Temporal Logic for Scenario-Based Specifications

Hillel Kugler[1], David Harel[2], Amir Pnueli[1,2] *, Yuan Lu[3], and Yves Bontemps[4]

[1] New York University, New York, NY, USA
{kugler,amir}@cs.nyu.edu
[2] The Weizmann Institute of Science, Rehovot, Israel
dharel@weizmann.ac.il
[3] Broadcom Corp., San Jose, CA, USA
ylu@broadcom.com
[4] University of Namur, Namur, Belgium
ybo@info.fundp.ac.be **

Abstract. We provide semantics for the powerful scenario-based language of live sequence charts (LSCs). We show how the semantics of live sequence charts can be captured using temporal logic. This is done by studying various subsets of the LSC language and providing an explicit translation into temporal logic. We show how a kernel subset of the LSC language (which omits variables, for example) can be embedded within the temporal logic CTL*. For this kernel subset the embedding is a strict inclusion. We show that existential charts can be expressed using the branching temporal logic CTL while universal charts are in the intersection of linear temporal logic and branching temporal logic LTL ∩ CTL. Since our translations are efficient, the work described here may be used in the development of tools for analyzing and executing scenario-based requirements and for verifying systems against such requirements.

1 Introduction

Understanding system and software behavior by looking at various "stories" or scenarios seems a promising approach, and it has focused intensive research efforts in the last few years. One of the most widely used languages for specifying scenario-based requirements is that of message sequence charts (MSCs), adopted long ago by the ITU [26], or its UML variant, sequence diagrams [25]. This paper addresses the relationship between scenario-based requirements and temporal logic [23]. As a scenario based language we focus on the language of live sequence charts (LSCs) [7] which is a powerful extension of classical message sequence charts.

LSCs distinguish between behaviors that may happen in the system (existential) from those that must happen (universal). A universal chart contains a

* This research was supported in part by the John von Neumann Minerva Center for the Verification of Reactive Systems, by the European Commission project OMEGA (IST-2001-33522) and by the Israel Science Foundation (grant No. 287/02-1).

** FNRS Research Fellow

prechart, which specifies the scenario which, if successfully executed, forces the system to satisfy the scenario given in the actual chart body.

Our contribution focuses on providing semantics for the powerful scenario-based language of live sequence charts, but the underlying approach and ideas are more general and can also be applied to other scenario based approaches including classical MSCs, UML sequence diagrams, triggered message sequence charts [24] and other variations. We show how the semantics of live sequence charts can be captured using temporal logics. This is done by studying various subsets of the LSC language and providing an explicit translation to temporal logic. We also show how some of the popular temporal logic "patterns" can be specified using live sequence charts.

In addition to gaining a better theoretical understanding of scenario-based languages, another motivation for this work is the development of tools for analyzing scenario based requirements and verifying systems against these requirements. Since our translations are efficient, the work described here may be used in tools that verify that a system satisfies a requirement specified using LSCs, in tools for executing scenarios directly, as suggested by the play-in/play-out approach [14, 15] and smart play-out [13], and in testing and synthesis tools.

2 Live Sequence Charts

2.1 Overview

Live sequence charts (LSCs) [7] have two types of charts: *universal* (annotated by a solid borderline) and *existential* (annotated by a dashed borderline). Universal charts are used to specify restrictions over all possible system runs. A universal chart typically contains a *prechart*, that specifies the scenario which, if successfully executed, forces the system to satisfy the scenario given in the actual chart body. Existential charts specify sample interactions between the system and its environment, and must be satisfied by at least one system run. They thus do not force the application to behave in a certain way in all cases, but rather state that there is at least one set of circumstances under which a certain behavior occurs. Existential charts can be used to specify system tests, or simply to illustrate longer (non-restricting) scenarios that provide a broader picture of the behavioral possibilities to which the system gives rise.

We will use an example of a cellular phone system to illustrate the main concepts and constructs of the language. The chart `OpenCover` appearing in Fig. 1 requires that whenever the user opens the `Cover`, as specified in the prechart (dashed hexagon), the `Speaker` must turn silent. Both the messages `Open` sent from the User to the Cover, and the self message `Sound(Silent)` of the Speaker are synchronous messages as denoted by the close triangular arrowheads.

The chart `CloseCover` appearing in Fig. 2 requires that whenever the user closes the `Cover`, The `Chip` will send the message `MakeSound(Silent)` to the `Speaker` and later the speaker will turn silent as designated by the self message

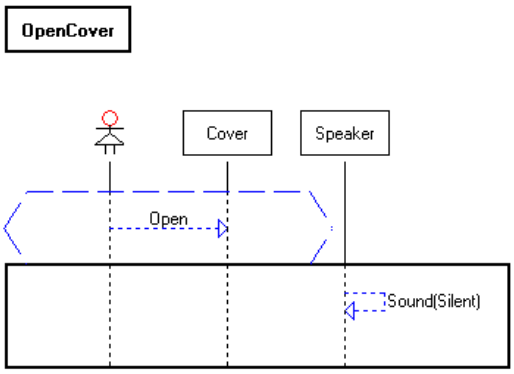

Fig. 1. Open Cover

`Sound(Silent)`. The `Display` should set its state to `Time` and later set its background to `Green`. An LSC induces a partial order which is determined by the order along an instance line, by the fact that a message can be received only after it is sent, and taking into account that a synchronous message blocks the sender until receipt. Thus in Fig. 2, message `ChangeBackground(Green)` must occur after message `SetState(Time)`, but both are unordered with respect to messages `MakeSound(Silent)` and `Sound(Silent)`. In the chart appearing in Fig. 2 all messages are synchronous, except message `MakeSound(Silent)` which is asynchronous, as denoted by the open arrowhead.

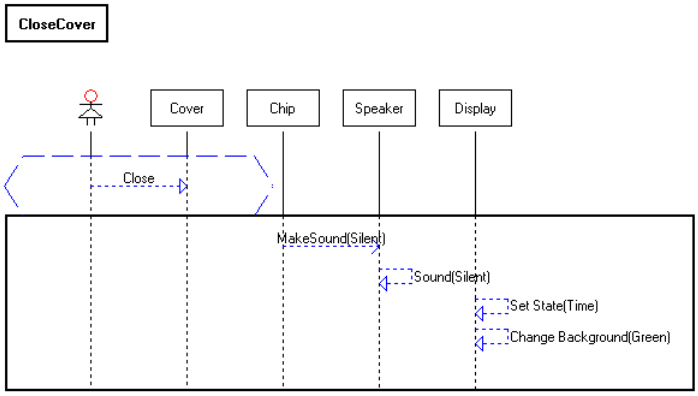

Fig. 2. Close Cover

The chart `Call911` appearing in Fig. 3 is an existential chart as denoted by the dashed borderline. It describes a scenario in which a user calls the number 911, opens the antenna and the call is answered. The chart in Fig. 3 introduces a

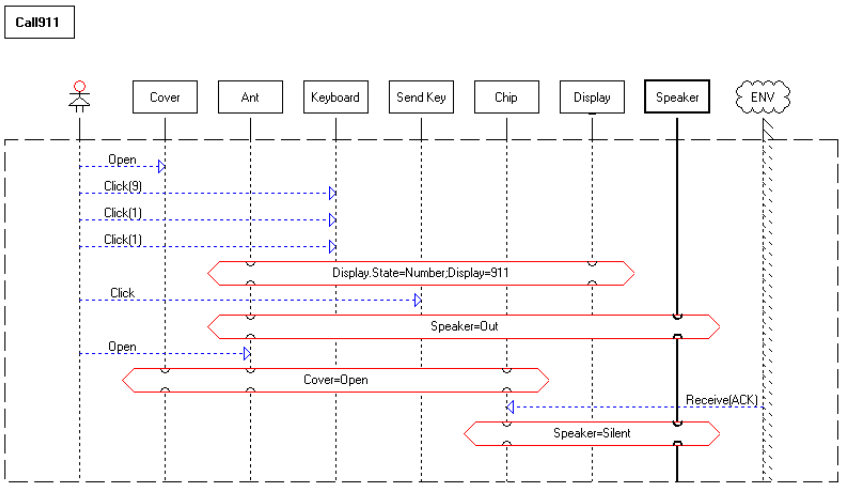

Fig. 3. Call 911

new element – a condition – denoted by a hexagon. The conditions in this chart are hot conditions, specifying assertions that must hold for the scenario to be satisfied. Existential charts do not have a prechart, and the meaning is that this is a possible scenario, that should be satisfied by at least one system run.

So far we have shown the main LSC features, which are at the focus of our research in this paper, and are also the ones most widely used. The LSC language supports additional features making it a rich and complex language. For a detailed description of live sequence charts the reader is referred to [7, 14].

2.2 Trace-based semantics for LSCs

This section defines the languages specified by a set of LSCs. Later in this paper we show how to provide equivalent semantics using temporal logics. For the ease of presentation and due to space limitations, the LSC definitions appearing here are a restricted and simplified version of the original LSC semantics [7]. These definitions provide the key ideas and concepts, allowing the reader to understand extensions that will be explained as we go along. The concept of an execution of a chart which is defined here will be used later in the temporal logic constructions.

We assume the LSC specification relates to an **object system** composed of a set of objects $\mathcal{O} = \{O_1 \dots O_n\}$. An object system corresponds to an implementation, and our goal while providing semantics for LSCs is to define if a given object system satisfies an LSC specification. The instance identifiers in the LSC charts are objects from $\mathcal{O}$, and possibly also the environment denoted *env*. The LSC specifies the behavior of the system in terms of the message communication between the objects in the system. We want to define the notion of satisfiability of an LSC specification. In other words, we want to capture the languages

$\mathcal{L} \subseteq A^* \cup A^\omega$ generated by the object systems that satisfy the LSC specification. The alphabet A used defines message communication between objects, $A = \mathcal{O} \times (\mathcal{O}.\Sigma)$ where Σ is the alphabet of messages.

Let $inst(m)$ be the set of all instance-identifiers referred to in chart m. With each instance i we associate a finite number of locations $dom(m,i) \subseteq \{0, ..., l\_max(i)\}$. We collect all locations of m in the set

$$dom(m) = \{< i, l >| \; i \in inst(m) \wedge l \in dom(m,i)\}$$

The messages appearing in m are triples

$$Messages(m) = dom(m) \times \Sigma \times dom(m),$$

where $(< i, l >, \sigma, < i', l' >)$ corresponds to instance i, while at location l, sending σ to instance i' at location l'. Each location can appear in at most one message in the chart. The relationship between locations and messages is given by the mapping

$$msg(m) : dom(m) \rightarrow Messages(m)$$

The msg function induces two Boolean predicates *send* and *receive*. The predicate *send* is true only for locations that correspond to the sending of a message, while the predicate *receive* is true only for locations that correspond to the receiving of a message. We define the binary relation $R(m)$ on $dom(m)$ to be the smallest relation satisfying the following axioms and closed under transitivity and reflexivity:

- order along an instance line:

$$\forall < i, l >\in dom(m), l < l\_max(i) \Rightarrow < i, l > R(m) < i, l+1 >$$

- order induced from message sending:

$$\forall msg \in Messages(m), msg = (< i, l >, \sigma, < i', l' >) \Rightarrow$$

$$< i, l > R(m) < i', l' >$$

- messages are synchronous; they block sender until receipt:

$$\forall msg \in Messages(m), msg = (< i, l >, \sigma, < i', l' >) \Rightarrow$$

$$< i', l' > R(m) < i, l+1 >$$

We say that the chart m is **well-formed** if the relation $R(m)$ is acyclic. We assume all charts to be well-formed, and use $\leq_m$ to denote the partial order $R(m)$.

We denote the **preset** of a location $< i, l >$ containing all elements in the domain of a chart smaller than $< i, l >$ by

$$^\bullet < i, l >= \{< i', l' >\in dom(m)| < i', l' > \leq_m < i, l >\}.$$

We denote the partial order induced by the order along an instance line by $\prec_m$, thus $< i,l > \prec_m < i',l' >$ iff $i = i'$ and $l < l'$.

A **cut** through m is a set c of locations, one for each instance, such that for every location $< i,l >$ in c, the preset $^\bullet < i,l >$ does not contain a location $< i',l' >$ such that $< j,l_j > \prec_m < i',l' >$ for some location $< j,l_j >$ in c. A cut c is specified by the locations in all of the instances in the chart:

$$c = (< i_1,l_1 >, < i_2,l_2 >, ..., < i_n,l_n >)$$

For a chart m with instances $i_1, ..., i_n$ the **initial cut** c_0 has location 0 in all the instances. Thus, $c_0 = (< i_1,0 >, < i_2,0 >, ..., < i_n,0 >)$. We denote $cuts(m)$ the set of all cuts through the chart m.

For chart m, some $1 \leq j \leq n$ and cuts c, c', with

$$c = (< i_1,l_1 >, < i_2,l_2 >, ..., < i_n,l_n >), c' = (< i_1,l'_1 >, < i_2,l'_2 >, ..., < i_n,l'_n >)$$

we say that c' is a $< j,l_j >$**-successor** of c, and write $succ_m(c, < j,l_j >, c')$, if c and c' are both cuts and

$$l'_j = l_j + 1 \ \wedge \ \forall i \neq j, l'_i = l_i$$

Notice that the successor definition requires that both c and c' are cuts, so that advancing the location of one of the instances in c is allowed only if the obtained set of locations remains unordered.

A **run** of m is a sequence of cuts, $c_0, c_1, ..., c_k$, satisfying the following:

- c_0 is an initial cut.
- for all $0 \leq i < k$, there is $1 \leq j_i \leq n$, such that $succ_m(c_i, < j_i, l_{j_i} >, c_{i+1})$.
- in the final cut c_k all locations are maximal.

Assume the natural mapping f between $(dom(m) \cup env) \times \Sigma \times dom(m)$ to the alphabet A, defined by

$$f(< i,l >, \sigma, < j,l' >) = (O_i, O_j.\sigma)$$

Intuitively, the function f maps a location to the sending object and to the message of the receiving object. Using this notation, $f(Messages(m))$ will be used to denote the letters in A corresponding to messages that are restricted by chart m:

$$f(Messages(m)) = \{f(v) \mid v \in Messages(m)\}$$

Definition 1. *Let $c = c_0, c_1, ..., c_k$ be a run. The* **execution trace***, or simply the* **trace** *of c, written $w = trace(c)$, is the word $w = w_1 \cdot w_2 \cdots w_k$ over the alphabet A, defined by:*

$$w_i = \begin{cases} f(msg(m)(< j,l_j >)) & \text{if } succ_m(c_{i-1}, < j,l_j >, c_i) \ \wedge \ send(< j,l_j >) \\ \epsilon & \text{otherwise} \end{cases}$$

We define the **trace language** *generated by chart m, $\mathcal{L}_m^{trc} \subseteq A^*$, to be*

$$\mathcal{L}_m^{trc} = \{w \mid \exists (c_0, c_1, ..., c_k) \in \text{Runs(m) s.t. } w = trace(c_0, c_1, ..., c_k)\}$$

There are two additional notions that we associate with an LSC, its **mode** and its **activation message**. These are defined as follows:

$$mod : m \rightarrow \{existential, universal\}$$

$$amsg : m \rightarrow dom(m) \times \Sigma \times dom(m)$$

The activation message of a chart designates when a scenario described by the chart should start, as we describe below. The charts and the two additional notions are now put together to form a specification. An **LSC specification** is a triple

$$LS = \langle M, amsg, mod \rangle,$$

where M is a set of charts, and $amsg$ and mod are the activation messages and modes of the charts, respectively.

The **language** of the chart m, denoted by $\mathcal{L}_m \subseteq A^* \cup A^\omega$, is defined as follows:

For an existential chart, $mod(m) = existential$, we require that the activation message is relevant (i.e., sent) at least once, and that the trace will then satisfy the chart:

$$\begin{aligned}\mathcal{L}_m = \{w = w_1 \cdot w_2 \cdots \mid & \exists i_0, i_1, ..., i_k \text{ and } \exists v = v_1 \cdot v_2 \cdots v_k \in \mathcal{L}_m^{trc}, \text{ s.t.} \\ & (i_0 < i_1 < ... < i_k) \wedge (w_{i_0} = f(amsg(m))) \wedge \\ & (\forall j, 1 \leq j \leq k, w_{i_j} = v_j) \wedge \\ & (\forall j', i_0 \leq j' \leq i_k, j' \notin \{i_0, i_1, ..., i_k\} \Rightarrow w_{j'} \notin f(Messages(m)))\}\end{aligned}$$

The formula requires that the activation message is sent once ($w_{i_0} = f(amsg(m))$), and then the trace satisfies the chart; i.e., there is a subsequence belonging to the trace language of chart m ($v = v_1 \cdot v_2 \cdots v_k = w_{i_1} \cdot w_{i_2} \cdots w_{i_k} \in \mathcal{L}_m^{trc}$), and all the messages between the activation message until the end of the satisfying subsequence ($\forall j', i_0 \leq j' \leq i_k$) that do not belong to the subsequence ($j' \notin \{i_0, i_1, ..., i_k\}$) are not restricted by the chart m ($w_{j'} \notin f(Messages(m))$).

For a universal chart, $mod(m) = universal$, we require that each time the activation message is sent the trace will satisfy the chart:

$$\begin{aligned}\mathcal{L}_m = \{w = w_1 \cdot w_2 \cdots \mid & \forall i, w_i = f(amsg(m)) \Rightarrow \exists i_1, i_2, ..., i_k \text{ and} \\ & \exists v = v_1 \cdot v_2 \cdots v_k \in \mathcal{L}_m^{trc}, \text{ s.t. } (i < i_1 < i_2 < ... < i_k) \wedge \\ & (\forall j, 1 \leq j \leq k, w_{i_j} = v_j) \wedge \\ & (\forall j', i \leq j' \leq i_k, j' \notin \{i_1, ..., i_k\} \Rightarrow w_{j'} \notin f(Messages(m)))\}\end{aligned}$$

The formula requires that after each time the activation message is sent ($\forall i, w_i = f(amsg(m))$), the trace will satisfy the chart m (this is expressed in the formula in a similar way to the case for an existential chart.)

Now come the main definitions, which finalize the semantics of our version of LSCs by connecting them with an object system:

Definition 2. *A system S* **satisfies** *the LSC specification $LS = \langle M, amsg, mod \rangle$, written $S \models LS$, if:*

1. $\forall m \in M, \quad mod(m) = universal \Rightarrow \ \mathcal{L}_S \subseteq \mathcal{L}_m$
2. $\forall m \in M, \quad mod(m) = existential \Rightarrow \ \mathcal{L}_S \cap \mathcal{L}_m \neq \emptyset$

3 Specifying temporal logic patterns in LSCs

We show how to specify some important temporal logic formulas using LSCs. Apart from the interest in specifying the properties, this can help getting more familiar with LSCs by seeing several examples.

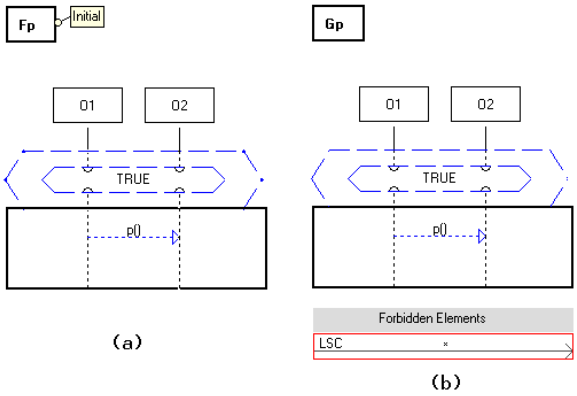

Fig. 4.

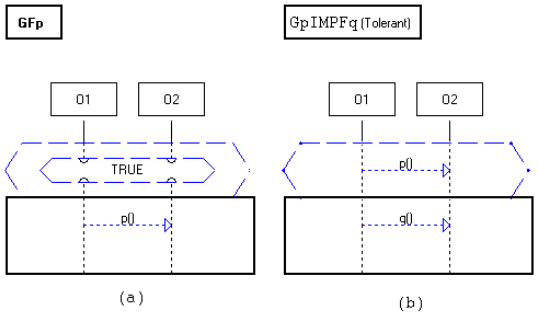

Fig. 5.

Consider the universal chart appearing in Fig. 4(a) specifying the temporal logic property Fp. The label `Initial` specifies that the chart can be activated

only once, at the beginning of the system run. The prechart (top dashed hexagon) has the condition `TRUE` as activation condition, this implies the main chart consisting of the message p occur eventually. The chart appearing in Fig. 4(b) specifies the temporal logic property Gp. The default interpretation without the label `Initial` is that whenever the prechart is activated, the main chart follows so p eventually occurs, and the forbidden element restricts all other messages from occurring so p always occurs. The properties GFp and $G(p \rightarrow Fq)$ are specified in Fig. 5(a), (b). Note the label `tolerant` in Fig. 5(b) that allows p to occur more than once before q happens without causing a violation of the chart.

4 Basic Translation

Embedding LSCs into CTL*

As a starting point for studying the relationship between LSCs and temporal logic, we consider a subset of the LSC language including only messages, and with at most one message in a prechart. We show that these LSC specifications can be embedded in the branching temporal logic CTL* [9]. This translation was first proposed in [12]. In this paper we will show how to support a wider subset of LSCs compared to [12] and in a much more efficient way.

Definition 3. *Let $w = m_1 m_2 m_3 ... m_k$ be a finite trace. Let $R = \{e_1, e_2, e_3 \cdots e_l\}$ be a set of events. The temporal logic formula ϕ_w^R is defined as:*

$$\phi_w^R = NU(m_1 \wedge (X(NU(m_2 \wedge (X(NU(m_3 ...)))))))),$$

where the formula N is given by $N = \neg e_1 \wedge \neg e_2 ... \wedge \neg e_l$.

Definition 4. *Let $LS = \langle M, amsg, mod \rangle$ be an LSC specification. For a chart $m \in M$, we define the formula ψ_m as follows:*

- *If $mod(m) = universal$, then $\psi_m = AG(amsg(m) \rightarrow X(\bigvee_{w \in \mathcal{L}_m^{trc}} \phi_w^R))$.*
- *If $mod(m) = existential$, then $\psi_m = EF(\bigvee_{w \in \mathcal{L}_m^{trc}} \phi_w^R)$.*

Here for a universal chart m we take R to be the events appearing in the prechart and in the main chart.

The following can now be proved

Proposition 1. *Given $LS = \langle M, amsg, mod \rangle$, let ψ be the CTL* formula $\bigwedge_{m \in M} \psi_m$, and let S be an object system. Then*

$$S \models \psi \quad \Leftrightarrow \quad S \models LS.$$

Proof. Follows from the definitions. Omitted from this version of the paper.

It is noteworthy that the reverse is not true: CTL* cannot be embedded in the language of LSCs. In particular, given the single level quantification mechanism of LSCs, the language cannot express general formulas with alternating path quantifiers. However, it shouldn't be too difficult to extend LSCs to allow certain kinds of quantifier alternation, as noted in [7]. This was not done there, since it was judged to have been too complex and unnecessary for real world usage of sequence charts.

5 Extending and Optimizing the Translation

5.1 Precharts with more than one message

In the language of live sequence charts, scenarios are a basic concept, so a prechart can itself describe a scenario which leads to the activation of the universal chart. We now consider the effect of this more general case on our translation. Since existential charts do not have precharts their translation is not affected, and we have to consider only the universal charts.

For a universal chart m we define the formula ψ_m as follows:

Definition 5.

$$\psi_m = G(\bigvee_{w \in \mathcal{L}^{trc}_{pch(m)}} \phi_w \rightarrow \bigvee_{w \in \mathcal{L}^{trc}_{pch(m)} \cdot \mathcal{L}^{trc}_{m}} \phi_w)$$

We denote $\mathcal{L}^{trc}_{pch(m)}$ the language of executions for the prechart m. The language $\mathcal{L}^{trc}_{pch(m)} \cdot \mathcal{L}^{trc}_{m}$ consists of concatenations of executions of the prechart and executions of the main chart.

Notice that this is a formula in linear temporal logic (LTL), as is the formula for a universal chart in the basic translation.

5.2 Improved Translation

The formula described in Definition 5 can be large, due to the possibility of having many different traces for the chart, which affects the number of clauses in the disjunction, and also due to the similarity of clauses at the different sides of the implication operator. We give an improved translation where the resulting temporal logic formulas are much more succinct (polynomial vs. exponential in the number of locations).

We consider the case where both the prechart and the main chart consist only of message communication, and denote $p_1, \cdots p_k$ the events appearing in the prechart, $m_1, \cdots m_l$ the events appearing in the main chart. Denote e_i any of these events, either in the prechart or in the main chart. Denote $e_i \prec e_j$ if e_i precedes e_j in the partial order induced by the chart and $e_i \not\prec e_j$ if e_i and e_j are unordered. We assume also that a message does not appear more than once in the same chart. It remains open whether an efficient translation exists for the most general case.

Definition 6.

$$\psi_m = G((\bigwedge_{p_i \prec p_j} \phi_{p_i,p_j} \wedge \bigwedge_{\forall p_i, m_j} \phi_{p_i,m_j} \wedge \bigwedge_{p_i \not\prec p_j} \neg\chi_{p_j,p_i}) \rightarrow$$

$$(\bigwedge_{m_i \prec m_j} \phi_{m_i,m_j} \wedge \bigwedge_{m_j \ is \ maximal} Fm_j \wedge \bigwedge_{\forall e_i, m_j} \neg\chi_{e_i,m_j}))$$

$\phi_{x_i,x_j} = \neg x_j U x_i$
$\chi_{x_i,x_j} = (\neg x_i \wedge \neg x_j) U (x_i \wedge X((\neg x_i \wedge \neg x_j) U x_i))$

Here the formula ϕ_{x_i,x_j} specifies that x_j must not happen before x_i which eventually occurs. The formula $\neg\chi_{x_i,x_j}$ specifies that x_i must not occur twice before x_j occurs. Note that this translation is polynomial in the number of messages appearing in the chart, while the translation in Definition 5 may be exponential in the number of messages appearing in the chart.

5.3 Past and Future

Another way to view LSCs is as a past formula implying a future formula, an approach similar to that of Gabbay [10]. An advantage of this view is that past formulas have simple and efficient canonical transformations to testers. A tester can be composed with a system and detect when a chart is activated due to the completion of the prechart. In this case the translation can be reduced to the simpler case of an activation message or activation condition rather than handling precharts explicitly. This view is formalized in the following definition:

Definition 7.

$$\psi_m = G((\bigwedge_{p_i \prec p_j} \tau_{p_i,p_j} \wedge \bigwedge_{\forall p_i, m_j} \tau_{m_j,p_i} \wedge \bigwedge_{p_i \not\prec p_j} \neg\xi_{p_j,p_i}) \rightarrow$$

$$(\bigwedge_{m_i \prec m_j} \phi_{m_i,m_j} \wedge \bigwedge_{m_j \ is \ maximal} Fm_j \wedge \bigwedge_{\forall e_i, m_j} \neg\chi_{e_i,m_j}))$$

$$\phi_{x_i,x_j} = \neg x_j U x_i$$
$$\chi_{x_i,x_j} = (\neg x_i \wedge \neg x_j) U (x_i \wedge X((\neg x_i \wedge \neg x_j) U x_i))$$
$$\tau_{x_i,x_j} = \neg x_i S x_j$$
$$\xi_{x_i,x_j} = (\neg x_i \wedge \neg x_j) S (x_j \wedge Y((\neg x_i \wedge \neg x_j) S x_j))$$

Here the formula τ_{x_i,x_j} specifies that x_i must not happen in the past before x_j which eventually occurs in the past. The formula $\neg\xi_{x_i,x_j}$ specifies that x_j must not occur twice in the past before x_i occurs. Y denotes the previous state, and is the past version of the operator X while S denotes the Since operator and is the past version of the Until operator U.

6 Expressing formulas in CTL

In this section we investigate the possibilities of expressing LSCs using the branching time logic CTL. The logic CTL is a restricted subset of CTL*. In CTL the temporal operators G, F, X and U must be immediately preceded by a path quantifier. We now show that the formulas in Definition 4 are in CTL, i.e. although syntactically they are not CTL formulas (In ϕ_w the X and U operators are not immediately preceded by a path quantifier) they have equivalent formulas that are CTL formulas.

Proposition 2. *For any formula ψ_m in Definition 4 there exists an equivalent CTL formula ψ'_m.*

Proof. We consider the two cases of existential and universal charts.
Existential chart

The formula given is $\psi_m = EF(\bigvee_{w \in \mathcal{L}_m^{trc}} \phi_w)$ We can simplify it as follows:

$$EF(\bigvee_{w \in \mathcal{L}_m^{trc}} \phi_w) \equiv \bigvee_{w \in \mathcal{L}_m^{trc}} EF(\phi_w)$$

And then go on to show that ϕ_w is equivalent to the CTL formula where the E path quantifier is added before each X and U temporal operator.
Universal chart

The formula given is $\psi_m = AG(amsg(m) \rightarrow X(\bigvee_{w \in \mathcal{L}_m^{trc}} \phi_w))$.

We show a proof for the special case of a single disjunct:
$AG(amsg(m) \rightarrow X\phi_w)$. It illustrates the main ideas and results that can be applied to the general case.

In order to explain the proof we consider an example with messages m_0, m_1, m_2, m_3 and prove the following lemma:

Lemma 1.

$$G(m_0 \rightarrow (X(NU(m_1 \wedge X(NU(m_2 \wedge X(NUm_3))))))) \equiv$$

$$AG(m_0 \rightarrow (AX(A(NU(m_1 \wedge AX(A(NU(m_2 \wedge AX(A(NUm_3))))))))))$$

Here $N = \neg m_0 \wedge \neg m_1 \wedge \neg m_2 \wedge \neg m_3$

In order to prove Lemma 1 we use a characterization obtained by Maidl of the common fragment of CTL and LTL [21]. In [21] an inductive definition of the ACTL [5] formulas that can be expressed in LTL is given. These formulas are called ACTL^{det}, and they are a restriction of ACTL.

ACTL^{det} is inductively defined as follows:

Definition 8. *ACTL^{det}*

- *p is a predicate.*
- *For ACTL^{det} formulas ϕ_1 and ϕ_2 and a predicate p:*
  $\phi_1 \wedge \phi_2$, $AX\phi_1$, $(p \wedge \phi_1) \vee (\neg p \wedge \phi_2)$, $A(p \wedge \phi_1)U(\neg p \wedge \phi_2)$, $A(p \wedge \phi_1)W(\neg p \wedge \phi_2)$.

Theorem 1. *(Maidl [21])*
Let ϕ be an ACTL formula. Then there exists an LTL formula ψ which is equivalent to ϕ iff ϕ can be expressed in ACTL^{det}.

We also use a theorem by Clarke and Draghicesku [6].

[5] ACTL is the fragment of those CTL formulas that contain, when in negation normal form, only A as a path quantifier.

Theorem 2. *(Clarke and Draghicesku [6]) For a CTL formula ϕ, we denote the result of removing all path quantifiers from ϕ by ϕ^d. Let ϕ be a CTL formula. Then there is an LTL formula ψ such that ϕ and ψ are equivalent iff ϕ is equivalent to ϕ^d.*

Proof. (of Lemma 1)

We now show that the formula in the right hand side of the equivalence in Lemma 1 is in ACTL^{det} by constructing it inductively according to Definition 8.

$\phi_0 = m_3$

$\phi_1 = A(\neg m_3 \wedge (\neg m_0 \wedge \neg m_1 \wedge \neg m_2))U(m_3 \wedge true) \equiv A(NUm_3)$

$\phi_2 = AX(\phi_1) \equiv AX(A(NUm_3))$

$\phi_3 = A((\neg m_2 \wedge (\neg m_0 \wedge \neg m_1 \wedge \neg m_3))U(m_2 \wedge \phi_2)) \equiv A(NU(m_2 \wedge AX(A(NUm_3)))$

$\phi_4 = AX(\phi_3) \equiv AX(A(NU(m_2 \wedge AX(A(NUm_3)))))$

$\phi_5 = A(\neg m_1 \wedge (\neg m_0 \wedge \neg m_2 \wedge \neg m_3))U(m_1 \wedge \phi_4)) \equiv A(NU(m_1 \wedge AX(A(NU(m_2 \wedge AX(A(NUm_3)))))))$

$\phi_6 = AX(\phi_5) \equiv AX(A(NU(m_1 \wedge AX(A(NU(m_2 \wedge AX(A(NUm_3))))))))$

$\phi_7 = (\neg m_0 \wedge TRUE) \vee (m_0 \wedge \phi_6) \equiv m_0 \rightarrow AX(A(NU(m_1 \wedge AX(A(NU(m_2 \wedge AX(A(NUm_3)))))))))$

$\phi_8 = A(true \wedge \phi_7)W(false) \equiv AG(m_0 \rightarrow AX(A(NU(m_1 \wedge AX(A(NU(m_2 \wedge AX(A(NUm_3))))))))))$

This shows that the formula in the right hand side of the equivalence in Lemma 1 is in ACTL^{det}, and therefore according to Theorem 1 the formula is in the common fragment of LTL and CTL.

As we showed the formula:

$$AG(m_0 \rightarrow AX(A(NU(m_1 \wedge AX(A(NU(m_2 \wedge AX(A(NUm_3))))))))))$$

is in ACTL^{det} therefore by theorem 1 the formula can be expressed in LTL, and by theorem 2 it is equivalent to the formula obtained by removing all path quantifiers:

$$G(m_0 \rightarrow (X(NU(m_1 \wedge X(NU(m_2 \wedge X(NUm_3)))))))$$

thus completing the proof of Lemma 1.

□

The proof of an equivalence like that of Lemma 1 for an execution of arbitrary length k, $w = m_1 m_2 m_3 ... m_k$ is by induction on k and is straightforward.

7 Extension for Additional LSC Constructs

We briefly outline how our translations can be extended to handle additional LSC constructs. A detailed treatment will appear in the full version of the paper.

7.1 Conditions

The LSC language allows using conditions, which are assertions on the variables of the system. Variables may be local to an instance or globally known. Conditions can be handled within the framework described previously, the main effect of conditions on the translation is on the languages of executions. We consider generalized executions, $w = x_1x_2x_3...x_k$ is an execution of m, a sequence of events (send or receive) or conditions (or their negation) satisfying the requirements of m.

As before

$$\phi_w = NU(x_1 \wedge (X(NU(x_2 \wedge (X(NU(x_3...)))))))),$$

where the formula N is given by $N = \neg m_1 \wedge \neg m_2... \wedge \neg m_l$. Each m_i in the formula is a proposition indicating the occurrence of a send or receive event, the conditions do not appear in N. Each x_i in the formula ϕ_w is a proposition indicating the occurrence of a send or receive event m_i, a condition holding c_i or not holding $\neg c_i$. Conditions can come in two flavors: mandatory (hot) and provisional (cold). If a system run encounters a false mandatory condition the run aborts abnormally, while a false provisional condition induces a normal exit from the enclosing charts. Conditions can also be shared by several instances, forming a synchronization barrier. These issues can be treated by our translation but are beyond the scope of this version of the paper.

7.2 Iteration

A loop construct is a subchart that is iterated a number of times. Fixed loops are annotated by a number or variable name, while unbounded loops are performed an a priory unknown number of times. The subchart can be exited when a cold condition inside it is violated. Bounded loops can be treated by unfolding techniques. Unbounded loops enhance the expressive power of LSCs and cannot be expressed in propositional temporal logic (PTL), since PTL does not allow counting modulo n, which can be specified by an LSC with unbounded loop with a certain message appearing n times inside the loop.

8 Related Work

A large amount of work has been done on scenario-based specifications. We briefly discuss the ones most relevant to our work. The idea of using sequence charts to discover design errors at early stages of development has been investigated in [1, 22] for detecting race conditions, time conflicts and pattern matching. The language used in these papers is that of classical message sequence charts, with semantics being simply the partial order of events in a chart. In order to describe system behavior, such MSC's are composed into hierarchal message sequence charts (HMSC's) which are basically graphs whose nodes are MSC's. As has been observed in several papers, e.g. [2], allowing processes to progress along

the HMSC with each chart being in a different node may introduce non-regular behavior and is the cause of undecidability of certain properties. Undecidability results and approaches to restrict HMSC's in order to avoid these problems appear in [16, 17, 11].

Live sequence charts have been used for testing and verification of system models. Lettrari and Klose [20] present a methodology supported by a tool called TestConductor, which is integrated into Rhapsody [18]. The tool is used for monitoring and testing a model using a restricted subset of LSCs. Damm and Klose [8, 19] describe a verification environment in which LSCs are used to describe requirements that are verified against a Statemate model implementation. The verification is based on translating an LSC chart into a timed Büchi automaton, as described in [19], and it also handles timing issues. Standard translations from Büchi automata to temporal logic can then be applied. Previous work on optimizing temporal logic formulas for model-checking appears in [3]. LSCs have also been applied to the specification and verification of hardware systems [4, 5].

In [14] a methodology for specifying and validating requirements, termed "play-in/play-out" is described. According to this approach, requirements are captured by the user playing in scenarios using a graphical interface of the system to be developed or using an object model diagram. The user "plays" the GUI by clicking buttons, rotating knobs and sending messages (calling functions) to objects in an intuitive manner. As this is being done, the supporting tool, called the Play-Engine, constructs a formal version of the requirements in the form of LSCs. Play-out is a complementary idea to play-in, which makes it possible to execute the requirements directly. Smart play-out [13] is an extension of the play-out mechanism using verification methods to drive the execution. The semantics described in this paper follows that of [14, 13], but the translation to temporal logic described here is new and was not used as part of (smart) play-out, a direction that we plan to investigate in future work.

References

1. R. Alur, G.J. Holzmann, and D. Peled. An analyzer for message sequence charts. *Software Concepts and Tools*, 17(2):70–77, 1996.
2. R. Alur and M. Yannakakis. Model checking of message sequence charts. In *10th International Conference on Concurrency Theory (CONCUR99)*, volume 1664 of *Lect. Notes in Comp. Sci.*, pages 114–129. Springer-Verlag, 1999.
3. Yves Bontemps. Automated verification of state-based specifications against scenarios (a step toward relating inter-object to intra-object specifications). Master's thesis, University of Namur, Belgium, June 2001.
4. A. Bunker and G. Gopalakrishnan. Verifying a VCI Bus Interface Model Using an LSC-based Specification. In *Proceedings of the Sixth Biennial World Conference on Integrated Design and Process Technology*, pages 1–12, 2002.
5. A. Bunker and K. Slind. Property Generation for Live Sequence Charts. Technical report, University of Utah, 2003.
6. E.M. Clarke and I. A. Draghicescu. Expressibility results for linear-time and branching-time logics. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, vol. 354 of *LNCS*, pages 428–437, 1988.

7. W. Damm and D. Harel. LSCs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19(1):45–80, 2001.
8. W. Damm and J. Klose. Verification of a radio-based signalling system using the statemate verification environment. *Formal Methods in System Design*, 19(2):121–141, 2001.
9. E.A. Emerson. Temporal and modal logics. In J. van Leeuwen, editor, *Handbook of theoretical computer science*, volume B, pages 995–1072. Elsevier, 1990.
10. D. Gabbay. The declarative past and imperative future. In *Temporal Logic in Specification*, volume 398 of *LNCS*, pages 407–448. Springer-Verlag, 1987.
11. E. L. Gunter, A. Muscholl, and D. Peled. Compositional message sequence charts. In *Proc. 7^{th} Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01), vol. 2031 of* LNCS*, Springer-Verlag*, pages 496–511, 2001.
12. D. Harel and H. Kugler. Synthesizing state-based object systems from LSC specifications. *Int. J. of Foundations of Computer Science (IJFCS).*, 13(1):5–51, Febuary 2002. (Also,*Proc. Fifth Int. Conf. on Implementation and Application of Automata (CIAA 2000)*, July 2000, Lecture Notes in Computer Science, Springer-Verlag, 2000.).
13. D. Harel, H. Kugler, R. Marelly, and A. Pnueli. Smart play-out of behavioral requirements. In *Proc. 4^{th} Intl. Conference on Formal Methods in Computer-Aided Design (FMCAD'02)*, volume 2517 of *LNCS*, pages 378–398, 2002.
14. D. Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer-Verlag, 2003.
15. D. Harel and R. Marelly. Specifying and Executing Behavioral Requirements: The Play In/Play-Out Approach. *Software and System Modeling (SoSyM)*, 2(2):82–107, 2003.
16. J.G. Henriksen, M. Mukund, K.N. Kumar, and P.S. Thiagarajan. On message sequence graphs and finitely generated regular MSC languages. In *Proc. 27th Int. Colloq. Aut. Lang. Prog.*, vol. 1853 of *LNCS*, pages 675–686. Springer-Verlag, 2000.
17. J.G. Henriksen, M. Mukund, K.N. Kumar, and P.S. Thiagarajan. Regular collections of Message Sequence Charts. In *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science(MFCS'2000)*, volume 1893 of *Lect. Notes in Comp. Sci.*, pages 675–686. Springer-Verlag, 2000.
18. Rhapsody. I-Logix, Inc., products web page. http://www.ilogix.com/products/.
19. J. Klose and H. Wittke. An automata based interpretation of live sequence charts. In *Proc. 7^{th} Intl. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01), volume 2031 of* LNCS*, Springer-Verlag*, 2001.
20. M. Lettrari and J. Klose. Scenario-based monitoring and testing of real-time UML models. In *4th Int. Conf. on the Unified Modeling Language, Toronto*, October 2001.
21. M. Maidl. The common fragment of CTL and LTL. In *Proc. 41st IEEE Symp. Found. of Comp. Sci.*, pages 643–652, 2000.
22. A. Muscholl, D. Peled, and Z. Su. Deciding properties for message sequence charts. In *Proc. of the 1st Int. Conf. on Foundations of Software Science and Computation Structures (FOSSACS '98)*, vol. 1378 of *LNCS*, pages 226–242. Springer-Verlag, 1998.
23. A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symp. Found. of Comp. Sci.*, pages 46–57, 1977.
24. B. Sengupta and R. Cleaveland. Triggered message sequence charts. In *Proceedings of the tenth ACM SIGSOFT symposium on Foundations of software engineering*, pages 167–176. ACM Press, 2002.
25. UML. Documentation of the unified modeling language (UML). Available from the Object Management Group (OMG), http://www.omg.org.
26. Z.120 ITU-TS Recommendation Z.120: Message Sequence Chart (MSC). ITU-TS, Geneva, 1996.