Sequence Diagrams Aided Security Policy Specification

Hui Shen, Ram Krishnan, and Jianwei Niu

Abstract—A fundamental problem in the specification of regulatory privacy policies such as the Health Insurance Portability and Accountability Act (HIPAA) is to state the policies precisely, consistent with their high-level intuition. In this paper, we propose UML Sequence Diagrams as a practical tool to graphically express privacy policies. A graphical representation allows decision-makers such as application domain experts and security architects to easily verify and confirm the expected behavior. Once intuitively confirmed, our work in this article introduces an algorithmic approach to formalizing the semantics of Sequence Diagrams in terms of Linear Temporal Logic (LTL) templates. In all the templates, different semantic aspects are expressed as separate, yet simple LTL formulas that can be composed to define the complex semantics of Sequence Diagrams. The formalization enables us to leverage the analytical powers of automated decision procedures for LTL formulas to determine if a collection of Sequence Diagrams is consistent, independent, etc. and also to verify if a system design conforms to the privacy policies. We evaluate our approach by modeling and analyzing a substantial subset of HIPAA rules using Sequence Diagrams.

Index Terms—Formal Verification, Sequence Diagram, Temporal Logic, Privacy Policy, HIPAA

1 INTRODUCTION

A number of privacy policy regulations have been enacted as public law. Prime examples of such policies include health privacy rules of HIPAA [1], consumer financial privacy rules of GLBA [3], student educational record privacy rules of FERPA [4], children online privacy rules of COPPA [2], etc. These privacy policies typically regulate sharing of private information of an individual between two or more organizations. In order to enforce such privacy policies in information systems, they must be codified precisely, consistent with the intuition behind those policies. However, in practice, the people who design such privacy policies are not experts in codifying those policies in computer systems while those who codify those policies are not experts in the legal parlance using which those privacy policies are often specified.

Consider HIPAA for example. There has been extensive prior work on formalizing HIPAA so it can be analyzed and codified in computer systems to regulate information sharing. Protection of private data often requires timely communication among multiple parties in a decentralized manner, and needs to accommodate the preferences of the subjects of private data. Several frameworks have been proposed for specifying and analyzing privacy policies using formal methods, including Contextual Integrity (CI) [7], Privacy APIs [25], PrivacyLFP [13], and work by Lam et al. [24]. To date, most of the work in this area has concentrated on methodologies for formally capturing privacy policies or analyzing privacy policies to determine whether they satisfy various properties. Much less emphasis is placed on whether the framework, comprising techniques and tools, enable domain experts to comprehend and check whether the codification of privacy rules is valid. In particular, such works assume that what has been codified is accurate with respect to the intuition behind legally stated HIPAA rules. This presents a fundamental security challenge that is hard to address. We propose that HIPAA domain experts should be involved while codifying those complex privacy rules.

One of the most prevalent tools that is used today for design specifications is a collection of Unified Modeling Language (UML) Sequence Diagrams [27]. Our premise is that UML Sequence Diagrams are intuitive ways to express privacy policies such as those in HIPAA which regulate interaction between various entities. The graphical nature of Sequence Diagrams allow domain experts to visually confirm the intuition, for example, behind HIPAA privacy rules. Once confirmed, those Sequence Diagrams still need to be formally codified in computer systems and analyzed. Our work in this article, addresses this precise issue. We propose Sequence Diagrams as a vehicle to intuitively express the semantics of privacy policies that can be confirmed by application domain experts. Once the Sequence Diagrams for the above tasks are finalized, our technique allows them to be algorithmically translated into Linear Temporal Logic (LTL) templates that precisely characterize the intuition behind graphical specification by privacy architects. This enables automated verification using
techniques such as model checking. We demonstrate these tasks using a HIPAA case-study in this article.

UML 2 Sequence Diagrams include Combined Fragments and Interaction Use, allowing multiple, complex scenarios to be aggregated. Combined Fragments permit different types of control flow, such as interleaving and branching, for presenting complex and concurrent behaviors, increasing a Sequence Diagram's expressiveness. Prior work in formalizing UML 2 Sequence Diagrams do not support all combined fragments within a single formal framework (see the survey by Micskei et al [26]). Specifically, what we lack is a unified framework for formally specifying nested combined fragments and interaction constraints. As we will see, these constructs are critical to specify real-world privacy policies such as HIPAA.

The following criteria specify our rationale behind selecting Sequence Diagrams as a framework of choice for security policy specification: (1) The framework should be intuitive and help bridge the gap between application domain experts and system architects or designers. (2) The framework must facilitate formal verification of security properties of the system using automated tools. (3) It would be convenient to utilize a single framework for specifying both the system design and its expected security properties.

A number of prior work in specification and verification of systems satisfy some of the above criteria, but not all. For instance, the large body of research work on utilizing process algebra based approaches for this purpose satisfies criterion (2), but not (1) or (3). For example, in [?], the authors propose an approach to translate UML into process algebra models. Such approaches have many practical limitations. For example, the tool developed in [?] can only discover deadlocks. Theoretically, one might translate process algebra constructs into Sequence Diagrams, but this is not a feasible approach in practice. Prior work on formalizing trace semantics [?], [?] satisfy criterion (1) and aspects of (2). Formalizing large-scale privacy policies such as HIPAA at individual trace level can become very cumbersome. Furthermore, we are not aware of tools that are readily available for formal verification of properties against systems using the trace theory approach in these works.

The contributions of this article are as follows:

- We develop an approach to formalize the semantics of Sequence Diagrams with all the Combined Fragments, including nested Combined Fragments and Interaction Constraints, using LTL. In the framework, different semantic aspects are expressed as separate, yet simple LTL formulas that can be composed to define the semantics of a Sequence Diagram. (Sections 4 and 5)
- We present how to use UML 2 Sequence Diagrams for formally specifying a substantial portion (>100 Sequence Diagrams) of the HIPAA privacy policies that are behavior-related. (HIPAA includes certain static definitions that we do not consider for our purpose.)
- We develop a tool suite that can generate LTL templates given the Sequence Diagrams as input. This tool enables system designers and security architects to conduct formal analysis. To this end, we utilize this tool and a model checker (Section 6) to verify security properties such as consistency and independence of HIPAA rules. We also show how to verify conformance of an enterprise's functional system with HIPAA rules.

2 UML 2 Sequence Diagram

In this section, we outline the syntax and semantics of a Sequence Diagram with Combined Fragments provided by OMG [27]. As the first step of defining a Sequence Diagram using LTL formulas, we precisely define the semantics of Sequence Diagram with Combined Fragments. We begin with the basic Sequence Diagram, then discuss the structured control constructs, including Combined Fragments and Interaction Use.

2.1 Basic Sequence Diagram

We refer to a Sequence Diagram without Combined Fragments as a basic Sequence Diagram (see figure 1a for an example with annotated syntactic constructs). A Lifeline is a vertical line representing a participating object. A horizontal line between Lifelines is a Message. Each Message is sent from its source Lifeline to its target Lifeline and has two endpoints, e.g., m1 is a Message sent from Lifeline L1 to Lifeline L2 in figure 1a. Each endpoint is an intersection with a Lifeline and is called an Occurrence Specification (OS), denoting a sending or receiving event within a certain context, i.e., a Sequence Diagram. OSs can also be the beginning or end of an Execution Specification, indicating the period during which a participant performs a behavior within a Lifeline, which is represented as a thin rectangle on the Lifeline.

The semantics of a basic Sequence Diagram is defined by a set of traces. A trace is a sequence of OSs expressing Message exchange among multiple Lifelines. We identify four orthogonal semantic aspects, each of which is expressed in terms of the execution order of concerned OSs, must be considered for the basic Sequence Diagram [26], [27]

1) Each OS can execute only once, i.e., each OS is unique within a Sequence Diagram.
2) On each Lifeline, OSs execute in their graphical order.
3) For a single Message, the sending OS must take place before the receiving OS does.
4) In a Sequence Diagram, only one object can execute an OS at a time, i.e., OSs on different Lifelines are interleaved.
Consider again figure 1a. All eight OSs are uniquely defined, which is prescribed by semantic aspect 1. OS s2 can not happen until OS r1 executes on Lifeline L1, which is prescribed by semantic aspect 2. For Message m1, OS r1 can not happen until OS s1 executes, which is imposed by semantic aspect 3. OS s3 and s4 can not happen at the same time, which is imposed by semantic aspect 4.

Messages are of two types: asynchronous and synchronous. The source Lifeline can continue to send or receive other Messages after an asynchronous Message is sent. If a synchronous Message is sent, the source Lifeline blocks until it receives a response from the target Lifeline [27].

2.2 Combined Fragments

Both Combined Fragments and Interaction Use are structured control constructs introduced in UML 2. A Combined Fragment (CF) is a solid-outline rectangle, which consists of an Interaction Operator and one or more Interaction Operands. Figure 1b shows example CFs with annotated syntactic constructs. A CF can enclose all, or part of, Lifelines in a Sequence Diagram. The Interaction Operands are separated by dashed horizontal lines. The Interaction Operator is shown in a pentagon in the upper left corner of the rectangle. OSs, CFs, and Interaction Operands are collectively called Interaction Fragments. An Interaction Operand may contain a boolean expression which is called an Interaction Constraint or Constraint. An Interaction Constraint is shown in a square bracket covering the Lifeline where the first OS will happen. The CFs can be classified by the number of their Interaction Operands. Alternatives, Parallel, Weak Sequencing and Strict Sequencing contain multiple Operands. Option, Break, Critical Region, Loop, Assertion, Negative, Consider, and Ignore contain a single Operand. The example in figure 1b contains two CFs: a Parallel with two Operands and a Critical Region with a single Operand.

An Interaction Use construct allows one Sequence Diagram to refer to another Sequence Diagram. The referring Sequence Diagram copies the contents of the referenced Sequence Diagram.

The semantics of the seq Sequence Diagram with CFs is defined by two sets of traces, one containing a set of valid traces, denoted as Val(seq), and the other containing a set of invalid traces, denoted as Inval(seq). Traces specified by a Sequence Diagram without a Negative CF are considered as valid traces. An empty trace is a valid trace. Invalid traces are defined by a Negative CF. Traces that are not specified as either valid or invalid are called inconclusive traces, denoted as Incon(seq). An Assertion specifies the set of mandatory traces in the sense that any trace that is not consistent with the traces of it is invalid, which is denoted as Mand(seq).

Along a Lifeline, OSs that are not contained in the CFs, are ordered sequentially. The order of OSs within a CF’s Operand which does not contain other CFs in it is retained if its Constraint evaluates to True. A CF may alter the order of OSs in its different Operands. We first identify three independent semantic rules general to all CFs, in the sense that, these rules do not constrain each other.

1) OSs and CFs, are combined using Weak Sequencing (defined below). On a single Lifeline, a CF’s preceding Interaction Fragment must complete the execution prior to the CF’s execution, and the CF’s succeeding Interaction Fragment must execute subsequently.

2) Within a CF, the order of the OSs and CFs within each Operand is maintained if the Constraint of the Operand evaluates to True; otherwise, (i.e., the Constraint evaluates to False) the Operand is excluded.

3) The CF does not execute when the Constraints of all the Operands evaluate to False. Thus, the CF’s preceding Interaction Fragment and succeeding Interaction Fragment are ordered by
2.3 Interaction Operator

The execution of OSs enclosed in a CF is determined by its Interaction Operator, which is summarized as follows:

- **Alternatives**: one of the Operands whose Interaction Constraints evaluate to True is nondeterministically chosen to execute.
- **Option**: its sole Operand executes if the Interaction Constraint is True.
- **Break**: its sole Operand executes if the Interaction Constraint evaluates to True. Otherwise, the remainder of the enclosing Interaction Fragment executes.
- **Parallel**: the OSs on a Lifeline within different Operands may be interleaved, but the ordering imposed by each Operand must be maintained separately.
- **Critical Region**: the OSs on a Lifeline within its sole Operand must not be interleaved with any other OSs on the same Lifeline.
- **Loop**: its sole Operand will execute for at least the minimum count (lower bound) and no more than the maximum count (upper bound) as long as the Interaction Constraint is True.
- **Assertion**: the OSs on a Lifeline within its sole Operand must occur immediately after the preceding OSs.
- **Negative**: its Operand represents forbidden traces.
- **Strict Sequencing**: in any Operand except the first one, OSs cannot execute until the previous Operand completes.
- **Weak Sequencing**: on a Lifeline, the OSs within an Operand cannot execute until the OSs in the previous Operand complete, the OSs from different Operands on different Lifelines may take place in any order (cf. Strict Sequencing).
- **Consider**: any message types other than what is specified within the CF is ignored.
- **Ignore**: the specified messages types are ignored within the CF.
- **Coregion**: the contained OSs and CFs on a Lifeline are interleaved.
- **General Ordering** imposes an order between two unrelated OSs on different Lifelines.

3 SEQUENCE DIAGRAM DECONSTRUCTION

In this section, we present the formal definitions of a Sequence Diagram. First, we give a textual representation of a Sequence Diagram. Then, we deconstruct a Sequence Diagram and CFs into fine-grained syntactic constructs to facilitate the semantic description of Sequence Diagram, in particular, Weak Sequencing among OSs and CFs.

3.1 Definition of Syntactic Structures

A Sequence Diagram consists of a set of Lifelines and a set of Messages. The textual representation of a Sequence Diagram is formally defined as below.

**Definition 1.** A Sequence Diagram is given by a three tuple \((L, MSG, FG)\), in which \(L\) is a non-empty set of Lifelines enclosed in the Sequence Diagram. \(MSG\) is a set of Messages directly enclosed in the Sequence Diagram, i.e., Messages that are not contained by any CF. \(FG\) is a set of CFs directly enclosed in the Sequence Diagram, i.e., the top level CFs, denoted as \(CF_1, CF_2, ..., CF_m\).

Messages that are directly enclosed in the top-level CFs will be defined in their respective CFs. Similarly CFs that are directly enclosed in top-level CFs are defined in their enclosing CFs. In this manner, a Sequence Diagram with CFs can be recursively defined.

A Message is the specification of an occurrence of a message type within the Sequence Diagram, while a message type is the signature of the form \((message\ name, source\ Lifeline, target\ Lifeline)\). Within a Sequence Diagram, a message type can occur multiple times, which are associated with multiple Messages. Accordingly, multiple OSs within a Sequence Diagram can be associated with an event. Each Message is defined by its sending OS and receiving OS. We associate each OS with a location of a Lifeline. As each location is uniquely defined, each OS is uniquely defined. Thus, each Message is uniquely defined by its sending OS and receiving OS.

**Definition 2.** A Message has the form \((name, mform, OS_s, OS_r)\), where name is the Message name, mform denotes it is either a synchronous or an asynchronous Message, \(OS_s\) denotes its sending OS and \(OS_r\) denotes its receiving OS. Each OS has the form \((l_i, loc_k, type)\), where \(l_i\) denotes its associated Lifeline, \(loc_k\) is the location where the OS takes places on Lifeline \(l_i\), and type denotes it is either a sending or a receiving OS.

Each Lifeline \(l_i \in L\) has a set of finite locations \(LOC(l_i) \subseteq N\) on it. The locations form a finite sequence \(1, 2, 3, ..., k, k \in N\). Each location is associated with an OS uniquely and vice versa, i.e., the relation between set \(LOC(l_i)\) and the set returned by function \(OSS(l_i)\) is a one-to-one correspondence. Function \(OSS(l_i)\) returns the set of OSs on Lifeline \(l_i\). For example, in figure 1b, the set \(LOC(l_2)\) contains seven locations, each of which is associated with an OS, i.e., OSs \(r_1, s_2, r_3, s_4, r_5, r_6, r_7\). Message \(msg1\) is expressed by \((l_1, asynch, s_1, r_1)\), and OS \(s_1\) is expressed by \((l_1, 1, send)\), where \(l_1\) represents a participating object of class \(L_1\).
Definition 3. A CF $CF_m$ has the form $(L, oper, OP)$. $L$ denotes the set of Lifelines enclosed by $CF_m$, including the Lifelines which may not intersect with the Messages of $CF_m$. $oper$ denotes the Interaction Operator of $CF_m$. $OP$ denotes the sequence of Interaction Operands within $CF_m$, i.e., $op_{m,1}$, $op_{m,2}$, ..., $op_{m,n}$.

Each $op_n \in OP$ has the form $(L, MSG, FG, cond)$, where $L$ denotes the set of Lifelines enclosed by $op_n$; $MSG$ denotes the set of Messages directly enclosed in $op_n$; $FG$ denotes the set of CFs directly enclosed in $op_n$; and $cond$ denotes the Interaction Constraint of $op_n$, which is True if there is no Interaction Constraint. Without loss of generality, $cond$ is represented by a boolean variable. Comparing the structure between a Sequence Diagram and an Operand, the Sequence Diagram does not have an Interaction Constraint. In order for an Operand and a Sequence Diagram to share the same form, we assign an Interaction Constraint (which evaluates to True) to a Sequence Diagram.

Consider figure 1b as an example. Sequence Diagram $seq$ is represented by $(\{l_1, l_2, l_3\}, \{msg_1, msg_2\}, \{CF_1\})$, where the set of Lifelines enclosed by $seq$ contains three Lifelines, $l_1, l_2, l_3$, the set of Messages directly enclosed in $seq$ contains two Messages, $msg_1, msg_2$, and the set of CFs directly enclosed in $seq$ contains one CF, $CF_1$, $msg_1, msg_2$, and $CF_1$ are combined using Weak Sequencing. $CF_1$ is represented by $(\{l_1, l_2, l_3\}, par, \{op_1, op_2\})$, where $l_1, l_2, l_3$ are Lifelines enclosed by $CF_1$, $par$ is the Interaction Operator of $CF_1$, and $op_1$ and $op_2$ are the Interaction Operands of $CF_1$. $op_1$ and $op_2$ preserve their execution order if their Interaction Constraints evaluate to True respectively, and the execution order between $op_1$ and $op_2$ are decided by Interaction Operator $par$. If both Constraints of $op_1$ and $op_2$ evaluate to False, $CF_1$ is excluded and Messages $msg_1$ and $msg_2$ are ordered by Weak Sequencing. Operand $op_1$ expresses the Messages and CFs directly enclosed in it, represented by $(\{l_1, l_2, l_3\}, \{msg_2\}, \{CF_2\}, cond_{1})$, where $cond_{1}$ is $op_1$’s Interaction Constraint. In this way, the syntax of $seq$ is described recursively.

3.2 Sequence Diagram Deconstruction

To facilitate codifying the semantics of Sequence Diagrams and nested CFs in LTL formulas, we show how to deconstruct a Sequence Diagram and CFs to obtain fine-grained syntactic constructs. Eichner et al. have defined the Maximal Independent Set in [16] to deconstruct a Sequence Diagram into fragments, each of which covers multiple Lifelines. Their proposed semantics defines that entering a Combined Fragment has to be done synchronously by all the Lifelines, i.e., each Combined Fragment is connected with adjacent OSs and CFs using Strict Sequencing. Recall that CFs can be nested within other CFs. OSs and CFs directly enclosed in the same CF or Sequence Diagram are combined using Weak Sequencing, constraining their orders with respect to each individual Lifeline only [27]. To express the semantics of Weak Sequencing, we further deconstruct a Sequence Diagram into syntactic constructs on each Lifeline, which also helps us to define the semantics of nested CFs.

We project every CF $cf_m$ onto each of its covered Lifelines $l_i$ to obtain a compositional execution unit (CEU), which is denoted by $cf_m \triangleright l_i$. (The shaded rectangle on Lifeline $L_1$ in figure 2 shows an example).

Definition 4. A CEU is given by a three tuple $(l_i, oper, setEU)$, where $l_i$ is the Lifeline, onto which we project the CF, $oper$ is the Interaction Operator of the CF, and $setEU$ is the set of execution units, one for each Operand $op_n$ enclosed in the CF on Lifeline $l_i$.

Every Operand $op_n$ of CF $cf_m$ is projected onto each of its covered Lifelines $l_i$ to obtain an execution unit (EU) while projecting $cf_m$ onto $l_i$, denoted by $op_n \triangleright l_i$. If the projected Interaction Operand contains a nested Combined Fragment, a hierarchical execution unit (HEU) is obtained; otherwise a basic execution unit (BEU) is obtained, i.e., an EU is a BEU if it does not contain any other EUs. (The lower shaded rectangle on Lifeline $L_2$ in figure 2 shows an example of a BEU and the shaded rectangle on Lifeline $L_3$ shows an example of an HEU).

Definition 5. A BEU $u$ is given by a pair, $(E_u, cond)$, in which $E_u$ is a finite set of OSs on Lifeline $l_i$ enclosed in Operand $op_n$, which are ordered by the locations associated with them, and $cond$ is the Interaction Constraint of the Operand. $cond$ is True when there is no Interaction Constraint.

Definition 6. An HEU is given by $(setCEU, setBEU, cond)$, where $setCEU$ is the set of CEUs directly enclosed in the HEU, i.e., the CEUs nested within any element of $setCEU$ are not considered. $setBEU$ is the set of BEUs that are directly enclosed in the HEU.

Projecting a Sequence Diagram onto each enclosing Lifeline also obtains an EU whose Constraint is True. The EU is an HEU if the Sequence Diagram contains CFs, otherwise, it is a BEU. In an HEU, we also group the OSs between two adjacent CEUs or prior to the first CEU or after the last CEU on the same level into BEUs, which inherit the parent HEU’s Constraint, $cond$. (The upper shaded rectangle on Lifeline $L_2$ in figure 2 shows an example). The constituent BEU(s) and CEU(s) within an HEU execute sequentially, complying with their graphical order, as do the OSs in the BEU.

In the example of figure 1b, Lifeline $L_2$ demonstrates the projections of the two CFs. The Parallel is projected to obtain a CEU. The first Operand of the Parallel is projected to obtain an HEU, containing the CEU projected from the Critical Region and the BEU composed of the sending OS of $m2$. The second Operand of the Parallel is projected to obtain a BEU. The CEU of the Critical Region contains a BEU projected from its single Operand. The OS prior to the
Fig. 2. Sequence Diagram Deconstruction

Parallel is grouped into a BEU.

We provide a metamodel to show the abstract syntax of relations among BEUs, HEUs, and CEUs in figure 3. An EU can be a BEU or an HEU, and one or more EU's compose a CEU. An HEU contains one or more CEUs.

Fig. 3. Execution Unit Metamodel

3.3 Nested Combined Fragment

The syntactical definitions and deconstruction enable us to express the semantics of Sequence Diagram as a composition of nested CFs at different levels. We consider the OSs and CFs directly enclosed in the Sequence Diagram as the highest-level Interaction Fragments, which are combined using Weak Sequencing. These OSs are grouped into BEUs on each enclosing Lifeline, which observe total order within each BEU. For each Message, its sending OS must occur before its receiving OS. To enforce the interleaving semantics among Lifelines, at most one OS may execute at a time within the Sequence Diagram. The semantics of the CFs are represented at a lower-level. Each CF contains one or more Operands, which are composed using the CF’s Interaction Operator. Each Interaction Operator determines its means of combining Operands without altering the semantics of each Operand. The semantics of anOperand within each CF are described at the next level. A Sequence Diagram can be considered as anOperand whose Constraint evaluates to True. Therefore, the semantics of eachOperand containing other CFs can be described in the same way with that of a Sequence Diagram with nested CFs. AnOperand containing no other CF is considered as the bottom-level, which has a BEU on each enclosing Lifeline. The Operand whose Constraint evaluates to False is excluded. In this way, the semantics of a Sequence Diagram with CFs can be described recursively.

4 Defining Trace Semantics in LTL

The semantics of a Sequence Diagram is given by valid and invalid traces. Each trace is a sequence of OSs (i.e., event occurrences within the context of the Sequence Diagram). A Sequence Diagram model specifies complete traces, each of which describes a possible execution of the system, whereas a CF of the Sequence Diagram defines a collection of their subtraces. These subtraces may interleave with other OSs appearing in the Sequence Diagram but outside the CF, connecting using Weak Sequencing to make complete traces of the Sequence Diagram [30]. A trace derived from a Sequence Diagram can be finite, denoted as σ[1,n] = σ1σ2...σn. The trace derived from a Sequence Diagram can also be infinite if it expresses the behavior of infinite iterations in terms of Loop with infinity upper bound, denoted as σ = σ1σ2...σn....

This paper presents a framework to characterize the traces of Sequence Diagram in Linear Temporal Logic (LTL). LTL is a formal language for specifying the orders of events and states in terms of temporal operators and logical connectives. We use LTL formulas to express the semantic rules prescribed by Sequence Diagram constructs, each of which defines the execution orders among OSs. Note that an LTL formula represents infinite traces. In the case that a Sequence Diagram expresses a set of finite traces, we need to handle the mismatch between an LTL formula and a Sequence Diagram’s finite trace semantics. To bridge the gap, we adapt the finite traces of Sequence Diagrams without altering their semantics by adding stuttering of a no-op after the last OS σn of each trace [18]. Then, LTL formulas can express these traces, each of which is denoted as Σ∗seqτ, where Σseq is the set of OSs of seq and Σ∗seq represents a finite sequence of OSs of seq. τ is a no-op, i.e., τ is an empty event occurrence and not observable, and τ represents an infinite sequence of no-ops.

A Sequence Diagram with Negative or Assertion CFs can specify desired properties as well as possible system executions in terms of traces. The Sequence Diagram for specifying desired properties only considers the OSs related to the properties. We represent the traces of properties with partial traces semantics, which allows other OSs do not appear in the Sequence Diagram but appear in the system executions to interleave the partial traces. Our framework supports partial traces semantics to express certain properties, including safety and consistency, with a Sequence Diagram.
5 Specifying Sequence Diagram in LTL

In this section, we describe how to use LTL formulas to codify the semantic rules of Sequence Diagrams as shown in section 2. Formalizing the semantics of a notation can be challenging, especially if we consider all semantic constraints at once. To reduce the complexity and to improve the readability, we devise an LTL framework, comprised of simpler definitions, we call templates, to represent each semantic aspect (i.e., the execution order of event occurrences imposed by individual constructs) as a separate concern. To capture the meanings of nested CFs, we provide a recursively defined template, in which each individual CF’s semantics is preserved (e.g., the inner CF’s semantics is not altered by other CFs containing it). These templates can then be composed using temporal logic operators and logical connectives to form a complete specification of a Sequence Diagram. In this way, if the notation evolves, many of the changes can still be localized to respective LTL templates.

To facilitate the representation of a Sequence Diagram in LTL, we define a collection of auxiliary functions (see table 1) to access information of a Sequence Diagram. We provide the algorithms to calculate some auxiliary functions in Appendix A. These functions are grouped into two categories. The functions within the first group return the syntactical constructs of a Sequence Diagram. For instance, function $\text{SND}(j)$ returns the sending OS of Message $j$. The functions within the second group return the constructs, either whose Constraints evaluate to True or which are contained in the constructs whose Constraints evaluate to True. For instance, for Parallel CF1 in figure 1b, function $\text{nested}(CF1)$ returns a singleton set containing Critical Region $CF2$ if the Constraint of the first Operand of $CF1$ evaluates to True. Otherwise, $\text{nested}(CF1)$ returns an empty set, and Critical Region $CF2$ is ignored to reflect the semantic rule 3 which is general to all CFs (see section 2.2). Functions $\text{MSG}(p)$, $\text{LN}(p)$, $\text{AOS}(q)$ are overloaded where $p$ can be an Interaction Operand, a CF, or a Sequence Diagram, and $q$ can be $p$, an EU, or a CEU.

<table>
<thead>
<tr>
<th>Function</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{LN}(p)$</td>
<td>return the set of all Lifelines in $p$.</td>
</tr>
<tr>
<td>$\text{MSG}(p)$</td>
<td>return the set of all Messages directly enclosed in $p$.</td>
</tr>
<tr>
<td>$\text{SN}(j)$</td>
<td>return the sending OS of Message $j$.</td>
</tr>
<tr>
<td>$\text{Rcv}(j)$</td>
<td>return the receiving OS of Message $j$.</td>
</tr>
<tr>
<td>$\text{Reply}(u)$</td>
<td>return the reply Message of a synchronous Message containing OS $u$.</td>
</tr>
<tr>
<td>$\text{typeOS}(u)$</td>
<td>return the type of OS $u$, which is a sending OS or a receiving OS.</td>
</tr>
<tr>
<td>$\text{TOP}(u)$</td>
<td>return the set of Interaction Operands whose Constraints evaluate to True.</td>
</tr>
<tr>
<td>$\text{TBEU}(u)$</td>
<td>return the set of all Interaction Operands enclosed in a CF containing a sole Operand.</td>
</tr>
<tr>
<td>$\text{pre}(u)$</td>
<td>return the set of OSs which are enabled (i.e., the Constraints associated with it evaluate to True) and chosen to execute in $q$.</td>
</tr>
<tr>
<td>$\text{AOS}(q)$</td>
<td>return the set of OSs which are enabled (i.e., the Constraints associated with it evaluate to True) and chosen to execute in $q$.</td>
</tr>
<tr>
<td>$\text{TOS}(u)$</td>
<td>return the set of OSs which are enabled (i.e., the Constraints associated with it evaluate to True) and chosen to execute in $q$.</td>
</tr>
<tr>
<td>$\text{post}(u)$</td>
<td>return the set of OSs which may happen right after CEU $u$, which can be calculated in a similar way as $\text{pre}(u)$.</td>
</tr>
<tr>
<td>$\text{nested}(u)$</td>
<td>return the set of CFs, which are directly enclosed in CF $u$’s Interaction Operands whose Constraints evaluate to True.</td>
</tr>
<tr>
<td>$\text{ABEU}(u)$</td>
<td>return the set of all Interaction Operands enclosed in CF $u$.</td>
</tr>
<tr>
<td>$\text{BEU}(u)$</td>
<td>return the set of all Interaction Operands enclosed in CF $u$.</td>
</tr>
</tbody>
</table>

5.1 Basic Sequence Diagram

We start with defining an LTL template, called $\Pi_{\text{seq}}^{\text{Basic}}$ (see figure 4), to represent the semantics of a basic Sequence Diagram. The semantic rules for basic Sequence Diagram $\text{seq}$ defined in section 2.1 are codified separately using formulas $\alpha_g$, $\beta_j$, and $\xi_{\text{seq}}$.

$\alpha_g$ focuses on the intra-lifeline behavior to enforce rules 1 and 2. Recall that when projecting basic Sequence Diagram $\text{seq}$ onto its covered Lifelines, $\text{LN}(\text{seq})$, we obtain BEU $g$ for each Lifeline $i$, denoted as $\text{seq}_i^\text{1}$. Each BEU $g$ contains a trace of OSs, $\sigma[r..(r+$
For a given basic Sequence Diagram, \( seq \), with \( j \) Messages and 2\( j \) event occurrences (each Message has a sending event occurrence and a receiving event occurrence), \( \Sigma_{sem}^{seq} \) is the set of event occurrences of \( seq \). \( \Sigma_{seq}^{seq} \subseteq \Sigma \), where \( \Sigma \) is the universe of event occurrences. The set of valid traces, \( (\Sigma_{seq}^{seq})^{*} \), contains finite traces derived from \( seq \) based on the semantic rules of Sequence Diagrams. \( \Sigma_{seq}^{LTL} \) is the set of event occurrences of LTL representation of \( seq \). \( \Pi_{seq}^{Basic} \), where \( \Sigma_{seq}^{LTL} = \Sigma_{sem}^{seq} \cup \{ \tau \} \), \( \tau \) is an invisible event occurrence which does not occur in \( seq \), i.e., \( \tau \in (\Sigma \setminus \Sigma_{seq}^{seq}) \). \( \Sigma_{seq}^{LTL}^{\omega} \) represents all infinite traces that satisfy \( \Pi_{Basic}^{seq} \). For each trace \( \sigma \in (\Sigma_{seq}^{LTL}^{\omega}) \), function \( pre_{i}(\sigma) \) returns the prefix of length \( i \) of \( \sigma \), i.e., \( \sigma[1..i] \). We lift function \( pre_{i}(\sigma) \) to \( PRE_{i}((\Sigma_{seq}^{LTL}^{\omega})) \) to apply to a set of traces. Function \( PRE_{i}((\Sigma_{seq}^{LTL}^{\omega})) \) returns the set of the prefixes of the traces within \( (\Sigma_{seq}^{LTL}^{\omega}) \), where the length of each prefix must be \( i \), i.e., \( PRE_{i}((\Sigma_{seq}^{LTL}^{\omega})) = \{ pre_{i}(\sigma) | \sigma \in (\Sigma_{seq}^{LTL}^{\omega}) \} \).

**Lemma 1:** For a given Sequence Diagram, \( seq \), with \( j \) Messages, if \( \sigma \in (\Sigma_{seq}^{LTL}^{\omega}) \), then \( \sigma \) must have the form, \( \sigma = \sigma[1..2j] \cdot \tau^{\omega} \), where \( \sigma[1..2j] \) contains no \( \tau \).

**Proof:** If \( \sigma = \Pi_{seq}^{Basic} \), then \( \sigma = \varepsilon_{seq} \). We can directly infer from sub-formula \( \varepsilon_{seq} \), that, in \( \sigma \), only one OS of \( seq \) can execute at a time, and \( \sigma \) should execute uninterrupted until all the OSs of \( seq \) have taken place. Similarly, we can infer from the assumption that \( \sigma = \bigwedge_{j \in MSG(seq)} \rho_{j} \). From sub-formula \( \bigwedge_{j \in MSG(seq)} \rho_{j} \), we can infer that each OS within \( seq \) can execute once and only once in \( \sigma \). \( seq \) contains \( j \) Messages with 2\( j \) OSs, so \( \sigma \) should have the form, \( \sigma = \sigma[1..2j] \cdot \tau^{\omega} \).

The semantics of a basic Sequence Diagram is given by a set of valid, finite traces, while LTL formulas describe infinite traces. To represent the semantics of a basic Sequence Diagram using LTL formulas, we need to bridge the gap by adding stuttering of \( \tau \) after each finite trace of the Sequence Diagram. For instance, for a given Sequence Diagram, \( seq \), \( \forall v \in (\Sigma_{seq}^{LTL}^{\omega}) \), \( v \) is extended to \( v \cdot \tau^{\omega} \) without changing the meaning of \( seq \).

We wish to prove that for a given Sequence Diagram, \( seq \), with \( j \) Messages, \( \forall v \in (\Sigma_{seq}^{LTL}^{\omega}) \), \( v \cdot \tau^{\omega} = \Pi_{seq}^{Basic} \), i.e., \( v \cdot \tau^{\omega} \in (\Sigma_{seq}^{LTL}^{\omega}) \). The semantic rule of \( seq \) defines that each OS occurs once and only once. Thus, \( \forall v \in (\Sigma_{seq}^{LTL}^{\omega}) \), \( |v| = 2j \). From Lemma 1, we learn that \( \forall v \in (\Sigma_{seq}^{LTL}^{\omega}) \), \( v = \sigma[1..2j] \cdot \tau^{\omega} \), where \( \sigma[1..2j] \) contains no \( \tau \). \( \sigma[1..2j] \in PRE_{2j}((\Sigma_{seq}^{LTL}^{\omega})) \). If \( \forall v \in (\Sigma_{seq}^{LTL}^{\omega}) \), \( v \cdot \tau^{\omega} \in (\Sigma_{seq}^{LTL}^{\omega}) \), we can infer that, \( v \in PRE_{2j}((\Sigma_{seq}^{LTL}^{\omega})) \), i.e., \( (\Sigma_{seq}^{LTL}^{\omega}) \subseteq PRE_{2j}((\Sigma_{seq}^{LTL}^{\omega})) \).

We also wish to prove that \( \forall \sigma \in (\Sigma_{seq}^{LTL}^{\omega}) \), \( \sigma[1..2j] \in (\Sigma_{seq}^{LTL}^{\omega}) \), i.e., \( PRE_{2j}((\Sigma_{seq}^{LTL}^{\omega})) \subseteq (\Sigma_{seq}^{LTL}^{\omega}) \).

**Theorem 1:** For a given Sequence Diagram, \( seq \), with \( j \) Messages, \( (\Sigma_{seq}^{LTL}^{\omega}) \) and \( PRE_{2j}((\Sigma_{seq}^{LTL}^{\omega})) \) are equal.

We provide the proof of theorem 1 in appendix A.
5.3 Combined Fragments

A Combined Fragment (CF) can modify the sequential execution of its enclosed OSs on each Lifeline. Moreover, a Sequence Diagram can contain multiple CFs that can be nested within each other. Though these features make a Sequence Diagram more expressive, they increase the complexity of representing all the features, making it more expressive.

The complexity of representing all the features makes a Sequence Diagram more expressive, which can be nested within each other. Though these over, a Sequence Diagram can contain multiple CFs and OSs present in a Sequence Diagram. We introduce a new template $\Phi_{CF}$ (see figure 7). We introduce a new template $\Phi_{CF}$ to assert the semantics of each CF directly enclosed in $seq$. Template $\Pi_{seq}$ is a conjunction of the formulas $\alpha_g$, $\beta_j$, $\Phi_{CF}$, and $\varepsilon_{seq}$, which is equivalent to the LTL template of basic Sequence Diagram if $seq$ does not contain any CF.

When multiple CFs and OSs present in a Sequence Diagram, they are combined using Weak Sequencing — CFs and OSs on the same Lifeline execute sequentially, whereas CFs and OSs on different Lifelines execute independently, except the pairs of OSs belonging to Messages. Thus, we project Sequence Diagram $seq$ with CFs onto Lifelines to obtain a collection of CEUs and EUUs, facilitating us to focus on OSs on each single Lifeline. The OSs directly enclosed in $seq$ are grouped into BEUs, whose semantics are enforced by a conjunction of $\alpha_g$ for each BEU $g$. The order of OSs within Messages directly enclosed in $seq$ are enforced by a conjunction of $\beta_j$ for each Message $j$. $\varepsilon_{seq}$ enforces that at most one OS can execute at a time for all the OSs within $seq$. One way to implement these formulas is provided in Appendix B. If $seq$ contains a Loop, the OSs of $seq$ includes OSs in each iteration of the Loop.

Template $\Phi_{CF}$ (see figure 8) considers three cases. Formula (1) asserts the case that the CF contains no...
Operand whose Constraint evaluates to True. Thus, the OSs within the CF are excluded from the traces. Semantics rule 3 for CFs states Weak Sequencing among the CF’s preceding Interaction Fragments and succeeding ones, which is enforce by formula $\eta^{CF}$. Functions $\text{pre}(CF \uparrow_i)$ and $\text{post}(CF \uparrow_i)$ return the set of OSs which may happen right before and after CEU $CF \uparrow_i$ respectively. The formula $\eta^{CF}$ enforces that the preceding set of OSs must happen before the succeeding set of OS on each Lifeline $i$, which sets to True if either $\text{pre}(CF \uparrow_i)$ or $\text{post}(CF \uparrow_i)$ returning empty set. Formula (2) asserts the case that $CF$ contains at least one Operand whose Constraint evaluates to True, and $CF$ is not an Alternatives or a Loop. The first conjunct $\Psi^{CF}$ defines the semantics of OSs directly enclosed in $CF$. The second conjunct states that the semantics of each $CF_i$, which is directly enclosed in $CF$, is enforced by each $\Phi^{CF}_i$. In this way, $\Phi^{CF}$ can be defined recursively until it has no nested CFs.

Template $\Psi^{CF}_i$ captures the semantics that is common to all CFs (except Alternatives and Loop) (see figure 9). Sub-formula $\gamma^{CF}_i$ enforces semantic rule 1, which defines the sequential execution on every Lifeline $i$. The first conjunct enforces that the preceding set of OSs must happen before each OS in $CF$ on Lifeline $i$, and the second conjunct enforces that the succeeding set of OSs must take place afterwards. $\theta^{CF}$ states semantic rule 2, which defines the order among OSs directly enclosed in $CF$. $\phi^{CF}$ is a conjunction of $\alpha_g$s and $\beta_j$s. The $\alpha_g$ is a conjunction of all $\alpha_g$ of each Lifeline, where $g$ is a BEU whose Constraint evaluates to True. The $\beta_j$ is a conjunction of $\beta_j$ of each Message.

Formula (3) asserts the case for Alternatives and Loop, which contain at least one Operand whose Constraint evaluates to True. For Alternatives, $\Psi^{CF}_\text{alt}$ defines the semantics of OSs and CFs directly enclosed in $CF$. $\Psi^{CF}_\text{alt}$ and $\phi^{CF}$, for $CF_i$ directly nested in the Alternatives form an indirect recursion (see figure 12). The semantics of Loop is defined in a similar way (see figure 17).

The semantic rule varies for CFs with different Operators, which is enforced by adding different semantics constraints on $\phi^{CF}$ for each individual CF respectively. The semantics specifics for different types of CF Operators are defined as below.

5.3.1 Concurrency
The Parallel represents concurrency among its Operands. The OSs of different Operands within Parallel can be interleaved as long as the ordering imposed by each Operand is preserved. Figure 1b is an example of Parallel with two Operands. The OSs within the same Operand respect the order along a Lifeline or a Message, whereas the OSs from different Operands may execute in any order even if they are on the same Lifeline. For instance, OS $r_5$ (i.e., the receiving OS of Message $m_5$) and OS $r_6$ on Lifeline $L_2$ maintain their order. OS $r_2$ and OS $s_5$ on Lifeline $L_1$ many execute in any order since they are in different Operands. Parallel does not add extra constraint to the general semantic rules of CF. Thus, the semantics of Parallel can be formally defined,

$$\Psi^{CF}_{\text{par}} = \theta^{CF} \land \bigwedge_{i \in LN(CF)} \gamma^{CF}_i$$

5.3.2 Branching
Collectively, we call Option, Alternatives, and Break branching constructs.

5.3.2.1 Representing Option: The Option represents a choice of behaviors that either the (sole) Operand happens or nothing happens. As Option does not add any extra constraint to the execution of its sole Operand, its semantics can be formally defined as the template,

$$\Psi^{CF}_{\text{opt}} = \theta^{CF} \land \bigwedge_{i \in LN(CF)} \gamma^{CF}_i$$

Figure 10 is an example of Option. The OSs within the Option execute if $\text{cond}$ evaluates to True. Otherwise, the Option is excluded, and its semantics is defined by formula $\eta$, i.e., Messages $m_1$ and $m_4$ are combined with Weak Sequencing.

5.3.2.2 Representing Alternatives: The Alternatives chooses at most one of its Operands to execute. Each Operand must have an explicit or an implicit or an “else” Constraint. The chosen Operand’s Constraint must evaluate to True. An implicit Constraint always evaluates to True. The “else” Constraint is the negation of the disjunction of all other Constraints in the enclosing Alternatives. If none of the Operands whose Constraints evaluate to True, the Alternatives is excluded. The translation of an Alternatives into an LTL formula must enumerate all possible choices of executions in that only OSs of one of the Operands, whose Constraints evaluate to True, will happen. LTL formula $\Phi^{CF}_{\text{alt}}$ in figure 12 defines the semantics of Alternatives, which is a conjunction of $\Phi^{m}_{\text{alt}}$. Each $\Phi^{m}_{\text{alt}}$ represents the semantics of Operand $m$, whose

$$\Pi_{\text{seq}} = \bigwedge_{i \in LN(seq)} \bigwedge_{g \in TBEU(seq)} \alpha_g \land \bigwedge_{j \in MSG(seq)} \beta_j \land \bigwedge_{CF \in \text{nested(seq)}} \Phi^{CF} \land \varepsilon_{\text{seq}}$$

Fig. 7. LTL templates for Sequence Diagram with Combined Fragments
Operand evaluates to False, its variable first and the third operand is chosen by evaluating operands evaluate to assume the constraints of the first and the third with three operands enclosing three lifelines. We True assertion, the first conjunct expresses that only one operand whose interaction constraint evaluates to clause) is using a boolean variable Alternatives.

\[
\Phi^\text{CF} = \begin{cases}
\eta^\text{CF} & \text{if } |\text{TOP}(CF)| = 0 \\
\eta^\text{CF} \land \Psi^\text{CF} \land \Phi^\text{CF}_i & \text{if } |\text{TOP}(CF)| > 0 \land (\text{type}\text{CF}(CF) \neq \text{alt}) \land (\text{type}\text{CF}(CF) \neq \text{loop}) \\
\eta^\text{CF} & \text{if } |\text{TOP}(CF)| > 0 \land ((\text{type}\text{CF}(CF) = \text{alt}) \lor (\text{type}\text{CF}(CF) = \text{loop}))
\end{cases}
\]

\[
\eta^\text{CF} = \bigwedge_{i \in \text{LN}(CF)} (\bigwedge_{OS_{\text{post}} \in \text{post}(CF_{\text{t}_i})} (\neg OS_{\text{post}}) \bigwedge_{OS_{\text{pre}} \in \text{pre}(CF_{\text{t}_i})} (\bigwedge_{OS_{\text{post}} \in \text{post}(CF_{\text{t}_i})} (\bigwedge_{OS_{\text{pre}} \in \text{pre}(CF_{\text{t}_i})})
\]

\[
\Psi^\text{CF} = \theta^\text{CF} \land \bigwedge_{i \in \text{LN}(CF)} \alpha_i \land \bigwedge_{j \in \text{MSG}(\text{TOP}(CF))} \beta_j
\]

\[
\gamma^\text{CF}_i = \bigwedge_{OS \in \text{TOS}(CF_{\text{t}_i})} (\neg OS \bigwedge_{OS_{\text{pre}} \in \text{pre}(CF_{\text{t}_i})} (\bigwedge_{OS_{\text{post}} \in \text{post}(CF_{\text{t}_i})} (\bigwedge_{OS_{\text{pre}} \in \text{pre}(CF_{\text{t}_i})}
\]

Constraint evaluates to True, which is achieved by function TOP(CF).

The semantics of the chosen operand (if clause) is described by \( \theta^\text{CF}_m \), \( \gamma^\text{CF}_m \), and \( \Phi^\text{CF}_i \), where \( \theta^\text{CF}_m \) defines the partial order of OSs within the chosen operand and \( \Phi^\text{CF}_i \) defines the semantics of CFs directly enclosed in the chosen operand. Functions \( \Psi^\text{CF}_m \) and \( \Phi^\text{CF}_i \) invoke each other to form indirect recursion. The sub-formula of the unchosen operand (else clause) returns True, i.e., the unchosen operand does not add any constraint. The weak sequencing of the alternatives is represented by \( \gamma^\text{CF}_i \) instead of \( \gamma^\text{CF}_m \), which enforces weak sequencing between the chosen operand and the preceding/succeeding OSs of the alternatives.

One way to implement the chosen operand (if clause) is using a boolean variable \( \text{exe} \) for each operand whose interaction constraint evaluates to True. The variable \( \text{exe} \) should satisfy the following assertion,

\[
\bigwedge_{i \in \{1..m\}} \text{exe}_i \land \bigwedge_{i \in \{1..m\}} (\text{exe}_i \rightarrow \text{cond}_i)
\]

The first conjunct expresses that only one \( \text{exe} \) sets to True, i.e., exactly one operand is chosen. The second conjunct enforces that interaction constraint of operand whose \( \text{exe} \) sets to True must evaluate to True. Figure 11 shows an example of an alternatives with three operands enclosing three lifelines. We assume the constraints of the first and the third operands evaluate to True, the one of the second operand evaluates to False. Only one between the first and the third operands is chosen by evaluating its variable \( \text{exe} \) to True.

5.3.2.3 Representing Break: The Break states that if the operand’s constraint evaluates to True, it executes instead of the remainder of the enclosing interaction fragment. Otherwise, the operand does not execute, and the remainder of the enclosing interaction fragment executes. A Break can be represented as an alternatives in a straightforward way. We rewrite the semantics interpretation of Break as an alternatives with two operands, the operand of Break and the operand representing the remainder...
of the enclosing Interaction Fragment. The Constraint of the second Operand is the negation of the first Operand’s Constraint. For example, the Interaction Fragment enclosing the Break is the first Operand of the Parallel rather than the Parallel (see figure 13). We rewrite the Sequence Diagram, using Alternatives to replace Break (see figure 14). cond3 is the Constraint of Break and cond4 is the negation of it. In this way, only oneOperand can be chosen to execute. Thus, the LTL representation of Break can be represented as the LTL formula for Alternatives with two Operands.

\[
\psi_{\text{CF}} = \bigwedge_{m \in \text{TOP}(CF)} \psi_{\text{alt}}^m
\]

\[
\psi_{\text{alt}}^m = \begin{cases} \hat{\theta}_{\text{CF}}^m \land \bigwedge_{i \in \text{LN}(CF)} \hat{\gamma}_{i,m}^CF \land CF_i \in \text{nested}(m) & \text{if } m \text{ is the chosen Operand} \\ \text{True} & \text{else} \end{cases}
\]

\[
\hat{\theta}_{\text{CF}}^m = \bigwedge_{i \in \text{LN}(m), g \in \text{BEU}(m^\pi_i)} \alpha_g \land \bigwedge_{j \in \text{MSO}(\text{TOP}(m))} \beta_j
\]

\[
\hat{\gamma}_{i,m}^CF = \bigwedge_{OS \in \text{OS}(m^\pi_i)} \left((\neg \text{OS}) \bigwedge_{OS_{\text{pre}} \in \text{pre}(CF^\pi_i)} (\diamond \text{OS}) \right) \land \left((\bigwedge_{OS_{\text{post}} \in \text{post}(CF^\pi_i)} (\neg \text{OS})) \bigwedge_{OS} (\text{OS}) \right)
\]

Fig. 12. LTL formula for Alternatives

5.3.4 Iteration

The Loop represents the iterations of the sole Operand, which are connected by Weak Sequencing. To restrict the number of iterations, the Operand’s Constraint may include a lower bound, minint, and an upper bound, maxint, i.e., a Loop iterates at least the minint number of times and at most the maxint number of times. If the Constraint evaluates to False after the minint number of iterations, the Loop will terminate. First, we consider fixed Loop. Figure 16 is an example of fixed Loop which iterates exactly three times.

5.3.3 Atomicity

The Critical Region represents that the execution of its OSs is in an atomic manner, i.e., restricting OSs within its sole Operand from being interleaved with other OSs on the same Lifeline. In the example of figure 1b, a Critical Region is nested in the first Operand of the Parallel. OSs s2, r5 and r6 can not interleave the execution of OSs r3 and s4. Formula \(\psi_{\text{CF}}^{\text{critical}}\) presents the semantics for Critical Region (see figure 15). \(\theta_{\text{CF}}^C\) and \(\gamma_{i}^CF\) have their usual meanings. \(\delta_{M_1, M_2}\) enforces that on each Lifeline, if any of the OSs within the CEU of Critical Region (representing as the set of \(M_1\)) occurs, no other OSs on that Lifeline (representing as the set of \(M_2\)) are allowed to occur until all the OSs in \(M_1\) finish. Thus, \(M_1\) is guaranteed to execute as an atomic region. Function “\(\sim\)” represents the removal of the set of OSs for Critical Region from the set of OSs for Sequence Diagram seq on Lifeline \(i\).

Fig. 13. Example for Break

Fig. 14. Representing Break using Alternatives

Fig. 15. Example for Loop
Fig. 15. LTL formula for Critical Region

number of iterations and \( n \), representing the current iteration number on Lifeline \( i \). The Loop in iteration \( n \) can be represented as \( \text{Loop}[n] \). For example, the Loop in figure 16 has three iterations, \( \text{Loop}[1] \), \( \text{Loop}[2] \) and \( \text{Loop}[3] \). Figure 17 shows an LTL formula for a Loop. \( \hat{\theta}_R \) overloads \( \theta^C_F \), which asserts the order of OSs during each iteration. \( \hat{\gamma}_{i, R} \) enforces the Weak Sequencing among Loop iterations and its preceding/following sets of OSs on each Lifeline \( i \), i.e., the first Loop iteration execute before the preceding set of OSs, and the last Loop iteration execute after the succeeding set of OSs. An OS and the value of \( n \) together represent the OS in a specific iteration, (e.g., the element \( (OS_k[n]) \) expresses \( OS_k \) in the \( n \)th iteration). The OSs within nested CFs are renamed with the same strategy. Template \( \kappa_{i, R} \) is introduced to enforce Weak Sequencing among Loop iterations, e.g., on the same Lifeline, \( OS_j[n+1] \) can not happen until \( OS_k[n] \) finishes execution.

If the Loop is not fixed and it does not have infinity upper bound, we need to evaluate the Interaction Constraint of the its sole Operand during each iteration. Similarly to fixed Loop, the finite but not fixed Loop can be unfolded by repeating iterations. To keep the Constraint of each iteration unique, an array is defined to rename the Constraint, e.g., the Constraint of iteration \( n \) is represented as \( \text{cond}[n] \). The order of OSs during each iteration is asserted as the fixed Loop. Two adjacent iterations are connected using Weak Sequencing. If \( n \leq \text{minint} \), \( \text{cond}[n] \) sets to True and the Loop executes. If \( \text{minint} < n < \text{maxint} \), the Loop executes only if \( \text{cond}[n] \) evaluates to True. Otherwise, the Loop terminates and the Constraints of remaining iterations (i.e., from \( \text{cond}[n+1] \) to \( \text{cond}[\text{maxint}] \)) set to False. The Loop no longer executes when its iteration reaches \( \text{maxint} \).

5.3.5 Negation

A Negative represents that the set of traces within a Negative are invalid. For example, there are three traces defined by the Negative in figure 18 [\( s_1, s_2, r_1, r_2 \), \( s_2, s_1, r_1, r_2 \), and \( s_1, r_1, s_2, r_2 \)], which are invalid traces. Formula \( \Psi^C_F = \theta^C_F \) formally defines the semantics of Negative CF, asserting the order of OSs directly enclosed in it. If the Interaction Constraint of the Negative evaluates to False, the traces within the Negative may be either invalid traces or the Operand is excluded (see subsection 5.5.1 for details).

5.3.6 Assertion

An Assertion representing, on each Lifeline, a set of mandatory traces, which are the only valid traces following the Assertion’s preceding OSs. Its semantics is formally defined as \( \Psi^C_F = \theta^C_F \) in figure 20. \( \theta^C_F \) and \( \gamma_{DF} \) have their usual meanings. Function \( \lambda_{\Delta E \left( \text{pre}(L_i), \text{in}(C_{i}) \right)} \) represents that on Lifeline \( i \), if all the OSs in the set of \( \text{pre} \), no other OSs in Sequence Diagram \( \text{seq} \) are allowed to happen until all the OSs in assertion complete their execution. The function prevents the Assertion and its preceding OSs from being interleaved by other OSs, which is required when the Assertion is nested within other CFs, such as Parallel. For example (see figure 19), an Assertion is nested within a Parallel. The OSs within the CEU of the Assertion execute right after their preceding OSs finish execution. On Lifeline \( L_3 \), after the execution of OS \( r_2 \), OSs \( s_3 \) and \( r_4 \) must happen without being interleaved by OS \( s_6 \).

5.3.7 Weak Sequencing

The Weak Sequencing restricts the execution orders among its Operands along each Lifeline. Figure 21 is an example of Weak Sequencing, where OS \( s_4 \) can not happen until OS \( s_3 \) execute, whereas OS \( s_4 \) and \( r_3 \) may...
The LTL definition of Weak Sequencing is given as below.

\[
\psi_{weak}^{CF} = \theta^{CF} \land \bigwedge_{i \in LN(CF)} \gamma_i^{CF} \land \bigwedge_{i \in LN(CF)} \kappa_i^{CF} \land \bigwedge_{n \in [1..R]} \Phi^{CF}[n]
\]

Templates \(\theta^{CF}\) and \(\gamma_i^{CF}\) have their usual meanings. \(\kappa_i^{CF}\) specifies the execution orders between adjacent Operands, as well as enforcing the Weak Sequencing between the CF and its preceding/succeeding Interaction Fragments \(\gamma_i^{CF}\). (The LTL formula keeps \(\gamma_i^{CF}\) for clarity and consistency.)

### 5.3.8 Strict Sequencing

The Strict Sequencing imposes an order among OSs within different Operands. For anOperand, all OSs must take place before any OS of its following Operand. In other words, any OS of an Operand can not execute until all OSs of the previous Operand finish execution. The Strict Sequencing enforces the synchronization among multiple Lifelines, *i.e.*, any covered Lifeline needs to wait other Lifelines to enter the second or subsequent Operand together. (Weak Sequencing enforces the order among Operands on each Lifeline.) For example, OS s4 will not execute until all OSs within the first Operand, including s1, r1, s2, r2, s3, and r3 complete execution.

Figure 23 presents the semantics of Strict Sequencing. Template \(\theta^{CF}\) has its usual meaning. The Strict Sequencing and its adjacent Interaction Fragments are connected using Weak Sequencing, which is expressed by template \(\gamma_i^{CF}\) as usual. Function \(\chi_k\) asserts the order between each Operand \(k\) and its preceding
Operand whose Constraint evaluates to True. Function $\text{preEU}(u)$ returns the set of OSs within EU $v$ which happen right before EU $u$, i.e., the Constraint of EU $v$ evaluates to True. Function $\text{NFTOP}(CF)$ returns the set of Interaction Operands whose Constraints evaluate to True within CF, excluding the first one.

5.3.9 Coregion

Fig. 24. Example for Coregion

A Coregion is an area of a single Lifeline, which is semantically equivalent to a Parallel that the OSs are unordered. Figure 24 shows an example of Coregion, where OS r3 and r4 may execute in any order. We represent the Coregion into an LTL formula in a similar way with a Parallel. Each OS within the Coregion is considered as an Operand of the Parallel, no order of OSs within a BEU needs to be defined. Template $\theta^{CF}$ is excluded because a Coregion does not contain any complete Messages. Complete messages are defined by the CF or Sequence Diagram which directly encloses them. $\gamma^{CF}$ describes the Weak Sequencing between Coregion and its preceding/succeeding set of OSs. The LTL formula does not describe the Messages containing the OSs of the Coregion.

$\psi^{CF}_{coregion} = \gamma^{CF}$

5.4 Ignore and Consider

So far, all the CFs define a collection of partial traces, which only interleave the OSs appearing in the Sequence Diagram to form a complete trace. The Ignore and Consider CFs allow other OSs that are not considered or ignored extend the traces. Ignore and Consider take into consideration the message types which do not appear in the Sequence Diagram. Generally, the interpretation of a Sequence Diagram only considers the message types explicitly shown in it. An Ignore specifies a list of message types which needs to be ignored within the CF. For instance, Messages whose type is m3 are ignored in the Ignore CF (see figure 25). A Consider specifies a list of considered message types, which is equivalent to specifying other possible message types to be ignored. For instance, the Consider CF only considers Messages whose types are m2, m3 or m5 (see figure 26). To design well-formed Ignore or Consider, some syntactical constraints need to be mentioned. For Consider, only Messages whose types specified by the list of considered Messages can appear in the CF [30]. For Ignore, the ignored message types are suppressed in the CF [30].

![Fig. 25. Example for Ignore](image)

![Fig. 26. Example for Consider](image)

Within the Ignore, the Messages appearing in the CF and the Messages which are explicitly ignored in the CF need to be constrained (see figure 27). $\theta^{CF}$ and $\gamma^{CF}$ have their usual meanings, which describe the semantics of Messages appearing in the Ignore. Each OS of the ignored Messages executes only once, which is enforced by $\tilde{\alpha}_{\text{ignoreOS}(CF)}$. We introduce function $\text{ignoreMsg}(CF)$ to return the set of Messages of the ignored message types which occur in CF, which can be finite or infinite. Function $\text{ignoreOS}(CF)$ returns the set of OSs associated with Messages of ignored message types, which can also be finite or infinite. Formula $\tilde{\beta}_{k}$ enforces that, for each ignored Message $k$, its sending OS must happen before its receiving OS. Formula $\gamma^{CF}_{\text{IgnoreOS}(CF)_{\uparrow}}$ extends $\gamma^{CF}_{k}$, which enforces any OS of the set of the ignored OSs can only happen within the CEU of the Ignore on each Lifeline, formally,

$$
\gamma^{CF}_{\text{IgnoreOS}(CF)_{\uparrow}} = \bigwedge_{OS \in S} ((\neg OS \tilde{\Upsilon} \bigwedge_{OS_{\text{pre}} \in \text{pre}(CF_{\uparrow})} (\otimes OS_{\text{pre}})) \land ((\bigwedge_{OS_{\text{post}} \in \text{post}(CF_{\uparrow})} (\neg OS_{\text{post}})) \tilde{\Upsilon} (\otimes OS)))
$$

where S can be replaced using $\text{ignoreOS}(CF \uparrow)$. Formula $\varepsilon_{\text{seq.ignoreOS}(CF)}$ extends $\varepsilon_{\text{seq}}$ to include the OSs of ignored Messages in the set of OSs of seq, formally,
Fig. 23. LTL formula for Strict Sequencing

Thus, function \( \varepsilon_{seq,ignoreOS} \) of Sequence Diagram with Ignore enforces the interleaving semantics among OSs appearing in seq and OSs of the ignored Messages.

As the dual Operator of ignore, the semantics of a CF with Operator consider is equivalent to ignoring all possible message types except the considered types. In this way, the LTL formula of Ignore can be adapted to represent the semantics of Consider (see figure 28). Function \( \text{AllMsg}(CF) \setminus \text{considerMsg}(CF) \) returns the Messages which are not considered but occur in CF, where \( \text{AllMsg}(CF) \) returns all possible Messages, including Messages of considered types and Messages of ignored types. \( \text{considerMsg}(CF) \) returns the Messages of considered types. Function \( \Sigma \setminus \text{considerOS}(CF) \) returns all possible OSs within CF except the OSs of Considered Messages, where \( \Sigma \) is the set of all possible OSs including considered OSs and ignored OSs, and \( \text{considerOS}(CF) \) returns the set of OSs of considered Messages. In this way, the Sequence Diagram with Consider or Ignore no longer derive complete traces.

5.5 Semantic Variations

OMG provides the formal syntax and semi-formal semantics for UML Sequence Diagrams, leaving semantic variation points for representing different applications. Mickei and Waeselynck have collected and categorized the interpretations of the variants [26]. In the following subsections, we discuss how to use our LTL framework to formalize the variations of Negative, Strict Sequencing, and Interaction Constraints.

5.5.1 Variations of Negative

Recall that the traces defined by a Negative are considered as invalid traces. For example, if the Operand of Negative \( S \), which does not contain any other Negative, defines a set of valid traces, then the set of traces defined by \( S \) are invalid traces. In the case that the Constraint of the Operand of \( S \) evaluates to \( False \), the interpretation of the semantics of \( S \) may be varied, depending on the requirement of applications. Formula \( \Psi_{neg}^{S} \) instantiates the template \( \Psi_{neg}^{CF} \) (see subsection 5.3.5) with \( S \), defining the traces of \( S \), which can be invalid or inconclusive. For example, three traces defined by the Negative (see figure 18), \([s1, s2, r1, r2],[s2, s1, r1, r2], \) and \([s1, r1, s2, r2] \), can be interpreted as invalid, or inconclusive traces if \( cond1 \) evaluates to \( False \).

In the case that, Negative \( S \) is enclosed in Sequence Diagram or non-Negative CF \( R \), the Messages which are not enclosed in \( S \) may interleave the sub-traces of \( S \). If the sub-traces of \( S \) are invalid, the traces of \( R \) can be interpreted as invalid or inconclusive traces. If the sub-traces of \( S \) are inconclusive traces (i.e., the Constraint of the Operand of \( S \) evaluates to \( False \)), the traces of \( R \) are also inconclusive traces. For Sequence Diagram \( R \), its traces are defined by formula \( \Pi_{R} \), which instantiates the template \( \Pi_{seq} \) (see figure 7). For non-Negative CF \( R \), its traces are defined by formula \( \Phi_{R} \), which instantiates the template \( \Phi_{CF} \) (see figure 8). For example, trace \([s1, s2, r2, r1, s3, r3] \) in figure 29 is interpreted as an invalid or an inconclusive trace.

Fig. 29. Example for variation of Negative Combined Fragment

For nested Negative CFs, i.e., Negative CF \( R \) encloses Negative CF \( S \), the traces of \( R \) are defined by \( \Phi_{R} \). These traces can be interpreted as valid, invalid, or inconclusive traces, depending on the Constraint of \( R \)'s Operand and the interpretation of the sub-traces of \( S \). The sub-traces of \( S \) are invalid or inconclusive depending on the value of its Constraint. Three different interpretations for the traces of \( R \) are provided: (1) If the sub-traces of \( S \) are invalid traces and the Constraint of \( R \)'s Operand evaluates to \( True \), the traces of \( R \) can be valid, invalid, or inconclusive traces. (2) If the sub-traces of \( S \) are invalid traces and the Constraint of \( R \)'s Operand evaluates to \( False \), the traces of \( R \) can be invalid or inconclusive traces. (3) If the sub-traces of \( S \) are inconclusive, the traces of \( R \) can be inconclusive traces in despite of the evaluation.
Consider $\Psi = \theta \land \gamma_i \land \neg \alpha \land \beta_k \land \gamma_{i+1}$

\[ \tilde{\alpha} = \bigwedge_{s \in S} (\neg OS \land \theta) (OS \land \bigcirc \neg OS) \]

\[ \Psi_{\text{ignore}} = \theta \land \gamma_i \land \neg \alpha \land \beta_k \land \gamma_{i+1} \]

\[ \Psi_{\text{consider}} = \theta \land \gamma_i \land \neg \alpha \land \beta_k \land \gamma_{i+1} \]

Figure 30 shows an example of nested Negative CFs. All the traces $[s_1, s_2, r_1, r_2]$, $[s_2, s_1, r_1, r_2]$, and $[s_1, r_1, s_2, r_2]$ of $R$ can be valid, invalid, or inconclusive traces depending on the value of cond1 and cond2.

5.5.2 Variations of Strict Sequencing

A Strict Sequencing CF represents an order among its Operands that any OS in an Operand can not execute until the previous Operand completes execution. However, the connection between the Strict Sequencing and its preceding/succeeding Interaction Fragments can be varied. According to the semantic rules general to all CFs, the Strict Sequencing is connected with its preceding/succeeding Interaction Fragments using Weak Sequencing. However, some applications may require that the Strict Sequencing are connected with its preceding/succeeding Interaction Fragments using Strict Sequencing. We modify the LTL formula of Strict Sequencing to formalize the variation (see figure 31). The only change we need to make is to replace $\gamma_i^{CF}$ that enforces Weak Sequencing between the Strict Sequencing and its preceding/succeeding Interaction Fragments with $\nu^{CF}$. Function $\nu^{CF}$ enforces the synchronization among multiple Lifelines when entering or leaving the Strict Sequencing, i.e., any covered Lifeline needs to wait others to enter or leave the Strict Sequencing together. The first conjunct enforces that the preceding set of OSs must happen before each OS within the Strict Sequencing, and the second conjunct enforces that the succeeding set of OSs must take place afterwards.

If an application requires Strict Sequencing to connect any CF with its preceding/succeeding Interaction Fragments, we can use function $\nu^{CF}$ to replace function $\gamma_i^{CF}$ in the LTL formula of the CF.

5.5.3 Variations of Interaction Constraint

There are two semantic interpretations of an Operand whose Interaction Constraint evaluates to False: (1) The Operand is excluded and its traces are inconclusive; (2) The traces expressed by the Operand are interpreted as invalid traces. Our LTL template chooses the first interpretation since that is the semantics provided by OMG (page 488 in [27]). For the second interpretation, the semantics of the Operand whose Constraint evaluates to False can be defined in the same way as the semantics of Negative CF (see subsection 5.5.1).

5.6 Other Control Constructs

5.6.1 General Ordering

General Ordering imposes order of two unorder OSs. We specify the two OSs of General Ordering as a pair of ordered OSs. In the LTL formula of General Ordering, $OS_p$ and $OS_q$ are two OSs connected by the General Ordering, which specifies that $OS_q$ can not execute until $OS_p$ completes execution.

\[ \gamma^{GO} = \neg OS_q \land OS_p \]

5.6.2 Interaction Use

Interaction Use embeds the content of the referred Interaction into the specified Interaction, thus composing a single, larger Interaction. We consider Interaction Use as a type of CF whose Interaction Operator is $\text{ref}$. Formula $\Psi_{\text{ref}}^{CF}$ represents the LTL representation of an Interaction Use. In $\Psi_{\text{ref}}^{CF}$, the first conjunct describes that the OSs directly enclosed in the referred Sequence Diagram obey their order. The second conjunct enforces that the referred Sequence Diagram and its adjacent OSs are ordered by Weak Sequencing, which is represented by $\gamma_i^{CF}$. 
Ψ_{strict}^{CF} = \theta^{CF} \land \bigwedge_{k \in NFTOP(CF)} \chi_{k} \land \nu^{CF}

\nu^{CF} = (\bigwedge_{i \in LN(CF)} \bigwedge_{OS \in \text{TOS}(CF_{i})} (\neg OS)) \cup (\bigwedge_{i \in LN(CF)} \bigwedge_{OS_{pre}\in\text{pre}(CF_{i})} (\neg OS_{pre}))

\bigwedge_{i \in LN(CF)} \bigwedge_{OS_{post}\in\text{post}(CF_{i})} (\neg OS_{post})) \cup (\bigwedge_{i \in LN(CF)} \bigwedge_{OS\in\text{TOS}(CF)} (\neg OS))

5.7 Discussion

![Diagram](image)

This paper does not address timed events, i.e., the events cannot represent the occurrence of an absolute time. The Messages are disallowed to cross the boundaries of CFs and their Operands [27]. Thereby, gates are not discussed in this paper. We only handle complete Messages, each of which has both sending and receiving OSs. The lost and found Messages are out of the scope of this paper.

For nested CFs, our syntactical constraints restrict that the borders of any two CFs cannot overlap each other, i.e., the inner CF cannot cover more Lifelines than the outer CF. The example in figure 32 is ill-formed. In this way, Coregion can only contain OSs and Coregions, no other CFs can be enclosed within a Coregion.

An Interaction Constraint of an Operand is located on a Lifeline where the first OS occurring within the Operand, i.e., the Interaction Constraint is positioned above the first OS occurring within the Operand. For example, figure 1b contains a Parallel covering three Lifelines. In the first Operand of the Parallel, os1, either OSs 2 or OS s3 may be the first OS to execute. As the Interaction Constraint of os1, cond1, located on Lifeline L2, OS s2 executes before OS s3.

However, if an Interaction Constraint of an Operand is located above a nested CF, it may not restrict an OS to be the first one to execute. In the example of figure 33, Interaction Constraint cond1 is located above a Parallel, which expresses that the first OS occurring within the Option’s Operand is contained by the Parallel on Lifeline L2. However, OS s1 and OS s2, which may be the first one to execute within the Parallel, are located on L1. To avoid the contradiction, we assume that an Interaction Constraint can restrict an OS to be the first one to execute only if it is located above an OS, not a nested CF.

For each Operand whose Constraint evaluates to True, the order between the first OS occurring within the Operand and any other OSs which are directly enclosed in the Operand is captured by an LTL formula (see figure 34). Function Init(m) returns the first OS occurring within Operand m, which may return an empty set if the Interaction Constraint is located above a nested CF.

![Diagram](image)

\[ \mu^{CF} = \bigwedge_{m \in \text{TOP}(CF)} \bigwedge_{OS_{p}\in\text{Init}(m) \land OS_{q}\in\text{TOS}(m)} (\neg OS_{q} \bigcup OS_{p}) \]

![Diagram](image)

Property patterns have been provided to assist software practitioners to map the system behaviors into formal specifications, such as LTL and CTL [15]. In their pattern system, some patterns can be adopted to codify the semantics of Sequence Diagram. For example, pattern precedence, which represents that S enables the occurrence of P, can be used to describe
the order between two OSs. The pattern shows the cases that the preceding OS may or may not happen, while the semantics shows the preceding OS must happen. In this way, our formula is succinct. Similarly, pattern bounded existence can be used to describe that an OS occurs at most once. The semantics defines that an OS executes once and in one state. Our formula represents the semantics more precisely than the pattern.

5.8 Proof for LTL Template of Sequence Diagram with Combined Fragments

We wish to prove that the NuSMV model for a Sequence Diagram with CFs capture the semantics of the Sequence Diagram. Recall the semantic rules general to all CFs have been presented in section 2.2, and the semantics of each CF Operator is shown in section 2.3. The LTL template for Sequence Diagram with CFs, $\Pi_{seq}$, is shown in figure 7.

We can write LTL template $\Pi_{seq}$ into $\Pi_{seq}$ (see figure 35) by replacing the sub-formula $\bigwedge_{i \in LN_{seq}(seq)} \alpha_{g}$ using sub-formulas $\bigwedge_{i \in LN_{seq}(seq)} \tilde{\alpha}_{g}$ and $\bigwedge_{j \in MSG(seq)} \rho_{j}$. The procedure (see figure 36) follows the one of rewriting LTL template $\Pi^{basic}_{seq}$. We can rewrite sub-formula $\theta_{CF}$ into $\theta_{CF}$ (see figure 37) to describe the semantics of CF’s Operands whose Constraints evaluate to True (see figure 38). In sub-formula $\theta_{CF}$, function $TBEU(CF_{\uparrow})$ returns the set of BEUs, whose Constraints evaluate to True, directly enclosed in the CE of CF on Lifeline $i$. It is equivalent to the set of BEUs directly enclosed in the EUs, which are obtained by projecting compacted CF’s Operands whose Constraints evaluate to True onto Lifeline $i$, i.e., $TBEU(CF_{\uparrow}) = \{beu | beu \in ABEU(op_{\uparrow}) \land op \in TOP(CF)\}$ (see line 1). Sub-formula $\bigwedge_{i \in LN_{seq}(CF)} \alpha_{g}$ is rewritten as the one of rewriting $\Pi^{basic}_{seq}$ (see line 4). We also rewrite sub-formula $\gamma_{i}^{CF}$ into $\tilde{\gamma}_{i}^{CF}$ (see figure 37) to enforce the sequential execution Lifeline $i$ (see figure 39). In sub-formula $\gamma_{i}^{CF}$, function $TOS(CF_{\uparrow})$ returns the set of OSs of the BEUs, whose Constraints evaluate to True, directly enclosed in the CEU of CF on Lifeline $i$. It is equivalent to the set of OSs directly enclosed in the Operands whose Constraints evaluate to True on Lifeline $i$, i.e., $TOS(CF_{\uparrow}) = \{os | os \in AOS(ABEU(op_{\uparrow})) \land op \in TOP(CF)\}$.

**Lemma 2**: A given Sequence Diagram with CFs, $seq$, directly contains $h$ Message. In the CFs, $p$ Messages are enclosed in Operands whose Interaction Constraints evaluate to True, i.e., if a Message is enclosed in multiple nested Operands, all the Interaction Constraints of the Operands evaluate to True. For other $q$ Messages within the CFs, each Message is enclosed in one Operand or multiple nested Operands, where at least one Operand’s Interaction Constraint evaluate to False. If $\sigma \in (\Sigma_{LTLL})_{\omega}$, then $\sigma$ must have the form, i.e., $\sigma = \sigma_{[1..2h+2p]} \cdot \tau_{\omega}$, where $\sigma_{[1..2h+2p]}$ contains no $\tau$.

**Proof**: If $\sigma \models \tilde{\Pi}_{seq}$, then $\sigma \models \bigwedge_{j \in MSG(seq)} \rho_{j}$ and $\sigma \models \bigwedge_{op \in TOP(CF)} (\bigwedge_{j \in MSG(seq)} \rho_{j})$. From sub-formula $\bigwedge_{j \in MSG(seq)} \rho_{j}$, we can infer that each OS of the Messages directly enclosed in seq can execute once and only once in $\sigma$. For each CF, we can infer from $\bigwedge (\bigwedge_{j \in MSG(seq)} \rho_{j})$ that each OS of the Messages directly enclosed in CF’s Operands whose Constraints evaluate to True can execute once and only once. Similarly, if $\sigma \models \Pi_{seq}$, we can deduce that $\sigma \models \varepsilon_{seq}$. It specifies that only one enabled OS (i.e., the OS is not enclosed in an Operand whose Constraint evaluates to False) can execute at a time, and $\sigma$ should execute uninterrupted until all the enabled OSs have taken place. seq directly contains $h$ Messages with $2h$ OSs. In the CFs within seq, the Operands whose Interaction Constraints evaluate to True contain $p$ Messages with $2p$ OSs. Therefore, $\sigma$ should have the form, $\sigma = \sigma_{[1..2h+2p]} \cdot \tau_{\omega}$, where $\sigma_{[1..2h+2p]}$ contains no $\tau$. $\square$

A given Sequence Diagram, $seq_{q}$, directly contains $k$ Lifelines, $h$ Messages and $r$ CFs, which contain $p + q$ Messages. Each CF does not contain other CFs. For the Messages within the CFs, $p$ Messages are enclosed in Operands whose Interaction Constraints evaluate to True, while $q$ Message are enclosed in Operands whose Interaction Constraints evaluate to False.

We wish to prove that $\forall v, u \in (\Sigma_{sem}^*)^*, u \cdot \tau_{\omega} \models \Pi_{seq}$, i.e., $v \cdot \tau_{\omega} \in (\Sigma_{LTLL})_{\omega}$. The semantic rules of seq define that each OS which is directly enclosed in seq or an Operand whose Constraint evaluates to True, occurs once and only once. Thus, $\forall v, u \in (\Sigma_{seq}^*)^*, |v| = 2h + 2p$. From lemma 2, we learn that $\forall \sigma. \sigma \in (\Sigma_{LTLL})_{\omega}, \sigma = \sigma_{[1..2h+2p]} \cdot \tau_{\omega}$, where $\sigma_{[1..2h+2p]}$ contains no $\tau$. $\sigma_{[1..2h+2p]} \in PRE_{2h+2p}((\Sigma_{seq}^*)^*)$. If $\forall v, u \in (\Sigma_{seq}^*)^*, u \cdot \tau_{\omega} \in (\Sigma_{LTLL})_{\omega}$, we can infer that, $u \in PRE_{2h+2p}((\Sigma_{seq}^*)^*)$, i.e., $(\Sigma_{seq}^*)^* \subseteq PRE_{2h+2p}((\Sigma_{LTLL})_{\omega})$.

We also wish to prove that $\forall \sigma. \sigma \in (\Sigma_{LTLL})_{\omega}, \sigma_{[1..2h+2p]} \in (\Sigma_{seq}^*)^*$, i.e., $PRE_{2h+2p}((\Sigma_{LTLL})_{\omega}) \subseteq (\Sigma_{seq}^*)^*$. **Theorem 2**: $(\Sigma_{seq}^*)^*$ and $PRE_{2h+2p}((\Sigma_{seq}^*)^*)$ are equal.

We provide the proof of theorem 2 in appendix B.

We consider the Sequence Diagram with nested CFs. A given Sequence Diagram, $seq_{q}$, directly contains $k$ Lifelines, $h$ Messages and $r$ CFs, which contain $p + q$ Messages. Each CF may contain other CFs. We use layer to define the location of the nested CFs. The Sequence Diagram’s layer is 0, while the layer of a CF directly enclosed in the Sequence Diagram is 1. If CF $cf_{m}$’s layer is $m$, then the layer of the CFs directly enclosed in $cf_{m}$ is $m + 1$. We assume the maximum layer of CF within $seq_{q}$ is $l$. For the Messages
\[ \Pi_{seq} = \bigwedge_{i \in LN(seq)} \bigwedge_{g \in TBEU(seq_{i_1})} \bar{\alpha}_g \wedge \bigwedge_{j \in MSG(seq)} \beta_j \wedge \bigwedge_{CF \in \text{nested}(seq)} \Phi_{CF} \wedge \varepsilon_{seq} \]

**Fig. 35.** Rewriting LTL templates for Sequence Diagram with Combined Fragments

\[ \Pi_{seq} = \bigwedge_{i \in LN(seq)} \bigwedge_{g \in TBEU(seq_{i_1})} \bar{\alpha}_g \wedge \bigwedge_{j \in MSG(seq)} \beta_j \wedge \bigwedge_{CF \in \text{nested}(seq)} \Phi_{CF} \wedge \varepsilon_{seq} \]

\[ = ( \bigwedge_{i \in LN(seq)} \bigwedge_{g \in TBEU(seq_{i_1})} \bar{\alpha}_g ) \wedge \bigwedge_{j \in MSG(seq)} \beta_j \wedge \bigwedge_{CF \in \text{nested}(seq)} \Phi_{CF} \wedge \varepsilon_{seq} \]

\[ = \Pi_{seq} \]

**Fig. 36.** Rewriting \( \Pi_{seq} \) into \( \tilde{\Pi}_{seq} \)

\[ \Psi_{CF} = \tilde{\theta}_{CF} \wedge \bigwedge_{i \in LN(CF)} \tilde{\gamma}_{i_{CF}} \wedge \tilde{\rho}_{CF} \]

\[ \tilde{\theta}_{CF} = \bigwedge_{op \in \text{TOP}(CF)} \left( ( \bigwedge_{i \in LN(CF)} \bigwedge_{g \in \text{ABEU}(op_{i_1})} \bar{\alpha}_g ) \wedge \bigwedge_{j \in \text{MSG}(op)} \beta_j \right) \]

\[ = \bigwedge_{op \in \text{TOP}(CF)} \left( ( \bigwedge_{i \in LN(CF)} \bigwedge_{g \in \text{ABEU}(op_{i_1})} \bar{\alpha}_g ) \wedge \bigwedge_{j \in \text{MSG}(op)} \beta_j \right) \]

\[ \tilde{\gamma}_{i_{CF}} = \bigwedge_{op \in \text{TOP}(CF)} \left( \bigwedge_{beu \in \text{ABEU}(op_{i_1})} (\neg \text{OS} \bar{\mu} (\bigwedge_{\text{OS} \text{pre} \in \text{pre}(CF_{i_1})} \bigwedge_{\text{OS} \text{post} \in \text{pre}(CF_{i_1})} \bar{\mu} (\diamond \text{OS}))) \right) \]

**Fig. 37.** Rewriting LTL template for OSs directly enclosed in Combined Fragment

\[ \tilde{\theta}_{CF} = \bigwedge_{op \in \text{TOP}(CF)} \left( \bigwedge_{i \in LN(CF)} \bigwedge_{g \in \text{ABEU}(op_{i_1})} \bar{\alpha}_g ) \wedge \bigwedge_{j \in \text{MSG}(op)} \beta_j \right) \]

\[ = \bigwedge_{op \in \text{TOP}(CF)} \left( \bigwedge_{i \in LN(CF)} \bigwedge_{g \in \text{ABEU}(op_{i_1})} \bar{\alpha}_g ) \wedge \bigwedge_{j \in \text{MSG}(op)} \beta_j \right) \]

\[ \tilde{\gamma}_{i_{CF}} = \bigwedge_{op \in \text{TOP}(CF)} \left( \bigwedge_{beu \in \text{ABEU}(op_{i_1})} (\neg \text{OS} \bar{\mu} (\bigwedge_{\text{OS} \text{pre} \in \text{pre}(CF_{i_1})} \bigwedge_{\text{OS} \text{post} \in \text{pre}(CF_{i_1})} \bar{\mu} (\diamond \text{OS}))) \right) \]

**Fig. 38.** Rewriting \( \tilde{\theta}_{CF} \) into \( \tilde{\rho}_{CF} \)

\[ \tilde{\gamma}_{i_{CF}} = \bigwedge_{op \in \text{TOP}(CF)} \left( \bigwedge_{beu \in \text{ABEU}(op_{i_1})} (\neg \text{OS} \bar{\mu} (\bigwedge_{\text{OS} \text{pre} \in \text{pre}(CF_{i_1})} \bigwedge_{\text{OS} \text{post} \in \text{pre}(CF_{i_1})} \bar{\mu} (\diamond \text{OS}))) \right) \]

**Fig. 39.** Rewriting \( \tilde{\gamma}_{i_{CF}} \) into \( \tilde{\gamma}_{i_{CF}} \)
within the CFs, $p$ Messages are enclosed in Operands whose Interaction Constraints evaluate to True, i.e., if a Message is enclosed in multiple nested Operands, all the Interaction Constraints of the Operands evaluate to True. For other $q$ Messages within the CFs, each Message is enclosed in one Operand or multiple nested Operands, where at least one Operand’s Interaction Constraint evaluate to False. We wish to prove that $\forall \upsilon, \nu \in (\Sigma_{sem})^*, \upsilon \cdot \rho^\omega = \Pi_{\text{nested}}$, i.e., $\upsilon \cdot \rho^\omega \in (\Sigma_{LTL})^\omega$. The semantic rules of seq\textsubscript{nested} define that each OS which is directly enclosed in seq\textsubscript{r}, or Operands whose Constraints evaluate to True, occurs once and only once. Thus, $\forall \upsilon, \nu \in (\Sigma_{sem})^*$, $|\upsilon| = 2h + 2p$. From lemma 2, we learn that $\forall \upsilon, \nu \in (\Sigma_{LTL})^\omega$, $\upsilon = \sigma_{[1,2h+2p]} \cdot \rho^\omega$, where $\sigma_{[1,2h+2p]}$ contains no $\tau$. $\sigma_{[1,2h+2p]} \in \text{PRE}_{2h+2p}(\sigma_{[1,2h+2p]} \cdot \rho^\omega)$. If $\upsilon, \nu \in (\Sigma_{sem})^*$, $\upsilon \cdot \rho^\omega \in (\Sigma_{LTL})^\omega$, we can infer that, $\upsilon \in \text{PRE}_{2h+2p}(\upsilon \cdot \rho^\omega)$, i.e., $(\Sigma_{sem})^* \subseteq \text{PRE}_{2h+2p}(\upsilon \cdot \rho^\omega)$.

We also wish to prove that $\forall \upsilon, \nu \in (\Sigma_{LTL})^\omega$, $\sigma_{[1,2h+2p]} \subseteq (\Sigma_{sem})^*$, i.e., $\text{PRE}_{2h+2p}(\Sigma_{LTL})^\omega \subseteq (\Sigma_{sem})^*$.

THEOREM 3: $(\Sigma_{sem})^*$ and $\text{PRE}_{2h+2p}(\Sigma_{LTL})^\omega$ are equal.

We provide the proof of theorem 3 in appendix B.

6 Case Study

In this section, we evaluate our formal framework and tool suite by modeling and analyzing a collection of the Health Insurance Portability and Accountability Act of 1996 (HIPAA) Privacy Rules [1] using Sequence Diagrams. We utilize the tool suite that takes Sequence Diagrams as input and generates LTL templates for formal analysis [32]. In particular, we show how to verify consistency and independence properties amongst a collection of HIPAA rules. We also show how to verify if a healthcare provider’s information sharing practices conforms with HIPAA rules. Since real data are difficult to obtain, we use a hospital’s publicly available information collection and patient authorization form (discussed later) as an example to demonstrate our approach.

HIPAA Overview. HIPAA provides national standards for insurance portability, fraud enforcement and administrative simplification of the healthcare industry [8]. It regulates the use and transmission of confidential health information, which is referred to as protected health information (PHI) among covered entities. Covered entities are the organizations required to comply with HIPAA, including hospitals, insurance companies, doctors and so on. The organizational policy rules of the covered entities should comply with HIPAA regulations, failure of which may result in severe penalties. For instance, Rite Aid Corporation paid $1 million for violations of the HIPAA Privacy Rule [28]. In another case, a former UCLA Health System employee was sentenced to prison and fined for unauthorized access to organizational electronic health record system [14].

Our work addresses the concerns discussed in section 1 in the following way. (1) Presenting the HIPAA rules using Sequence Diagrams is a user-friendly, yet precise, way to understand compliance requirements. In this section, we model a collection of HIPAA policy rules using Sequence Diagrams. (2) Our tool suite can help engineers to detect the HIPAA violations in the system automatically. For instance, Sequence Diagrams that reflect the system design can be generated based on system logs and administrators can check them against related HIPAA regulations automatically. They can also verify if their organizational policies conform to the HIPAA privacy rules and make amendments as necessary, which can prevent financial losses to the company.

6.1 Mapping Strategy

HIPAA Privacy Rule consists of 17 sections, each of which may contain standards and implementation specifications. A standard is a statement which reflects an organization’s intention. Implementation specifications define processes by which the intention is implemented. We model the portion of HIPAA which are related to information sharing. Specifically, we model all transmission-related requirements in 11 of these sections. The remaining rules do not concern about information transmission and are static in nature, such as definitions, conditions, and contents. The transmission-related rules can be separated into two groups. One group restricts the use or disclosure of PHI for covered entities, including sections §164.502, §164.506, §164.508, §164.510, §164.512, and §164.514. The other group restricts the request from the individual to the covered entities, including sections §164.520, §164.522, §164.524, §164.526, and §164.528.

HIPAA Rule Structure. We base the modeling of the HIPAA Privacy Rule using Sequence Diagrams on its structure. Each section contains multiple paragraphs, which are listed using lower case letters, numbers, roman letters, upper case letters, and italic numbers for different levels. For example, in section §164.512 (see [1] for a full description of this section), the standard of disclosures about victims of abuse, neglect or domestic violence is labeled using “(c)”. This standard consists of two paragraphs, which are labeled as “(1) Permitted disclosures” and “(2) Informing the individual”. The details of permitted disclosures are described in three paragraphs, which are labeled using roman letters “(i), (ii), (iii)”, and two paragraphs from lower level describe the details of paragraph “(iii)”, where they are labeled using upper case letters “(A), (B)”. To keep the diagrams clear and readable, we model each paragraph listed with a lower case letter, a number, or a roman letter using a Sequence Diagram. A paragraph labeled with upper case letters or italic number can be modeled
as a separate Sequence Diagram only if it is referred to by other paragraphs since it mainly represents the purposes or conditions of rules.

**Rule Types.** We categorize the paragraphs in a HIPAA rule into three groups. (1) “At least one” rules or Permission rules (all system traces each satisfies at least one of the rules): Each paragraph provides a number of possible means to regulate the behavior, e.g., “§164.512(a)(1) a covered entity may use or disclose protected health information to the extent that such use or disclosure is required by law...”. (2) “All” rules or Mandatory rules (all system traces each satisfies all such rules): Each paragraph provides mandatory means to regulate the behaviors, e.g., “§164.512(a)(2) A covered entity must meet the requirements described in paragraph (c), (e), or (f) of this section for uses or disclosures required by law.”. (3) Restrictive rules or Prohibition rules (all system traces each is restrictive from violating any of such rule): Each paragraph provides a forbidden means to regulate the behaviors, e.g., “§164.512(j)(2) A use or disclosure pursuant to paragraph (j)(1)(ii)(A) of this section may not be made if...”. Any of the three types of rules can be combined with exceptions, each of which enumerates the alternative cases of a rule. If system traces satisfy one alternative case, they do not need to satisfy the rule. For instance, “§164.524(b)(2)(ii) Except as provided in paragraph (b)(2)(ii) of this section...”, where §164.524(b)(2)(ii) is an exception of §164.524(b)(2)(i). In addition, a paragraph may refer to others to reuse that clause.

**Mapping.** We model each paragraph listed with roman letters using a Sequence Diagram. Particularly, a paragraph expressing a restrictive rule is modeled using a Sequence Diagram with a Negative CF. A paragraph with exceptions is modeled using a Sequence Diagram with an Alternatives CF, where each Operand refers to the Sequence Diagram of each exception rule. Each paragraph listed with numbers is also modeled using a Sequence Diagram, which expresses the composition of lower level paragraphs with CFs. Multiple “at least one” rules can be combined using an Alternatives CF. This procedure is recursively applied until all sections are mapped.

Each Sequence Diagram expressing a paragraph may have Constraints, which represent the purposes and the predicates. A predicate represents a condition which needs to be evaluated by external actors. Each actor is modeled using a Lifeline, where the actor’s role is modeled using the instance’s class. For instance, a Lifeline’s head can be p1: coveredEntity, which represents that the role of actor p1 is a covered entity. The roles of actors are hierarchical, i.e., general roles can be subdivided into specific roles. For example, a covered entity can be replaced as a health plan or a healthcare provider. The most general roles in HIPAA are defined as “HIPAA-role”, which can be replaced with a specific role in a rule. Messages transmitted among actors is modeled using asynchronous Message transmission, where the Message name indicates the information it carries, i.e., the attributes of an actor. For policy rules with time constraints, we add a “timer” Lifeline.

### 6.2 Sequence Diagrams for HIPAA

We illustrate our mapping strategies using 2 examples for safety and liveness properties separately. A third safety property example is demonstrated in our technical report [31]. §164.512(j)(2)(i) defines a safety property by expressing forbidden behaviors. It regulates that “A use or disclosure pursuant to paragraph (j)(1)(ii)(A) of this section may not be made if the information described in paragraph (j)(1)(ii)(A) of this section is learned by the covered entity in the course of treatment to affect the propensity to commit the criminal conduct that is the basis for the disclosure under paragraph (j)(1)(ii)(A) of this section.” We model this paragraph using a Sequence Diagram with a Negative CF (see figure 40). The Negative CF describes that if the condition is satisfied, the scenarios expressed by the Interaction Use are invalid. Para §164.512(j)(2)(i)(A) is represented as nested combined fragment in fig 40.

§164.524(b)(2)(i) expresses a liveness property, which regulates that “(i) Except as provided in paragraph (b)(2)(ii) of this section, the covered entity must act on a request for access no later than 30 days after receipt of the request as follows. (A) If the covered entity grants the request, in whole or in part, it must inform the individual of the acceptance of the request and provide the access requested, in accordance with paragraph (c) of this section. (B) If the covered entity denies the request, in whole or in part, it must provide the individual with a written denial, in accordance with paragraph (d) of this section.” This paragraph expresses a rule with exception. We model the policy rule and its exception using the Operands of an Alternatives CF (see second Alternatives CF in figure 41). The first Operand describes the exception case described paragraph §164.524(b)(2)(ii) regarding PHI not being available on-site. The second Operand describes the response for an access request in any other case. Two possible responses are combined using an Alternatives CF. A “timer” Lifeline is introduced to track the time interval. The timer is set when the request is sent, and it will notify the individual when
Fig. 41. Paragraph 164.524(b)(2)(i) – (ii)

the specified amount of time has elapsed.

6.3 HIPAA Verification

We have modeled more than one hundred HIPAA policy rules using Sequence Diagrams. Yet, it is hard to check manually that if the policy rules contradict each other (consistency), or if some policy rules are redundant (independence). With the help of our tool suite, we can analyze Sequence Diagram based policy rules automatically. However, due to the state explosion problem, it is difficult to check all the policy rules at the same time. To simplify the verification process, we group the policy rules into sets of related rules and verify each set independently. We define a set of related policy rules to (1) share the same purpose; and (2) at least two actors with the same or hierarchically related roles, e.g., “HIPAA-role” and “law-enforcement-official” are hierarchically related because “HIPAA-role” can be replaced using “law-enforcement-official” in specific rules.

Consistency Checking: For each set of rules, we build a free model to define all the variables in the LTL formulas of the rules. Then we check the free model against the negation of conjunction of the LTL formulas. If no counter-examples are generated (i.e., there are no property violations), the policy rules are inconsistent. Otherwise, the policy rules are consistent and a counterexample which satisfies all the LTL formulas (i.e., all the HIPAA privacy rules belonging to the same purpose) is provided by the model checker. As a proof-of-concept, we tested the consistency of rules for 3 purposes with 6, 5 and 8 rules respectively and they all held true. These rules are large with multiple nested paragraphs. A major verification across entire HIPAA is research endeavor in its own right.

Independence Checking: We build the free model for each set of rules and check it against the independence properties. We tested independence for 3 purposes with 6, 5 and 8 rules respectively and they held true.

Conformance Checking: We can use LTL models generated with LTL templates to verify that a system conform to, or satisfies, an existing policy or regulation. To exemplify this, we have adapted a model from forms that is common in many hospitals regarding authorization for release of PHI. Figure 42 models the possible traces regarding authorization of PHI disclosure present in our modified version of such forms. The parallel combined fragment allows for Message 3, the notification to the individual by the covered entity of their right to revoke authorization, to occur at any time between Message 4 and 5. This means Message 3 can occur before Message 4, between Messages 4 and 5, or after Message 5. Message 8, the disclosure of PHI to the HIPAA role can only occur if authorization has not been revoked by the individual.

§164.508(c)(2)(i) of HIPAA describes the requirement for authorization forms to provide notice to the individual of their right to revoke authorization for disclosure of PHI. Furthermore, §164.508(b)(2)(i) describes the invalidity of authorizations after the expiration date as required by §164.508(c)(1)(v). Figure 43 models the acceptable traces for disclosure of PHI with regard to both of these HIPAA regulations. The alternative combined fragment allows for two pos-
remuneration is involved.”

A third party, the authorization must state that such direct or indirect remuneration to the covered entity from any relevant traces. The regulation states that “...a covered entity must obtain an authorization for any order within the second operand of the combined fragment which allows Message 3 to be interleaved in any order within the second operand of the combined fragment. This represents a single possible trace which can violate the HIPAA rule which outlines a specific ordering of the Messages in the second operand of the alternative combined fragment in figure 43, which the hospital policy fulfills due to its constraints.

This example shows how a PHI authorization form can be checked for conformance with two HIPAA regulations. Because the form also regards the possibility of disclosure of PHI for marketing purposes, it could also be checked against HIPAA §164.508(a)(3) as seen in figure 44, after being modified to include any relevant traces. The regulation states that “...a covered entity must obtain an authorization for any use or disclosure of protected health information for marketing, except if the communication is in the form of (A) a face-to-face communication made by a covered entity to an individual; or (B) a promotional gift of nominal value provided by a covered entity.” Furthermore, “if the marketing involves direct or indirect remuneration to the covered entity from a third party, the authorization must state that such remuneration is involved.”

Similar instances can be checked to validate conformance to other privacy policy regulations such as the Gramm-Leach-Bliley Act (GLBA) regulating financial institutions. For example, §502 of GLBA describes regulations for disclosure of personal information. Privacy policies for financial institutions can be modelled and checked for conformance to GLBA in same way as we have modelled hospital policies and checked for conformance to HIPAA. This method provides a means to validate whether a system’s design satisfies existing security policies for any relevant set of traces that can be modelled as a Sequence Diagram.

7 Related Work

Several frameworks have been proposed to specifying and analyzing privacy rules. Barth et al. [7] propose Contextual Integrity, which is a framework for specifying privacy rules, such as HIPAA and Gramm-Leach-Bliley Act (GLBA). They introduce two norms of transmission, positive norm and negative norm, corresponding to “all” and “at least one” rules. Similarly, May et al. [25] present Privacy API, which is an extension of access control matrix model. They translate section §164.506 of the 2000 and 2003 HIPPA rules and use SPIN model checker for analysis. DeYoung et al. [12], [13] present a logic framework, PrivacyLFP, to formalize HIPAA privacy rule. Their work based on first order logic, but do not consider the “all” rules with “may not”. They also provide an auditing algorithm for detecting violations of policy rules [17]. Lam et al. [24] present pLogic based on a fragment of stratified Datalog. They formalize section §164.502, §164.506, and §164.508 of HIPAA privacy rules. Breaux et al. [9], [10] present a methodology for extracting formal description of rules which regulate actions from policies. They apply the methodology to the HIPAA Privacy Rule to help engineers identify the essential elements. Similarly, our work
helps engineers to understand the HIPAA Privacy Rule, and also check that if the organizational rules made by the engineers comply with HIPAA rules. To specify security policies for distributed object system, Damianou et al. [11] introduced a declarative, object-oriented language, Ponder. The language supports access control policies and obligation policies, which can also be managed to reflect the organizational structure of large enterprises. The Enterprise Privacy Authorization Language (EPAL) [29] is the first language that allows organizations to express privacy rules directly in a XML-based markup language.

To the best of our knowledge, our technique is the first to support all CFs and the nested CFs of Sequence Diagrams. Working towards similar goals, Kugler et al. [22] and Kumar et al. [23] have presented translations from a LSC to temporal logic formulas, which does not support all CFs. LSC extend MSC using universal charts and existential charts. Universal charts specify behavior over all possible system runs, where existential charts specify behavior which must be satisfied by at least one system run. They expressed universal charts using both LTL and Computation tree logic (CTL), and expressed existential charts using CTL since existential charts only consider possible behavior. Their approach formalizes global behaviors using synchronous Messages, while our work focuses on the execution of events on each Lifeline and supports both synchronous and synchronous Messages, which is more specific for concurrent systems. Harel and Maoz [19] propose a Modal Sequence Diagram (MSD) to specify semantics of Negation and Assertion Operators, providing an avenue for us to define liveness and safety properties. To specify and formalize temporal logic properties, Autili et al. [5], [6] propose the Property Sequence Chart (PSC), which is an extension of UML 2 Sequence Diagrams. Their approach eases software engineers' efforts for defining properties. Our method can be adapted for PSC to support a larger set of properties.

Micskei and Waeselynck survey comprehensively formal semantics proposed for Sequence Diagrams by 13 groups and present the different options taken in [26]. In these groups, [21] presents an operational semantics for a translation of an Interaction into automata, which is used to model check the communication produced by UML state machines with SPIN or UPPAAL. Grosu and Smolka [18] propose a formal semantics using Büchi automata and represent positive and negative automata as liveness and safety properties respectively. The approaches of both groups do not support all CFs and the interpretation of automata restricts the specification of CF constraints. Eichner et al. introduce a compositional formal semantics of UML 2 Sequence Diagrams using colored high-level Petri Nets [16]. The semantics represents a subset of the CFs of Sequence Diagrams. Haugen et al. present the formal semantics of the UML 2 Sequence Diagram through an approach named STAIRS [20]. STAIRS provides a trace-based representation for a subset of Sequence Diagram CFs, which focuses on the specific definition of refinement for Interactions. Whittle presents a three-level notation with formal syntax and semantics for specifying use cases in [33]. Each use case is defined by a set of UML Interactions in level-2 and the details of each Interaction are defined in level-3. With this three-level notation, Whittle and Jayaraman present an algorithm for synthesizing well-structured hierarchical state machines from scenarios [34]. The generated hierarchical state machines are used to simulate scenarios and improve readability. Our work focuses on Sequence Diagrams in level-3.

8 Conclusion

In this paper, we demonstrate that LTL templates can be used to specify the semantics of Sequence Diagrams. This formalization will enable software practitioners to verify if a collection of security policies satisfies specified properties and if they are consistent and independent. It also allows practitioners to test organizational policies and operations against required policies specified using Sequence Diagrams and check for inconsistencies. To evaluate the framework, we performed a major case study of modeling and analyzing 100s of HIPAA privacy rules using Sequence Diagrams. Developing Sequence Diagrams allows a domain-expert without much technical background to validate the policies, yet, our work enables generation of precise and automated verifiable specifications that can be utilized by system administrators with a guarantee on conformance of interpretation of high-level policies. We strongly believe that our approach can be utilized in similar fashion to analyze other policies such as GLBA [3], FERPA [4] and COPPA [2] which is part of future work.

References

of the 7th International Conference on Software Paradigm Trends (ICSOFT), pages 44–54, July 2012.


APPENDIX A

PROOF OF THEOREM 1

Theorem 1. For a given Sequence Diagram, seq, with j Messages, \((\Sigma_{seq}^*)^*\) and \(PREF_2(\Sigma_{seq}^*)^*\) are equal.

Proof: We use mathematical induction, which is based on the number of Messages, \(j\), within \(seq\).

Base step. Basic Sequence Diagram \(seq_1\) contains only one Message, \(m_1\), \((j = 1)\).

- Case 1. Sending OS \(s_1\), and receiving OS \(r_1\) of Message \(m_1\) locate on two Lifelines \(L_1, L_2\) respectively (see figure 45).

\[
\Sigma_{seq_1}^* = \{s_1, r_1\}, \text{ where } \Sigma_{seq_1}^* \subseteq \Sigma. \text{ The semantic aspects of } seq_1 \text{ define that, for } m_1, r_1 \text{ can only happen after } s_1. \text{ Only one trace, } v = < s_1, r_1 > \text{ of size 2, can be derived from seq}_1, \text{ i.e., } (\Sigma_{seq_1}^*)^* = \{< s_1, r_1 >\}.
\]

We wish to prove that \(< s_1, r_1 > \rightarrow \tau^\omega \models \widehat{\Pi}_{seq_1}^\text{Basic}, \text{ in which } \widehat{\Pi}_{seq_1}^\text{Basic} \text{ for } seq_1 \text{ is shown as below.}

\[
\widehat{\Pi}_{seq_1}^\text{Basic} = \alpha_{seq_1} L_1 \land \rho_m \land \beta_{m_1} \land \varepsilon_{seq_1}
\]

\[
\rho_m = (\neg s_1 \tilde{U}(s_1 \land \square \neg s_1))
\]

\[
\land (\neg r_1 \tilde{U}(r_1 \land \square \neg r_1))
\]

\[
\beta_{m_1} = (\neg r_1 s_1 \tilde{U})
\]

\[
\varepsilon_{seq_1} = \Box((s_1 \land r_1) \lor (s_1 \land \neg r_1) \lor ((\tilde{S}s_1) \land (\tilde{S}r_1)))
\]

Sub-formula \(\alpha_{seq_1} L_1\) returns true because Lifeline \(L_1\) contains only one OS, \(s_1, < s_1, r_1 > \rightarrow \tau^\omega\) satisfies sub-formula \(\rho_m\) because \(s_1\) and \(r_1\) only occur once. It satisfies sub-formula \(\beta_{m_1}\), because \(s_1\) happens before \(r_1\) does. It also satisfies sub-formula \(\varepsilon_{seq_1}\) because only one OS happens at a time and \(< s_1, r_1 > \) executes uninterrupted. Thus, \(< s_1, r_1 > \rightarrow \tau^\omega \models \widehat{\Pi}_{seq_1}^\text{Basic}.

We wish to prove that \(\sigma \land \sigma \in \Sigma^\omega, \text{ if } \sigma \in (\Sigma_{seq_1}^*)^*\), then \(\sigma_{[1,2]} \in (\Sigma_{seq_1}^*)^*\).

\(\sigma\) satisfies sub-formula \(\rho\), which constrains that \(s_1\) and \(r_1\) can occur once and only once respectively. Therefore, \(\sigma_{[1,2]}\) can be \(< s_1, r_1 > \) or \(< r_1, s_1 > \).

![Fig. 45. Case 1 for basic Sequence Diagram with single Message](image-url)
Sub-formula $\beta_{m_1}$ represents that $r_1$ cannot occur until $s_1$ does. Therefore, $\sigma[1..2]$ can only be $< s_1, r_1 >$, which is an element of $(\Sigma_{sem})^*$. In this way, we can prove $\sigma[1..2] \in (\Sigma_{sem})^*$.

- Case 2. Sending OS $s_1$ and receiving OS $r_1$ of Message $m_1$ locate on a single Lifeline $L_1$ (see figure 46).

![Fig. 46. Case 2 for basic Sequence Diagram with single Message](image)

Besides the semantic aspects discussed in case 1, the OSs on $L_1$ respect their graphical order, i.e., $s_1$ occurs before $r_1$. Trace $v = (s_1, r_1)$ of size 2 can be derived from $seq_1$, i.e., $(\Sigma_{seq})^* = \{< s_1, r_1 >\}$.

$\Pi_{Basic}$ is reduced to $\Pi_{Basic}$ for $seq_1$ as below.

\[
\Pi_{Basic}^{seq_1} = \alpha_{seq_1} \uparrow_{L_1} \land \beta_{m_1} \land \rho_{m_1} \land \varepsilon_{seq_1}
\]

\[
\alpha_{seq_1} \uparrow_{L_1} = \neg r_1 \uparrow s_1
\]

\[
\beta_{m_1} = \neg s_1 \wedge (s_1 \land \emptyset \neg r_1)
\]

\[
\rho_{m_1} = \neg r_1 \wedge (r_1 \land \emptyset \neg r_1)
\]

\[
\varepsilon_{seq_1} = \emptyset \neg (s_1 \wedge r_1) \lor (s_1 \land \neg r_1) \lor (\emptyset (s_1 \wedge r_1))
\]

Comparing to $\Pi_{Basic}$ in case 1, only sub-formula $\alpha_{seq_1} \uparrow_{L_1}$ is changed. $\alpha_{seq_1} \uparrow_{L_1}$ represents that $s_1$ happen before $r_1$, which enforces the same order as sub-formula $\beta_{m_1}$. Trace $< s_1, r_1 > \gamma$ can be generated from $\Pi_{seq}$, i.e., $(\Sigma_{seq})^{\omega} = \{< s_1, r_1 > \gamma\}$.

Similarly, we wish to prove that $\forall v, v \in \Sigma^n$, if $v \in (\Sigma_{seq})^*$, then $v \cdot \gamma \Pi_{Basic}$, and $\forall \sigma, \sigma \in \Sigma^n$, if $\sigma \in (\Sigma_{seq})^{\omega}$, then $\sigma[1..2n] \in (\Sigma_{seq})^*$.

The proof follows the case of one 1.

To sum up, for a basic Sequence Diagram with one Message, $(\Sigma_{seq})^*$ and $pre((\Sigma_{seq})^{\omega})$ are equal.

Inductive step. Basic Sequence Diagram $seq_n$ contains $n$ Messages, which are graphically-ordered, i.e., $(m_{i-1}$ locates above $m_i$ $(2 \leq i \leq k))$. The Messages have $2n$ OSs, which locate on $k$ Lifelines. We assumes  $\forall v, v \in \Sigma^n$, if $v \in (\Sigma_{seq})^*$, then $v \cdot \omega \Pi_{Basic}$; and $\forall \sigma, \sigma \in \Sigma^n$, if $\sigma \in (\Sigma_{seq})^{\omega}$, then $\sigma[1..2n] \in (\Sigma_{seq})^*$ $(j = n)$.

We add a Message, $m_{n+1}$, at the bottom of $seq_n$ graphically to form a new Sequence Diagram, $seq_{n+1}$, with $n + 1$ Messages. We wish to prove $\forall v, v' \in \Sigma^n$, if $v \in (\Sigma_{seq_{n+1}})^*$, then $v' \cdot \omega.v \Pi_{seq_{n+1}}$ and $\forall \sigma, \sigma \in \Sigma^n$, if $\sigma \in (\Sigma_{seq_{n+1}})^*$, then $\sigma[1..2n+2] \in (\Sigma_{seq_{n+1}})^*$ $(j = n+1)$.

(a) We wish to prove $\forall v, v' \in \Sigma^n$, if $v' \in (\Sigma_{seq_{n+1}})^*$, then $v' \cdot \omega \Pi_{seq_{n+1}}$.

The semantic aspects of $seq_{n+1}$ enforce that only one OS occurs at a time, and each OS happens once and only once. $\Sigma_{seq_{n+1}}^{\omega}$, $\Sigma_{seq_{n+1}}^*$ is $\Sigma_{seq_n}$ $\cup \{s_{n+1}, r_{n+1}\}$, where $|\Sigma_{seq_n}| = 2n$ and $|\Sigma_{seq_{n+1}}| = 2n + 2$. If $v' \in (\Sigma_{seq_{n+1}})^*$, then $v'$ is a finite trace of size $2n + 2$, which contains OSs in $\Sigma_{seq_{n+1}}$. Adding $m_{n+1}$ at the bottom of $seq_n$ does not change the structure of $seq_n$. Thus, for trace $v'$, the order of OSs in $\Sigma_{seq_{n+1}}$ is still preserved. Message $m_{n+1}$ restricts that $s_{n+1}$ must happen before $r_{n+1}$, i.e., $s_{n+1}$ locates before $r_{n+1}$ in $v'$.

$\Pi_{seq}$ is reduced to $\Pi_{Basic}$ and $\Pi_{seq_{n+1}}$ for $seq_{n}$ and $seq_{n+1}$ respectively (see figure 47). We group the sub-formulas of $\Pi_{seq_{n+1}}$ using $\Pi_{seq}$, $\Pi_{seq_{n+1}}$, and $\varepsilon_{seq_{n+1}}$. In order to prove $v' \cdot \omega \Pi_{seq_{n+1}}$, we wish to prove that $v' \cdot \omega$ satisfies all sub-formulas of $\Pi_{seq_{n+1}}$. Sub-formula $\Pi_{seq}$ enforces the order of OSs within $seq_n$, which includes the order of OSs along each Lifeline, and the order between OSs of each Message. We assume that if $v \in (\Sigma_{seq_n})^*$, then $v \cdot \omega \Pi_{seq_n}$. It is easy to observe that $v \cdot \omega$ also satisfies $\varepsilon_{seq_n}$. As we discussed, the order of OSs within $seq_n$ is still preserved in $v'$. Thus, $v' \cdot \omega$ satisfies $\varepsilon_{seq_n}$. Sub-formula $\varphi_{m_{n+1}}$ enforces the order between OSs of $m_{n+1}$, i.e., $s_{n+1}$ and $r_{n+1}$ happen only once respectively, and $s_{n+1}$ must occur before $r_{n+1}$. $v' \cdot \omega$ satisfies $\varphi_{m_{n+1}}$ because (1) only one $s_{n+1}$ and one $r_{n+1}$ are in $v'$, and (2) $s_{n+1}$ locates before $r_{n+1}$ in $v'$. Sub-formula $\varepsilon_{seq_{n+1}}$ enforces that only one OS of $seq_{n+1}$ can execute at once, and the trace should execute uninterrupted. As we discussed, in $v' \cdot \omega$, each OS of $seq_{n+1}$ only executes once, and the execution of $v'$ does not interleaved by $\tau$. Therefore, $v' \cdot \omega$ satisfies $\varepsilon_{seq_{n+1}}$.

(b) Case 1: Two OSs of $m_{n+1}$ locate on two new Lifelines, $L_{k+1}$ and $L_{k+2}$ (see figure 48a); or two OSs of $m_{n+1}$ locate on one new Lifeline, $L_{k+1}$ (see figure 48b).

The OSs of $m_{n+1}$ locate on one or two new Lifelines, so $m_{n+1}$ and the existing Messages, $m_1, m_2 \ldots m_n$, are interleaved. Therefore, in trace $v' \in (\Sigma_{seq_{n+1}})^*$, $s_{n+1}$ or $r_{n+1}$ can locate (1) between any two OSs of $seq_n$, or (2) before all OSs of $seq_n$, or (3) after all OSs of $seq_n$. Thus, $s_{n+1}$ can be the $2n$ OS of $v'$, where $1 \leq s \leq 2n + 1$; and $r_{n+1}$ can be the $2n$ OS of $v'$, where $s < r \leq 2n + 2$.

$\zeta_{seq, m_{n+1}} = \alpha_{seq} \uparrow_{L_{k+1}} \land \alpha_{seq} \uparrow_{L_{k+2}}$

Sub-formula $\zeta_{seq, m_{n+1}}$ is a conjunction of $\alpha_{seq} \uparrow_{L_{k+1}}$ and $\alpha_{seq} \uparrow_{L_{k+2}}$. Only one OS locates on
Fig. 47. LTL formulas for seqn and seqn+1

Lifeline $L_{k+1}$. Therefore, $\bar{\alpha}_{seqn_{k+1}}$ returns true as defined by sub-formula $\bar{\alpha}_g$. Similarly, $\bar{\alpha}_{seqn_{k+2}}$ returns true. Thus, $\bar{\alpha}_{seqn_{m+1}+1}$ returns true.

Case 2: Sending OS $s_{n+1}$ locates on a new Lifeline, $L_{k+1}$, and receiving OS $r_{n+1}$ locates on an existing Lifeline, $L_i$ ($1 \leq i \leq k$) (see figure 48c).

In seqn, we assume the last OS on $L_i$ is OSpre. After adding $m_{n+1}$ at the bottom of seqn, $r_{n+1}$ becomes the last OS on $L_i$. Therefore, OSpre should happen before $r_{n+1}$. seqn+1 locates on a new Lifeline, so it is interleaved with the OSs of seqn. However, $s_{n+1}$ must happen before $r_{n+1}$. In trace $v' \cdot \tau^\omega$, if OSpre is the $p$th OS, where $1 \leq p \leq 2n + 1$. Then $s_{n+1}$ is the $s$th OS of $v'$, where $1 \leq s \leq 2n + 1$ and $s \neq p$; $r_{n+1}$ is the rth OS of $v'$, where $r < r_{n+1} = 2n + 2$ and $p < r_{n+1}$. Sub-formula $\zeta_{seqn_{m+1}+1}$ defines that $s_{n+1}$ cannot happen before OSpre. Only one or none OSs of $s_{n+1}$ locate on an existing Lifeline $L_i$ ($1 \leq i \leq k$) (see figure 48e).

Similarly, we assume the last OS on $L_i$ in seqn is OSpre. In seqn+1, if $s_{n+1}$ locates on $L_i$, OSpre should happen before $s_{n+1}$ because OSpre locates above $s_{n+1}$ graphically. For $m_{n+1}, r_{n+1}$ must happen after $s_{n+1}$. In trace $v' \in (\Sigma_{seqn_{m+1}})\ast$, if OSpre is the $p$th OS, where $1 \leq p \leq 2n$. Then $s_{n+1}$ is the $s$th OS of $v'$, where $p < s \leq 2n + 1$; $r_{n+1}$ is the rth OS of $v'$, where $r < r_{n+1} = 2n + 2$. Sub-formula $\zeta_{seqn_{m+1}+1}$ defines that $s_{n+1}$ cannot happen before OSpre. Only one or none OSs of $s_{n+1}$ locate on an existing Lifeline $L_i$ ($1 \leq i \leq k$) (see figure 48f).

In seqn, we assume the last OS on $L_i$ is OSpre, and the last OS on $L_{i+1}$ is OSpre+. After adding $m_{n+1}$ at the bottom of seqn, $s_{n+1}$ locates on an existing Lifeline $L_i$ ($1 \leq i \leq k$), receiving OS $r_{n+1}$ locates on Lifeline $L_j$ ($1 \leq j \leq k$) (see figure 48g). In seqn, we assume the last OS on $L_i$ is OSpre, and the last OS on $L_{i+1}$ is OSpre+. After adding $m_{n+1}$ at the bottom of seqn, the last OS of $L_i$, and $r_{n+1}$ becomes the last OS of $L_j$. In trace $v' \in (\Sigma_{seqn_{m+1}})\ast$, if OSpre is the $p$th OS, where
Fig. 48. Examples for basic Sequence Diagram with $n + 1$ Messages
1 \leq p_s \leq 2n$, and OS
pre is the $p_s$-th OS, where
1 \leq p_s \leq 2n + 1. Then $s_{n+1}$ is the $s$th OS of $v'$, where
$p_s < s \leq 2n + 1$; $r_{n+1}$ is the $r$th OS of $v'$, where
$p_r < r \leq 2n + 2$.

$$s_{seq, m_{n+1}} = (-s_{n+1} \tilde{U} OS_{pre}) \land (-r_{n+1} \tilde{U} OS_{pre})$$

The first conjunct of sub-formula $s_{seq, m_{n+1}}$ defines that $s_{n+1}$ cannot happen until $OS_{pre}$ executes. In $v'$, $\tau^m$, $OS_{pre}$ locates before $s_{n+1}$, i.e., $p_s < s$. Therefore, $v' \land \tau^m$ satisfies $s_{seq, m_{n+1}}$. Similarly, we can prove that $v' \land \tau^m$ satisfies $s_{seq, m_{n+1}}$, i.e., $v' \land \tau^m \models s_{seq, m_{n+1}}$.

Now we have proven that for all cases, $v' \land \tau^m \models s_{seq, m_{n+1}}$.

To conclude, $v' \land \tau^m \models s_{seq, m_{n+1}}$. It is easy to infer that $s_{seq}$ satisfies $\ast$. Sub-formula $t_{seq}$ enforces the order of OSs in $\Sigma_{seq}$ and each OS should execute once and only once. We can also infer that $v' \land \tau^m \models s_{seq}$ from $s_{seq}$. If $\sigma$ does not contain an OS in $\Sigma_{seq}$, then $\sigma$ does not satisfy $t_{seq}$, which defines that each OS in $\Sigma_{seq}$ should happen once. Therefore, all OSs in $\Sigma_{seq}$ execute once and only once in $v'$. We wish to prove that in $s_{seq}$, all OSs in $\Sigma_{seq}$ respect their order defined by semantic aspects of $\Sigma_{seq}$. In $s_{seq}$, the semantic aspects of $\Sigma_{seq}$ define that $OS_p$ must happen before $OS_q$. In $s_{seq}$, we assume that the OSs do not respect the same order, i.e., $OS_p$ occurs after $OS_q$. $t_{seq}$ codifies the semantic aspects of $\Sigma_{seq}$, so it contradicts that $OS_p$ should take place after $OS_q$. To satisfy $t_{seq}$, $OS_p$ must occur before $OS_q$ in $\sigma'$, which contradicts our assumption. Therefore, in $s_{seq}$, the OSs in $\Sigma_{seq}$ respect the order defined by semantic aspects of $\Sigma_{seq}$, i.e., if we remove the OSs not in $\Sigma_{seq}$ from $s_{seq}$, we obtain a new trace $\sigma''$ then $\sigma'' \in (\Sigma_{seq} \ast)$.

Sub-formula $\vartheta_{m_{n+1}}$ specifies that $s_{n+1}$ must occur before $r_{n+1}$, and both OSs can occur only once. $s_{n+1}$ and $r_{n+1}$ may not locate on the same Lifeline. Thus, $\vartheta_{m_{n+1}}$ codifies the semantic of Message $m_{n+1}$ in $s_{seq}$. In $s_{seq}$, $s_{n+1}$ and $r_{n+1}$ represent the semantics of $m_{n+1}$. We have proven each OSs in $\Sigma_{seq}$ should happen once and only once in $s_{seq}$, where $|\Sigma_{seq}| = 2n$, and both of $s_{n+1}$ and $r_{n+1}$ occur only once. Thus, we can deduce that $\vartheta_{m_{n+1}}$ captures the semantics, which defines only one OS executing at a time and the $\sigma'_{[1, 2n+2]}$ should execute uninterrupted.

Now we wish to prove that sub-formula $s_{seq, m_{n+1}}$ codifies the order between the OSs within $\Sigma_{seq}$ and the OSs of $m_{n+1}$, which is discussed using four cases as below.

- **Case 1**: Two OSs of $m_{n+1}$ locate on two new Lifelines, $L_{k+1}$ and $L_{k+2}$ (see figure 48a); or two OSs of $m_{n+1}$ locate on one new Lifeline, $L_{k+1}$ (see figure 48b).

$$s_{seq, m_{n+1}} = (-s_{n+1} \tilde{U} OS_{pre}) \land (-r_{n+1} \tilde{U} OS_{pre})$$

Sub-formula $s_{seq, m_{n+1}}$ is a conjunction of $\ast_{L_{k+1}}$ and $\ast_{L_{k+2}}$. It returns true only if none or at most one OS locates on each Lifeline. Therefore only one OS locates on $L_{k+1}$ and $L_{k+2}$ respectively. $s_{seq, m_{n+1}}$ represents that $m_{n+1}$ and the Messages of $seq_n$ are interleaved. No specific order is defined between the OSs of $seq_n$ and the OSs of $m_{n+1}$. Thus, $s_{seq, m_{n+1}}$ codifies the order between the OSs of $seq_n$ and the OSs of $m_{n+1}$ in $s_{seq}$. In $s_{seq}$, the OSs of $seq_n$ and the OSs of $m_{n+1}$ respect the order defined by the semantic aspects of $seq_n$.

- **Case 2**: Sending OS $s_{n+1}$ locates on a new Lifeline, $L_{k+1}$, receiving OS $r_{n+1}$ locates on an existing Lifeline, $L_i$ ($i \leq k$) (see figure 48c).

$$s_{seq, m_{n+1}} = (-s_{n+1} \tilde{U} OS_{pre}) \land \ast_{L_{k+1}}$$

Sub-formula $s_{seq, m_{n+1}}$ defines that $r_{n+1}$ cannot happen until $OS_{pre}$ executes, where $OS_{pre}$ is the OS which occurs right before $r_{n+1}$ on Lifeline $L_i$. As the semantic aspect of $seq_n$ defined, $r_{n+1}$ should locate right below $OS_{pre}$ on Lifeline $L_i$ and OSs execute in their graphical order. $\ast_{L_{k+1}}$ returns true. It denotes that only $s_{n+1}$ locates on $L_{k+1}$. Thus, $s_{seq, m_{n+1}}$ codifies the order between the OSs of $seq_n$ and the OSs of $m_{n+1}$ in $seq_{n+1}$. In $s_{seq}$, the OSs of $seq_n$ and the OSs of $m_{n+1}$ respect the order defined by the semantic aspects of $seq_n$.

- **Case 3**: Sending OS $s_{n+1}$ locates on an existing Lifeline, $L_i$ ($i \leq k$), and receiving OS $r_{n+1}$ locates on a new Lifeline, $L_{k+1}$ (see figure 48d). or two OSs of $m_{n+1}$ locate on an existing Lifeline $L_i$ ($i \leq k$) (see figure 48e).

$$s_{seq, m_{n+1}} = (-s_{n+1} \tilde{U} OS_{pre}) \land \ast_{L_{k+1}}$$

Similarly, sub-formula $s_{seq, m_{n+1}}$ defines that $s_{n+1}$ cannot happen until $OS_{pre}$ executes, where $OS_{pre}$ is the OS which occurs right before $s_{n+1}$ on Lifeline $L_i$. As the semantic aspect of $seq_n$ defined, $s_{n+1}$ should locate right below $OS_{pre}$ on Lifeline $L_i$ and OSs execute in their graphical order. $\ast_{L_{k+1}}$ returns true. It denotes that none or only one OS locates on $L_{k+1}$. Therefore $r_{n+1}$ may locate on $L_{k+1}$ or below $s_{n+1}$ on $L_i$. Thus,
\[ \sigma_{seq,n+1} = (\neg s_{n+1} \tilde{u} OS_{pre}) \land (\neg r_{n+1} \tilde{u} OS_{pre}) \]

**Case 4:** Two OSs of \( m_{n+1} \) locate on two existing Lifelines. Without loss of generality, we assume that sending OS \( s_{n+1} \) locates on Lifeline \( L_i \) (\( i \leq k \)), receiving OS \( r_{n+1} \) locates on Lifeline \( L_j \) (\( j \leq k \)) (see Figure 48f).

\[ \Sigma_{seq,n+1} = (\Sigma_{seq})^{n+1} \]

**APPENDIX B**

**PROOF OF THEOREM 2 AND THEOREM 3**

**Theorem 2.** \((\Sigma_{sem}^{eq})^*\) and \( PRED_{2h+2p}(\Sigma_{LTL}^{eq})^*\) are equal.

**Proof:** We use mathematical induction, which is based on the number of CFs, \( r \), directly enclosed in \( seq \).

Base step. The sequence Diagram contains at most one CF, \( cf_1 \). (\( r \leq 1 \))

- Case 1. Sequence Diagram \( seq_0 \) contains no CF. \( (r = 0) \)

  The proof follows the one for basic Sequence Diagram.

- Case 2. Sequence Diagram \( seq_1 \) contains only one CF, \( cf_1 \). (\( r = 1 \))

\[ \tilde{P}_{seq_1} = (\bigwedge_{i \in LN(seq_1)} \tilde{a}_i) \land (\bigwedge_{j \in MSG(seq_1)} \tilde{b}_j) \land (\bigwedge_{j \in MSG(seq_1)} \tilde{d}_j) \land \tilde{c}_i \]

- Case 2.1 We assume that \( cf_1 \) has \( a \) Operands whose Interaction Constraint evaluate to \( False \). The \( b \)th Operand contains \( q_b \) Messages, where \( 1 \leq b \leq a \).

\[ \phi^{cf_1} = \eta^{cf_1} = \bigwedge_{i \in LN(seq_1)} OS_{post} \in post(cf_1 \uparrow_i) \land \bigwedge_{i \in LN(seq_1)} (\neg OS_{post}) \]

(a) We wish to prove that, \( \forall \cdot \tau \in \Sigma_{LTL}^* \), \( \tau \models \Phi \)

First, consider the semantic aspects of the OSs directly enclosed in \( seq_1 \). We project \( seq \) onto each of its covered Lifelines to obtain a EU. We also project \( cf_1 \) onto each of its covered Lifeline to obtain a CEU. Therefore, each EU of \( seq_2 \) may contain a CEU of \( cf_1 \) and BEUs grouped by the OSs directly enclosed in the EU. Similar to the semantics of an EU within a basic Sequence Diagram, the semantics of any BEU directly enclosed in the EU of \( seq_1 \) specifies that OSs are ordered as their graphical order. If \( \tau \models \Phi \), we can easily infer that \( \tau \models \Phi \)

Then, we consider the semantics of \( cf_1 \). It defines that \( cf_1 \) does not execute when the Constraints of all the Operands evaluate to \( False \). \( cf_1 \)'s preceding Interaction Fragments and succeeding Interaction Fragments are ordered by Weak Sequencing. In this case, \( cf_1 \)'s preceding OS must happen before its succeeding OS on each Lifeline. We use LTL formula \( \eta^{cf_1} \) to capture \( cf_1 \)'s semantics. \( \eta^{cf_1} \) does not specify the order of OSs within Operands because the Operands whose Constraints evaluate to \( False \) are excluded. We assume that if \( \tau \models \Phi \), then \( \eta^{cf_1} \) specifies that, on Lifeline \( i \), \( cf_1 \)'s preceding OS, \( OS_{pre} \), occurs after \( cf_1 \)'s succeeding OS, \( OS_{post} \). However, \( \eta^{cf_1} \) specifies that, on each Lifeline covered by \( cf_1 \), its succeeding OS cannot happen until its preceding OS finishes execution. Functions \( pre(cf_1 \uparrow_i) \) and \( post(cf_1 \uparrow_i) \) return the set of OSs which may happen right before and after CEU \( cf_1 \uparrow_i \).

In this case, each set contains at most one OS. Thus, \( OS_{pre} \) must happen before \( OS_{post} \), which contradicts our assumption. In this way, we can prove that \( \tau \models \Phi \). Finally, we consider the interleaving semantics of \( seq_1 \). No OS in \( cf_1 \) can execute, so only the OSs directly enclosed in \( seq_1 \) can
Now we have proven that if $\nu \cdot \tau^\omega \models \varepsilon_{\text{seq}}$, The proof follows the one for basic Sequence Diagram.

Now we have proven that if $\nu \in (\Sigma_{\text{sem}}^\omega)^*$, then $\nu \cdot \tau^\omega \models \Pi_{\text{seq}}$.

(b) We wish to prove that, $\forall \sigma, \sigma \in \Sigma^\omega$, if $\sigma \in (\Sigma_{\text{LT L}}^\omega)|\sigma|_{1 \cdot 2h} \in (\Sigma_{\text{sem}})^*$. In $\Sigma_{\text{LT L}}$, no OS within $cf_1$ is enabled to execute because all the Constraints of $cf_1$’s Operand evaluate to False. If $\sigma \in (\Sigma_{\text{LT L}}^\omega)$, then $\sigma = [seq] \cdot [\tau^\omega]$, which follows Lemma 2. We wish to prove that $\sigma|_{1 \cdot 2h}$ respects all the semantics of $seq_1, \sigma \models \Pi_{\text{seq}_1}$, so $\sigma$ satisfies all sub-formulas of $\Pi_{\text{seq}_1}$. We prove that the sub-formulas capture the semantic aspects as below.

First, we discuss the sub-formulas $\tilde{\alpha}_g$, $\rho_j$, and $\beta_j$ for $seq_1$. Function $\text{TB EU}(seq_1 \uparrow)$ returns the BEUs directly enclosed in $seq_1$ on Lifeline $i$. These BEUs, which are separated using CEU of $cf_1$ on Lifeline $i$, are formed by the OSs directly enclosed in $seq_1$. Function $\text{MSG}(cf_1)$ returns the set of Messages directly enclosed in $cf_1$. We can prove that these sub-formulas capture the semantics of OSs directly enclosed in $seq_1$. The proof follows the one for OSs within basic Sequence Diagram.

Next, we discuss the sub-formula $\eta^{cf_1}$. It defines that, on Lifeline $i$, OSs in $\text{post}(cf_1 \uparrow)$ cannot happen until OSs in $\text{pre}(cf_1 \uparrow)$ finish execution. We assume that, if $\eta^{cf_1}$ does not capture the semantics of $cf_1$, then on a Lifeline, $i$, the preceding OS of $cf_1$, OS$_{\text{pre}_i}$, happens after the succeeding OS of $cf_1$, OS$_{\text{post}_i}$. However, the semantics of $\eta^{cf_1}$ defines the Weak Sequencing between $cf_1$’s preceding OSs and succeeding OSs, i.e., its preceding OSs must happen before its succeeding OS on the same Lifeline. Therefore, OS$_{\text{pre}_i}$ must happen before OS$_{\text{post}_i}$, which contradicts our assumption. In this way, we can prove that $\eta^{cf_1}$ captures the semantics of $cf_1$.

Finally, we discuss the sub-formula $\varepsilon_{\text{seq}_1}$. It represents that only one OS in $|AOS(seq_1)|$ execute at a time, or all OSs in $|AOS(seq_1)|$ have executed. In this case, function $|AOS(seq_1)|$ returns the set of OSs directly enclosed in $seq_1$. We can prove that $\varepsilon_{\text{seq}_1}$ captures the interleaving semantics of $seq_1$ by following the proof for basic Sequence Diagram.

Now we have proven that $\forall \sigma, \sigma \in \Sigma^\omega$, if $\sigma \in (\Sigma_{\text{LT L}}^\omega)$, it respects all the semantic aspects of $seq_1$, i.e., $\sigma|_{1 \cdot 2h} \in (\Sigma_{\text{sem}})^*$.

To conclude, $\forall \nu, \nu \in \Sigma^*$, if $\nu \in (\Sigma_{\text{sem}})^*$, then $\nu \cdot \tau^\omega \models \Pi_{\text{seq}_1}$, and $\forall \nu, \nu \in \Sigma^\omega$, if $\nu \in (\Sigma_{\text{LT L}}^\omega)$, then $\sigma|_{1 \cdot 2h} \in (\Sigma_{\text{sem}})^*$.

- Case 2.2 We assume that $cf_1$ has at least one Operand whose Constraint evaluates to True. The Operator of $cf_1$ is not “alt” or “loop”.

\[ \phi^{cf_1} = \psi^{cf_1} - \theta^{cf_1} \land \bigwedge_{i \in LN(cf_1)} \hat{\tau}_{i}^{cf_1} \land \hat{\nu}^{cf_1} \]

* Case 2.2.1 We assume that, $cf_1$ has two Operands. One Operand contains $p$ Messages, and its Interaction Constraint evaluates to True. The other Operand contains $q$ Messages, and its Interaction Constraint evaluate to False. (see figure 49, where cond1 evaluate to True, and cond2 evaluates to False).

(a) We wish to prove that, $\forall \nu, \nu \in \Sigma^*$, if $\nu \in (\Sigma_{\text{seq}})^*$, then $\nu \cdot \tau^\omega \models \Pi_{\text{seq}_1}$.

First, we consider the semantic aspects of the OSs within each Operand of $cf_1$. The semantic aspects specify that only the order of the OSs within each Operand whose Constraint evaluates to True is maintained. The Operands whose Constraints evaluate to False are excluded. Each Operand can be considered as a basic Sequence Diagram with Constraint. The OSs within each Operand respect the same order as the OSs within a basic Sequence Diagram. Sub-formula $\theta^{cf_1}$ describes the semantics of the Operands whose Constraints evaluate to True using function $\text{T OP}(cf_1)$, where the formula for each Operand follows the formula for a basic Sequence Diagram, i.e., a conjunction of $\tilde{\alpha}_g$s, $\beta_j$s, and $\rho_j$s. Therefore, we can prove that $\nu \cdot \tau^\omega \models \theta^{cf_1}$ by following the proof of basic Sequence Diagram.

Next, we consider the semantic aspects which describe the order between $cf_1$ and its adjacent OSs. $cf_1$ and its adjacent OSs are connected using Weak Sequencing, i.e., for Lifeline $i (1 \leq i \leq j)$, $cf_1$’s preceding OSs must execute before its CEU’s execution, and $cf_1$’s succeeding OSs must execute afterwards. Function $\text{pre}(cf_1 \uparrow)$ returns the set of OSs which may happen right before CEU $cf_1 \uparrow$. The semantics aspect of $seq_1$ defines that, for Lifeline $i (1 \leq i \leq j)$, any OS within $cf_1 \uparrow$ cannot execute until all OSs in $\text{pre}(cf_1 \uparrow)$ finish execution. We wish to prove that
the semantic aspect is captured by the first conjunct of sub-formula \( \tilde{\gamma}^{cf_1} \). We assume that, if \( v \cdot \tau^\omega \) does not satisfy the first conjunct of \( \tilde{\gamma}^{cf_1} \), then \( \tilde{\gamma}^{cf_1} \) defines that, on Lifeline \( i \), at least one OS, \( r_{c+d} \) (see figure 49), occurs before \( OS_{pre} \). \( OS_{pre} \) is an OS in \( pre(cf_1 \uparrow i) \). The first conjunct of \( \tilde{\gamma}^{cf_1} \) specifies that any OS within \( cf_1 \) on Lifeline \( i \) cannot execute until the OSs in \( pre(cf_1 \uparrow i) \) finish execution, so \( OS_{pre} \) must happen before \( r_{c+d} \), which contradicts our assumption. In this way, we can prove that \( v \cdot \tau^\omega \) satisfies the first conjunct of \( \tilde{\gamma}^{cf_1} \). Similarly, we can also prove that \( v \cdot \tau^\omega \) satisfies the second conjunct of \( \tilde{\gamma}^{cf_1} \). Hence, \( v \cdot \tau^\omega \models \tilde{\gamma}^{cf_1} \).

Finally, we consider the semantic aspect for the \( seq_1 \). We define the OSs which are directly enclosed in \( seq_1 \) or Operands whose Constraints evaluate to True as enabled OSs. We define the OSs which are directly enclosed in \( seq_1 \) or Operands whose Constraints evaluate to True as enabled OSs, i.e., these OSs can be enabled to occur. Function \( AOS(seq_1) \) returns the set of enabled OSs within \( seq_1 \). The semantic aspect specifies that only one enabled OS can execute at a time, and all the enabled OSs should execute uninterrupted. If \( v \in (\Sigma_{seq_1}^*) \), we can deduce that \( |v| = |AOS(seq_1)| = 2h + 2p \). It is easy to infer that \( v \cdot \tau^\omega \models \Pi_{seq_1} \).

Now we have proven that if \( v \in (\Sigma_{seq_1}^*) \), then \( v \cdot \tau^\omega \models \Pi_{seq_1} \).

(b) We wish to prove that, \( \forall \sigma \sigma \in \Sigma^\omega \), if \( \sigma \in (\Sigma_{seq_1}^*)^\omega \), \( \sigma[1..2h+2p] \in (\Sigma_{seq_1})^* \).

If \( \sigma \in (\Sigma_{seq_1}^*)^\omega \), then \( \sigma = \sigma[1..2h+2p] \cdot \tau^\omega \), which follows Lemma 2. We wish to prove that \( \sigma[1..2h+2p] \) respects all the semantics of \( seq_1 \). \( \sigma \models \Pi_{seq_1} \), so \( \sigma \) satisfies all sub-formulas of \( \Pi_{seq_1} \). We prove that the sub-formulas capture the semantic aspects as below.

First, we discuss the sub-formulas \( \tilde{\alpha}_{_g}, \rho_{_j}, \) and \( \beta_{_j} \) for \( seq_1 \). We can prove that these sub-formulas capture the semantics of OSs directly enclosed in \( seq_1 \). The proof follows the one in case 2.1.

Then, we discuss the sub-formula \( \tilde{\beta}^{cf_1} \). Function \( \bigwedge_{op \in T\cap P(cf_1)} \) returns the set of Operands whose Constraints evaluate to True within \( cf_1 \). Hence, \( \tilde{\beta}^{cf_1} \) only captures the semantics of Operands whose Constraints evaluate to True. It is consistent with the semantic aspect of \( cf_1 \), which excludes the Operands whose Constraints evaluate to False. For each Operand whose Constraints evaluate to True, we wish to prove that sub-formulas \( \tilde{\alpha}_{_g}, \rho_{_j}, \) and \( \beta_{_j} \) capture its semantics. \( cf_1 \) contains no other CFs, so \( ABEU(op \uparrow i) \) returns the BEU of \( op \) on Lifeline \( i \). We can consider an Operand with no nested CFs as a basic Sequence Diagram with Interaction Constraint. In this way, we can prove that these sub-formulas capture the Operand’s semantics.
by following the proof of basic Sequence Diagram. Therefore, we have proven that \( \gamma_{\preccurlyeq f1} \) captures the semantics of Combined Fragment \( cf_1 \).

Next, we discuss the sub-formula \( \gamma_{\preccurlyeq f1} \) for Lifeline \( i \). We wish to prove that it captures the order of CEU \( cf_1 \uparrow_i \) and its preceding/succeeding OSs on Lifeline \( i \). The first conjunct of \( \gamma_{\preccurlyeq f1} \) defines that any OS in \( \preceq(cf_1 \uparrow_i) \) cannot happen before all OSs in \( \preceq(cf_1 \uparrow_i) \) finish execution. If it does not capture the semantic aspect, then we assume that at least an OS in \( \preceq(cf_1 \uparrow_i) \), \( OS_{preu} \), occurs after an OS in \( cf_1 \uparrow_i \), \( r_{c+d} \).

Function \( \preceq(cf_1 \uparrow_i) \) returns the set of OSs which may happen right before CEU \( cf_1 \uparrow_i \). The semantics defines that all OS in \( \preceq(cf_1 \uparrow_i) \) must happen before all OS within CEU \( cf_1 \uparrow_i \). Thus, \( OS_{preu} \) must occur before \( r_{c+d} \) which contradicts our assumption. In this way, we have proven that the first conjunct of \( \gamma_{\preccurlyeq f1} \) captures the order of CEU \( cf_1 \uparrow_i \) and its preceding OSs on Lifeline \( i \).

Therefore, we have proven that \( \gamma_{\preccurlyeq f1} \) captures the order of CEU \( cf_1 \uparrow_i \) and its preceding/succeeding OSs on Lifeline \( i \).

Finally, we discuss the sub-formula \( \varepsilon_{seqi} \). It represents that only one OS in \( |AOS(seqi)| \) executes at a time, or all OSs in \( |AOS(seqi)| \) have executed. Function \( |AOS(seqi)| \) returns the set of OSs which can be enabled to execute in \( seqi \), i.e., it returns a set which includes the OSs directly enclosed in \( seqi \) and the OSs within \( cf_i \)'s Operand whose Constraint evaluates to True. In \( seqi \), \( |AOS(seqi)| = 2h + 2p \). From lemma 2, if \( \sigma \models \varepsilon_{seqi} \), then \( \sigma = \sigma_{[1, \leq 2h+2p]} \cdot \tau^\omega \). Therefore, \( \varepsilon_{seqi} \) captures the semantic aspect, which enforces that only one object can execute an OS at a time and all enabled OSs of \( seqi \) execute uninterrupted.

Now we have proven that \( \forall v, \sigma \in \Sigma^\omega, \) if \( \sigma \in (\Sigma_{LTTL})^\omega \) respects all the semantic aspects of \( seqi \), i.e., \( \sigma_{[1, \leq 2h+2p]} \in (\Sigma_{sem})^\omega \).

If \( cf_1 \) contains more than two Operands, \( p \) Messages may be enclosed in multiple Operands whose Interaction Constraints evaluate to True, and \( q \) Messages may be enclosed in multiple Operands whose Interaction Constraints evaluate to False. The proof follows the one for \( cf_1 \) with two Operands.

To conclude, \( \forall u, v \in \Sigma^\omega, \) if \( v \in (\Sigma_{sem})^\omega \), then \( v \cdot \tau^\omega \models \Pi_{seqi} \), and \( \forall v, \sigma \in \Sigma^\omega, \) if \( \sigma \in (\Sigma_{LTTL})^\omega \), then \( \sigma_{[1, \leq 2h+2p]} \in (\Sigma_{sem})^\omega \).

We have proven that the semantic rules general to all CFs can be captured by our LTL templates. The semantic rules for each CF with different Operators can be enforced by adding different semantic constraints, which are captured using LTL template \( gCF \). Parallel defines that the OSs within different Operands may be interleaved. Its semantics does not introduce additional semantic rule. Thus, we have proven that our LTL templates capture the semantics of Parallel.

We use Strict Sequencing as an example to prove that the semantic rule for each Operator can be captured by our LTL templates. The cases for CFs with other Operators can be proven similarly.

* Case 2.2.2 We assume that, a given Strict Sequencing, \( cf_{1\text{strict}} \), has two Operands whose Interaction Constraints evaluate to True. The first Operand contains \( p_1 \) Messages, and the second Operand contains \( p_2 \) Messages. \( cf_{1\text{strict}} \) covers \( i \) Lifelines.

(a) We wish to prove that, if \( u \in \Sigma^\omega, \) if \( v \in (\Sigma_{seqi})^\omega \), then \( v \cdot \tau^\omega \models \Pi_{seqi} \).

The Strict Sequencing imposes an order among OSs within different Operands. For an Operand (not the first Operand), any OS cannot occur before the OSs within the previous Operand finish execution. Function \( \preceq EU(u) \) returns the set of OSs within \( EU \) which happen right before \( EU \), i.e., the Constraint of \( EU \) evaluates to True. In this case, \( \preceq EU(u) \) returns the last OS in \( EU \). The semantic aspect of Strict Sequencing can be considered as that, any OS in Operand \( k \) cannot happen until the OSs in all \( \preceq EU(u) \), where \( u \) is the EU of \( Operand \) \( k \) on Lifeline \( j \) in \( \leq i \), finish execution. We introduce sub-formula \( \chi_k \) to capture the semantics of \( Operand \) \( k \). We assume that, if \( v \cdot \tau^\omega \) does not satisfies \( \bigwedge_{k \in NFTOP(cf_{1\text{strict}})} \chi_k \) then \( \chi_k \) defines that at least one OS, \( OS_{si} \), in \( Operand \) \( k \) occurs before \( OS_{preu} \), which is an OS in \( \preceq EU((k-1) \uparrow_i) \), where \( 1 \leq j \leq i \). Sub-formula \( \chi_k \) specifies that any OS within \( \preceq EU(u) \) on all the Lifelines covered by the Strict Sequencing must happen before the OSs within \( Operand \) \( k \). Therefore, \( OS_{preu} \) must happen before \( OS_{si} \), which contradicts our assumptions. Thus, we can prove that \( v \cdot \tau^\omega \models \bigwedge_{k \in NFTOP(cf_{1\text{strict}})} \chi_k \).
We have proven that \( \nu \cdot \tau^\omega \) satisfies other general sub-formulas of \( \Pi_{seq.i} \) in case 2.1.2(1). Hence, we can prove that \( \nu \cdot \tau^\omega \models \Pi_{seq.i} \).

(b) We wish to prove that, \( \forall \sigma, \tau \in \Sigma^\omega, \) if \( \sigma \in (\Sigma_{seq.1})^\omega, \) then \( \sigma \in (\Sigma_{seq.1})^\omega \).

If \( \sigma \in (\Sigma_{seq.1})^\omega \), then \( \sigma = \sigma_{[1, 2h + 2p] + 2p} \) · \( \tau^\omega \), which follows Lemma 2. We wish to prove that \( \sigma_{[1, 2h + 2p] + 2p} \) respects all the semantics of \( seq.1 \), \( \sigma \models \Pi_{seq.1} \), so \( \sigma \) satisfies all sub-formulas of \( \Pi_{seq.1} \). We have proven that the sub-formulas \( \bar{\sigma}_g \), \( \beta_j \), and \( \beta_j \) capture the semantics of OS directly enclosed in \( seq.1 \) sub-formulas of \( \theta_{f \triangleright m} \) and \( \gamma_{i,m} \). Each OS within \( seq.1 \) sub-formulas \( \theta_{f \triangleright m} \) and \( \gamma_{i,m} \) captures the general semantic aspects of \( cf_{f \triangleright m} \), sub-formula \( \varepsilon_{seq.1} \) captures the interleaving semantics of \( seq.1 \) (see case 2.1.2(1)). Now we need to prove that sub-formula \( \bigwedge_{k \in NFTOP(cf_{f \triangleright m})} \chi_k \) captures the semantics of Strict Sequencing.

Sub-formula \( \bigwedge_{k \in NFTOP(cf_{f \triangleright m})} \chi_k \) asserts the order between each Operand \( k \) of Strict Sequencing (\( k \) is not the first Operand), and its preceding Operand. Function \( preEU(u) \) returns the set of OSs within EU \( u \) which happen right before EU \( u \). Each OS within \( k \) cannot happen until all OS within \( preEU(u) \) on all the Lifelines covered by the Strict Sequencing. If the sub-formula does not capture the semantics of Strict Sequencing, we assume the semantics defines that at least an OS in \( preEU((k - 1) \uparrow) \) \( (1 \leq j \leq i) \), \( OS_{pre} \), occurs after an OS in Operand \( k \), \( OS_{s} \). Actually, the semantics of Strict Sequencing defines that in any Operand except the first one, OSs cannot execute until the previous Operand completes. Therefore, \( OS_{pre} \) must happen before \( OS_{s} \), which contradicts our assumption. In this way, we can prove that sub-formula \( \bigwedge_{k \in NFTOP(cf_{f \triangleright m})} \chi_k \) captures the semantics of Strict Sequencing. Hence, we can prove that \( \sigma_{[1, 2h + 2p]} \in (\Sigma_{seq.1})^\omega \).

To conclude, \( \forall \nu, \sigma, \tau \in \Sigma^\omega, \) if \( \nu \in (\Sigma_{seq.1})^\omega \), then \( \nu \cdot \tau^\omega \models \Pi_{seq.1} \), and \( \forall \sigma, \tau \in \Sigma^\omega, \) if \( \sigma \in (\Sigma_{seq.1})^\omega \), then \( \sigma \models \Pi_{seq.1} \).

- Case 2.3 The semantics of Alternatives defines that at most one of its Operand whose Constraints evaluate to True is chosen to execute. The Operands whose Constraints evaluate to False are still excluded. To capture its semantics, we need to specify the semantics of the chosen Operand and the connection between the chosen Operand and its adjacent OSs. We use LTL formula \( \Psi_{alt} \) to capture the semantic of Alternatives. Sub-formulas \( \theta_{m} \) and \( \gamma_{i,m} \) can be rewritten into \( \theta_{m} \) and \( \gamma_{i,m} \) by following the same procedures of rewriting sub-formulas \( \theta_{cf_{f \triangleright m}} \) and \( \gamma_{cf_{f \triangleright m}} \). The LTL formula of Alternatives, \( \Psi_{alt} \), with rewritten sub-formulas is shown in figure 50.

We assume that, a given Alternatives, \( cf_{alt} \), has two Operands whose Interaction Constraints evaluate to True. The first Operand contains \( p_1 \) Messages, and the second Operand contains \( p_2 \) Messages. \( cf_{alt} \) covers \( i \) Lifelines.

\[
\psi_{f \downarrow 1} = \psi_{c f_{alt} \downarrow 1}
\]

(a) We wish to prove that, \( \forall \nu, \psi \in \Sigma^\omega, \) if \( \nu \in (\Sigma_{seq.1})^\omega \), then \( \nu \cdot \tau^\omega \models \Pi_{seq.1} \).

For Alternatives. We only consider the Operands whose Constraints evaluate to True as defined by the general semantics rules. If more than one Operand’s Constraint evaluates to True, at most one Operand is chosen and the order of the OSs within it should be specified. Sub-formula \( \psi_{m} \) defines the semantics of Operand \( m \) whose Constraint evaluates to True. If \( m \) is chosen, its semantics is captured by sub-formula \( \theta_{f \downarrow m} \) and \( \gamma_{f \downarrow i,m} \). Otherwise, \( \psi_{m} \) evaluates to True, denoting that \( m \) is excluded. We can prove that \( \theta_{f \downarrow m} \) describes the order among OSs within \( m \) by following the proof for sub-formula \( \theta_{f \downarrow m} \).

Similarly, we can prove that \( \gamma_{f \downarrow i,m} \) describes the order among OSs within \( m \) and the Alternatives’ adjacent OSs by following the proof for sub-formula \( \gamma_{f \downarrow i,m} \). Therefore, \( \nu \cdot \tau^\omega \) satisfies \( \psi_{f \downarrow i,m} \).

We have proven that \( \nu \cdot \tau^\omega \) satisfies \( \bar{\sigma}_g \), \( \beta_j \), and \( \beta_j \) for \( seq.1 \) in case 2.1.2(1). For sub-formula \( \varepsilon_{seq.1} \), function \( AOS(seq.1) \) returns the enabled and chosen OSs, i.e., for Alternatives, only the OSs within the chosen Operand are returned. We can prove that \( \nu \cdot \tau^\omega \) satisfies \( \varepsilon_{seq.1} \) by following the proof in case 2.1.2(1). Hence, we can prove that \( \nu \cdot \tau^\omega \models \Pi_{seq.1} \).

(b) We wish to prove that, \( \forall \sigma, \tau \in \Sigma^\omega, \) if \( \sigma \in (\Sigma_{seq.1})^\omega \), then \( \sigma \models \Pi_{seq.1} \).

If \( \sigma \in (\Sigma_{seq.1})^\omega \), then \( \sigma \models \Pi_{seq.1} \) so \( \sigma \) satisfies all sub-formulas of \( \Pi_{seq.1} \). We have proven that the sub-formulas \( \bar{\sigma}_g \), \( \beta_j \), and \( \beta_j \) capture the semantics of OS directly enclosed in \( seq.1 \); sub-formula \( \varepsilon_{seq.1} \) captures the interleaving semantics of \( seq.1 \) (see case 2.1.2(1)). We need
Fig. 50. Rewriting LTL formula for Alternatives

to prove that sub-formula \( \Psi_{\text{alt}}^{\text{alt}} \) captures the semantics of Alternatives.

Sub-formula \( \Psi_{\text{alt}}^{\text{alt}} \) is a conjunction of sub-formula \( \Psi_{\text{alt}}^{\text{alt}} \), where \( m \) is an Alternatives’s Operand whose Constraint evaluates to True.

Therefore, the Operands whose Constraints evaluate to False are excluded. \( \Psi_{\text{alt}} \) evaluates to False if \( m \) is unchosen, which captures the semantics that the unchosen Operands are excluded. \( \Psi_{\text{alt}} \) is a conjunction of sub-formulas \( \tilde{\psi}^{\text{alt}} \) and \( \tilde{\gamma}^{\text{alt}} \) when \( m \) is the chosen Operand. We can prove that sub-formula \( \tilde{\psi}^{\text{alt}} \) captures the order among OSs within \( m \) by following the proof of \( \tilde{\psi}^{\text{alt}} \). We can also prove that sub-formula \( \tilde{\gamma}^{\text{alt}} \) captures the order between OSs within \( m \) and the Alternatives’s adjacent OSs by following the proof of \( \tilde{\gamma}^{\text{alt}} \).

In this way, we can prove that sub-formula \( \tilde{\psi}^{\text{alt}} \) captures the semantics of Alternatives. Hence, we can prove that \( \sigma[1..2h+2p_n] \in (\Sigma_{\text{seq}})^* \).

To conclude, \( \forall v,v' \in \Sigma^* \), if \( v \in (\Sigma_{\text{seq}})^* \), then \( v \cdot \tau^v = \bar{\Pi}_{\text{seq}} \), and \( \forall \sigma, \sigma' \in \Sigma^\omega \), if \( \sigma \in (\Sigma_{\text{LTL}})^\omega \), then \( \sigma[1..2h+2p_n] \in (\Sigma_{\text{seq}})^* \).

Case 2.4 The Loop represents the iterations of its sole Operand. We capture the semantics of Loop using LTL formula \( \Psi_{\text{loop}}^{\text{alt}} \) which unfolds the iterations and connects them using Weak Sequencing. Each iteration can be considered as an Operand, whose semantics can be captured by sub-formulas \( \alpha_g, \beta_j, \rho_j \) and \( \gamma_i \) as proven. We need to prove that sub-formula \( \bigwedge \gamma_i \in \text{NFTOP}(CF) \) Sequencing among iterations. The proof is quite similar as the proof for sub-formula \( \bigwedge \gamma_i \in \text{NFTOP}(CF) \).

Inductive step. A given Sequence Diagram, \( \text{seq}_n \), directly contains \( n \) CFs. For the Messages within the CFs, \( p_n \), Messages are chosen and enabled in Operands whose Interaction Constraints evaluate to True. We assume \( \forall v,v' \in \Sigma^* \), if \( v \in (\Sigma_{\text{seq}})^* \), then \( v \cdot \tau^v = \bar{\Pi}_{\text{seq}} \).

(1) \( \forall \sigma, \sigma' \in \Sigma^\omega \), if \( \sigma \in (\Sigma_{\text{LTL}})^\omega \), then \( \sigma[1..2h+2p_n] \in (\Sigma_{\text{seq}})^* \).

We add a CF, \( cf_{n+1} \), in \( \text{seq}_n \) to form a new Sequence Diagram, \( \text{seq}_{n+1} \), with \( n + 1 \) CFs. \( cf_{n+1} \) is directly enclosed in \( \text{seq}_{n+1} \). In \( cf_{n+1} \), \( p_{n+1} \) Messages are chosen and enabled in Operands whose Interaction Constraints evaluate to True. We wish to prove that, \( \forall v', v'' \in \Sigma^* \), if \( v'' \in (\Sigma_{\text{seq}_{n+1}})^* \), then \( v'' \cdot \tau^{v''} = \bar{\Pi}_{\text{seq}_{n+1}} \).

First, we consider the semantic aspects of the OSs directly enclosed in \( \text{seq}_{n+1} \). We can prove that \( v'' \cdot \tau^{v''} = \bar{\Pi}_{\text{seq}_{n+1}} \).

Next, we consider the semantic aspects of \( cf_{n+1} \), which is captured by formula \( \Phi_{cf} \). For \( \Phi_{cf} \),
sub-formulas $\tilde{\theta}f_{n+1}$, $\tilde{\gamma}f_{n+1}$, and the additional ones for each Operator still define the semantics we have proven in base case. The order of OSs within each CF is not changed. Therefore, $v' \cdot \tau'$ satisfies $\tilde{\theta}f_{n+1}$ and the additional sub-formulas for each Operand.

Sub-formula $\tilde{\gamma}f_{n+1}$ still specifies the Weak Sequencing between $f_{n+1}$ and its preceding/succeeding Interaction Fragments. Comparing to base case, $f_{n+1}$’s preceding/succeeding Interaction Fragments can be OSs or CFs. We wish to prove that our algorithms for calculating $\text{pre}(f_{n+1} \uparrow_i)$ and $\text{post}(f_{n+1} \uparrow_i)$ are correct.

Function $\text{pre}(f_{n+1} \uparrow_i)$ returns the set of OSs which happen right before $f_{n+1}$, and $\text{post}(f_{n+1} \uparrow_i)$ returns the OS in the bottom of $f_{n+1}$, respectively. We have proven in base case $2.2.1$ and $2.2.2$. On Lifeline $i$, functions $\text{pre}(f_{n+1} \uparrow_i)$ and $\text{post}(f_{n+1} \uparrow_i)$ return the set of OSs which are directly contained in $f_{n+1}$, and all enabled OSs should execute uninterrupted. If $v' \in (\Sigma^{seq+n+1})$, we can deduce that $|v'| = |AOS(seq_{n+1})| = 2h + 2p + 2p_{n+1}$. It is easy to infer that $v' \cdot \tau' = \tilde{\gamma}f_{n+1}$.

Now we have proven that if $v' \in (\Sigma^{seq+n+1})$, then $v' \cdot \tau' = \tilde{\Pi}_{seq+n+1}$.

(b) We wish to prove that, $\forall \sigma' \in \Sigma^n$, if $\sigma' \in (\Sigma_{LTL}^{seq+n+1})$, then $\sigma'_{[1,2h+2p+2p_{n+1}]} \in (\Sigma^{seq+n+1})$.

If $\sigma' \in (\Sigma_{LTL}^{seq+n+1})$, then $\sigma' = \sigma'_{[1,2h+2p+2p_{n+1}]} \cdot \tau'$, which follows Lemma 2. We wish to prove that $\sigma'_{[1,2h+2p+2p_{n+1}]}$ respects all the semantics of $seq_{n+1}$. $\sigma' = \tilde{\Pi}_{seq_{n+1}}$, so $\sigma'$ satisfies all sub-formulas of $\tilde{\Pi}_{seq_{n+1}}$. We prove that the sub-formulas capture the semantic aspects as below.

First, we discuss the sub-formulas $\tilde{\alpha}_g, \rho_j$, and $\beta_j$ for $seq_{n+1}$. We can prove that these sub-formulas capture the semantics of OSs directly enclosed in $seq_{n+1}$. The proof follows the one in case 2.1.

Then, we discuss the sub-formula $\Phi_{CF}. In seq_{n+1}$, the sub-formula captures the semantics of $n$ CFs. In $seq_{n+1}$, adding $f_{n+1}$ does not change the semantics of the existing CFs. It is easy to infer that, sub-formula $\tilde{\Phi}_{CF}$ still captures the semantics of the CFs except for $f_{n+1}$.

Next, we discuss the sub-formula $\tilde{\theta}f_{n+1}$, which is a conjunction of sub-formulas $\tilde{\theta}f_{n+1}$, and the additional one for each Operator. With the proof of base case, $\tilde{\theta}f_{n+1}$ captures the semantics of $f_{n+1}$’s Operands, while the additional sub-formula captures the semantics of $f_{n+1}$’s Operator. Sub-formula $\tilde{\gamma}f_{n+1}$ may be different from the base case, since the preceding/succeeding Interaction Fragment of $f_{n+1}$ can be other CFs. On Lifeline $i$, functions $\text{pre}(f_{n+1} \uparrow_i)$ and $\text{post}(f_{n+1} \uparrow_i)$ return the set of OSs which may happen right before and after $f_{n+1}$, respectively. We have proven that our algorithms can calculate $\text{pre}(f_{n+1} \uparrow_i)$ and $\text{post}(f_{n+1} \uparrow_i)$ for all the cases. Thus, we can infer that $\tilde{\gamma}f_{n+1}$ still captures the Weak Sequencing between $f_{n+1}$ and its preceding/succeeding Interaction Fragments.

Finally, we discuss the sub-formula $\tilde{\epsilon}_{seq_{n+1}}$. It represents only one OS in $|AOS(seq_{n+1})|$ execute at a time, or all OSs in $|AOS(seq_{n+1})|$ have executed. Function $|AOS(seq_{n+1})|$ returns the set of OSs which are chosen and enabled to execute in $seq_{n+1}$. In $seq_{n+1}$, $|AOS(seq_{n+1})| = 2h + 2p + 2p_{n+1}$. From lemma 2, if
σ |= ε_{seq_{n+1}}, then σ = σ[1..2h+2p_n+2p_{n+1}] \cdot τ^ω. Therefore, ε_{seq_{n+1}} captures the interleaving semantics of seq_{n+1}.

Now we have proven that ∀σ',σ' ∈ Σ^ω, if σ' ∈ (Σ_{LTL}^ω)^n+1, i.e., σ'[1..2h+2p_n+2p_{n+1}] ∈ (Σ_{LTL}^ω)^n+1.

To conclude, ∀u',u' ∈ Σ^ω, if u' ∈ (Σ_{LTL}^ω)^n+1, then u' · τ^ω = \bar{Π}_{\text{seq}_{n+1}}, and ∀σ',σ' ∈ Σ^ω, if σ' ∈ (Σ_{LTL}^ω)^n+1, then σ'[1..2h+2p_n+2p_{n+1}] ∈ (Σ_{LTL}^ω)^n+1.

Interaction Constraints evaluate to True. We wish to prove that, ∀u',u' ∈ Σ^ω, if u' ∈ (Σ_{seq_{n+1}}^ω)^n+1, then u' · τ^ω = \bar{Π}_{\text{seq}_{n+1}}, ∀σ',σ' ∈ Σ^ω, if σ' ∈ (Σ_{LTL}^ω)^n+1, then σ'[1..2h+2p_n+2p_{n+1}] ∈ (Σ_{LTL}^ω)^n+1.

When we add cf_u into seq_{n+1}, then order of the OSs directly enclosed in seq_{n+1} keep unchanged. Thus, the semantics of the OSs directly enclosed in seq_{n+1} can still be captured using the corresponding sub-formulas of \bar{Π}_{\text{seq}_{n+1}}. The LTL templates \bar{Π}_{\text{seq}_{n+1}} and \bar{Π}_{\text{seq}_{n+1}} are shown as,

\bar{Π}_{\text{seq}_{n+1}} = \bigwedge_{i \in \text{L}(\text{seq}_{n+1})} \bar{\alpha}_g \wedge \bigwedge_{j \in \text{M}(\text{seq}_{n+1})} (j \in \text{CF}_{\text{seq}_{n+1}}) \wedge (j \in \text{CF}_{\text{seq}_{n+1}}) \wedge (\bar{\alpha}_f \wedge \bar{\beta}_j) \wedge (\bar{\alpha}_g \wedge \bar{\alpha}_f)

Theorem 3. \( (Σ_{LTL}^ω)^n+1 \) and \( \text{PREF}_{2h+2p_n}(Σ_{LTL}^ω) \) are equal.

Proof: We use mathematical induction, which is based on the maximal layer of CF, l, within seq_{n+1}.

Base step. seq_{n+1} directly contains r CFs, each of which does not contain other CFs. (l = 1)

The proof follows the one for theorem 2.

Inductive step. seq_{n+1} directly contains r CFs. We assume that cf_u, which is a CF directly enclosed in seq_{n+1}, contains cf_{u'}, which is a CF with the maximal layer within seq_{n+1}. The maximal layer of CF within seq_{n+1} is n. For the Messages within the CFs, p_n Messages are chosen and enabled in Operands whose Interaction Constraints evaluate to True. We assume ∀u',u' ∈ (Σ_{LTL}^ω)^n+1, u' · τ^ω = \bar{Π}_{\text{seq}_{n+1}}, ∀σ',σ' ∈ (Σ_{LTL}^ω)^n+1, then σ'[1..2h+2p_n+2p_{n+1}] ∈ (Σ_{LTL}^ω)^n+1.

We add a CF, cf_{u'}, in seq_{n+1} to form a new Sequence Diagram, seq_{n+1}, where cf_u contains cf_{u'}.

The layer of cf_u becomes n + 1, which is the maximal layer of CF within seq_{n+1}. In seq_{n+1}, p_{n+1} Messages are chosen and enabled in Operands whose

(a) We wish to prove that, ∀u',u' ∈ Σ^ω, if u' ∈ (Σ_{seq_{n+1}}^ω)^n+1, then u' · τ^ω = \bar{Π}_{\text{seq}_{n+1}}.

We wish to prove that u' · τ^ω satisfies all sub-formulas of \bar{Π}_{\text{seq}_{n+1}}.

First, we consider the CFs directly enclosed in seq_{n+1}.

Then, we consider the CFs (except cf_u) directly enclosed in seq_{n+1}.

Next, we consider CF cf_u whose semantics is captured using \Phi_{cf_u}. We discuss sub-formula \Phi_{cf_u} using cases. (1) If all the Constraints of cf_u’s Operands evaluate to False, \Phi_{cf_u} = \Phi_{cf_u}. We can prove that u' · τ^ω satisfies \Phi_{cf_u}. The proof follows the one for base case. (2) If not all the Constraints of cf_u’s Operands
evaluate to False, and the Operator of $cf_u$ is not alt or loop, $\Phi^{cf_u} = \Psi^{cf_u} \land \Phi^{cf_v}$. The semantics of $cf_u$ is not altered by adding $cf_f$. Hence, we can infer that $u' \cdot \tau^u \models \Phi^{cf_f}$, $\Psi^{cf_u} = \Phi^{cf_f} \land \bigwedge_{i \in LN(cf_u)} \tilde{\gamma}^{cf_u}_i$. We can prove that $u' \cdot \tau^u \models \tilde{\Phi}^{cf_u}$ and $\bigwedge_{i \in LN(cf_u)} \tilde{\gamma}^{cf_u}_i$. The proof follows the one for base case. (3) If not all the Constraints of $cf_u$’s Operands evaluate to False, and the Operator of $cf_u$ is alt or loop, $\Phi^{cf_u} = \Psi^{cf_u} \land \Phi^{cf_f}$ or $\Psi^{cf_u} = \Phi^{cf_f}$ respectively. Similarly, we can prove that $u' \cdot \tau^u \models \tilde{\Phi}^{cf_f}$ and $\Phi^{cf_f}$.

Finally, we consider the interleaving semantics of $seq_{n+1}^{nested}$. Function $AOS(seq_{n+1}^{nested})$ returns the set of chosen and enabled OSs within $seq_{n+1}^{nested}$. Sub-formula $\varepsilon_{seq_{n+1}^{nested}}$ specifies that only one OS execute at a state, or all OS have executed. If $u' \in (\Sigma_{sem}^{seq_{n+1}^{nested}})^*$, we can deduce that $u' \models AOS(seq_{n+1}^{nested}) = 2h + 2p_{n+1}$. It is easy to infer that $u' \cdot \tau^u \models \varepsilon_{seq_{n+1}^{nested}}$.

Now we have proven that if $u' \in (\Sigma_{sem}^{seq_{n+1}^{nested}})^*$, then $u' \cdot \tau^u \models \prod_{seq_{n+1}^{nested}}$.

(b) We wish to prove that, $\forall \sigma', \sigma' \in \Sigma^w$, if $\sigma' \in (\Sigma_{LTL}^{seq_{n+1}^{nested}})^w$, then $\sigma'_1 = 2i_2 + 2p_{n+1} \in (\Sigma_{sem}^{seq_{n+1}^{nested}})^*$. If $\sigma' \in (\Sigma_{LTL}^{seq_{n+1}^{nested}})^w$, then $\sigma' = \sigma'_1 \cdot \tau^u$, which follows Lemma 2. We wish to prove that $\sigma'_1 = 2i_2 + 2p_{n+1}$ respects all the semantics of $seq_{n+1}^{nested}$. $\sigma' \models \prod_{seq_{n+1}^{nested}}$, so $\sigma'$ satisfies all sub-formulas of $\prod_{seq_{n+1}^{nested}}$. We prove that the sub-formulas capture the semantic aspects as below.

First, we discuss the sub-formulas $\tilde{\alpha}_i$, $\rho_j$, and $\beta_j$ for $seq_{n+1}^{nested}$. These sub-formulas are not changed, so they still capture the semantics of OSs directly enclosed in $seq_{n+1}^{nested}$. We can also infer that these sub-formulas capture the semantics of OSs directly enclosed in $seq_{n+1}^{nested}$.

Then, we discuss the sub-formula $\bigwedge {CF_{n+1}^{seq_{n+1}^{nested}} CF_{n+1}^{seq_{n+1}^{nested}}}$. For $seq_{n+1}^{nested}$, the sub-formula captures the semantics of the CFs (except for $cf_f$) directly enclosed in it. In $seq_{n+1}^{nested}$, adding $cf_f$ does not change the semantics of the CFs except for $cf_f$. It is easy to infer that, the sub-formula still captures the semantics of the CFs except for $cf_f$.

Next, we discuss the sub-formula formula $\Phi^{cf_f}$ using three cases. (1)$\Psi^{cf_f} = \eta^{cf_f}$. We can prove that the sub-formula captures the semantics of $cf_f$ when all the Constraints of $cf_f$’s Operands evaluate to False. The proof follows the one for base case. (2) $\Phi^{cf_f} = \Psi^{cf_f} \land \Phi^{cf_f}$. We wish to prove that the sub-formula captures the semantics of $cf_f$ if not all the Constraints of $cf_f$’s Operands evaluate to False, and the Operator of $cf_f$ is not alt or loop. With our assumption, $\Phi^{cf_f}$ still captures the semantics of $cf_f$. $\Psi^{cf_f} = \tilde{\Phi}^{cf_f} \land \bigwedge_{i \in LN(cf_f)} \tilde{\gamma}^{cf_f}_i$, $\tilde{\Phi}^{cf_f}$ captures the order of OSs directly enclosed in $cf_f$, while $\bigwedge_{i \in LN(cf_f)} \tilde{\gamma}^{cf_f}_i$ captures the order between $cf_f$ and its preceding/succeeding Interaction Fragments. The proof follows the one for base case. The semantics of the OSs directly enclosed in $cf_f$ and the semantics of $cf_f$ are independent. Therefore $\Phi^{cf_f}$ and $\Psi^{cf_f}$ are connected using conjunction. In this way, we can prove that $\tilde{\Phi}^{cf_f}$ captures the semantics of $cf_f$. (3)$\Psi^{cf_f} = \Phi^{cf_f} \land \Phi^{cf_f}$. Similarly, we can prove that the sub-formula captures the semantics of $cf_f$ if not all the Constraints of $cf_f$’s Operands evaluate to False, and the Operator of $cf_f$ is alt or loop.

Finally, we discuss the sub-formula $\varepsilon_{seq_{n+1}^{nested}}$. It represents that only one OS in $AOS(seq_{n+1}^{nested})$ executes at a time, or all OSs in $AOS(seq_{n+1}^{nested})$ have executed. Function $AOS(seq_{n+1}^{nested})$ returns the set of chosen and enabled OSs within $seq_{n+1}^{nested}$, where $AOS(seq_{n+1}^{nested}) = 2h + 2p_{n+1}$. From lemma 2, if $\sigma^u \models \varepsilon_{seq_{n+1}^{nested}}$, then $\sigma^u = 1 \cdot 2h + 2p_{n+1} \cdot \tau^u$. Therefore, $\varepsilon_{seq_{n+1}^{nested}}$ captures the interleaving semantics of $seq_{n+1}^{nested}$.

Now we have proven that $\forall \sigma', \sigma' \in \Sigma^w$, if $\sigma' \in (\Sigma_{LTL}^{seq_{n+1}^{nested}})^w$, respects all the semantic aspects of $seq_{n+1}^{nested}$, i.e., $\sigma' \cdot 1 \cdot h + 2p_{n+1} \in (\Sigma_{sem}^{seq_{n+1}^{nested}})^*$.

To conclude, $\forall u' \cdot u' \models \Sigma^w$, if $u' \in (\Sigma_{sem}^{seq_{n+1}^{nested}})^*$, then $u' \cdot \tau^u \models \prod_{seq_{n+1}^{nested}}$, and $\forall \sigma', \sigma' \in \Sigma^w$, if $\sigma' \in (\Sigma_{LTL}^{seq_{n+1}^{nested}})^w$, then $\sigma' \cdot 1 \cdot 2h + 2p_{n+1} \in (\Sigma_{sem}^{seq_{n+1}^{nested}})^*$. □

APPENDIX C

HIPAA POLICY TRANSLATIONS

<table>
<thead>
<tr>
<th>pf</th>
<th>p2: HIPAA-role</th>
</tr>
</thead>
<tbody>
<tr>
<td>ref</td>
<td>164.502(a)(1)(ii)</td>
</tr>
</tbody>
</table>

Fig. 52. Paragraph 164.502(a)(1)(ii)
Fig. 53. Paragraph 164.506(a)

Fig. 54. Paragraph 164.506(b)

Fig. 55. Paragraph 164.506(c)(1)

Fig. 56. Paragraph 164.506(c)(2)

Fig. 57. Paragraph 164.506(c)(3)

Fig. 58. Paragraph 164.506(c)(5)

Fig. 59. Paragraph 164.506(c)
Fig. 60. Paragraph 164.506

Fig. 61. Paragraph 164.508(a)(2)
Fig. 75. Paragraph 164.512(b)(1)(ii)

Fig. 76. Paragraph 164.512(b)(1)(iii)

Fig. 77. Paragraph 164.512(b)(1)(iv)

Fig. 78. Paragraph 164.512(b)(1)(v)

Fig. 79. Paragraph 164.512(b)(1)

Fig. 80. Paragraph 164.512(c)(1)
Fig. 81. Paragraph 164.512(c)(2)

Fig. 82. Paragraph 164.512(c)

Fig. 83. Paragraph 164.512(d)(1)

Fig. 84. Paragraph 164.512(e)(1)(i)

Fig. 85. Paragraph 164.512(e)(1)(ii)
Fig. 86. Paragraph 164.512(e)(1)(iv)

Fig. 87. Paragraph 164.512(e)

Fig. 88. Paragraph 164.512(f)(1)(i)

Fig. 89. Paragraph 164.512(f)(1)(ii)
Fig. 90. Paragraph 164.512(f)(1)

Fig. 91. Paragraph 164.512(f)(2)

Fig. 92. Paragraph 164.512(f)(3)(1)

Fig. 93. Paragraph 164.512(f)(3)(ii)

Fig. 94. Paragraph 164.512(f)(3)
Fig. 95. Paragraph 164.512(f)(4)

Fig. 96. Paragraph 164.512(f)(5)

Fig. 97. Paragraph 164.512(f)(6)

Fig. 98. Paragraph 164.512(f)
Fig. 104. Paragraph 164.512(i)(1)(ii)

Fig. 105. Paragraph 164.512(i)(1)(iii)

Fig. 106. Paragraph 164.512(i)(1)

Fig. 107. Paragraph 164.512(j)(1)(i)
Fig. 117. Paragraph 164.512

Fig. 118. Paragraph 164.520(a)(1)

Fig. 119. Paragraph 164.520(a)(2)(i)
Fig. 120. Paragraph 164.520(a)(2)(ii)

Fig. 121. Paragraph 164.520(a)(2)(iii)

Fig. 122. Paragraph 164.520(a)(2)

Fig. 123. Paragraph 164.520(a)(3)

Fig. 124. Paragraph 164.520(c)(1)(i)

Fig. 125. Paragraph 164.520(c)(1)(ii)
Fig. 126. Paragraph 164.520(c)(1)(iii)

Fig. 127. Paragraph 164.520(c)(1)(iv)

Fig. 128. Paragraph 164.520(c)(2)(i)

Fig. 129. Paragraph 164.520(c)(2)(ii)

Fig. 130. Paragraph 164.522(a)(1)(i)

Fig. 131. Paragraph 164.522(a)(1)(ii)
Fig. 140. Paragraph 164.522(b)(2)(i)

Fig. 141. Paragraph 164.522(b)(2)(iii)

Fig. 142. Paragraph 164.524(b)(2)(i)

Fig. 143. Paragraph 164.524(b)(2)(ii)

Fig. 144. Paragraph 164.526(a)(1)

Fig. 145. Paragraph 164.526(a)(2)(i)

Fig. 146. Paragraph 164.526(a)(2)(ii)

Fig. 147. Paragraph 164.526(a)(2)(iii)

Fig. 148. Paragraph 164.526(a)(2)(iv)
Fig. 158. Paragraph 164.526(d)(5)(ii)

Fig. 159. Paragraph 164.526(d)(5)(iii)

Fig. 160. Paragraph 164.526(e)

Fig. 161. Paragraph 164.528(a)(1)

Fig. 162. Paragraph 164.528(a)(2)(i)

Fig. 163. Paragraph 164.528(a)(2)(ii)

Fig. 164. Paragraph 164.528(a)(3)
Fig. 165. Paragraph 164.528(c)(1)(i)

Fig. 166. Paragraph 164.528(c)(1)(ii)

Fig. 167. Paragraph 164.528(c)(2)

ACKNOWLEDGMENTS

The authors would like to thank...The authors would like to thank...The authors would like to thank...