# Improving Translation of Live Sequence Charts to Temporal Logic

Rahul Kumar[1] Eric G Mercer[2]

*Computer Science Department, Brigham Young University, Provo, Utah 84606, USA*

Annette Bunker[3,4]

*Intel Corporation, Portland, Oregon 97123, USA*

**Abstract**

An efficient and mathematically rigorous translation from Live Sequence Charts (LSCs) to temporal logic is essential to providing an end-to-end specification and verification method for System on Chip (SoC) protocols. Without mathematical rigor, no translation can be trusted to completely represent the LSC specification, while inefficiency renders even provably sound translations useless in verifying the correctness of industrial-strength protocols. Previous work shows that the LSC-to-temporal logic and LSC-to-automata translations can be automated and formalized for the LSC language. In the LSC-to-temporal logic translation, the extraordinary size of the resulting formula limits the scalability of the charts that can be translated and verified. Our work, on the other hand, leverages intuitive temporal logic reductions to generate a formula that is at most quadratic in the size of the chart and demonstrates the benefits of the improved translation on several examples.

*Keywords:* live sequence chart, temporal logic, translation, linear, polynomial, spin, nusmv, reduction, scenario, verification, specification, ip core

## 1 Introduction

Recently, development trends have shifted towards building systems by integrating numerous heterogeneous *Intellectual Property (IP) cores* on a single chip. Such *system on chip* (SoC) designs implement multiple communication protocols that are necessary for the IP cores to interface and interact with each other. Given this trend, it becomes important to not only verify the individual IP cores, but the interface design and implementation as well. Verification of the interface provides the IP integrators the peace-of-mind guarantee of the IP core's communication

[1] Email:`rahul@cs.byu.edu`

[2] Email:`egm@cs.byu.edu`

[3] Email:`annette.bunker@intel.com`

[4] Annette Bunker performed this work while at USU.

*This paper is electronically published in*
*Electronic Notes in Theoretical Computer Science*
*URL:* `www.elsevier.nl/locate/entcs`

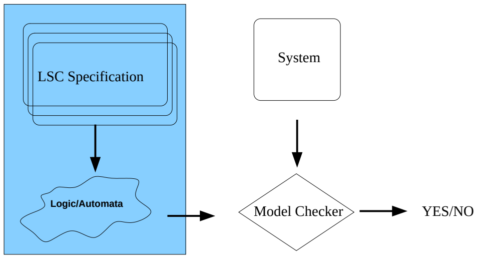


Fig. 1. The process of using scenario-based specifications to verify systems.

protocol as well as ease of integration in the SoC; thus, providing benefits to the IP core developers and integrators.

Pivotal to any verification effort is the ability to develop specifications against which the system in question is to be verified. Traditionally, English has been the default language choice for specifying and describing communication protocols. In our experience, English is not an ideal medium for expressing protocol specifications because of its context sensitive, imprecise, and cumbersome nature.

*Protocol Live Sequence Charts (PLSCs)* are a scenario-based language especially targeted for protocol design and verification of SoC systems implementing protocols [6]. Scenario-based languages, like PLSCs, provide a more intuitive and mathematically precise language for describing system interactions. Strictly speaking, PLSCs are a restricted form of Live Sequence Charts (LSCs) [5,7]. Additionally, they provide syntactic constructs for easily specifying certain protocol-specific behaviors such as clocks and invariants. As an example of the conciseness and expressive power of PLSCs, the Virtual Component Interface (VCI) SoC communication protocol has been translated from its 60 page English format to a one page PLSC chart describing all possible interactions [6]. As PLSCs are a subset of LSCs, we concern ourselves with LSCs for the remainder of the paper.

LSCs can be used at various stages of the development and verification process. Initially, LSCs can be used to verify properties of the communication protocol to guarantee behaviors of the protocol. Further in the development and testing stages of the system, LSCs can provide a specification against which implementations are verified. LSCs can also be published as the supported interface of the IP core.

We focus on the problem of formally verifying systems against LSC specifications. Fig. 1 shows a high level overview of the process of verifying a system against an LSC specification as developed and presented in [6,15,11]. An LSC specification and the system under test are given as input. The scenario-based specification is translated to automata or temporal logic that is used to verify the system with the help of a model checker (symbolic or explicit), as shown in the shaded portion of the figure.

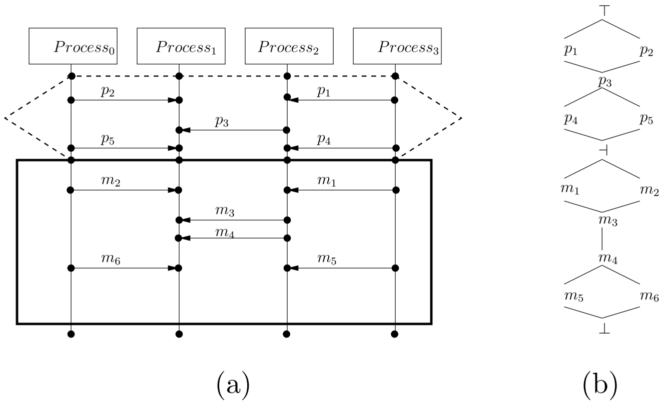
Inefficient translations to automata or temporal logic directly affect the verification task complexity, thus, motivating the need to improve methods to translate LSCs to temporal logic or automata. We present a translation of LSCs to temporal logic that generates a temporal logic formula which is of at most quadratic size with respect to the number of maximal messages of the LSC. Earlier translations produce formulas that are of quadratic size with respect to the size of the chart [15,18], which in the average case, is much larger compared to the number of maximal messages in the chart. The translation uses logic minimization over the *until* (**U**) operator to improve upon earlier translations. We prove that our improved translation covers the same set of behaviors as those specified by the quadratic translation in [15] and is more than competitive with the work presented in [11], which translates LSCs directly to automata. Further, we extend the translation to other constructs of the LSC grammar that have not been directly translated to temporal logic in earlier research. Finally, we present results in explicit and symbolic model checking that show the benefits of using a smaller formula during verification as generated by our improved translation. Specifically, the verification time and state space are greatly reduced, especially in cases where counterexamples need to be generated.

## 2 Related Work

LSCs extend the semantics of MSCs to be able to specify provisional behavior [16,7,5]. Additionally, LSCs have also been used in the past to model and verify systems. Work in [4] describes the modeling of an air traffic control system and the verification performed on the system. Work in [14] describes the modeling of an automotive system using LSCs. Other such examples utilize the expressive power of LSCs to concisely describe and verify systems and relate them to other specification languages as well [3,12,8].

There is ongoing interest in the model checking of LSCs and translation of LSCs to automata for the automatic synthesis of systems [13,1,9,10]. Additionally there has been significant work done in translating LSCs to temporal logic for verification of systems. The major limitation of translating LSCs to temporal logic is the sheer size of the resulting temporal logic formula [15,18]. In [6], small and efficient-to-verify ordering properties are generated from LSCs but no formal relationship is established between the system and the specification. Although the system satisfies the ordering properties, the properties do not imply that the system implements the LSC specification. In [15], an LSC is translated to an LTL formula that is quadratic in the size of the LSC. The size of the LTL formula is large enough to hinder the verification of anything but small LSCs. The work in [15] forms the basis of our work.

The work in [11], and [17,18] give an alternate verification approach in an effort to avoid the explosion encountered during the LSC-to-temporal logic translation. The approach unwinds the LSC to create an automaton with size proportional to the number of reachable states in the LSC, and it uses the automaton in an multi-tiered verification effort comprised of four different model checking procedures ordered by least-to-worst algorithmic complexity: reachability analysis with safety observer, ACTL model checking with and without observer, and finally LTL model checking. If a less powerful technique is not able to complete the verification, then the approach moves to the next procedure until it arrives at full LTL model checking. Although the approach deals with the full semantic model of LSCs, except Kleene stars, reachability is never powerful enough in systems that are non-timed as seen in the results from [11] where at least three procedures are run before obtaining an actual verification result in the non-timed charts. The need for the more powerful model checking procedures is critical to the work in this paper because the proce-



(a)

(b)

Fig. 2. A LSC illustrating most of the language features supported in this work with a lattice defining the partial order on messages in the chart. (a) The LSC. (b) The partial order defined on the messages in the LSC.

dures require a temporal logic formula. The size of the formula produced from the automaton-to-temporal logic translation in [11] results in such a large formula that ACTL and LTL verification are only feasible on small non-concurrent charts.

The LSC-to-temporal logic translation in this paper produces an LTL formula that is small enough to be practical in even highly concurrent charts, and we show this by presenting results for LTL verification on charts larger and more concurrent than those presented in [11]. In other words, a direct verification can be performed using our translation without having to use the multi-tiered approach in [11] since we address the original problem of formula size. Smaller formula sizes yield better verification times and greater scalability to larger charts and systems.

## 3 Live Sequence Charts

In this section we formalize our presentation of LSCs before relating them to LTL and verification. For simplicity and space purposes, the semantics only deal with a restricted set of the full LSC grammar; although, the translation extends to most of the LSC semantic constructs as discussed in Section 7.

An LSC is a graphical language for specifying communication between agents in a system [16,7,5]. Fig. 2(a) is an example LSC that contains the relevant language features discussed in this section. The boxes at the top of the figure are *agents* in the system. Each agent has a vertical *instance line* descending from the agent. The horizontal lines with filled arrowheads are *synchronous messages*. Synchronous messages force the send and receive events of the message to occur in the same state; thus, the sender and receiver have to be at their respective send and receive locations in the chart.

Messages in the LSC are divided into a pre-chart and a main chart. The pre-chart is the activation condition for the main chart and indicates the scenario in which the main chart operates. It is represented by the dashed hexagon in the figure. The main chart (directly below the pre-chart) is represented by the rectangle. A *universal* chart that specifies mandatory behavior is drawn with a solid line and an *existential* chart specifying provisional behavior is drawn with a dashed line. Universal charts force the occurrence of the main chart events after every occurrence of the pre-chart. Existential charts only assert the presence of one instance of the pre-chart and main chart events, and do not force the main chart to occur after every occurrence of the pre-chart. For the LSC shown in Fig. 2(a), the pre-chart is satisfied by a trace that contains the messages $\{p_1, p_2\}$ in any order, followed by $p_3$, and finally $\{p_4, p_5\}$ in any order. Note that the messages $p_1$ and $p_2$ are not ordered with respect to each other but are ordered with respect to $p_3$. The same observation holds for the messages $\{p_4, p_5\}$ and $p_3$. Once the pre-chart has been satisfied, the mandatory behavior of the universal chart forces the main chart to occur after the pre-chart. The main chart is satisfied if the messages $\{m_1, m_2\}$ occur in any order, followed by $m_3$, $m_4$, and finally the messages $\{m_5, m_6\}$ in any order.

We use the symbol $c$ to denote an individual chart like that in Fig. 2(a). The set of instance lines for a chart is given by $\mathit{inst}(c)$. An instance line has *locations* to track the state of the associated agent. For an instance line $i$ and chart $c$, $\mathit{dom}(c, i) = \{l_0, l_1, \ldots, l_{max}(i)\}$ is the set of locations for $i$ in $c$ with the first location (top most location for a process line) given by $l_0$ and the last location given by $l_{max}(i)$ when moving from the top to the bottom of the instance line. As locations are not uniquely labeled in the chart across instance lines, the set of pairs $\mathit{dom}(c) = \{\langle i, l\rangle \mid i \in \mathit{inst}(c) \wedge l \in \mathit{dom}(c, i)\}$ represents all instance and location pairs in a chart $c$. The complete state of a chart is the state of the individual instances as denoted by their current location. The initial state has every instance in its first location. The state of the chart evolves as instances move from one location to another down the chart.

The symbol $AP$ denotes the set of messages in the system. Communication is specified as a triple of the form $(\langle i, l\rangle, e, \langle i', l'\rangle)$, where $e \in AP$ is the message communicated from $\langle i, l\rangle$ to $\langle i', l'\rangle$. The set of all message communications for a chart is given by the relation $R(c)$. Well-formed charts are charts where the relation $R(c)$ is acyclic. This work, like [15], only considers well-formed charts.

The pre-chart begins at the first location, $l_0$, for each instance line. The main chart is the box below the pre-chart. A special location is reserved to denote the start of the main chart (end of the pre-chart). For each instance line, $i$, the last location, $l_{max}(i)$, marks the end of the main chart. As such, we require three unique locations in a chart that are not used by messages: the start of the pre-chart, the start of the main chart (end of the pre-chart), and the end of the main chart; all other locations must be used by messages. We use $p$ to represent messages in the pre-chart, $m$ to represent messages in the main chart, and $e$ to represent any message in the chart regardless of position in the chart.

A chart defines a partial order on its messages as its state changes on location transitions. To be specific, we introduce the symbols $\top$ (top), $\dashv$ (middle), and $\bot$ (bottom) to synchronize the agents when the chart starts, completes its pre-chart, and completes its main chart, respectively. The function shown in Fig. 3 returns the letter that is communicated in a message or the symbol $\top$, $\dashv$, or $\bot$. For convenience, we define $\mathit{msg}(c)$, $\mathit{msg}_p(c)$ and $\mathit{msg}_m(c)$ to be the messages of the entire chart, the pre-chart, and the main chart respectively.

We define an order relation, $\lhd$, to capture the sequences of messages and symbols as specified in the instance lines when starting at the first locations and traversing

Fig. 3. Function that returns letter given a location

$$msg(c)(\langle i,l\rangle) = \begin{cases} e & \text{if } \exists e,i',l' : (\langle i,l\rangle, e, \langle i',l'\rangle) \in R(c) \vee (\langle i',l'\rangle, e, \langle i,l\rangle) \in R(c) \\ \top & \text{if } l = l_0 \\ \bot & \text{if } l = l_{max}(i) \\ \dashv & \text{otherwise} \end{cases}$$

the lines to their last locations:

$$\forall \langle i,l_i\rangle, \langle i,l_{i+1}\rangle \in dom(c),\ \ msg(c)(\langle i,l_i\rangle) \lhd msg(c)(\langle i,l_{i+1}\rangle).$$

We do not explicitly describe the rules of moving from one state to another. Rather, we relate the message sequences as observed along each instance line in the order relation. In other words, the order relation describes the sequence of messages observed in the scope of the individual instance lines. The locations that communicate each message in the chart connect the sequences observed in one agent to those of the other agents. A partial order, $\prec$, on messages and symbols is created from the order relation ($\lhd$) by adding to it the reflexive terms and then computing its transitive closure. The partial order induced by the chart in Fig. 2(a) is shown in Fig. 2(b).

The lattice formed by the partial order shows those messages that are unordered with respect to other messages in the chart. The observation is key to our improved translation because these events must be ordered before synchronizing events but not relative to each other. For convenience in writing the necessary ordering properties, we also define $max_m(c) = \{e \mid e \lhd \bot\}$ and $max_p(c) = \{e \mid e \lhd \dashv\}$ to be the maximal messages of the main chart and the pre-chart respectively. Intuitively, $max_m(c)$ is the set of messages that occurs last in the main chart and $max_p(c)$ is the set of messages that occurs last in the pre-chart and immediately before the start of the main chart.

An LSC specification is a set of scenarios, $C$, defined in individual charts, $c$. For a given LSC specification, the equivalent temporal logic specification is given as $\psi = \bigwedge_{c\in C} \psi_c$ where $\psi_c$ is the temporal logic formula for chart $c$ using one of the translation approaches described in this work. A proof for the relationship between a system, an LSC specification, and the generated temporal logic formula is given in [15]. We summarize the proposition here and assert that since our improved translation is equivalent to the translation in [15], the proposition holds for our improved translation as well.

**Proposition 3.1** *Given a set of LSCs, $C$, for a specification, let $\psi$ be the temporal logic formula $\bigwedge_{c\in C} \psi_c$, $S$ be a system implementing the scenarios in $C$, and $s_0$ be the initial state of the system. Then*

$$S, s_0 \models \psi \Leftrightarrow \forall_{c\in C} S, s_0 \models c.$$

The translation, as presented in [15], writes a property for every entry in the partial order induced by the chart; thus, producing a formula that is at least quadratic in the size of the chart.

## 4 Quadratic Translation of LSCs to LTL

Two kinds of properties are generated in the quadratic translation of [15]: properties that establish the order between any two messages in the chart ($\phi$ properties) and properties that guarantee the uniqueness of each message in the chart ($\chi$ properties). For any two messages $x_i$ and $x_j$, such that $x_i \prec x_j$ in the partial order imposed by the chart, the property $\phi_{x_i,x_j} = \neg x_j \;\mathbf{U}\; x_i$ is generated stating that the message $x_j$ does not occur until message $x_i$ has occurred. For any two messages $x_i$ and $x_j$ such that $x_i \not\prec x_j$, the $\chi_{x_i,x_j} = (\neg x_i \wedge \neg x_j) \;\mathbf{U}\; (x_i \wedge \;\mathbf{X}\; ((\neg x_i \wedge \neg x_j) \;\mathbf{U}\; x_i))$ property states that the message $x_i$ occurs twice before $x_j$. The negation of this property states that message $x_i$ does not occur twice before message $x_j$.

The quadratic translation for a universal chart as defined in [15] is given in Equation 1. Again, we use the letters $e, p$ and $m$ to denote the events in the chart in general, pre-chart, and main chart respectively. Equation 1 is divided into two parts by the implication; if the pre-chart is correctly satisfied (left), then the main chart has to follow (right). The top terms on each side of Equation 1 describe the order of the pre-chart and main chart messages. The $\phi$ properties are generated for all pairs belonging to the $\prec$-relation as restricted by the $p, m$ and $e$ notation. The middle term on the left side guarantees that no main chart events occur in the pre-chart, thus, correctly framing the pre-chart. The middle term on the right guarantees the occurrence of the maximal messages (since they do not occur on the right side of any $\phi$ formula). The framing is important because it is responsible for correctly triggering the verification of the main chart events. In other words, we do not check for main chart events until we have correctly observed the pre-chart events. Finally, the bottom terms on each side of Equation 1 guarantee that each message only occurs once in the chart. For the example LSC shown in Fig. 2(a), Equation 1 generates 50 pre-chart properties and 80 main chart properties. In this context, a property refers to a single $\phi$ or $\chi$ term generated by the translation.

$$\psi_c = \mathbf{G}\left(\left(\begin{array}{c} \bigwedge\limits_{p_i \prec p_j} \phi_{p_i,p_j} \\ \wedge \\ \bigwedge\limits_{\forall p_i, m_j} \phi_{p_i,m_j} \\ \wedge \\ \bigwedge\limits_{p_i \not\prec p_j} \neg\chi_{p_j,p_i} \end{array}\right) \Rightarrow \left(\begin{array}{c} \bigwedge\limits_{m_i \prec m_j} \phi_{m_i,m_j} \\ \wedge \\ \bigwedge\limits_{m_j \text{ is max}} \mathbf{F}\; m_j \\ \wedge \\ \bigwedge\limits_{\forall e_i, m_j} \neg\chi_{e_i,m_j} \end{array}\right)\right) \tag{1}$$

The complexity analysis of the translation is divided into two parts: analyzing terms that establish order (top left, middle left, top right, and middle right terms) and uniqueness (bottom right and bottom left) in Equation 1. The total number of individual properties required to establish order is bounded by $|msg_p(c)|^2 +$

$(|msg_p(c)| \times |msg_m(c)|) + |msg_m(c)|^2 + |max_m(c)|$. For establishing uniqueness, the number of properties is bounded by $|msg_p(c)|^2 + (|msg_p(c)| + |msg_m(c)|) \times |msg_m(c)|$. Combining these bounds gives us a complexity that is at least quadratic in the size of the chart.

Despite the quadratic bound of the translation presented in [15], which improves on the classical exponential translation, the resulting formula is too large to be practical. The large size is due to the use of the partial order in Equation 1. Building properties from the $\prec$-relation includes redundant ordering formulas in the pre-chart and the main chart. For example, if $\neg p_3 \mathbf{~U~} p_2$ and $\neg p_2 \mathbf{~U~} p_1$ hold, then by transitivity, we know that $\neg p_3 \mathbf{~U~} p_1$ holds, and do not need to explicitly establish the relation. Additionally, checking for uniqueness of an event with respect to every other event includes multiple redundant checks. For example, if message $m_5$ does not occur until after $m_3$, and message $m_1$ occurs once before $m_5$ and $m_3$, then it is implied that message $m_1$ occurs only once before $m_3$. These reductions are formalized in the next section.

## 5 Temporal Logic Reductions

The improved translation makes use of transitivity to reduce the size of the quadratic formula presented in [15]. It eliminates both ordering ($\phi$) and uniqueness ($\chi$) properties from the formula. These reductions are discussed in the following sub-sections. [1]

### *5.1 Reducing Ordering Properties*

The following reduction result for the until (**U**) LTL operator is a general modal logic calculation which has been restated for this domain from [2]; it forms the basis of our reduction in the number of $\phi$ formulas in the final translation.

**Lemma 5.1** *For any three messages $x_t$, $x_u$, and $x_v$ and a trace $\pi = s_0, s_1, \ldots$*

$$M, \pi \models (\neg x_v \mathbf{~U~} x_u) \wedge (\neg x_u \mathbf{~U~} x_t) \Rightarrow M, \pi \models (\neg x_v \mathbf{~U~} x_t).$$

Intuitively, the **U**-operator forces the existence of the right hand predicate, and ensures that the left hand predicate holds until the state where the right side predicate occurs. The above result, takes advantage of the transitivity of the **U**-operator to eliminate properties that do not need to be explicitly verified but are part of the partial order on messages induced by the chart.

Furthermore, since there are events that may be succeeded by multiple unordered events, it is useful to be able to collapse multiple ordering properties relative to a single event into a single ordering property. An example of the reduction is seen in Fig. 2(a) where we would not expect to see $p_5$ or $p_4$ until $p_3$; and $p_5$ and $p_4$ are unordered relative to each other. In essence, the reduction collapses expressions that share the right hand term of the **U**-operator into a single property.

[1] All proofs are available in the full version of the paper available at http://vv.cs.byu.edu/~rahul/LSCTOLTL.pdf

**Lemma 5.2** *For a given set of messages $N$ and a message $x_i$ such that $\forall x_j \in N, x_i \prec x_j$*

$$M, \pi \models \bigwedge_{x_j \in N} (\neg x_j \ \mathbf{U} \ x_i) \Leftrightarrow M, \pi \models (\bigwedge_{x_j \in N} \neg x_j) \ \mathbf{U} \ x_i.$$

For our example, Lemma 5.2 results in a single property, rather than two properties, that does not allow either $p_5$ or $p_4$ until it sees $p_3$.

We now show via application of Lemma 5.1 and Lemma 5.2 how the total number of ordering properties can be reduced in the quadratic translation of [15]. First, we define the function $next(c)(e_i) = \{e_j \mid (e_i \lhd e_j) \wedge (e_j \notin \{\top, \dashv, \bot\})\}$. The $next(c)$ function, given a message $e_i$, returns the set of immediate successors according to $\lhd$-relation induced by the instance lines involved in the message communication. Using this function, we directly reference Lemma 5.2 to coalesce $\phi$ properties. We also define the $\phi'$ helper function as follows:

$$\phi'_{x_i, N} = (\bigwedge_{o \in N} \neg o) \ \mathbf{U} \ x_i.$$

The $\phi'$ function is a modified version of the $\phi$ function presented earlier. The second argument of the $\phi'$ function is a set rather than a single event and relies on Lemma 5.2 to produce a single ordering property when $N$ meets the necessary conditions. We now present the main reduction for this section.

**Corollary 5.3** *Given a chart, c, the ordering imposed by the formulas including the transitive and not including the transitive properties in the partial order is exactly the same.*

$$M, \pi \models \bigwedge_{p_i \prec p_j} \phi_{p_i, p_j} \Leftrightarrow M, \pi \models \bigwedge_{p \in msg(c)} \phi'_{p, next(c)(p)}.$$

The corollary reduces the set of properties needed to specify the order of events by omitting transitive properties which are implied by Lemma 5.1, and it proves the equivalence to the set of properties that explicitly writes the order between every event. Intuitively, the right formula is written from the $\lhd$-relation while the left formula is written from the $\prec$-relation. The **U**-operator implies by transitivity the presence of the extra orderings included in the $\prec$-relation.

### *5.2 Reducing Uniqueness Properties*

The $\chi$ formula in the previous section states that message $x_i$ occurs twice before message $x_j$, and the negative form of the $\chi$ formula states that message $x_i$ does not occur twice before message $x_j$. The next reduction result eliminates redundant properties when establishing uniqueness of a message in a chart (main or pre-chart). The reduction result is stated as follows:

**Lemma 5.4** *Given $k$ messages in a set $N$ that occur in the order $e_1 \prec \ldots \prec e_k$, if a message $e_i$ does not re-occur between its occurrence and the final message $e_k$, then it does not re-occur between its occurrence and any intermediate message $e_j$.*

$$M, \pi \models \bigwedge_{e_i \not\prec e_j} \neg \chi_{e_j, e_i} \Leftrightarrow M, \pi \models \bigwedge_{e_i \in N} \neg \chi_{e_i, e_k}.$$

Intuitively, Lemma 5.4 states that given a validated message ordering, we only need to establish the uniqueness of a message with respect to the last message in the ordering, as opposed to establishing uniqueness with respect to every message in the ordering.

## 6 Improved Translation of LSCs to LTL

The improved translation is produced by using the reduction results of Lemmas 5.1, 5.2, 5.4, and Corollary 5.3. The improved formula, $\psi'_c$, for universal charts is shown in Equation. 2. The improved translation has a structure similar to the structure of the quadratic translation shown in Equation 1. The key differences are in the use of the $\lhd$-relation (used by $next(c)$ function) for specifying order, and the use of the $\max_m(c)$ function for specifying fewer uniqueness properties. For explicit state model checking, the formula in its negated form is synthesized to automata for the verification of systems. Any violation provides a counterexample of the chart, which represents a flaw in the system implementing the chart. For symbolic model checking, the formula is not negated but directly verified on the system.

$$\psi'_c = G\left(\left(\begin{array}{c} \bigwedge\limits_{e\in msg_p(c)} \phi'_{e,next(c)(e)} \\ \wedge \\ \bigwedge\limits_{e\in max_p(c)} \phi'_{e,msg_m(c)} \\ \wedge \\ \bigwedge\limits_{(e,p)\in msg_p(c)\times max_p(c)} \neg\chi_{e,p} \end{array}\right) \Rightarrow \left(\begin{array}{c} \bigwedge\limits_{e\in msg_m(c)} \phi'_{e,next(c)(e)} \\ \wedge \\ \bigwedge\limits_{(e,m)\in msg(c)\times max_m(c)} \neg\chi_{e,m} \end{array}\right)\right) \tag{2}$$

Equation 3 shows the translation of existential charts. Since existential charts (provisional behavior) have to be satisfied by some trace of the system, the **EF** operator is used [11]. The existential formula states that there exists a trace in the future that satisfies the sequence of events as described in the existential chart. Similar to the earlier formulas, the $\phi'$ properties establish the order of the messages of the pre-chart and the main chart, and the $\chi$ properties establish the uniqueness of the messages with respect to the maximal messages in the chart. Equation 3 is a CTL* formula. Negating this CTL* formula gives us an LTL formula that has the form **AG** $(\neg(\theta_{order} \wedge \theta_{uniqueness}))$ where $\theta_{order}$ are the properties used to specify the order of events and $\theta_{uniqueness}$ are the properties that specify uniqueness of messages in the chart. This LTL formula is used for explicit or symbolic state verification. A violation of the formula provides a *witness* to the existential chart, which is the desired result, since an existential chart should be satisfied by at least one run of the system. If, on the other hand, no witness is generated, then the implementation violates the specification. Again, it should be noted that we start with a CTL* formula but actually perform LTL verification since the negated formula is in LTL.

$$\mathbf{EF}\left(\bigwedge_{e\in msg(c)} \phi'_{e,next(c)(e)} \wedge \bigwedge_{(e,m)\in msg(c)\times max_m(c)} \neg\chi_{e,m}\right) \tag{3}$$

**Theorem 6.1** *The improved translation as presented in Equation 2 and Equation 3 is equivalent to the quadratic translation presented in [15].*

$$M,\pi \models \psi_c \Leftrightarrow M,\pi \models \psi'_c.$$

For the example LSC in Fig. 2, the improved translation generates 13 properties for the pre-chart and 28 properties for the main chart. This provides a dramatic reduction over the 50 and 80 properties generated using the translation presented in [15].

## 7 Translating Additional Constructs

The work presented in this paper can translate all constructs except non-bonded conditions (conditions not tied to a specific message), temperatures (progress is forced or unforced), tolerant behavior (allowing multiple instances of an event within a chart), and multiplicities (Kleene star). In our experience, these constructs are rare in practice, and charts omitting these constructs are more than expressive enough to specify IP core interactions. We briefly present extensions to remaining constructs.

**Asynchronous Messages**: Asynchronous messages are drawn in charts with an open ended arrow head. They are used to specify messages where the receiver may not be ready to receive the message; thus, an asynchronous message produces two events as opposed to a single event as observed for synchronous messages. We build a new $\lhd$-relation that ranges over the individual events rather than the letters of the messages and forces the send event to always occur before the receive event. Properties are then generated from this new $\lhd$-relation.

**Co-regions**: Co-regions, specified by dashed lines parallel to the agent instance line, are used to express unordered events. To translate co-regions into temporal logic, we do not explicitly specify the order of all the events. Instead, we specify the order of the events that are to occur before and after the co-region and force the existence of the co-region events in the correct order.

**Bonded Conditions**: Bonded conditions are boolean predicates that are checked with a message in a simultaneous region (all events occur in the same state); thus, they are bonded to the message. Since a message always occurs with the condition (simultaneous region), the temporal logic translation only needs to consider the conjunction of the boolean predicate of the condition and the message.

**Invariants:** Invariants are boolean predicates that can be specified for a certain region of the LSC. We treat invariants as a set of bonded conditions. For each event in the region of invariance, the invariant predicate is attached to the event during translation. Using this approach, the invariant is only checked when an event occurs.

# 8 Analysis

The formulas presented in Equation2, Equation 3 and their extensions to additional constructs are quadratic in the number of maximal messages in the main chart, with all except one term being linear. In this context, linear implies a one-to-one relation between the LSC event and a $\phi'$ or $\chi$ formula. As before, we break up the analysis into two parts: one for the number of individual properties needed to establish event order and the second for the number of individual properties needed to establish uniqueness. The terms used to specify order are bounded by $|msg_p(c)| + |max_p(c)| + |msg_m(c)|$ properties, which is linear as compared to the quadratic number of properties generated by the quadratic translation presented in [15]. Terms used for specifying uniqueness are bounded by $(|msg_p(c)| \times |max_p(c)|) + (|msg(c)| \times |max_m(c)|)$ properties, which is directly proportional to the number of maximal messages in the main chart and pre-chart as opposed to the translation in [15] where the corresponding bound is directly proportional to the total number of messages in the main chart. In the typical case, the total number of main chart messages is much greater than the number of maximal messages (which can be greater than one only if the chart ends in a set of concurrent messages). Equation 3 has similar bounds.

# 9 Results

We are interested in understanding the performance of our improved LSC-to-temporal logic translation in both explicit and symbolic state model checking in terms of total verification time. Ideally, we would like to directly compare verification time using our translation to that in [15] and verification using the iterative approach in [11]. Such a direct comparison to [11] is not possible, however, because we do not have access to the implementation of the approach (which builds on VIS). Since the specification descriptions of [11] are the worst-case specifications for the translation process, we design an experiment using the specification descriptions from the empirical study in [11], and only indirectly compare the results.

The independent variables (inputs) in our experiment are the two translating approaches, the model checkers, the specifications, and the implementations. The model checkers are SPIN for explicit state model checking and NuSMV for symbolic state model checking. We generate specifications `SpecB`, and `SpecC` containing five to seven messages, and we use the `A2`, `A3`, and `A4` specifications from [11]. The `A`$x$ specifications are highly concurrent specifications consisting of 3 non-timed sequential co-regions with $x$ messages in each co-region. For example, the `A4` specification has twelve total messages that appear in three groups of four concurrent messages. The `A`$x$ specifications yield the worst-case formula size for our translation.

The implementations of the specifications in our experiment are broken into two categories: those that correctly implement the specification and those that do not as indicated by the _e annotation on the model name (created by introducing an error in the initial stages of the main chart). All of the `Spec*` systems directly implement the message sequences in the specification. The `A`$x$ implementations, however, follow the pattern described in [11] in that they first perform arbitrary computation by solving

Table 1
Verification results using the quadratic and improved translations and SPIN.

| Specification | Test | Quadratic Translation | | Improved Translation | |
|---|---|---|---|---|---|
| | | States | Time (seconds) | States | Time (seconds) |
| **SpecB** | SysA | 2612 | 0.02 | 2158 | 0.02 |
| | SysA_e | 2446 | 0.07 | 1965 | 0.06 |
| **SpecC** | SysB | – | | 4175 | 0.03 |
| | SysB_e | – | | 4589 | 0.12 |
| **A2** | soko | 3847560 | 104 | 1557700 | 36 |
| | soko_e | 1479320 | 32 | 620902 | 12 |
| **A3** | soko | – | | 2840220 | 69 |
| | soko_e | – | | 1031970 | 22 |

a puzzle and then implement the message sequence described in the specification. The pattern is similar to one observed in SoC designs using IP cores that perform some local computation followed by an exchange of information as specified by the IP core's interface and communication protocol. Since the puzzles nor the sizes of the implementations in [11] could be obtained, we use the *sokoban* block sliding puzzle (`soko`), the *bridge crossing* puzzle (`bridge`), and the *Alternating Bit Protocol* (`abp4`) to represent arbitrary computation. [5]

The dependent variables (outputs) in all our experiments are the number of states explored and the verification time, which we refer to as wall clock time. Wall clock time does not include LTL synthesis time for explicit state model checking. All of our experiments are run on an Intel Pentium 4 3.0 GHz machine with 2 GB of main memory. LTL synthesis is handled using LTL2BA. The results of our experiments are presented in Table 1 (SPIN) and Table 2 (NuSMV). A '–' entry in a table indicates that LTL synthesis timed out (> 2 hours). The quadratic translation is the translation from [15] while the improved translation is the translation as presented in this paper.

The explicit state model checking results in Table 1 show the verification time and number of states explored is much smaller for the improved translation compared to the quadratic translation. For specification `A2`, the improved translation completes in less than half the time of the quadratic translation and produces half as many states, thus, supporting our claim that verification cost is directly proportional to the size of the formula. Note that LTL synthesis times out for the quadratic translation for two of the models. LTL2BA can be a limiting factor for explicit state model checking since LTL synthesis also times out for both the quadratic and the improved translations for models and specifications not included in the table.

Table 2 shows that the improved translation results in a much smaller verification time in symbolic model checking. For specification `A4`, the improved translation completes the verification in less than a fourth of the time required by the quadratic translation. The performance improvement is noticeably larger when a counterexample trace is generated for the model. More importantly, the improved translation readily verifies `A5` implementations which are not verified in [11] due to the large formula size from the translation. In fact, the improved translation allows direct

[5] All of our models, and specification formulas can be downloaded at http://vv.cs.byu.edu/∼rahul/experiments.tar.gz

Table 2
Verification results using the quadratic and improved translations and NuSMV.

| Specification | Model | States | Quadratic Time (seconds) | Improved Time (seconds) |
|---|---|---|---|---|
| **Specification A2** | bridge | 76992 | 14 | 5 |
| | abp4 | 2236420 | 30 | 11 |
| | bridge_e | 76992 | 29 | 13 |
| | abp4_e | 2236420 | 76 | 27 |
| **Specification A3** | bridge | 76992 | 22 | 8 |
| | abp4 | 2236420 | 59 | 20 |
| | bridge_e | 76992 | 56 | 20 |
| | abp_e | 2236420 | 146 | 50 |
| **Specification A4** | bridge | 76992 | 49 | 11 |
| | abp4 | 2236420 | 174 | 29 |
| | bridge_e | 76992 | 132 | 26 |
| | abp_e | 2236420 | 337 | 67 |
| **Specification A5** | bridge | 76992 | 175 | 26 |
| | abp4 | 2236420 | 555 | 73 |
| | bridge_e | 76992 | 509 | 56 |
| | abp_e | 2236420 | 1271 | 131 |

LTL verification of the A$x$ implementations which in the multi-tiered approach of [11] require three separate verification runs before obtaining a meaningful result.

There are three key threats to validity in this empirical study: first, the omitted times for LTL synthesis in explicit state model checking; second, the indirect comparison to the multi-tiered approach of [11], and third, the model checker implementations. The LTL synthesis time is a one time cost that either completes or does not complete. We assume that once synthesized, the formula is reused over several verification runs as the system is implemented. In our experiment, all LTL synthesis occurred in less than three minutes of wall clock time. Our experiments try to duplicate the models in [11] as closely as possible. Anecdotally, our results scale to the A4 and A5 specifications in very little time and with great ease in NuSMV, which is of prime importance, since the specifications represent our worst-case for this translation. Real world specifications tend to be much less concurrent and smaller as compared to the $Ax$ specifications. As for the model checkers, default decisions in model checker implementations sometimes affect performance. It is not known if a different model checker such as VIS with different default decisions would give different results to invalidate those presented here. We suspect such a result to be highly unlikely since we know that formula size (as well as nesting depth) directly affects the verification cost, and the formula generated by the improved translation is much smaller than any other known translation.

## 10 Conclusions and Future Work

We present an improved translation of LSCs to temporal logic for a subset of LSCs by capitalizing on temporal logic reductions to reduce the number of individual properties needed for specifying event order and uniqueness. The known quadratic translation is directly proportional to the number of messages in the main chart as opposed to the improved translation that is proportional to the number of maximal messages in the main chart, which in the common case, is much smaller than the total number of messages in the main chart. We present results that show the benefits of using the improved translation during automata synthesis and verification in both the explicit and symbolic domains. Verification time, number of states, and automata synthesis benefit from the improved translation because of the smaller formula size. Future work in this area involves extending the translation to the LSC constructs of non-bonded conditions and temperatures. We are also investigating methods to decompose the formula into smaller pieces, as well as creating specialized algorithms to take advantage of the formula structure for optimized LTL synthesis. By doing so, we plan to complete the verification chain for SoC interface design and validation.

# References

[1] Alur, R. and M. Yannakakis, *Model checking of Message Sequence Charts*, Proc. of the 10th Inter. Conf. on Concurrency Theory (1999), pp. 114–129.

[2] Blackburn, P., M. de Rijke and Y. Venema, "Modal Logic," Cambridge University Press New York, NY, USA, 2001.

[3] Bohn, J., W. Damm, J. Klose, A. Moik, H. Wittke, H. Ehrig, B. Kramer and A. Ertas, *Modeling and validating train system applications using Statemate and Live Sequence Charts*, Proc. of the Conf. on Integrated Design and Process (2002).

[4] Bontemps, Y., P. Heymans and H. Kugler, *Applying LSCs to the specification of an air traffic control system*, Proc. of the 2nd Int. Work. on Scenarios and State Machines: Models, Algorithms and Tools (SCESM03), at the 25th Int. Conf. on Soft. Eng.(ICSE03), Portland, OR, USA, May (2003).

[5] Brill, M., W. Damm, J. Klose, B. Westphal and H. Wittke, *Live Sequence charts: An introduction to lines, arrows, and strange boxes in the context of formal verification*, Lec. notes in Computer Science (2001), pp. 374–399.

[6] Bunker, A. 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*, Society of Design and Process Science, 2002, p. 48.

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

[8] Damm, W. and J. Klose, *Verification of a radio-based signaling system using the STATEMATE verification environment*, Formal Methods in System Design **19** (2001), pp. 121–141.

[9] Harel, D., H. Kugler and A. Pnueli, *Synthesis Revisited: Generating statechart models from scenario-based requirements*, Formal Methods in Software and Systems Modeling (2005), pp. 309–324.

[10] Heymans, P. and Y. Bontemps, *Turning high-level Live Sequence Charts into automata*, Proc. of the First Inter. Work. on Scenarios and State Machines (SCESM) (2002).

[11] Klose, J., T. Toben, B. Westphal and H. Wittke, *Check it out: On the efficient formal verification of live sequence charts*, in: *CAV*, 2006, pp. 219–233.

[12] Klose, J. and B. Westphal, *Relating LSC specifications to UML models*, Proc. of the Second Inter. Work. on Integration of Specification Techniques for Applications in Eng. (INT 2002), Grenoble, France (2002).

[13] Klose, J. and H. Wittke, *An automata based interpretation of live sequence charts*, in: *Proc. of the 7th Inter. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS01)* (2001), pp. 512–527.

[14] Knieke, C., M. Huhn and U. Goltz, *Modeling and simulation of an automotive system using LSCs*, CSDUML (2005).

[15] Kugler, H., D. Harel, A. Pnueli, Y. Lu and Y. Bontemps, *Temporal Logic for Scenario-Based Specifications*, Proc. of the 11th Inter. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS05) (2005).

[16] Rudolph, E., P. Graubmann and J. Grabowski, *Tutorial on Message Sequence Charts*, Computer Networks and ISDN Systems **28** (1996), pp. 1629–1641.

[17] Toben, T. and B. Westphal, *Concurrent LSC Verification: On Decomposition Properties of Partially Ordered Symbolic Automata*, Electr. Notes Theor. Comput. Sci. **145** (2006), pp. 95–111.

[18] Toben, T. and B. Westphal, *On the expressive power of LSCs*, in: *SOFSEM* (2006), pp. 33–43.