatenation of pre- and main-chart.

Secondly, we only consider *un-timed* LSCs, that is, we exclude timer-set and -reset and timeout elements as well as timing intervals which LSCs inherit from MSCs. General timed LSCs require a timed Temporal Logic. They can be treated similar to the Real time Symbolic Timing Diagrams (RSTD) in [22] since they use the same kind of automata that we use for LSCs [3].

And thirdly, we consider only hot asynchronous messages, i.e. sending *and* receipt has to be observed in contrast to cold asynchronous messages, which admit that the message is lost. Cold asynchronous messages add irregular transitions to the Symbolic Automaton, which are tedious to consider but effectively don't harm the automaton properties we use in the proofs. The translation to temporal logic extends directly to possible asynchronous messages, for the translation back we might need to exclude them.

The new representation of the abstract syntax of LSCs we introduce in the following Section 2.1 is equivalent to the one used in [3] in that it is closely related to the actual graphical charts like the one shown in Figure 1, but it is much more concise than [3]. Note that for our purpose, it is not sufficient to use a more abstract notion of abstract syntax like, for instance, partial orders on the set of LSC elements as demonstrated by [14]. The reason is that the proof of equivalence between a fragment of first-order prenex CTL* (cf. Section 3) and a subset of LSCs in Section 3.2 inductively constructs an LSC, and to this end needs a rather detailed abstract syntax.

As mentioned in the introduction, our definition of the LSC semantics in Section 2.2 is equivalent to the one of [3] but closed, instead of in terms of a translation algorithm, and thus much more appropriate to our needs.

2.1 Syntax.

Formally, annotations of messages, conditions, and local invariants of an LSC are boolean expressions, i.e. core LSCs are defined over a signature. A *signature* $\mathcal{S} = (\mathcal{V}, \mathcal{P}, \chi)$ comprises a set of variables \mathcal{V} , 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 the three sets of *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 messages and conditions 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 $\psi ::= true \mid p_0 \mid p(x_1, \dots, x_n) \mid \neg\psi_1 \mid \psi_1 \vee \psi_2$ where p_0 is a 0-ary predicate and p of arity $n > 0$. We shall use the common abbreviations *false*, \wedge , \rightarrow , and \leftrightarrow . 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 $\psi \in Expr_{\mathcal{S}}$ is standard given a structure $(\mathcal{U}, \mathcal{I})$ and a valuation from $Vak_{\mathcal{U}}(\mathcal{S})$, the set of all valuations of \mathcal{V} .

A central piece of information in the concrete syntax of an LSC as given in Figure 1 is the order of elements along a single instance line as their order shall be preserved unless relaxed by coregions. As coregions mustn't be nested, the order of elements is actually a scenario order as defined in the following. An LSC instance line is then simply a set equipped with a scenario order and a function that assigns each element a temperature from $Temp := \{hot, cold\}$.

Definition 1 (LSC Instance Line). *Let A be a finite, non-empty set. The tuple (A, \prec) is called instance line if and only if $\prec \subseteq A \times A$ is a scenario order (or direct predecessor relation) on A , that is, if and only if*

- (i) $\exists! a^\perp \in A \forall a \in A : a^\perp \prec^* a$ (Unique Minimum)
where \prec^* denotes the reflexive transitive closure of \prec .
- (ii) $\forall a, a_1, a_2 \in A : a \prec a_1 \wedge a \prec a_2 \implies a_1 \not\prec^* a_2$ (Unordered Successors)
where $a_1 \not\prec^* a_2$ denotes that a_1, a_2 are unordered, i.e. $a_1 \not\prec^* a_2$ and $a_2 \not\prec^* a_1$.
- (iii) $\forall a_1, a_2 \in A : (\exists a_0 \in A : a_0 \prec a_1 \wedge a_0 \prec a_2) \implies (\forall a_3 \in A : a_1 \prec a_3 \implies a_2 \prec a_3)$. (Diamond Property)

A triple (A, \prec, ϑ) with $\vartheta : A \rightarrow Temp$ is called LSC instance line if and only if (A, \prec) is an instance line. The elements $a \in A$ are then called (tempered) atoms. When dealing with multiple instance lines, we use $a_1 \bowtie a_2$ to denote that the atoms a_1 and a_2 belong to the same instance line. \diamond

The following definition of core LSCs captures the essence of an LSC picture like Figure 1, in particular the set of elements and their order on the instance lines. Formally, a core LSC is structured into the body and the information found in the head and given by the frame around the body, namely the activation condition, the activation mode, the interpretation, and the quantification. The body is further structured and comprises a set of LSC instance lines together with three sets of the elements: messages, conditions, and local invariants (cf. Figure 2). Messages in addition have a synchronicity from $Sync := \{inst, asyn\}$, conditions and local invariants are equipped with an *obligation* from $Obl := \{mand, poss\}$ (the formal names for *hot* and *cold*), and a local invariant start- and end-atom has a *containedness* from $Cont := \{incl, excl\}$.

Definition 2. *Let $\mathcal{S} = (\mathcal{V}, \mathcal{P}, \chi)$ be a signature. A core LSC over \mathcal{S} is a tuple $L = (\ell, ac, am, int, quant)$ with activation condition $ac \in Expr_{\mathcal{S}}$, activation mode $am \in \{initial, invariant, iterative\}$, interpretation $int \in \{strict, weak\}$, quantification $quant \in \{existential, universal\}$, and body*

$$\ell = (\{(A_1, \prec_1, \vartheta_1), \dots, (A_n, \prec_n, \vartheta_n)\}, Msg_L, Cond_L, LocInv_L), n \geq 1, \text{ where}$$

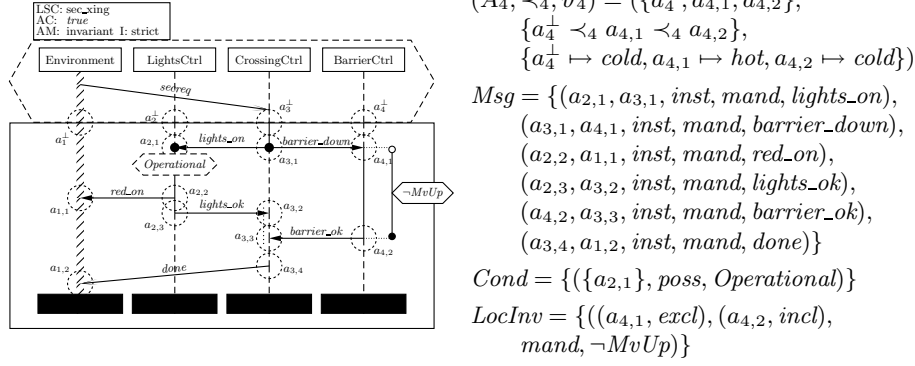


Fig. 2. Abstract syntax of the LSC from Figure 1. For brevity, we only consider the LSC body, give the order and temperature only for the right-most instance line, and in messages omit the receive expressions as they are equal to the send expressions.

- $\{(A_1, \prec_1, \vartheta_1), \dots, (A_n, \prec_n, \vartheta_n)\}$ is a set of disjoint LSC instance lines.

We set $Inst(L) := \{1, \dots, n\}$, $A_L := \bigcup_{i \in Inst(L)} A_i$, $\prec_L := \bigcup_{i \in Inst(L)} \prec_i$, and $\vartheta_L := \bigcup_{i \in Inst(L)} \vartheta_i$. We denote by a_i^\perp the minimum of \prec_i , $i \in Inst(L)$, also called instance head, and set $A_L^\perp := \{a_i^\perp \mid i \in Inst(L)\}$. By $A|_i := A \cap A_i$ we denote the projection of a set $A \subseteq A_L$ onto instance $i \in Inst(L)$.

If the LSC L is clear by context we shall simply write, e.g., \prec instead of \prec_L .

- $(m \in) Msg_L$ is a set of messages,

$$m = (a_s, a_r, \varsigma, \psi_s, \psi_r) \in A \times A \times Sync \times Expr_S \times Expr_S,$$

each comprising the message send and receive atoms a_s and a_r , the message synchronicity ς , and the message send and receive expressions $\psi_s = p(x_1, \dots, x_n)$ with $p \in \mathcal{P}_{snd}$ and $\psi_r = \chi(p)(x_1, \dots, x_n)$, $n \geq 0$, $x_i \in \mathcal{V}$. By $Msg_{inst}(L) := \{m \in Msg(L) \mid \varsigma(m) = inst\}$ and $Msg_{asyn}(L) := \{m \in Msg(L) \mid \varsigma(m) = asyn\}$ we denote the sets of instantaneous and asynchronous messages of L and set $atoms(m) := \{a_s, a_r\}$;

- $(c \in) Cond_L$ is a set of conditions,

$$c = (A_c, \kappa, \psi_c) \in (2^A \setminus \{\emptyset\}) \times Obl \times Expr_S,$$

each comprising the set of condition atoms A_c with at most one atom per instance line², i.e. $|(A_c|_i)| \leq 1$ for $i \in Inst(L)$, the condition mode κ , and the condition expression ψ_c . We set $atoms(c) := A_c(c)$;

- $(l \in) LocInv_L$ is a set of local invariants,

$$l = ((a_s, \gamma_s), (a_e, \gamma_e), \kappa, \psi) \in (A \times Cont) \times (A \times Cont) \times Obl \times Expr_S,$$

² in general, conditions are not limited to single instance lines as shown in Figure 1, but may span multiple instance lines; a condition spanning multiple instance lines synchronises the participating components

each comprising the local invariant start and end atoms a_s and a_e with containedness γ_s and γ_e , the local invariant mode κ , and the local invariant expression ψ . We set $atoms(l) := \{a_s, a_e\}$.

The set $elems(L) := Msg_L \cup Cond_L \cup LocInv_L$ is called the set of elements of L . To denote the components of a given element we use functional notations, like $a_s(m)$ for a message m , etc. These functions and ‘atoms’ are canonically extended from single elements to subsets of $elems(L)$ yielding sets of components and atoms, respectively, and we set $atoms(L) := atoms(elems(L))$.

A core LSC is called closed if and only if $atoms(elems(L)) \cup A_L^\perp = A_L$, i.e. if there are no atoms not used by elements except for instance heads. In the following we will only consider closed core LSCs. \diamond

Note that all expressions in an LSC may use variables from \mathcal{V} . Thereby we cover dynamic binding of core LSCs as introduced in [23, 24] as an extension of the static binding core LSCs in [3]. The example in Figure 1 is statically bound. To extend it, for example, to cover systems with four barriers, each having its own barrier controller, we would write ‘*barrier_down(b)*’ where b is a free variable ranging over the identities of barrier controllers in the system.

Definition 3 (LSC Specification). Let $Lsc = \{L_1, \dots, L_n\} \neq \emptyset$ be a set of core LSCs over signature \mathcal{S} . Lsc is then called core LSC specification over \mathcal{S} . \diamond

2.2 Semantics.

The central concept of the LSC semantics of [3] is the *cut* that represents how far each instance line has been observed. If the LSC has been activated and no element has been observed yet, then the cut is empty. Any other cut comprises at least one atom per instance line. It may comprise multiple atoms from one instance line if they all belong to the same core region.

Definition 4 (Cut). Let \mathcal{S} be a signature and L a core LSC over \mathcal{S} . A set of atoms $\alpha \subseteq atoms(L)$ is called cut if and only if

$$(i) \ \alpha \neq \emptyset \implies \forall i \in Inst(L) : \alpha|_i \neq \emptyset \text{ and } (ii) \ \forall a_1, a_2 \in \alpha : a_1 \bowtie a_2 \implies a_1 \not\prec^* a_2.$$

We call $\alpha_0 := \emptyset$ the initial cut, $\alpha_\perp(L) := A_L^\perp$ the instance heads cut, and $\alpha_{fin}(L)$, the maximal cut α with $\forall a \in \alpha \ \forall a' \in atoms(L) : a \prec^* a' \implies a' = a$, the final cut. The temperature of α , denoted by $\vartheta(\alpha)$, is ‘cold’ if $\alpha = \alpha_{fin}(L)$ or $\forall a \in \alpha : \vartheta(a) = cold$ and ‘hot’ otherwise. $Cuts(L)$ is the set of all cuts of L . \diamond

The unit by which a cut can be advanced is the *simultaneous class* (*simclass* for short). Simultaneity is transitively induced by synchronous messages and by conditions that span multiple instance lines. All atoms of these elements are supposed to be observed at the same point in time.

Definition 5 (Simclass). Let \mathcal{S} be a signature and L a core LSC over \mathcal{S} . Two atoms $a_1, a_2 \in atoms(L)$ are called simultaneous, denoted by $a_1 \sim a_2$, if and only if

- (i) $a_1 = a_2$, or (iii) $\exists e \in \text{Cond}(L) \cup \text{Msg}_{\text{inst}}(L) : \{a_1, a_2\} \subseteq \text{atoms}(e)$, or
(ii) $\{a_1, a_2\} \subseteq A_L^\perp$, or (iv) $\exists a_3 \in \text{atoms}(L) : a_1 \sim a_3 \wedge a_3 \sim a_2$.

For each $a \in \text{atoms}(L)$, $[a] := \{a' \in \text{atoms}(L) \mid a' \sim a\}$ denotes the equivalence class of 'a' with respect to \sim . The elements of the set $\text{Simclass}(L) := \text{atoms}(L) / \sim$ of all equivalence classes of atoms from $\text{atoms}(L)$ are called simclasses.

By $\text{elems}(scl) := \{e \in \text{elems}(L) \mid \text{atoms}(e) \cap scl \neq \emptyset\}$ we denote the set of LSC elements that share an atom from the simclass $scl \in \text{Simclass}(L)$. \diamond

A cut α can be advanced by observing a set of *enabled* simclasses. A simclass scl is enabled by a cut α if each atom in scl has all of its direct predecessors in α or belongs to a coregion and there is at least one other atom from the same coregion in α . The intuition of asynchronous messages is explicitly added by saying that a simclass is only enabled if for each asynchronous message receive atom in scl the sending has been observed in α or earlier. The *step function* formalises the advancement of α by a non-empty set of enabled simclasses. We note without a proof that Step_L yields a proper cut if applied to a cut and a non-empty set of enabled simclasses, and that it strictly advances the cut.

Definition 6 (Ready-set and Step_L). Let L be a core LSC over signature \mathcal{S} , $\alpha \in \text{Cuts}(L)$, and $scl \in \text{Simclass}(L)$. We say α enables scl , denoted $\alpha \triangleright scl$, if and only if

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

where $\text{prereq}(a) := \{a' \in \text{atoms}(L) \mid a' \prec a\}$ is the prerequisite of a .

A non-empty set of simclasses $\text{Scl} \subseteq \text{Simclass}(L)$ such that each simclass $scl \in \text{Scl}$ is enabled by α , i.e. $\alpha \triangleright scl$, is called *fired-set* of α . The set $\text{Ready}_L(\alpha)$ of all fired-sets of α is called the *ready-set* of α .

For $\emptyset \neq \{scl_1, \dots, scl_n\} \subseteq \text{Simclass}(L)$, the step function of L is defined as $\text{Step}_L(\alpha, \{scl_1, \dots, scl_n\}) := \text{Max}(\alpha \cup scl_1 \cup \dots \cup scl_n)$ where $\text{Max}(A) := A \setminus \{a \in A \mid \exists a' \in A : a \prec^+ a'\}$. \diamond

The semantics of an LSC L is defined in terms of \mathcal{A}_L , the Symbolic Automaton of its body. Symbolic Automata are a variant of Büchi automata whose transitions are labelled by expressions over a signature \mathcal{S} instead of by elements of an alphabet. They accept sequences of interpretations of the predicates in \mathcal{S} on a fixed universe and under a fixed valuation of the variables in \mathcal{S} .

Definition 7 (Symbolic Automata). A Symbolic Automaton 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 $\rightsquigarrow \subseteq Q \times \text{Expr}_{\mathcal{S}} \times Q$, and the accepting states $F \subseteq Q$. We write $q_i \rightarrow q_j$ if and only if $(q_i, \psi, q_j) \in \rightsquigarrow$ for some ψ and $q_i \xrightarrow{\sim} q_j$ if and only if $q_i \rightarrow q_j$ and $q_i \neq q_j$.

\mathcal{A} is called *partially ordered*, or POSA, if the reflexive transitive closure of \rightarrow is antisymmetric. It is called *deterministic* if $(q, \psi_1, q_1) \in \rightsquigarrow$ and $(q, \psi_2, q_2) \in \rightsquigarrow$, $q_1 \neq q_2$, implies $\mathcal{M}, \sigma \models \neg(\psi_1 \wedge \psi_2)$ for any \mathcal{M} and σ .

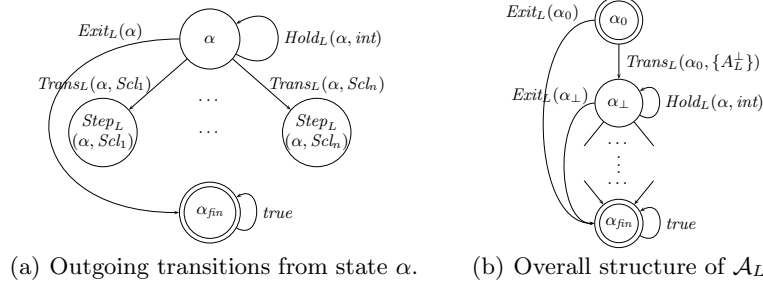


Fig. 3. Structure of the LSC body automaton. Double lined states are in F .

For a universe \mathcal{U} , $\vec{Int}_{\mathcal{U}}(\mathcal{S})$ is the set of all interpretation sequences, i.e. sequences $\vec{t} = \iota_0 \iota_1 \iota_2 \dots$ of interpretations ι_i of the predicates \mathcal{P} in \mathcal{S} . By $\vec{t}/k := \iota_k \iota_{k+1} \dots$ we denotes the suffix of \vec{t} starting at position k and set $\vec{t}^k := \iota_k$.

An infinite sequence $r = q_0 q_1 q_2 \dots$ of states $q_i \in Q$ is called a run of \mathcal{A} over \vec{t} under σ if and only if $q_0 = q_s$ and for $i \in \mathbb{N}_0$ there is a $(q_i, \psi, q_{i+1}) \in \rightsquigarrow$ such that $(\mathcal{U}, \iota_i), \sigma \models \psi$.

The set of runs of \mathcal{A} over \vec{t} under σ is denoted by $\Pi_\sigma^{\vec{t}}(\mathcal{A})$. The language accepted by \mathcal{A} is $\mathcal{L}_\sigma(\mathcal{A}) := \{\vec{t} \in \vec{Int}_{\mathcal{U}}(\mathcal{P}) \mid \exists r \in \Pi_\sigma^{\vec{t}}(\mathcal{A}) : inf(r) \cap F \neq \emptyset\}$ where $inf(r)$ is the set of states occurring infinitely often in r . \diamond

The states of \mathcal{A}_L are the cuts of L and each state gets three kinds of outgoing transitions, a self-loop, progress transitions to the following cuts, and a legal exit transition if possible conditions have to be considered (cf. Figure 3).

To construct the transition annotations, we use a number of abbreviations. The following five abbreviations select relevant elements from a simclass and, point-wise extended, from sets of simclasses.

$$\begin{aligned} Cond(scl) &:= Cond(L) \cap elems(scl), & Msg(scl) &:= Msg(L) \cap elems(scl), \\ Cond_{poss}(scl) &:= \{c \in Cond(scl) \mid \kappa(c) = poss\}, & Msg_{snd}(scl) &:= \\ & \{m \in Msg(scl) \mid a_s(m) \in scl\}, & Msg_{rcv}(scl) &:= \{m \in Msg(scl) \mid a_r(m) \in scl\}. \end{aligned}$$

A local invariant $l \in LocInv(L)$ affects the transition annotation of a cut α if it is *active beyond* α , i.e. $\exists a, a' \in \alpha : a_s(l) \prec^* a \wedge a' \prec^+ a_e(l)$ or if it is *active at* α , i.e. it ends inclusively at α or it is active beyond α and not starting exclusively at α . By $ali_{L,=}(\alpha)$ ($ali_{L,>}(\alpha)$) we denote all local invariants *active at* (*beyond*) α and by $ali_{L,=}^{poss}(\alpha)$ ($ali_{L,>}^{poss}(\alpha)$) those $l \in ali_{L,=}(\alpha)$ ($ali_{L,>}(\alpha)$) with $\kappa(l) = poss$.

The expression

$$A_{\{scl_1, \dots, scl_n\}} := \bigwedge (\psi_s(Msg_{snd}(scl_1 \cup \dots \cup scl_n)) \cup \psi_r(Msg_{rcv}(scl_1 \cup \dots \cup scl_n)))$$

characterise the simultaneous occurrence of all messages of simclasses scl_1, \dots, scl_n and

$$N_{\{scl_1, \dots, scl_n\}} := \neg \bigvee (\psi_s(Msg_{snd}(scl_1 \cup \dots \cup scl_n)) \cup \psi_r(Msg_{rcv}(scl_1 \cup \dots \cup scl_n)))$$

the absence of these messages where $\bigvee\{\psi_1, \dots, \psi_n\} := (\psi_1 \vee \dots \vee \psi_n)$, $\bigvee\emptyset := false$, $\bigwedge\{\psi_1, \dots, \psi_n\} := (\psi_1 \wedge \dots \wedge \psi_n)$, and $\bigwedge\emptyset := true$.

The first kind of transitions is labelled with a *hold predicate*

$$Hold_L(\alpha, int) := N_{ExclMsg(int)} \wedge \bigwedge \psi(al_{L,>}(\alpha))$$

which allows the automaton to remain in a state α while none of the awaited messages is observed and all local invariants active beyond α hold. The progress transitions are labelled with *transition predicates*

$$\begin{aligned} Trans_L(\alpha, int, Scl) := & A_{Scl} \wedge N_{ExclMsg(int) \setminus Scl} \\ & \wedge \bigwedge \psi_c(Cond(Scl)) \wedge \bigwedge \psi(al_{L,=} (Step_L(\alpha, Scl))) \end{aligned}$$

where $ExclMsg(weak) = \{scl \in Simclass(L) \mid \exists Scl \in Ready_L(\alpha) : scl \in Scl\}$ and $ExclMsg(strict) = Simclass(L)$, and $F = \{\alpha \in Cuts(L) \mid \vartheta(\alpha) = cold\}$, allows a transition from α to $\alpha' = Step_L(\alpha, Scl)$ if all messages required by the fired-set Scl and none of the messages from any other fired-set in the ready-set are observed and if the conditions co-located with the relevant messages and the local invariants active in α' hold. And legal exit transitions with *exit predicates*

$$\begin{aligned} Exit_L(\alpha, int) := & \bigvee_{scl \in Ready_L(\alpha)} \left[N_{\{scl\}} \wedge \neg \bigvee \psi(al_{L,>}^{poss}(\alpha)) \vee A_{\{scl\}} \wedge \right. \\ & \left. \left(\neg \bigwedge \psi_c(Cond_{poss}(scl)) \vee \neg \bigwedge \psi(al_{L,=}^{poss}(Step_L(\alpha, scl))) \right) \right] \end{aligned}$$

which allow to take the legal exit from α if a possible local invariant is violated while the awaited messages are not yet observed or if a possible condition or local invariant at a target cut is violated when observing the relevant messages. The disjunction over the complete ready-set avoids parallel exit edges in the automaton.

Definition 8. Given a core LSC L over signature \mathcal{S} with interpretation ‘ int ’, the Symbolic Automaton of L is $\mathcal{A}_L := (Q, q_s, \rightsquigarrow, F)$ with $Q = Cuts(L)$, $q_s = \alpha_0$,

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

Figure 4 shows the automaton \mathcal{A}_L of the body of the LSC from Figure 1 according to Definition 8, omitting the exit transitions annotated with *false* and all states not reachable from the initial cut.

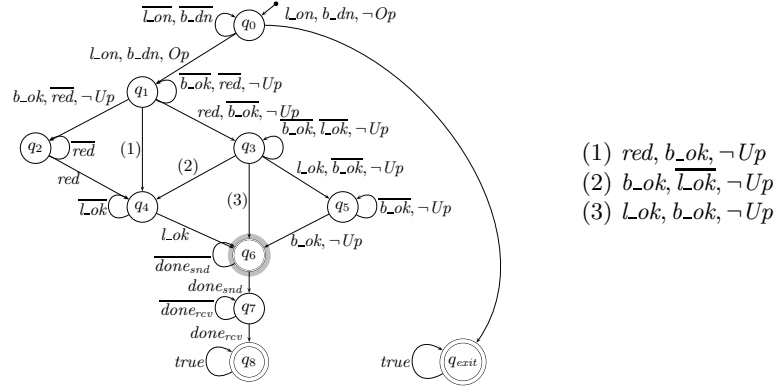


Fig. 4. For lack of space, message and condition names are abbreviated, negation of message-observation predicates is expressed by over-lining, and a comma is used for conjunction. E.g. q_0 's loop fires if neither 'lights_on' nor 'barrier_down' are observed.

Note that the interpretation *strict* or *weak* (called *tolerant* in [2]) has no effect on the structure of the automaton but only on the annotations. A *strict* LSC restricts the occurrence of the messages used in the LSC to exactly those points in time where they are supposed to occur according to the scenario. For example, if a level-crossing system sends 'red_on' twice (to play safe) then the system wouldn't satisfy Figure 1 in the strict interpretation. In the *weak* interpretation, the specification is satisfied if each necessary message occurs at least once where it is supposed to.

By the following Lemma, the Symbolic Automaton of a (bonded) LSC is a (deterministic) POSA. We provide the rather technical proof in the appendix.

Lemma 1 (POSA). *Let L be a core LSC over signature \mathcal{S} . Then \mathcal{A}_L is a POSA. If L is bonded, i.e. if all condition and local invariant atoms in L are co-located with at least one message atom, then \mathcal{A}_L is a deterministic POSA. \diamond*

In practice, LSCs are nearly always bonded. Non-bonded LSCs, i.e. those with loose conditions, are highly counter-intuitive and it is significantly harder to understand counter-examples obtained from model-checking because \mathcal{A}_L runs of a given counter-example need not be unique. Already [3] explicitly recommends to avoid loose conditions.

The semantics of a complete LSC L is obtained by quantifying the interpretation sequences accepted by its \mathcal{A}_L according to the activation mode and quantification.

Definition 9 (LSC Semantics). *Let $L = (\ell, ac, am, int, quant)$ be a core LSC over signature \mathcal{S} and \mathcal{U} a universe. A set of interpretation sequences $\vec{I} \subseteq \overrightarrow{Int}_{\mathcal{U}}(\mathcal{S})$ is said to satisfy L , denoted $\vec{I} \models_{LSC} L$, if and only if*

- *quant = existential and*

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

– *quant = universal and*

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

The language accepted by L is $\mathcal{L}(L) := \{\vec{I} \subseteq \vec{Int}_{\mathcal{U}}(\mathcal{P}) \mid \vec{I} \models_{Lsc} L\}$. \diamond

A set \vec{I} of interpretation sequences satisfies a core LSC specification Lsc if and only if it satisfies all core LSCs in Lsc . The language of Lsc is the intersection of the languages of the core LSCs in Lsc .

3 The Temporal Logics of Core LSCs

In order to cover symbolic LSCs with free variables, the temporal logic has to provide first-order quantification over logical variables. And in order to cover both, existential and universal LSCs, LTL is not sufficient but quantification over paths is necessary. Consequently, we basically use first-order CTL* as the destination temporal logic.

For convenience, the following definition already introduces a fragment of CTL* which we call FOP-CTL* (first-order prenex CTL*). Its expressive power is sufficient for our purposes since LSCs only need top-level path and logical quantifiers and the semantics (which is standard) can be explained using a set of (system) runs instead of a computation tree for general CTL*.

Definition 10 (FOP-CTL*). *The set of first-order prenex CTL* (FOP-CTL*) formulae over signature \mathcal{S} is defined by the grammar*

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

where $\psi \in \text{Expr}_{\mathcal{S}}$. We shall use the abbreviations $\wedge, \rightarrow, \leftrightarrow, F, G,$ and W . \diamond

The formulae we construct for core LSCs in Section 3.1 are in FOP-CTL* but we can identify further structure. We will find that general core LSCs translate to formulae from the FOP-CTL* fragment CSCTL and the bonded ones translate to DCSCCTL. Formulae of the latter fragment even provide enough information to establish a way back to LSCs (cf. Section 3.2).

Definition 11 (CSCTL, DCSCCTL). *The set of communication sequence FOP-CTL* (CSCTL) formulae over signature \mathcal{S} is*

$$\begin{aligned} \zeta &::= \xi_E \mid \xi_A \mid \zeta \wedge \zeta & \xi_E &::= E(\psi \rightarrow \pi) \mid EG(\psi \rightarrow \pi) \mid \exists x. \xi_E \\ \pi &::= \eta \text{U} \hat{\pi} \mid \eta W \hat{\pi} & \xi_A &::= A(\psi \rightarrow \pi) \mid AG(\psi \rightarrow \pi) \mid \forall x. \xi_A \\ \eta &::= \neg\mu \wedge \eta \mid \psi & \hat{\pi} &::= \hat{\pi}_1 \vee \hat{\pi}_2 \mid \tau \wedge X\pi \mid \text{false} \\ \tau &::= \neg\mu \wedge \tau \mid \mu \wedge \tau \mid \psi & \mu &::= p_{msg}(x_1, \dots, x_n) \end{aligned} \tag{1}$$

where $\psi \in \text{Expr}_{\mathcal{S}}$, $x_i \in \mathcal{V}$, $1 \leq i \leq n$, and $p_{msg} \in \mathcal{P}_{msg}$ is a message predicate.

Deterministic *CSCTL* (*DCSCTL*) comprises the formulae obtained from grammar (1) with the $\hat{\pi}$ production changed to

$$\hat{\pi} ::= \tau \vee \phi \mid (\tau \wedge \mathbf{X} \pi) \vee \phi, \phi ::= \text{false} \mid \tau \mid \phi_1 \vee \phi_2,$$

that satisfy

- (i) occurrences of $p \in \mathcal{P}_{snd}$ and $\chi^{-1}(p)$ in ξ_E and ξ_A are injectively related,
- (ii) if $\neg\mu_1$ and μ_2 occur on both sides of an \mathbf{U} or \mathbf{W} , then $\mu_1 = \mu_2$, and
- (iii) in each $\hat{\pi}$, any μ in τ is disjoint to ϕ , i.e. $\models \neg(\mu \wedge \phi)$. \diamond

3.1 From Core LSCs to Temporal Logic...

Lemma 2 (Schlör [19]). *Let $\mathcal{A} = (Q, q_s, \rightsquigarrow, F)$ 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 an LTL formula $\phi_{\mathcal{A}}$ over \mathcal{S} with $\vec{t} \in \mathcal{L}_{\sigma}(\mathcal{A}) \iff \vec{t}, \sigma \models \phi_{\mathcal{A}}$. \diamond*

The proof is by induction over the distance of states from the states whose only outgoing transitions are self-loops, in case of \mathcal{A}_L this is only $\alpha_{fin}(L)$. The POSA property ensures that the sequence of the sets of states with distance $1, 2, \dots$ is ascending with respect to \subseteq . The formula constructed in the course of the proof is recursively defined as $\phi_{\mathcal{A}} := \phi_{q_s}$ with

$$\phi_q := \psi(q, q) \mathbf{U}_q \bigvee_{q \hat{\rightarrow} q'} (\psi(q, q') \wedge \mathbf{X} \phi_{q'})$$

for $q \in Q$ where $\psi(q_1, q_2)$ denotes the (well-defined) transition predicate of a transition $q_1 \rightarrow q_2$ between locations $q_1, q_2 \in Q$. The temporal operator \mathbf{U}_q is \mathbf{W} (unless or weak until) if $q \in F$ and \mathbf{U} (until or strong until) otherwise.

As an example consider the outgoing transitions of a typical automaton state as shown in Figure 3(a). We may stay at state α as long as the hold condition $\text{Hold}_L(\alpha, int)$ holds and may leave α if any of the outgoing transitions can fire. Then, recursively, we may stay at the destination state as long as the destination state's hold condition holds etc. The formula directly follows this structure (cf. Figure 5).

Theorem 1. *Let L be a core LSC over signature \mathcal{S} . There is a CSCTL formula ϕ_L over \mathcal{S} with $\mathcal{L}(L) = \mathcal{L}(\phi_L)$. If L is bonded, then there is an equivalent formula in DCSCTL. \diamond*

$$\begin{aligned} \phi_{\alpha} = & \text{Hold}_L(\alpha, int) \mathbf{U} \left((\text{Trans}_L(\alpha, Scl_1) \wedge \mathbf{X} \phi_{\text{Step}_L(\alpha, Scl_1)}) \vee \dots \vee \right. \\ & \left. (\text{Trans}_L(\alpha, Scl_n) \wedge \mathbf{X} \phi_{\text{Step}_L(\alpha, Scl_n)}) \vee (\text{Exit}_L(\alpha) \wedge \mathbf{X} \phi_{\alpha_{fin}}) \right) \end{aligned}$$

Fig. 5. Schlör formula of the location α shown in Figure 3(a).

By Lemma 1, we can apply Lemma 2 to the automaton \mathcal{A}_L of an LSC, which represents the LSC body. Adding path and variable quantifiers according to the LSC activation mode and quantification completes the proof of the first claim. For example, an LSC $L = (\ell, ac, am, int, quant)$ with $am = invariant$ and $quant = universal$ over a signature $\mathcal{S} = (\{x_1, \dots, x_n\}, \mathcal{P}, \chi)$ becomes $\phi_L ::= \forall x_1 \dots \forall x_n. \mathbf{AG}(ac \rightarrow \phi_{\mathcal{A}_L})$. The second claim is established by a result of [19] that transforms the CSCTL formula to the desired DCSCTL form if the transition expressions, here $\psi(q, q')$, are mutually disjoint which is the case by Lemma 1. The finer structure of DCSCTL formulae is obtained by close examination of the construction of the translation relation \rightsquigarrow for \mathcal{A}_L . Theorem 1 extends to an LSC specification Lsc by conjoining the formulae of all LSCs in Lsc .

As a first observation, the formula for bonded LSCs is actually in deterministic ACTL, and thus in LTL, as also observed in [15]. They also claim that non-bonded LSCs in their interpretation are in LTL which is not the case for ours as non-bonded LSCs introduce non-determinism via self-loops annotated only with *true*. Restricted to messages, the core LSCs studied here and the kernel LSCs studied in [15] coincide (as expected).

LSCs are strictly weaker than general first-order CTL* as they can't express alternating path quantifiers [15]. The following lemma shows that there are simpler patterns not expressible by core (and kernel) LSCs. Intuitively, core (and kernel) LSCs only consider non-temporal properties as hold-conditions, i.e. before the “until” operator. Thus they can in general not express that some sub-scenario shall be repeated until the main-scenario continues. In contrast to [3], the LSC dialect of [2] provides (bounded and unbounded) loops, so the full version of [15] will show whether that extension is sufficient to make LSCs equivalent to LTL.

Lemma 3. *FOP-CTL* over \mathcal{S} is strictly more expressive than core LSCs.* \diamond

Proof. Assume there were an equivalent LSC for the formula $\varphi = (\mathbf{XX}p) \mathbf{U} q$ from the LTL fragment of CTL*. The interpretation sequence $\vec{t} = \bar{p}\bar{q} \bar{p}\bar{q} p q p \dots$ satisfies φ . Then ϕ_L , a formula equivalent to L , has by Theorem 1 a conjunctive term of the form $\mathbf{A}(\psi \rightarrow \eta \mathbf{U} \hat{\pi})$ or $\mathbf{A}(\psi \rightarrow \eta \mathbf{W} \hat{\pi})$ that is satisfied by \vec{t} . This implies $\bar{p}\bar{q} \rightarrow \psi$ and $\bar{p}\bar{q} \rightarrow \eta$ since ψ and η don't comprise temporal operators. Consequently $\vec{t}' \models L$ with $\vec{t}' = \bar{p}\bar{q} \bar{p}\bar{q} \bar{p}\bar{q} \bar{p}\bar{q} \dots$ but $\vec{t}' \not\models \varphi$ in contradiction to the equivalence assumption. \square

3.2 ...and Back

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

The constructive proof exploits that ζ is a DCSCCTL formula, i.e. has the properties (i)–(iii) from Definition 11. Intuitively, a sub-formula of ζ of the form

$$\pi = \underbrace{\neg\mu}_{\eta} \wedge \psi_1 \mathbf{U} ((\underbrace{\mu \wedge \psi_2}_{\tau} \wedge \mathbf{X} \pi_1) \vee \phi)$$

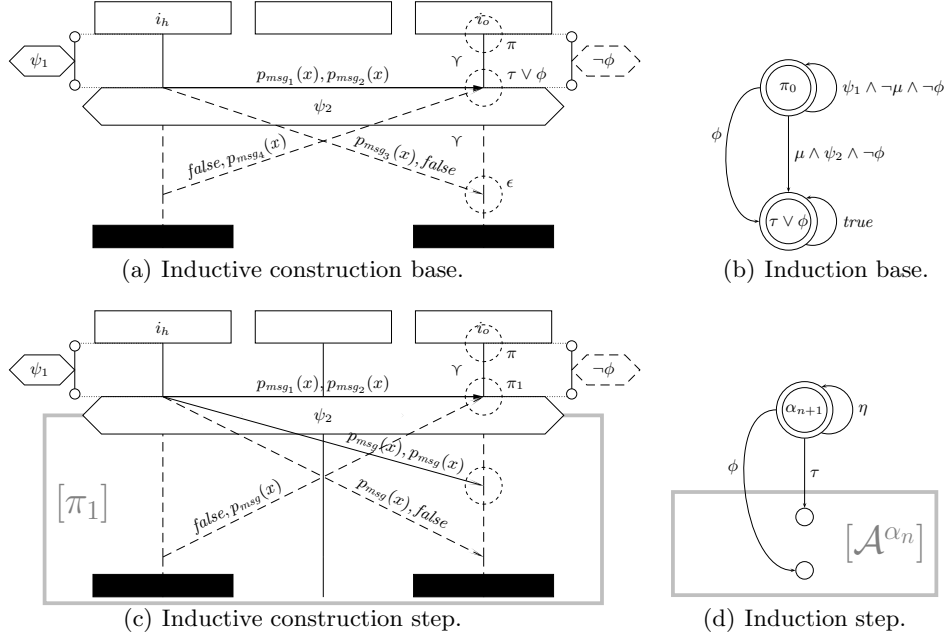


Fig. 6. Construction of an LSC from a DCSCCTL formula and its automaton structure.

says that η is supposed to hold until either τ is observed and the (system) run continues as required by π_1 , or ϕ holds which indicates that the current (system) run exhibits a different message order than the one accepted by π . If this different message order is legal, then there is another sub-formula of ζ that is satisfied by the (system) run. Transferring this intuition to LSCs, we inductively construct a core LSC specification of bonded core LSCs with (for convenience) two instance lines i_h and i_o . Each LSC accepts one particular message order, the atoms are pairs of the instance line name and the sub-formula that is observed up to this location. For example, the instance head atoms are (i_h, π) and (i_o, π) .

The expression ψ_1 in π becomes a mandatory local-invariant that is required to hold until τ , one or more messages co-located with a condition ψ_2 , is observed, unless a parallel cold local-invariant with expression $\neg\phi$ is violated, indicating that this core LSC can not accept the (system) run (cf. Figure 6(a) and 6(c)).

We omit the (tedious) formal inductive construction, but focus on the proof of language equivalence to the formula.

Proof. Part 2. Without loss of generality we assume that ζ accepts exactly one message order since the general case is a conjunction of such formulae. Let L be the core LSC specification constructed for ζ by the procedure described above. By construction, all instantaneous messages and conditions in L either share all atoms or none, so all simclasses and all consistent cuts (see below) are of the form $\{(i_h, \pi), (i_o, \pi)\}$.

We define the *nesting depth* of a formula by $dep(\eta U \tau \vee \phi) = 0$ and $dep(\eta U \tau \wedge \pi \vee \phi) = n + 1$ if $dep(\pi) = n$ with $U \in \{U, W\}$. There is at most one consistent cut $\{(i_h, \pi), (i_o, \pi)\} \in Cuts(L)$ with $dep(\pi) = n$. A cut is called *consistent* if $\forall a \in \alpha, a' \in [a] \exists a'' \in \alpha : a' \prec^* a''$.

Let $\mathcal{A}_L = (Q, q_s, \rightsquigarrow, F)$ be the automaton of L . The sequence $(T_n)_{n \in \mathbb{N}_0}$ with $T_n := \{q \in Q \mid dep(q) \leq n \wedge q \text{ consistent}\}$ is monotone. We prove by induction

$$\forall n \in \mathbb{N}_0 \forall \alpha_n \in T_n \forall \sigma \in Val_{\mathcal{U}}(\mathcal{S}), \vec{\tau} \in \overrightarrow{Int}_{\mathcal{U}}(\mathcal{P}) : \vec{\tau} \in \mathcal{L}_{\sigma}(\mathcal{A}_L^{\alpha_n}) \iff \vec{\tau}, \sigma \models \varphi_{\alpha_n},$$

where $\mathcal{A}_L^q := (Q, q, \rightsquigarrow, F)$ is the automaton that coincides with \mathcal{A}_L except for the start location $q \in Q$ and where $\varphi_{\alpha_n} := \pi$ if $\alpha_n = \{(i_h, \pi), (i_o, \pi)\}$.

Let N be the nesting depth of the largest π sub-formula of ζ . Then $\pi = \varphi_{\alpha_N}$. Examining the quantifiers in ζ and in the semantics of L according to Definition 9, we obtain the desired equivalence between L and ζ .

Induction base. T_0 comprises only the cut $\{(i_h, \pi_0), (i_o, \pi_0)\}$ with $\pi_0 = \eta W(\tau \vee \phi) = (\neg\mu \wedge \psi_1) W(\mu \wedge \psi_2 \vee \phi)$ (the until ('U') case follows analogously). The automaton $\mathcal{A}_L^{\alpha_0}$ is depicted in Figure 6(b). Its corresponding formula $\phi_{\mathcal{A}_L^{\alpha_0}}$ obtained by the construction of Lemma 2 is

$$\begin{aligned} \phi_{\mathcal{A}_L^{\alpha_0}} &= (\neg\mu \wedge \psi_1 \wedge \neg\phi) W(\mu \wedge \psi_2 \wedge X(true W false) \vee \phi \wedge X(true W false)) \\ &\iff (\neg\mu \wedge \psi_1 \wedge \neg\phi) W(\mu \wedge \psi_2 \vee \phi) \iff (\neg\mu \wedge \psi_1) W(\mu \wedge \psi_2 \vee \phi) = \pi_0. \end{aligned}$$

Induction step. Now let $\alpha_{n+1} = \{(i_h, \pi), (i_o, \pi)\} \in T_{n+1}$ with

$$\pi = \eta W(\tau \wedge X(\pi_1) \vee \phi) = (\neg\mu \wedge \psi_1) W(\mu \wedge \psi_2 \wedge X(\pi_1) \vee \phi)$$

('U' case analogously). By construction of \prec_L , the only simclass that can possibly be enabled by α_{n+1} is $scl := \{(i_h, \pi_1), (i_o, \pi_1)\}$. It is actually enabled since all its prerequisites are obviously part of α_{n+1} and all the sendings of asynchronous receptions lie strictly before α_{n+1} by definition of DCSCCTL (Definition 11.(i)) and construction of L . By definition of the $Step_L$ function, α_{n+1} has a unique successor cut, namely $Step_L(\alpha_{n+1}, \{scl\}) = scl = \alpha_n$ with $dep(\varphi_{\alpha_n}) = n$.

The automaton $\mathcal{A}_L^{\alpha_{n+1}}$ is shown in Figure 6(d). The τ transition leads to the state $Step_L(\alpha_{n+1}, \{scl\})$ which we just identified to lie in T_n and the ϕ transition leads to the final cut that lies in T_0 by definition. Hence we can use the induction hypothesis as follows, where (*) exploits the style of outgoing transitions of α_{n+1} :

$$\begin{aligned} \vec{\tau}, \sigma \models \varphi_{\alpha_{n+1}} &\iff \vec{\tau}, \sigma \models \eta W(\tau \wedge X(\varphi_n) \vee \phi) \\ &\iff \vec{\tau}, \sigma \models G(\eta) \vee (\eta U(\tau \wedge X(\varphi_n) \vee \phi)) \\ &\iff \forall j \in \mathbb{N}_0 : \vec{\tau}^j \models \eta \vee (\exists k \in \mathbb{N}_0 : (\forall 0 \leq j < k : \vec{\tau}^j, \sigma \models \eta) \\ &\quad \wedge (\vec{\tau}^{k+1}, \sigma \models \phi \vee \vec{\tau}^{k+1}, \sigma \models \tau \wedge \vec{\tau}/k + 2, \sigma \models \varphi_{\alpha_n})) \\ &\iff (\exists r \in \Pi_{\sigma}^{\vec{\tau}}(\mathcal{A}_L^{\alpha_{n+1}}) : r = \alpha_{n+1} \alpha_{n+1} \alpha_{n+1} \dots \\ &\quad \vee (\exists \tilde{r} \in \Pi_{\sigma}^{\vec{\tau}}(\mathcal{A}_L^{\alpha_{fin}(L)}) \exists k > 0 : r = \alpha_{n+1}^k \tilde{r} \wedge inf(\tilde{r}) \cap F \neq \emptyset) \\ &\quad \vee (\exists \tilde{r} \in \Pi_{\sigma}^{\vec{\tau}}(\mathcal{A}_L^{\alpha_n}) \exists k > 0 : r = \alpha_{n+1}^k \tilde{r} \wedge inf(\tilde{r}) \cap F \neq \emptyset)) \\ &\stackrel{(*)}{\iff} \exists r \in \Pi_{\sigma}^{\vec{\tau}}(\mathcal{A}_L^{\alpha_{n+1}}) : inf(r) \cap F \neq \emptyset \iff \vec{\tau} \in \mathcal{L}_{\sigma}(\mathcal{A}_L^{\alpha_{n+1}}) \quad \square \end{aligned}$$

Recall that intuitively the translation considers single paths through the LSC specification, that is a translation back and forth yields as many LSCs as there are paths through the original ones. Thus the back-translation makes the possible paths and combinations of conditions and local invariants explicitly visible and thus may help in the debugging of LSC specifications. But in presence of concurrency introduced, for instance, by independent parts or coregions, a single original LSC can be exponentially more succinct than the back and forth translation.

4 Conclusion

Our new concise formalisation of the LSCs of [3] makes it possible to compare these LSCs to temporal logic. General core LSCs are at most as powerful as the fragment CSCTL of FOP-CTL* for which we provided a syntactical characterisation. The practically relevant set of *bonded* core LSCs is exactly as powerful as the smaller fragment DCSCTL because we can construct an LSC specification for a given formula. The embedding into first-order prenex CTL* is strict even without resorting to nesting of path quantifiers [15].

These results have a number of applications. Section 3.1 formally justifies the practice of LSC model-checking using formulae [3] and allows to compare both dialects of LSCs that emerged from the original proposal [1]. Section 3.2 provides for the first time an instrument to decide whether a given formula has an equivalent LSC specification. This is useful since experts in formal methods sometimes easier come up with a formula for a given requirement while for discussion within a more general audience it is highly desirable to present the requirement in form of LSCs. Another aspect stems from the observation of [25] that their distributed LTL model-checker performs extraordinarily well on formulae that are a chain of right-nested “Until” operators without noticing that this is exactly the structure of (D)CSCTL. Section 3 would justify a restriction of their input language to (D)CSCTL, thus obtaining an LSC model-checker, and raise the question whether this restriction could yield further speedup. Finally, the fact that we use results from the theory of Symbolic Timing Diagrams [19, 22] (STD) raises the question whether STDs also qualify as a scenario language and, for example, whether they can be played out [26].

Further work comprises the extension to the full LSC language of [3], that is, pre-charts, real-time, and possible asynchronous messages, as outlined in the introduction of Section 2.

Acknowledgements. The authors want to express their gratitude to Matthias Brill and Hartmut Wittke for clarifying discussions on the intricacies of LSCs, sharing of expertise, and valuable hints on related literature.

Note. An abridged version of this paper appeared at the SofSem’06 poster session [27].

References

1. Damm, W., Harel, D.: LSCs: Breathing Life into Message Sequence Charts. *Formal Methods in System Design* **19** (2001) 45–80
2. Harel, D., Marelly, R.: *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer (2003)
3. Klose, J.: *Live Sequence Charts: A Graphical Formalism for the Specification of Communication Behavior*. PhD thesis, C. v. O. Universität Oldenburg (2003)
4. Weidenhaupt, K., Pohl, K., Jarke, M., Haumer, P.: Scenarios in system development: Current practice. *IEEE Software* **15** (1998) 34–45
5. Amyot, D., Eberlein, A.: An evaluation of scenario notations and construction approaches for telecommunication systems development. *Telecommunications Systems Journal* **24** (2003) 61–94
6. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: *ICSE'99. Proceedings of the 1999 International Conference on Software Engineering*, May 16-22, 1999, Los Angeles, CA, USA., ACM (1999) 411–420
7. Bitsch, F.: Safety patterns - the key to formal specification of safety requirements. In Voges, U., ed.: *Computer Safety, Reliability and Security, 20th International Conference, SAFECOMP 2001, Budapest, Hungary, September 26-28, 2001, Proceedings*. Volume 2187 of *Lecture Notes in Computer Science.*, Springer (2001) 176–190
8. ITU-T: *ITU-T Rec. Z.120: Message Sequence Chart (MSC)*. ITU-T, Geneva (1999)
9. Knieke, C., Huhn, M., Goltz, U.: Modelling and simulation of an automotive system using LSCs. In Houmb, S.H., Jürjens, J., eds.: *Proc. CSDUML'2005, TUM (2005)* 0–0 TUM-TR.
10. Combes, P., Harel, D., Kugler, H.: 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)
11. Bunker, A., Gopalakrishnan, G., Slind, K.: Live Sequence Charts applied to hardware requirements specification and verification: A VCI bus interface model. *Software Tools for Technology Transfer* **7** (2004) 341–350
12. Bontemps, Y., Heymans, P., Kugler, H.: Applying LSCs to the specification of an air traffic control system. In: *Proc. SCESM'03*. (2003)
13. Bohn, J., Damm, W., Wittke, H., Klose, J., Moik, A.: Modelling and validating train system applications using statemate and live sequence charts. In: *Proc. IDPT 2002, Society for Design and Process Science* (2002)
14. Bontemps, Y.: *Relating Inter-Agent and Intra-Agent Specifications*. PhD thesis, University of Namur (Belgium) (2005)
15. Kugler, H., Harel, D., Pnueli, A., Lu, Y., Bontemps, Y.: Temporal logic for scenario-based specifications. In Halbwachs, N., Zuck, L.D., eds.: *Proc. TACAS 2005*. Volume 3440 of *LNCS*. (2005)
16. Klose, J., Toben, T., Westphal, B., Wittke, H.: Check it out: On the efficient formal verification of Live Sequence Charts. In Ball, T., Jones, R.B., eds.: *Proc. CAV 2006*. Volume 4144 of *Lecture Notes in Computer Science.*, Springer-Verlag (2006) 219–233
17. Schinz, I., Toben, T., Mrugalla, C., Westphal, B.: The Rhapsody UML Verification Environment. In Cuellar, J.R., Liu, Z., eds.: *Proc. SEFM 2004*. (2004) 174–183
18. Klose, J., Wittke, H.: An automata based interpretation of Live Sequence Charts. In Margaria, T., Yi, W., eds.: *Proc. TACAS 2001*. Number 2031 in *LNCS* (2001) 512–527

19. Schlör, R.C.: Symbolic Timing Diagrams: A Visual Formalism for Model Verification. PhD thesis, C. v. O. Universität Oldenburg (2000)
20. Maidl, M.: The common fragment of ctl and ltl. In: IEEE Symp. on Foundations of Computer Science. (2000) 643–652
21. Westphal, B., Toben, T.: The good, the bad and the ugly: Well-formedness of Live Sequence Charts. In Baresi, L., Heckel, R., eds.: Proc. FASE 2006. Volume 3922 of Lecture Notes in Computer Science., Springer-Verlag (2006) 230–246
22. Feyerabend, K., Josko, B.: A visual formalism for real time requirement specification. In Bertran, M., Rus, T., eds.: Proc. ARTS'97. Volume 1231 of LNCS. (1997) 158–168
23. Klose, J., Westphal, B.: Relating LSC specifications to UML models. In Ehrig, H., Grosse-Rhode, M., eds.: Proc. INT'02. (2002)
24. Damm, W., Westphal, B.: Live and let die: LSC-based verification of UML-models. Science of of Computer Programming **55** (2005) 117–159
25. Barnat, J.: Distributed Memory LTL Model Checking. PhD thesis, Faculty of Informatics, Masaryk University Brno (2004)
26. Harel, D.: personal communication (2005)
27. Toben, T., Westphal, B.: On the expressive power of LSCs. In Wiedermann, J., Tel, G., Pokorný, J., Bieliková, M., Štuller, J., eds.: Proc. SofSem 2006. Volume 2., Institute of Computer Science AS CR, Prague (2006) 33–43

Appendix

Definition 12. Let L be a core LSC over signature \mathcal{S} and $\alpha, \alpha' \in \text{Cuts}(L)$. The cut α' is said to be larger than α , denoted by $\alpha \sqsubset \alpha'$, if and only if

$$\begin{aligned} & \forall i \in \text{Inst}(L) \forall a \in \alpha|_i \exists a' \in \alpha'|_i : \\ & a \prec^* a' \wedge \exists i \in \text{Inst}(L) : \alpha|_i \subsetneq \alpha'|_i \vee \alpha' \neq \emptyset \wedge \forall a \in \alpha|_i, a' \in \alpha'|_i : a \prec^+ a'. \end{aligned}$$

Note that \sqsubset is a strict partial order (irreflexive, asymmetric, and transitive). \diamond

Proof of Lemma 1.

Part 1. Let $\mathcal{A}_L = (Q, q_s, \rightsquigarrow, F)$. We have to prove antisymmetry of \rightarrow^* , i.e. $(q \rightarrow^* q') \wedge (q' \rightarrow^* q) \implies q = q'$ for $q, q' \in Q$. Consider the interesting case where $q \neq \alpha_{\text{fin}}(L) \neq q'$. Then there is a minimal $n \in \mathbb{N}_0$ such that

$$q_0 \rightarrow \dots \rightarrow q_{k-1} \rightarrow q_k \rightarrow q_{k+1} \rightarrow \dots \rightarrow q_n$$

with $q_0 = q_n = q$ and $q_k = q'$ and none of the transitions is a self-loop, i.e. $q_k \neq q_{k+1}$, $0 \leq k < n$.

Also none of the transitions is an exit loop, i.e. a transition annotated by Exit_L , since then one of intermediate states were the final cut $\alpha_{\text{fin}}(L)$, thus $q = \alpha_{\text{fin}}(L)$ by (*) in contradiction to the assumption.

Thus all transitions are by regular transitions defined by Step_L , i.e. there are fired-sets $\text{Scl}_k \in \text{Ready}_L(q_k)$ such that $q_{k+1} = \text{Step}_L(q_k, \text{Scl}_k)$, $0 \leq k < n$. Since Step_L strictly advances the cut, we have $q_k \sqsubset q_{k+1}$, $0 \leq k < n$ and thus

$$q = q_0 \sqsubset q_1 \sqsubset \dots \sqsubset q_{n-1} \sqsubset q_n = q$$

in contradiction to the fact that the cut order is a strict partial order.

Part 2. If L is bonded, i.e. $\forall e \in Cond \cup LocInv \forall a \in atoms(e) \exists m \in Msg : \kappa(m) = mand \wedge [a] \cap atoms(m) \neq \emptyset$, we have that every simclass Scl contains at least one message send or message receive atom.

To prove determinism of \mathcal{A}_L , we have to show for all $\alpha \in Q$

- (i) $\models \neg(Hold_L(\alpha, int) \wedge Trans_L(\alpha, , intf))$,
- (ii) $\models \neg(Trans_L(\alpha, int, f_1) \wedge Trans_L(\alpha, int, f_2))$,
- (iii) $\models \neg(Hold_L(\alpha, int) \wedge Exit_L(\alpha, int))$, and
- (iv) $\models \neg(Exit_L(\alpha, int) \wedge Trans_L(\alpha, int, f))$

for arbitrary $f, f_1, f_2 \in Ready_L(\alpha)$ with $f_1 \neq f_2$. Most of the cases can be proved by exploiting the observation that $\models \neg(A_{\{Scl\}} \wedge N_{\{Scl\}})$ if $Msg_{snd}(Scl) \cup Msg_{rcv}(Scl) \neq \emptyset$. In the other cases, the condition and local invariant expressions ensure the mutual disjointness of the transition predicates. \square