Sequence Diagrams Aided Security Policy Specification

Hui Shen, Ram Krishnan, Rocky Slavin, and Jianwei Niu

Abstract—A fundamental problem in the specification of regulatory privacy policies such as the Health Insurance Portability and Accountability Act (HIPAA) in a computer system 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
UML 2 Sequence Diagrams do not support Diagram’s expressiveness. Prior work in formalizing as interleaving and branching, for presenting comments permit different types of control flow, such as interleaving and branching, for presenting comments and Interaction Use, allowing multiple, 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., $m_1$ is a Message sent from Lifeline $L_1$ to Lifeline $L_2$ 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 examples 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 Weak Sequencing.
The semantics of each CF Operator determines the execution order of all the Operands. Each Operator has its specific semantic implications regarding the execution of the OSs enclosed by the CF on the covered Lifelines as described in the next subsection.

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 Constructs

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 \( \langle L, MSG, FG \rangle \), 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 \( CF_1, CF_2, \ldots, CF_m \).

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 \( \langle \text{message name}, \text{source Lifeline}, \text{target Lifeline} \rangle \). 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. 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 \( \langle \text{name}, \text{mform}, \text{OS}_i, \text{OS}_j \rangle \), where \( \text{name} \) is the Message name, \( \text{mform} \) denotes it is either a synchronous or an asynchronous Message, \( \text{OS}_i \) denotes its sending OS and \( \text{OS}_j \) denotes its receiving OS. Each OS has the form \( \langle l_i, \text{loc}_{l_i}, \text{type} \rangle \), where \( l_i \) denotes its associated Lifeline, \( \text{loc}_{l_i} \) is the location where the OS takes places on Lifeline \( l_i \), and \( \text{type} \) denotes it is either a sending or a receiving OS.

Each Lifeline \( l_i \in L \) has a set of finite locations \( LOC(l_i) \subseteq \mathbb{N} \) on it. The locations form a finite sequence \( 1, 2, 3, \ldots, k, k \in \mathbb{N} \). 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 \( \text{OSS}(l_i) \) is a one-to-one correspondence. Function \( \text{OSS}(l_i) \) 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 \( \langle m_1, \text{async}, s_1, r_1 \rangle \), and OS \( s_1 \) is expressed by \( \langle l_1, 1, \text{send} \rangle \), where \( l_1 \) represents a participating object of class \( L_1 \).
Definition 3. A CF $CF_m$ has the form $\langle L, oper, OP \rangle$. $L$ 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., $op_n \preceq_1, op_n \preceq_2, \ldots, op_n \preceq_n$. Each $op_n \in OP$ has the form $\langle L, MSG, FG, cond \rangle$, 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 $\langle \{l_1, l_2, l_3\}, \{msg_1, msg_7\}, \{CF_1\} \rangle$, 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, $msg_1, msg_7$, and the set of CFs directly enclose in $seq$ contains one CF, $CF_1$, $msg_1, CF_1$, and $msg_7$ are combined using Weak Sequencing. $CF_1$ is represented by $\langle \{l_1, l_2, l_3\}, \text{par}, \{op_1, op_2\} \rangle$, where $l_1, l_2, l_3$ are Lifelines enclosed by $CF_1$, $\text{par}$ is the Interaction Operator of $CF_1$, and $op_1, op_2$ 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 $op_2$ is decided by Interaction Operator $\text{par}$. If both Constraints of $op_1$ and $op_2$ evaluate to False, $CF_1$ is excluded and Messages $msg_1$ and $msg_7$ are ordered by Weak Sequencing. Operand $op_1$ expresses the Messages and CFs directly enclosed in it, represented by $\langle \{l_1, l_2, l_3\}, \{msg_2\}, \{CF_2\}, \text{cond_1} \rangle$, where $\text{cond_1}$ 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 $cf_m \uparrow_{\text{i}}$. (The shaded rectangle on Lifeline $L_1$ in figure 2 shows an example).

Definition 4. A CEU is given by a three tuple $\langle l_i, oper, setEU \rangle$, 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 $op_n \uparrow_{\text{i}}$. 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, $\langle Eu, \text{cond} \rangle$, in which $Eu$ 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 $\text{cond}$ is the Interaction Constraint of the Operand. $\text{cond}$ is True when there is no Interaction Constraint.

Definition 6. An HEU is given by $\langle \text{setCEU}, \text{setBEU}, \text{cond} \rangle$, 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, $\text{cond}$. (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
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 EUs compose a CEU. An HEU contains one or more CEUs.

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 an Operand within each CF are described at the next level. A Sequence Diagram can be considered as an Operand whose Constraint evaluates to True. Therefore, the semantics of each Operand containing other CFs can be described in the same way with that of a Sequence Diagram with nested CFs. An Operand 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 $\sigma[1..n] = \sigma_1\sigma_2...\sigma_n$. 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 $\sigma = \sigma_1\sigma_2...\sigma_n,...$

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 $\sigma_n$ of each trace [18]. Then, LTL formulas can express these traces, each of which is denoted as $\Sigma^*_\text{seq}\tau^\omega$, where $\Sigma^*_\text{seq}$ is the set of OSs of $\text{seq}$ and $\Sigma^*_\text{seq}$ represents a finite sequence of OSs of $\text{seq}$. $\tau$ is a no-op, i.e., $\tau$ is an empty event occurrence and not observable, and $\tau^\omega$ 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.
We include a summary of temporal operators that are sufficient to understand our LTL template. $\square p$ means that formula $p$ will continuously hold in all future states. $\diamond p$ means that formula $p$ holds in some future state. $\lozenge p$ means formula $p$ holds in the next state. $\bigcirc p$ means that formula $p$ holds in the previous state. $\lozenge p$ means that formula $p$ holds in some past state. $\diamond p \equiv \bigcirc \lozenge p$ means that formula $p$ holds in some past state, excluding current state. $p \bigcirc q$ means that formula $p$ holds until some future state where $q$ becomes true, and $p$ can be either $\text{True}$ or $\text{False}$ at that state. The macro $p \bigcup q \equiv p \bigcup (q \wedge p)$ states that in the state when $q$ becomes $\text{True}$, $p$ stays $\text{True}$.

## 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 $\text{True}$ or which are contained in the constructs whose Constraints evaluate to $\text{True}$. For instance, for Parallel $CF$ in figure 1b, function $\text{nested}(CF)$ returns a singleton set containing Critical Region $CF$ if the Constraint of the first Operand of $CF$ evaluates to $\text{True}$. Otherwise, $\text{nested}(CF)$ returns an empty set, and Critical Region $CF$ 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 CEU, or a Sequence Diagram, and $q$ can be $p$, an EU, or a CEU.

### Table 1: Auxiliary functions

<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>$\text{SND}(j)$</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>$\text{TOE}(u)$</td>
<td>return the Interaction Operator of CF $u$.</td>
</tr>
<tr>
<td>$\text{TOP}(u)$</td>
<td>return the set of Interaction Operands whose Constraints evaluate to $\text{True}$ within CF $u$, i.e., ${\text{op}</td>
</tr>
<tr>
<td>$\text{nested}(u)$</td>
<td>return the set of CFs, which are directly enclosed in CF $u$’s Interaction Operands whose Constraints evaluate to $\text{True}$. It can be overloaded to an Interaction Operator or a Sequence Diagram.</td>
</tr>
<tr>
<td>$\text{TBEU}(u)$</td>
<td>for CEU or EU $u$, return a set of directly enclosed BEUs whose Constraints evaluate to $\text{True}$, i.e., ${\text{beu}</td>
</tr>
<tr>
<td>$\text{AOS}(q)$</td>
<td>return the set of OSs which are enabled (i.e., the Constraints associated with it evaluate to $\text{True}$) and chosen to execute in $q$.</td>
</tr>
<tr>
<td>$\text{TOS}(u)$</td>
<td>return the set of OSs of the BEUs directly enclosed in CEU or EU $u$ whose Constraints evaluates to $\text{True}$, i.e., ${\text{os}</td>
</tr>
<tr>
<td>$\text{pre}(u)$</td>
<td>return the set of OSs which may happen right before CEU $u$. The set contains an OS if a BEU whose Constraint evaluates to $\text{True}$ prior to $u$ on the same Lifeline. If a CEU executes prior to $u$ on the same Lifeline, the set may contain a single or multiple OSs depending on the CEU’s Operator and nested CEUs (if there are any nested CEUs). If an HEU executes prior to $u$ on the same Lifeline, the set is determined by the last CEU or BEU nested within the HEU. 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>
</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 seq defined in section 2.1 are codified separately using formulas $\alpha_g$, $\beta_j$, and $\delta_{\text{seq}}$. $\alpha_g$ focuses on the intra-lifeline behavior to enforce rules 1 and 2. Recall that when projecting basic Sequence Diagram seq onto its covered Lifelines, $\text{LN(seq)}$, we obtain BEU $g$ for each Lifeline $i$, denoted as $\text{seq}^i$. Each BEU $g$ contains a trace of OSs, $\sigma[r..(r+}
For a given basic Sequence Diagram, \( \text{seq} \), with \( j \) Messages and \( 2j \) event occurrences (each Message has a sending event occurrence and a receiving event occurrence), \( \Sigma_{\text{seq}}^{\text{sem}} \) is the set of event occurrences of \( \text{seq}. \) \( \Sigma_{\text{seq}}^{\text{sem}} \subseteq \Sigma \), where \( \Sigma \) is the universe of event occurrences. The set of valid traces, \( (\Sigma_{\text{seq}}^{\text{sem}})^* \), contains finite traces derived from \( \text{seq} \) based on the semantic rules of Sequence Diagrams. \( \Sigma_{\text{seq}}^{\text{LTL}} \) is the set of event occurrences of LTL representation of \( \text{seq}. \) \( \Pi_{\text{seq}}^{\text{LTL}} \), where \( \Sigma_{\text{LTL}}^{\text{seq}} = \Sigma_{\text{seq}}^{\text{sem}} \cup \{\tau\}. \) \( \tau \) is an invisible event occurrence which does not occur in \( \text{seq} \), i.e., \( \tau \in (\Sigma \setminus \Sigma_{\text{seq}}^{\text{sem}}). \) \( (\Sigma_{\text{seq}}^{\text{LTL}})^\omega \) represents all infinite traces that satisfy \( \Pi_{\text{Basic}}^{\text{seq}}. \) For each trace \( \sigma \in (\Sigma_{\text{seq}}^{\text{LTL}})^\omega \), function \( \text{pre}_{i}(\sigma) \) returns the prefix of length \( i \) of trace \( \sigma \), i.e., \( \sigma[1..i] \). We lift function \( \text{pre}_{i}(\sigma) \) to \( \text{PRE}_{i}((\Sigma_{\text{seq}}^{\text{LTL}})^\omega) \) to apply to a set of traces. Function \( \text{PRE}_{i}((\Sigma_{\text{seq}}^{\text{LTL}})^\omega) \) returns the set of the prefixes of the traces within \( (\Sigma_{\text{seq}}^{\text{LTL}})^\omega \), where the length of each prefix must be \( i \), i.e., \( \text{PRE}_{i}((\Sigma_{\text{seq}}^{\text{LTL}})^\omega) = \{\text{pre}_{i}(\sigma) | \sigma \in (\Sigma_{\text{seq}}^{\text{LTL}})^\omega\} \).

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

**Proof:** If \( \sigma \models \Pi_{\text{seq}}^{\text{Basic}} \), then \( \sigma \models \varepsilon_{\text{seq}} \). We can directly infer from sub-formula \( \varepsilon_{\text{seq}} \) that, in \( \sigma \), only one OS of \( \text{seq} \) can execute at a time, and \( \sigma \) should execute uninterruptedly until all the OSs of \( \text{seq} \) have taken place. Similarly, we can infer from the assumption that \( \sigma \models \bigwedge_{j \in \text{MSG}(\text{seq})} \rho_{j} \). From sub-formula \( \bigwedge_{j \in \text{MSG}(\text{seq})} \rho_{j} \), we can infer that each OS within \( \text{seq} \) can execute once and only once in \( \sigma \). \( \text{seq} \) contains \( j \) Messages with \( 2j \) 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, \( \text{seq} \), \( \forall \nu. \nu \in (\Sigma_{\text{seq}}^{\text{LTL}})^\omega \), \( \nu \) is extended to \( \nu \cdot \tau^\omega \) without changing the meaning of \( \text{seq} \).

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

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

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

We provide the proof of theorem 1 in appendix A.
Fig. 4. LTL templates for basic Sequence Diagram

\[
\Pi_{\text{seq}}^{\text{Basic}} = ( \bigwedge_{i \in LN(\text{seq})} \alpha_i ) \land ( \bigwedge_{j \in MSG(\text{seq})} \beta_j ) \land \varepsilon_{\text{seq}}
\]

\[
\alpha_i = ( \bigwedge_{k \in [r..+|AOS(g)|-1]} (\neg OS_{k+1} U OS_k) ) \land \bigwedge_{OS_e \in AOS(g)} (\neg OS_e U (OS_e \land \Box \neg OS_e))
\]

\[
\beta_j = \neg RCV(j) \ U SND(j)
\]

\[
\varepsilon_{\text{seq}} = \Box ( \bigvee_{OS_m \in AOS(\text{seq})} OS_m ) \lor ( \bigvee_{OS_m \in AOS(\text{seq})} (\Diamond OS_m) )
\]

Fig. 5. Rewriting LTL templates for basic Sequence Diagram

\[
\bigwedge_{i \in LN(\text{seq})} \alpha_i = \bigwedge_{i \in LN(\text{seq})} \left( \bigwedge_{g=seq_1} (\neg OS_{k+1} U OS_k) \land \bigwedge_{OS_c \in AOS(g)} (\neg OS_c U (OS_c \land \Box \neg OS_c)) \right)
\]

\[
= \bigwedge_{i \in LN(\text{seq})} \left( \bigwedge_{g=seq_1} (\neg OS_{k+1} U OS_k) \land \bigwedge_{OS_c \in AOS(g)} (\neg OS_c U (OS_c \land \Box \neg OS_c)) \right)
\]

\[
= \left( \bigwedge_{i \in LN(\text{seq})} \alpha_i \land ( \bigwedge_{j \in MSG(\text{seq})} (\neg SND(j) U (SND(j) \land \Box \neg SND(j))) \right)
\]

\[
\land (\neg RCV(j) U (RCV(j) \land \Box \neg RCV(j)))
\]

\[
= \left( \bigwedge_{i \in LN(\text{seq})} \alpha_i \land ( \bigwedge_{j \in MSG(\text{seq})} \beta_j ) \right)
\]

Fig. 6. Rewriting \( \Pi_{\text{seq}}^{\text{Basic}} \) into \( \Pi_{\text{seq}}^{\text{Basic}} \)

### 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 make a Sequence Diagram more expressive, that can be nested within each other. Though these execution of its enclosed OSs on each Lifeline. More-

More-

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 traces of CFs. To capture these features, we generalize \( \Pi_{\text{seq}}^{\text{Basic}} \) to \( \Pi_{\text{seq}} \) for expressing Sequence Diagram with CFs (see figure 7). We introduce a new template \( \Phi_{CF} \) to assert the semantics of each CF directly enclosed in \( \text{seq} \). Template \( \Pi_{\text{seq}} \) is a conjunction of the formulas \( \alpha_{CF}, \beta_{CF}, \varepsilon_{CF} \) and 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 \( \text{seq} \) with CFs onto Lifelines to obtain a collection of CEUs and EUs, facilitating us to focus on OSs on each single Lifeline. The OSs directly enclosed in \( \text{seq} \) 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 \( \text{seq} \) are enforced by a conjunction of \( \beta_j \) for each Message \( j \). \( \varepsilon_{\text{seq}} \) enforces that at most one OS can execute at a time for all the OSs within \( \text{seq} \). One way to implement these formulas is provided in Appendix B. If \( \text{seq} \) contains a Loop, the OSs of \( \text{seq} \) 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
Fig. 7. LTL templates for Sequence Diagram with Combined Fragments

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 \( \eta^{CF}_i \). 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 \( \eta^{CF}_i \) 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 \( \Psi^{CF}_i \) defines the semantics of OSs directly enclosed in CF. The second conjunct states that the semantics of each CF, which is directly enclosed in CF, is enforced by each \( \Phi^{CF}_i \). In this way, \( \Phi^{CF}_i \) 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. \( \theta^{CF}_i \) states semantic rule 2, which defines the order among OSs directly enclosed in CF. \( \theta^{CF}_i \) is a conjunction of \( \alpha_g^i \)s and \( \beta_j^i \)s. The \( \alpha_g^i \)s is a conjunction of all \( \alpha_g \) of each Lifeline, where \( g \) is a BEU whose Constraint evaluates to True. The \( \beta_j \)s 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, \( \Psi^{CF}_i \) defines the semantics of OSs and CFs directly enclosed in CF. \( \Psi^{CF}_{alt} \) and \( \Phi^{CF}_i \), for CF, 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 \( \Psi^{CF}_i \) 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 r5 (i.e., the receiving OS of Message m5) and OS r6 on Lifeline L2 maintain their order. OS r2 and OS s5 on Lifeline L1 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,

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

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,

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

Figure 10 is an example of Option. The OSs within the Option execute if \( cond1 \) evaluates to True. Otherwise, the Option is excluded, and its semantics is defined by formula \( \eta \), i.e., Messages m1 and m4 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 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 \( \Psi^{CF}_{alt} \) in figure 12 defines the semantics of Alternatives, which is a conjunction of \( \Phi^{CF}_i \). Each \( \Phi^{CF}_i \) represents the semantics of Operand \( m \), whose
Constraint evaluates to True, which is achieved by function TOP(CF).

The semantics of the chosen Operand (if clause) is described by $\theta^C_F$, $\gamma^C_F$, and $\Phi^C_F$, where $\theta^C_F$ defines the partial order of OSs within the chosen Operand and $\Phi^C_F$ defines the semantics of CFs directly enclosed in the chosen Operand. Functions $\Psi^{alt}_m$ and $\Phi^{CF}_m$ 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 $\gamma^{CF}_i$ instead of $\gamma^C_F$, 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 $exe$ for each Operand whose Interaction Constraint evaluates to True. The variable $exe$ should satisfy the following assertion,

$$\forall i \in \left[1..m\right] \left( exe_i \land \bigwedge_{i \in \left[1..m\right]} \left( exe_i \rightarrow \text{cond}_i \right) \right)$$

The first conjunct expresses that only one $exe$ sets to True, i.e., exactly one Operand is chosen. The second conjunct enforces that the Interaction Constraint of Operand whose $exe$ 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 $exe$ 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 one Operand can be chosen to execute. Thus, the LTL representation of Break can be represented as the LTL formula for Alternatives with two Operands.

\[
\psi_{alt}^{CF} = \bigwedge_{m \in \text{TOP}(CF)} \psi_{alt}^m \\
\psi_{alt}^m = \begin{cases} 
\theta_{m}^{CF} \land \bigwedge_{i \in \text{LN}(CF)} \neg \bar{\theta}_{i,m}^{CF} \land \bigwedge_{\Phi_i \in \text{nested}(m)} \Phi_i & \text{if } m \text{ is the chosenOperand} \\
\text{True} & \text{else}
\end{cases}
\]

\[
\bar{\theta}_{m}^{CF} = \bigwedge_{i \in \text{LN}(m)} \bigwedge_{y \in \text{BEU}(m_{\uparrow}^{\text{TOP}}(i))} \alpha_{y} \land \bigwedge_{j \in \text{MSG}(TOP(m))} \beta_j
\]

\[
\gamma_{i,m}^{CF} = \bigwedge_{OS \in \text{OS}(m_{\uparrow}^{\text{TOP}})} \left(\neg \bar{\theta} \left(\bigwedge_{OS_{\text{pre}} \in \text{pre}(CF_{\uparrow}^{\text{TOP}})}(\bigwedge_{OS_{\text{post}} \in \text{post}(CF_{\uparrow}^{\text{TOP}})}(\neg OS_{\text{post}}) \tilde{U} (\bigwedge_{OS_{\text{pre}}}(\bigwedge_{OS_{\text{post}}}(\neg OS_{\text{post}})))\right)ight)
\]

Fig. 12. LTL formula for Alternatives

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 \(\psi_{critical}^{CF}\) presents the semantics for Critical Region (see figure 15). \(\theta^{CF}\) and \(\gamma^{CF}\) have their usual meanings. \(\delta_{M_i, M_j}\) 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 \(i\).

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.

Fig. 14. Representing Break using Alternatives

Fig. 15. Example for Loop

Each OS is an instance of an event, which is unique within a Sequence Diagram. To keep each OS within different iterations of a Loop unique, one way to implement an OS is defining an array to rename the OS of each iteration. We introduce \(R_i\), representing the
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, \( \text{Loop}[1] \) Loop \( [2] \) and Loop \( [3] \). Figure 17 shows an LTL formula for a Loop. \( \delta_{R} \) overloads \( \theta_{CF} \), which asserts the order of OSs during each iteration. \( \gamma_{i,R} \) 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 \( OS_{k} \) 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, \( OS_{i} \) or \( OS_{i+1} \) can not happen until \( OS_{k} \) 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 \( \Psi_{CF} = \theta_{CF} \) 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 \( \Psi_{assert} \) in figure 20. \( \theta_{CF} \) and \( \gamma_{i,CF} \) have their usual meanings. Function \( \lambda_{\text{pre}(\text{CF}_{1}), AOS(CF_{1})} \) represents that on Lifeline \( i \), if all the OSs in the set of \( \text{pre} \) happen, 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
Fig. 17. LTL formula for fixed Loop

\[
\psi_{\text{loop,R}}^{CF} := \theta_R \land \bigwedge_{i \in LN(CF)} \gamma_i \land \bigwedge_{i \in LN(CF)} \kappa_i \land \bigwedge_{n \in [1..R]} \Phi_{\text{CF}[n]}
\]

\[
\theta_R = \bigwedge_{i \in LN(CF)} ( \bigwedge_{g \in TBEU(CF[i])} \alpha_i \land \bigwedge_{j \in MSG(TOP(k))} \beta_j)
\]

\[
\alpha_i = \bigwedge_{k \in [r..r+1]} \bigwedge_{OS \in (CF[i])} ((\neg OS_k[n]) \land \bigwedge_{OS \in (CF[1..r])} OS_i \land \bigwedge_{n \in [1..R]} \neg OS_c[n])
\]

\[
\beta_j = \bigwedge_{n \in [1..R]} ((\neg RCV(j)[n]) \land \bigwedge_{OS \in (CF[1..r])} OS_j \land \bigwedge_{OS \in (CF[1..r])} OS_j \land \bigwedge_{n \in [1..R]} \neg OS_c[n])
\]

\[
\gamma_i = \bigwedge_{OS \in (CF[1..r])} (\neg OS \land \bigwedge_{OS \in (CF[1..r])} (\neg OS_{pre}) \land \bigwedge_{OS \in (CF[1..r])} OS_{post})
\]

\[
\kappa_i = \bigwedge_{OS \in (CF[1..r])} (\neg OS \land \bigwedge_{OS \in (CF[1..r])} OS_{pre} \land \bigwedge_{OS \in (CF[1..r])} OS_{post})
\]

Fig. 20. LTL formula for Assertion

\[
\psi_{\text{assert,CF}}^{CF} = \theta^{CF} \land \bigwedge_{i \in LN(CF)} \gamma_i \land \bigwedge_{i \in LN(CF)} \lambda_{(\pre{CF[i]},\text{AOS}(CF[i]))}^{i,seq}
\]

\[
\lambda_{N_1,N_2}^{i,seq} = \Box (\bigwedge_{OS \in N_1} (\neg OS_p) \rightarrow ((\bigwedge_{OS \in \text{AOS}(seq_{1}),N_2} \neg OS_q) \land \bigwedge_{OS \in N_2} (\neg OS)))
\]

Fig. 21. Example for Weak Sequencing

happen in any order as they are on different Lifelines. The LTL definition of Weak Sequencing is given as below.

\[
\psi_{\text{weak}}^{CF} = \theta^{CF} \land \bigwedge_{i \in LN(CF)} \gamma_i \land \bigwedge_{i \in LN(CF)} \bigwedge_{m \in \text{TOP}(CF)} \gamma_i^m
\]

Templates \(\theta^{CF}\) and \(\gamma_i^{CF}\) have their usual meanings. \(\gamma_i^m\) 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 an Operand, 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.

Fig. 22. Example for Strict Sequencing

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 \( preEU(u) \) 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 \( NFTOP(CF) \) 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](image)

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. \( \gamma_{\text{Seq}}^{CF} \) 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.

\[
\psi_{\text{Coregion}}^{CF} = \gamma_{\text{Seq}}^{CF}
\]

### 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

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 \( \gamma_{\text{Seq}}^{CF} \) 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 \( \delta_{\text{Ignore}(CF)} \). We introduce function \( ignoreMsg(CF) \) to return the set of Messages of the ignored message types which occur in CF, which can be finite or infinite. Function \( ignoreOS(CF) \) returns the set of OSs associated with Messages of ignored message types, which can also be finite or infinite. Formula \( \beta_k \) enforces that, for each ignored Message \( k \), its sending OS must happen before its receiving OS. Formula \( \gamma_{\text{Ignore}(CF)}^{CF} \) extends \( \gamma_{\text{Seq}}^{CF} \), which enforces any OS of the set of the ignored OSs can only happen within the CEU of the Ignore on each Lifeline, formally,

\[
\gamma_{\text{Ignore}(CF)}^{CF} = \bigwedge_{OS \in S} ((-OS \tilde{u} \bigwedge_{OS_{pre} \in pre(CF)} (\otimes OS_{pre})) \bigwedge ((-OS_{post}) \tilde{u} (\otimes OS)))
\]

where \( S \) can be replaced using \( ignoreOS(CF) \). 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,
\[ \psi_{\text{strict}} = \phi_{\text{CF}} \land (\forall i \in \text{LN}(\text{CF}) \, \bigwedge_{S \in \text{AOS}(i)} \gamma_{i}^{\text{CF}} \land (\exists k \in \text{NFTOP}(\text{CF}) \, \chi_{k}) \]

\[ \chi_{k} = ((\bigcap_{S \in \text{AOS}(k)} (\neg \text{OS})) \bigcup (\bigcup_{i \in \text{LN}(\text{CF})} \text{OS}_{\text{pre}} \in \text{preEU}(\text{CF}_{i})) \]

Fig. 23. LTL formula for Strict Sequencing

\[ \varepsilon_{\text{seq.ignoreOS}(\text{CF})} = \square((\bigvee_{S \in \text{AOS}(\text{seq}) \cup \text{ignoreOS}(\text{CF})} \text{OS}_{p}) \bigvee (\bigcup_{S \in \text{AOS}(\text{seq}) \cup \text{ignoreOS}(\text{CF})} (\bigcirc \text{OS}_{p})) \]

Thus, function \( \varepsilon_{\text{seq}} \) 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 \( \text{AllMsg}(\text{CF}) \setminus \text{considerMsg}(\text{CF}) \) returns the Messages which are not considered but occur in CF, where \( \text{AllMsg}(\text{CF}) \) returns all possible Messages, including Messages of considered types and Messages of ignored types. Function \( \text{considerMsg}(\text{CF}) \) returns the Messages of considered types. Function \( \Sigma \setminus \text{considerOS}(\text{CF}) \) 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 \( \text{considerOS}(\text{CF}) \) 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 \( \psi_{\text{neg}}^{S} \) instantiates the template \( \psi_{\text{neg}}^{S} \) (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 \( \text{cond1} \) evaluates to True.

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 True), the traces of \( R \) are also inconclusive traces. For Sequence Diagram \( R \), its traces are defined by formula \( \Pi_{R} \), which instantiates the template \( \Pi_{\text{seq}} \) (see figure 7). For non-Negative CF \( R \), its traces are defined by formula \( \Phi^{R} \), which instantiates the template \( \Phi^{\text{CF}} \) (see figure 8). For example, traces \([s1, s2, r2, r1, s3, r3] \) in figure 29 is interpreted as an invalid or an inconclusive trace.

![Example for variation of Negative Combined Fragment](image)

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.
Figure 30 shows an example of nested Negative CFs. All the traces \([s1, s2, r1, r2], [s2, s1, r1, r2],\) and \([s1, r1, s2, r2]\) of \(R\) can be valid, invalid, or inconclusive traces depending on the value of \(cond1\) and \(cond2\).

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 \(\text{False}\): (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 \(\text{False}\) can be defined in the same way as the semantics of Negative CF (see subsection 5.5.1).

5.6 Deadlock Property with Synchronous Messages

A deadlock can occur within a Sequence Diagram with synchronous Messages, where each synchronous Message must have an explicit reply Message (see example in figure 32). We want to detect if multiple Lifelines are blocked, waiting on each other for a reply.

We formalize the semantics of Sequence Diagram with synchronous Messages, shown in figure 33.
We add a constraint $\xi_{seq}$ to the LTL formula for a Sequence Diagram with asynchronous Messages to express that if a Lifeline sends a synchronous Message, it can not send or receive any other synchronous Message until it receives a reply Message. We define some helper functions, where $type(OS_{SP})$ returns that $OS_{P}$ is a sending OS or a receiving OS, and $Reply(OS_{P})$ returns the reply Message of a synchronous Message containing $OS_{P}$. Constraint $\xi_{seq}$ can be used as a property to check if a Sequence Diagram is deadlock, in the sense that if all the reply Messages are received, i.e., the receiving OSs of the reply Messages have executed, no Lifeline is blocked and deadlock does not happen in the Sequence Diagram. In the example of figure 32, all of the Lifelines eventually deadlock since they all send Messages and are all awaiting replies.

5.7 Other Control Constructs

5.7.1 General Ordering

General Ordering imposes order of two unordered 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.

$$T^{GO} = \neg OS_{Q} \Diamond OS_{P}$$

5.7.2 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 $ref$. Formula $\Psi_{CF}^{ref}$ represents the LTL representation of an Interaction Use. In $\Psi_{CF}^{ref}$, the first conjunct describes that the OSs directly enclosed in the referred Sequence Diagram obeys their order. The second conjunct enforces that the referred Sequence Diagram and its adjacent OSs are ordered by Weak Sequencing, which is represented by $\psi_{CF}^{ref}$.

$$\Psi_{CF}^{ref} = \theta_{CF}^{ref} \land \bigwedge_{i \in LN(CF)} \psi_{CF}^{ref}$$

Fig. 31. LTL formula for variation of Strict Sequencing

Fig. 34. Example for Overlapped Combined Fragments

5.8 Discussion

This paper does not address timed events, i.e., the events can not 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 can not overlap each other, i.e., the inner CF can not cover more Lifelines than the outer CF. The example in figure 34 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, $op1$, either OS s2 or OS s3 may be the first OS to execute. As the Interaction Constraint of $op1$, $cond1$, 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 35, 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 36). 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.

Fig. 35. Example for Combined Fragments with Interaction Constraints

\[ \Pi_{seq}^{\text{sync}} = \Pi_{seq} \land \xi_{seq}^{\text{sync}} \]
\[ \xi_{seq}^{\text{sync}} = \bigcap_{i \in \text{LN}(seq)} \left( \bigcap_{OS_p \in \text{AOS}(seq_i)} \bigcap_{\text{type}=\text{OS}(OS_p)=\text{send}} (OS_p \rightarrow ((\bigcap_{OS_q \in \text{AOS}(seq_i)} \neg \text{OS}_q) \cup \text{RCV}(\text{Reply}(OS_p)))) \right) \]

Fig. 33. LTL formula for Sequence Diagram with synchronous Messages

\[ \Pi_{seq}^{\text{sync}} = \Pi_{seq} \land \xi_{seq}^{\text{sync}} \]
\[ \xi_{seq}^{\text{sync}} = \bigcap_{i \in \text{LN}(seq)} \left( \bigcap_{OS_p \in \text{AOS}(seq_i)} \bigcap_{\text{type}=\text{OS}(OS_p)=\text{send}} (OS_p \rightarrow ((\bigcap_{OS_q \in \text{AOS}(seq_i)} \neg \text{OS}_q) \cup \text{RCV}(\text{Reply}(OS_p)))) \right) \]

5.9 Proof for LTL Template of Sequence Diagram with Combined Fragments

We wish to prove that the NuSMV model for a Sequence Diagram with CFs captures 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 \( \Pi_{seq}^{\text{basic}} \) (see figure 37) by replacing the sub-formula \( \bigcap_{i \in \text{LN}(seq)} \bigcap_{\text{BEU}(seq_i)} \alpha_g \) using sub-formulas \( \bigcap_{i \in \text{LN}(seq)} \bigcap_{\text{BEU}(seq_i)} \hat{\alpha}_g \) and \( \bigcap_{j \in \text{MSG}(seq)} \rho_j \). The procedure (see figure 38) follows the one of rewriting LTL template \( \Pi_{seq}^{\text{basic}} \). We can rewrite sub-formula \( \theta^{\text{CF}} \) into \( \hat{\theta}^{\text{CF}} \) (see figure 39) to describe the semantics of CF’s Operands whose Constraints evaluate to True (see figure 40). In sub-formula \( \theta^{\text{CF}} \), function \( \text{BEU}(CF \uparrow_i) \) returns the set of BEUs, whose Constraints evaluate to True, directly enclosed in the CEU of CF on Lifeline i. It is equivalent to the set of BEUs directly enclosed in the EUs, which are obtained by projecting CF’s Operands whose Constraints evaluate to True onto Lifeline i, i.e., \( \text{BEU}(CF \uparrow_i) = \{ \text{beu} | \text{beu} \in \text{BEU}(op \uparrow_i) \land \text{op} \in \text{TOP}(CF) \} \) (see line 1). Sub-formula \( \bigcap_{i \in \text{LN}(CF)} \bigcap_{\text{BEU}(seq_i)} \alpha_g \) is rewritten as the one of rewriting \( \Pi_{seq}^{\text{basic}} \) (see line 4). We also rewrite sub-formula \( \gamma^{\text{CF}} \) into \( \hat{\gamma}^{\text{CF}} \) (see figure 39) to enforce the sequential execution Lifeline i (see figure 41). In sub-formula \( \gamma^{\text{CF}} \), function \( \text{TOS}(CF \uparrow_i) \) 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., \( \text{TOS}(CF \uparrow_i) = \{ \text{os} | \text{os} \in \text{AOS}(BEU(op \uparrow_i)) \land \text{op} \in \text{TOP}(CF) \} \).

**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
\[\Pi_{\text{seq}} = (\bigwedge_{i \in \text{LN(seq)}} (\bigwedge_{g \in \text{TBEU(seq)}} \alpha_g) \wedge (\bigwedge_{j \in \text{MSG(seq)}} \beta_j) \wedge (\bigwedge_{\text{CF} \in \text{nested(seq)}} \phi^{\text{CF}}) \wedge \varepsilon_{\text{seq}})\]

Fig. 37. Rewriting LTL templates for Sequence Diagram with Combined Fragments

\[\Pi_{\text{seq}} = \bigwedge_{i \in \text{LN(seq)}} (\bigwedge_{g \in \text{TBEU(seq)}} \alpha_g) \wedge (\bigwedge_{j \in \text{MSG(seq)}} \beta_j) \wedge (\bigwedge_{\text{CF} \in \text{nested(seq)}} \phi^{\text{CF}}) \wedge \varepsilon_{\text{seq}}\]

\[= (\bigwedge_{i \in \text{LN(seq)}} (\bigwedge_{g \in \text{TBEU(seq)}} \tilde{\alpha}_g) \wedge (\bigwedge_{j \in \text{MSG(seq)}} \beta_j) \wedge (\bigwedge_{\text{CF} \in \text{nested(seq)}} \phi^{\text{CF}}) \wedge \varepsilon_{\text{seq}})\]

\[= \tilde{\Pi}_{\text{seq}}\]

Fig. 38. Rewriting \(\Pi_{\text{seq}}\) into \(\tilde{\Pi}_{\text{seq}}\)

\[\psi^{\text{CF}} = \tilde{\theta}^{\text{CF}} \wedge \bigwedge_{i \in \text{LN(CF)}} \gamma_{\tilde{\delta}^{\text{CF}}} \wedge \delta^{\text{CF}}\]

\[\tilde{\theta}^{\text{CF}} = \bigwedge_{\text{op} \in \text{TOP(CF)}} \left( \left( \bigwedge_{i \in \text{LN(CF)}} \tilde{\alpha}_g \right) \wedge (\bigwedge_{j \in \text{MSG(op)}} \beta_j) \right)\]

\[\gamma_{\tilde{\delta}^{\text{CF}}} = \bigwedge_{\text{op} \in \text{TOP(CF)}} \left( \left( \bigwedge_{\text{beu} \in \text{ABEU(op)}} (\neg \text{OS} \tilde{u}) \wedge (\bigwedge_{\text{pre} \in \text{pre(CF)}} \text{OS} \pre \wedge (\bigwedge_{\text{post} \in \text{post(CF)}} \text{OS} \post)) \right) \wedge \left( \bigwedge_{\text{beu} \in \text{AOS(beu)}} (\neg \text{OS} \tilde{u}) \wedge (\bigwedge_{\text{pre} \in \text{pre(CF)}} \text{OS} \pre \wedge (\bigwedge_{\text{post} \in \text{post(CF)}} \text{OS} \post)) \right) \right)\]

Fig. 39. Rewriting LTL template for OSs directly enclosed in Combined Fragment

\[\theta^{\text{CF}} = (\bigwedge_{i \in \text{LN(CF)}} (\bigwedge_{g \in \text{TBEU(CF)}} (\bigwedge_{\text{op} \in \text{TOP(CF)}} \alpha_g) \wedge (\bigwedge_{j \in \text{MSG(TOP(CF))}} \beta_j))\]

\[= (\bigwedge_{i \in \text{LN(CF)}} (\bigwedge_{\text{op} \in \text{TOP(CF)}} \alpha_g) \wedge (\bigwedge_{j \in \text{MSG(TOP(CF))}} \beta_j))\]

(1)

\[= (\bigwedge_{\text{op} \in \text{TOP(CF)}} (\bigwedge_{i \in \text{LN(CF)}} \alpha_g) \wedge (\bigwedge_{j \in \text{MSG(TOP(CF))}} \beta_j))\]

(2)

\[= \tilde{\theta}^{\text{CF}}\]

Fig. 40. Rewriting \(\theta^{\text{CF}}\) into \(\tilde{\theta}^{\text{CF}}\)

\[\gamma_{\tilde{\delta}^{\text{CF}}} = \bigwedge_{\text{os} \in \text{TOS(CF)}} (\bigwedge_{\text{pre} \in \text{pre(CF)}} (\bigwedge_{\text{post} \in \text{post(CF)}} ((\neg \text{OS} \tilde{u}) \wedge (\bigwedge_{\text{pre} \in \text{pre(CF)}} (\bigwedge_{\text{post} \in \text{post(CF)}} \text{OS} \pre \wedge (\bigwedge_{\text{post} \in \text{post(CF)}} \text{OS} \post)) \wedge (\bigwedge_{\text{pre} \in \text{pre(CF)}} (\bigwedge_{\text{post} \in \text{post(CF)}} \text{OS} \pre \wedge (\bigwedge_{\text{post} \in \text{post(CF)}} \text{OS} \post)) \right))\)

\[= \gamma_{\tilde{\delta}^{\text{CF}}}\]

Fig. 41. Rewriting \(\gamma_{\tilde{\delta}^{\text{CF}}}\) into \(\tilde{\gamma}_{\tilde{\delta}^{\text{CF}}}\)
evaluate to False. If \( \sigma \in (\Sigma_{\text{seq}})^{LTL}_{\omega} \), then \( \sigma \) must have the form, i.e., \( \sigma = [1..2h+2p] \cdot \tau^\omega \), where \( [1..2h+2p] \) contains no \( \tau \).

**Proof:** If \( \sigma \models \tilde{\Pi}_{\text{seq}} \), then \( \sigma \models \bigwedge_{j \in MS(\sigma)} \rho_j \) and \( \sigma \models \bigwedge_{op \in \text{TOP}(CF)} \bigwedge_{j \in MS(\sigma)} \rho_j \). From sub-formula \( \bigwedge_{j \in MS(\sigma)} \rho_j \), 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 \( \bigwedge_{op \in \text{TOP}(CF)} \bigwedge_{j \in MS(\sigma)} \rho_j \) 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 \( \sigma \models \Pi_{\text{seq}} \), we can deduce that \( \sigma \models \varepsilon_{\text{seq}} \). 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 Os. 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, \( \sigma = [1..2h+2p] \cdot \tau^\omega \), where \( [1..2h+2p] \) contains no \( \tau \). 

A given Sequence Diagram, seq, 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 \( \forall \upsilon. \upsilon \in (\Sigma_{\text{sem}})^* \), \( \upsilon \cdot \tau^\omega \models \tilde{\Pi}_{\text{seq}} \), i.e., \( \upsilon \cdot \tau^\omega \models (\Sigma_{\text{seq}})^{LTL}_{\omega} \). 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, \( \forall \upsilon. \upsilon \in (\Sigma_{\text{sem}})^* \), \( |\upsilon| = 2h + 2p \). From lemma 2, we learn that \( \forall \upsilon. \upsilon \in (\Sigma_{\text{seq}})^{LTL}_{\omega} \), \( \upsilon \cdot \tau^\omega \models (\Sigma_{\text{seq}})^{LTL}_{\omega} \). For other \( \upsilon \), we can infer that \( \upsilon \models (\Sigma_{\text{seq}})^{LTL}_{\omega} \), i.e., \( (\Sigma_{\text{seq}})^{LTL}_{\omega} \subseteq (\Sigma_{\text{seq}})^{\omega} \).

We also wish to prove that \( \forall \upsilon. \upsilon \in (\Sigma_{\text{seq}})^{LTL}_{\omega} \), \( \upsilon \models (\Sigma_{\text{seq}})^{\omega} \), i.e., \( (\Sigma_{\text{seq}})^{\omega} \subseteq (\Sigma_{\text{seq}})^{LTL}_{\omega} \).

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 [31]. 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)”. Furthermore, 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 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 rule (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 to 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)(i) 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 p: coveredEntity, which represents that the role of actor p 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 us-
Fig. 42. Paragraph 164.512(j)(2)(i)

Fig. 43. Insurance Service Information System Safety Property

Fig. 44. Paragraph 164.524(b)(2)(i) – (ii)

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 44). 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 the specified amount of time has elapsed.

6.2 Sequence Diagrams for HIPAA

We illustrate our mapping strategies using two examples of safety properties and one example of a liveness property.

§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 42). 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 42.

Figure 43 describes a safety property present in an insurance software system between a web form and the information system’s server.

§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...
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 45 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 46 models the acceptable traces for disclosure of PHI with regard to both of these HIPAA regulations. The alternative combined fragment allows for two possibilities in response to the request for PHI from a HIPAA role: 1) if a previous authorization has not yet passed the expiration date and the authorization has not been revoked by the individual, Message 2, disclosure of the requested PHI can occur. 2) in any other case Messages 3, 4, and 5, verification of authorization, notification of an individual’s right to revoke authorization, and authorization by the individual must occur in that exact order before disclosure of PHI to the HIPAA-role can occur. Note that Messages 2 and 6 are the same with respect to possible traces.

To check for conformity of the hospital policy in figure 45 with the HIPAA rules in figure 46, we must show that figure 46’s traces contain all of the traces defined by figure 45 ignoring the extra OSs defined in 45 not appearing in 46 (e.g., startTimer and CheckTimer). Such OSs that are not related to the HIPAA policy are not considered as per the partial trace semantics described in section 4. A free model defining all of the OSs in both Sequence Diagrams must be generated by defining boolean variables for each OS. Then, conformance can be validated by checking if the LTL formula describing figure 45 implies the LTL formula describing figure 46. If the resulting formula returns true, this implies that all possible traces in the hospital policy (excluding irrelevant OSs) are a subset of the possible traces in the HIPAA rule, i.e., the hospital policy conforms to the HIPAA rule.

For our example, the fictional hospital policy would not conform to the HIPAA rule due to the possibility of the notification of an individual’s right to revoke authorization occurring after authorization. This is apparent in figure 45 due to the parallel 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 46, 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)
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 asynchronous 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.

Fig. 47. Paragraph 164.508(a)(3)

as seen in figure 47, 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.
tomata, which is used to model check the communication produced by UML state machines with SPIN or UPAL. 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 [32]. 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 [33]. 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

APPENDIX A

PROOF OF THEOREM 1

Theorem 1. For a given Sequence Diagram, seq, with j Messages, \((\Sigma_{seq})^s\) and \(PRE_2((\Sigma_{LTL})^w)\) are equal.

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

Base step. Basic Sequence Diagram seq1 contains only one Message, m1, (j = 1)

- Case 1. Sending OS s1, and receiving OS r1 of Message m1 locate on two Lifelines L1, L2 respectively (see figure 48).

\[
\Sigma_{seq1} = \{s_1, r_1\}
\]

where \(\Sigma_{seq1} \subseteq \Sigma\). The semantic aspects of seq1 define that, for \(m_1, r_1\) can only happen after s1. Only one trace, \(v = <s_1, r_1>\) of size 2, can be derived from seq1, i.e., \((\Sigma_{sem})^s = \{<s_1, r_1>\}\).

We wish to prove that \(<s_1, r_1> \cdot \tau^w = \tilde{\Pi}_{seq1}\), in which \(\tilde{\Pi}_{seq1}\) for seq1 is shown as below:

\[
\tilde{\Pi}_{seq1} = \tilde{\alpha}_{seq1 \downarrow L_1} \land \rho_{m_1} \land \beta_{m_1} \land \varepsilon_{seq1}
\]

\[
\rho_{m_1} = (-s_1 \tilde{u} (s_1 \land \Box \lnot s_1)) \land (-r_1 \tilde{u} (r_1 \land \Box \lnot r_1))
\]

\[
\beta_{m_1} = -r_1 \tilde{u} s_1
\]

\[
\varepsilon_{seq1} = \Box (((-s_1 \land r_1) \lor (s_1 \land \lnot r_1)) \lor ((\tilde{u} s_1) \land (\tilde{u} r_1)))
\]

Sub-formula \(\tilde{\alpha}_{seq1 \downarrow L_1}\) returns true because Lifeline \(L_1\) contains only one OS, \(s_1, <s_1, r_1> \cdot \tau^w\) satisfies sub-formula \(\rho_{m_1}\) 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 \(\varepsilon_{seq1}\) because only one OS happens at a time and \(<s_1, r_1> \cdot \tau^w\) executes uninterrupted. Thus, \(<s_1, r_1> \cdot \tau^w = \tilde{\Pi}_{seq1}\).

We wish to prove that \(\forall \sigma, \sigma \in \Sigma^w\), if \(\sigma \in (\Sigma_{LTL})^w\), then \(\sigma(1..., 2) \in (\Sigma_{sem})^s\).

\(\sigma\) 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 \(<r_1, s_1>\).

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 \((\Sigma_{sem})^s\).

In this way, we can prove \(\sigma(1..., 2) \in (\Sigma_{sem})^s\).

- Case 2. Sending OS s1, and receiving OS r1 of Message m1 locate on a single Lifeline L1 (see figure 49).

Besides the semantic aspects discussed in case 1, the OSs on L1 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 seq1, i.e., \((\Sigma_{sem})^s = \{<s_1, r_1>\}\). \(\Pi_{seq1}\) is reduced to \(\tilde{\Pi}_{seq1}\) for seq1 as below.

\[
\tilde{\Pi}_{seq1} = \tilde{\alpha}_{seq1 \downarrow L_1} \land \beta_{m_1} \land \rho_{m_1} \land \varepsilon_{seq1}
\]

\[
\rho_{m_1} = (-s_1 \tilde{u} (s_1 \land \Box \lnot s_1)) \land (-r_1 \tilde{u} (r_1 \land \Box \lnot r_1))
\]

\[
\beta_{m_1} = -r_1 \tilde{u} s_1
\]

\[
\varepsilon_{seq1} = \Box (((-s_1 \land r_1) \lor (s_1 \land \lnot r_1)) \lor ((\tilde{u} s_1) \land (\tilde{u} r_1)))
\]

Comparing to \(\tilde{\Pi}_{seq1}\) in case 1, only sub-formula \(\tilde{\alpha}_{seq1 \downarrow L_1}\) is changed. \(\tilde{\alpha}_{seq1 \downarrow L_1}\) represents that \(s_1\) happen before \(r_1\), which enforces the same order as sub-formula \(\beta_{m_1}\). \(\tilde{\alpha}_{seq1 \downarrow L_1}\) Trace \(<s_1, r_1> \cdot \tau^w\) can be generated from \(\tilde{\Pi}_{seq1}\), i.e., \((\Sigma_{LTL})^w = \{<s_1, r_1>\cdot \tau^w\}\).

Similarly, we wish to prove that \(\forall v, v \in \Sigma^w\), if \(v \in (\Sigma_{sem})^s\), then \(v \cdot \tau^w = \tilde{\Pi}_{seq1}\); and \(\forall \sigma, \sigma \in \Sigma^w\),
if $\sigma \in (\Sigma_{\text{seq}})^* \setminus \omega$, then $\sigma[1..2] \in (\Sigma_{\text{sem}})^*$. The proof follows the one of case 1.

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

Inductive step. Basic Sequence Diagram $\text{seq}_n$ contains $n$ Messages, which are graphically-ordered, i.e., $(m_1, \ldots, m_{n-1})$ locates above $m_i$ ($2 \leq i \leq k$)). The Messages have 2\n Os, which locate on $k$ Lifelines. We assume $\forall \nu \in \Sigma^*$, if $\nu \in (\Sigma_{\text{seq}})^*$, then $\nu \cdot \tau = \Pi_{\text{Basic}}$; and $\forall \sigma \in \Sigma^*$, if $\sigma \in (\Sigma_{\text{seq}})^\omega$, then $\sigma[1..2n] \in (\Sigma_{\text{sem}})^*$ ($j = n$).

We add a Message, $m_{n+1}$, at the bottom of $\text{seq}_n$ graphically to form a new Sequence Diagram, $\text{seq}_{n+1}$, with $n + 1$ Messages. We wish to prove $\forall \nu' \in \Sigma^*$, if $\nu' \in (\Sigma_{\text{seq}})^*$, then $\nu' \cdot \tau = \Pi_{\text{Basic}}$. Let $\text{seq}_{n+1}$ be a conjunction of $\text{seq}_n$ and $\text{seq}_{n+1}$ respectively (see figure 50). We group the sub-formulas of $\Pi_{\text{Basic}}$ using $\text{seq}_n \cdot \text{seq}_n \cdot \text{seq}_n \cdot \text{seq}_n \cdot \text{seq}_n$ and $\text{seq}_{n+1}$, $\sigma_{\text{seq}_{n+1}}$. In order to prove $\nu' \cdot \tau = \Pi_{\text{Basic}}$, $\text{seq}_{n+1}$, we wish to prove that $\nu' \cdot \tau$ satisfies all sub-formulas of $\Pi_{\text{Basic}}$. Sub-formula $\nu_{\text{seq}_{n+1}}$ enforces the order of OSs within $\text{seq}_{n+1}$, which includes the order of OSs along each Lifeline, and the order between OSs of each Message. We assume that $\nu \in (\Sigma_{\text{seq}})^*$, then $\nu \cdot \tau = \Pi_{\text{Basic}}$. It is easy to observe that $\nu \cdot \tau$ also satisfies $\nu_{\text{seq}_{n+1}}$. As we discussed, the order of OSs within $\text{seq}_{n+1}$ is still preserved in $\nu'$. Thus, $\nu' \cdot \tau$ satisfies $\nu_{\text{seq}_{n+1}}$. Sub-formula $\sigma_{\text{seq}_{n+1}}$ 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 $r_{n+1}$. $\nu' \cdot \tau$ satisfies $\sigma_{\text{seq}_{n+1}}$ because (1) only one $s_{n+1}$ and one $r_{n+1}$ are in $\nu'$, and (2) $s_{n+1}$ locates before $r_{n+1}$ in $\nu'$. Sub-formula $\sigma_{\text{seq}_{n+1}}$ enforces that only one OS of $\text{seq}_{n+1}$ can execute at once, and the trace should execute uninterrupted. As we discussed in $\nu' \cdot \tau$, each OS of $\text{seq}_{n+1}$ only executes once, and the execution of $\nu'$ does not interleave by $\tau$. Therefore, $\nu' \cdot \tau$ satisfies $\sigma_{\text{seq}_{n+1}} \cdot \text{seq}_{n+1} \cdot \text{seq}_{n+1} \cdot \text{seq}_{n+1} \cdot \text{seq}_{n+1}$ 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 51a); or two OSs of $m_{n+1}$ locate on one new Lifeline, $L_{k+1}$ (see figure 51b).

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 $\nu \in (\Sigma_{\text{seq}})^*$, $s_{n+1}$ or $r_{n+1}$ can locate (1) between any two OSs of $\text{seq}_n$, or (2) before all OSs of $\text{seq}_n$, or (3) after all OSs of $\text{seq}_n$. Thus, $s_{n+1}$ can be the $\text{th}$ OS of $\nu'$, where $1 \leq s \leq 2n + 1$; and $r_{n+1}$ can be the $\text{th}$ OS of $\nu'$, where $s < r \leq 2n + 2$.

$\nu_{\text{seq}_{n+1}} = \alpha_{\text{seq}_{L_{k+1}}} \land \alpha_{\text{seq}_{L_{k+2}}}$

Sub-formula $\nu_{\text{seq}_{n+1}}$ is a conjunction of $\alpha_{\text{seq}_{L_{k+1}}}$ and $\alpha_{\text{seq}_{L_{k+2}}}$. Only one OS locates on Lifeline $L_{k+1}$. Therefore, $\alpha_{\text{seq}_{L_{k+1}}}$ returns $true$ as defined by sub-formula $\alpha_{\text{seq}}$. Similarly, $\alpha_{\text{seq}_{L_{k+2}}}$ returns true. Thus, $\nu_{\text{seq}_{n+1}}$ returns true, $\nu' \cdot \tau$ satisfies $\nu_{\text{seq}_{n+1}}$, i.e., $\nu' \cdot \tau = \Pi_{\text{Basic}}$.

- **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 51c). In $\text{seq}_n$, we assume the last OS on $L_i$ is $\text{OS}_{\text{pn}}$. After adding $m_{n+1}$ at the bottom of $\text{seq}_n$, $r_{n+1}$ becomes the last OS on $L_i$. Therefore, $\text{OS}_{\text{pn}}$ should happen before $r_{n+1}$, $s_{n+1}$ locates on a new Lifeline, so it is interleaved with the OSs of $\text{seq}_n$. However, $s_{n+1}$ must happen before $r_{n+1}$. In trace $\nu' \in (\Sigma_{\text{seq}})^*$, if $\text{OS}_{\text{pn}}$ is the $\text{th}$ OS, where $1 \leq p \leq 2n + 1$. Then $s_{n+1}$ is the $\text{th}$ OS of $\nu'$, where $1 \leq s \leq 2n + 1$ and $s \neq p$; $r_{n+1}$ is the $\text{th}$ OS of $\nu'$, where $s < r \leq 2n + 2$.

$\nu_{\text{seq}_{n+1}} = (\neg s_{n+1} \cdot \bar{U} \text{ OS}_{\text{pn}}) \land \alpha_{\text{seq}_{L_{k+1}}}$

Sub-formula $\nu_{\text{seq}_{n+1}}$ defines that $r_{n+1}$ does not happen until $\text{OS}_{\text{pn}}$ does. $\alpha_{\text{seq}_{L_{k+1}}}$ returns true because only one OS locates on Lifeline $k + 1$. In $\nu' \cdot \tau$, $\text{OS}_{\text{pn}}$ locates before $\text{OS}_{n+1}$, i.e., $p < r$. Thus, $\nu' \cdot \tau$ satisfies $\nu_{\text{seq}_{n+1}}$, i.e., $\nu' \cdot \tau = \nu_{\text{seq}_{n+1}}$.

- **Case 3:** Sending OS $s_{n+1}$ locates on an existing Lifeline, $L_i$ ($1 \leq i \leq k$), and receiving OS $r_{n+1}$ locates on a new Lifeline, $L_{k+1}$ (see figure 51d); or two OSs of $m_{n+1}$ locate on an existing Lifeline $L_i$ ($1 \leq i \leq k$) (see figure 51e).

Similarly, we assume the last OS on $L_i$ in $\text{seq}_n$ is $\text{OS}_{\text{pn}}$. In $\text{seq}_{n+1}$, if $\text{seq}_{n+1}$ locates on $L_i$, $\text{OS}_{\text{pn}}$ should happen before $s_{n+1}$ because $\text{OS}_{\text{pn}}$ locates above $s_{n+1}$ graphically. For $m_{n+1}$, $r_{n+1}$ must happen after $s_{n+1}$. In trace $\nu' \in (\Sigma_{\text{seq}})^*$, if $\text{OS}_{\text{pn}}$ is the $\text{th}$ OS, where $1 \leq p \leq 2n$. Then $s_{n+1}$ is the $\text{th}$ OS of $\nu'$, where $p < s \leq 2n + 1$; $r_{n+1}$ is the $\text{th}$ OS of $\nu'$, where $s < r \leq 2n + 2$.

$\nu_{\text{seq}_{n+1}} = \neg (s_{n+1} \cdot \bar{U} \text{ OS}_{\text{pn}}) \land \alpha_{\text{seq}_{L_{k+1}}}$

Sub-formula $\nu_{\text{seq}_{n+1}}$ defines that $s_{n+1}$ cannot happen before $\text{OS}_{\text{pn}}$. Only one or none OS locates on Lifeline $k + 1$, so $\alpha_{\text{seq}_{L_{k+1}}}$ returns true.
In $v' \cdot \tau^w$, $s_{n+1}$ locates after $OS_{pre}$, i.e., $p < s$. Therefore, $v' \cdot \tau^w$ satisfies $s_{seq,m+1}$, i.e., $v' \cdot \tau^w = s_{seq,m+1}$.

- 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 $L_i$ $(1 \leq i \leq k)$, receiving OS $r_{n+1}$ locates on Lifeline $L_j$ $(1 \leq j \leq k)$ (see figure 51f).

In $seq$, we assume the last OS on $L_i$ is $OS_{pre}$, and the last OS on $L_j$ is $OS_{pre}$. After adding $m_{n+1}$ to the bottom of $seq$, $s_{n+1}$ becomes the last OS of $L_i$, and $r_{n+1}$ becomes the last OS of $L_j$. In trace $v' \in (\Sigma_{seq})^*$, if $OS_{pre}$ is the $p$th OS, where $1 \leq p_s \leq 2n$, and $OS_{pre}$ is the $p$th OS, where $1 \leq p_r \leq 2n + 1$. 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 $p_r < r \leq 2n + 1$.

To conclude, $\forall v', v' \in \Sigma^*$, if $v' \in (\Sigma_{seq})^*$, then $v' \cdot \tau^w = \Pi_{Basic, seq_{n+1}}$.

(b) We wish to prove $\forall \sigma, \sigma' \in \Sigma^*$, if $\sigma' \in (\Sigma_{seq})^*$, then $\sigma' \in (\Sigma_{seq})^*$. If $\sigma' \in (\Sigma_{seq})^*$, we wish to prove that $\sigma'$ respects the same order of $seq_{n+1}$. For $\Pi_{Basic, seq_{n+1}}$, we still group the sub-formulas using $t_{seq_{n+1}, m_{n+1}}, s_{seq_{n+1}, m_{n+1}}$ and $\varepsilon_{seq_{n+1}}$, i.e.,

$$\Pi_{Basic, seq_{n+1}} = t_{seq_{n+1}} \land \varepsilon_{seq_{n+1}} \land s_{seq_{n+1}, m_{n+1}}$$

We assume that if $\sigma \in (\Sigma_{seq})^*$, then $\sigma' \in (\Sigma_{seq})^*$. It is clear that $\sigma$ satisfies $t_{seq_{n+1}}$. Subformula $t_{seq_{n+1}}$ enforces the order of OSs in $\Sigma_{seq}$ and each OS should execute once and only once. We can also infer that trace $\sigma'$ satisfies $t_{seq_{n+1}}$ from $\sigma' \in (\Sigma_{seq})^*$. If $\sigma'$ does not contain an OS in $\Sigma_{seq}$, then $\sigma'$ does not satisfy $t_{seq_{n+1}}$, which defines that each OS in $\Sigma_{seq}$ should happen once. Therefore, all OSs in $\Sigma_{seq}$ executes once and only once in $\sigma'$. We wish to prove that, in $\sigma'$, all OSs in $\Sigma_{seq}$ respect their order defined by semantic aspects of $seq_{n+1}$.

Note that $\Pi_{Basic, seq_{n+1}}$
Fig. 51. Examples for basic Sequence Diagram with $n + 1$ Messages
must occur before $OS_q$ in $\sigma'$, which contradicts our assumption. Therefore, in $\sigma'_{[1..2n+2]}$, the OSs in $\Sigma_{seq}$ respect the order defined by semantic aspects of $seq_n$, i.e., if we remove the OSs not in $\Sigma_{seq}$ from $\sigma'_{[1..2n+2]}$ to obtain a new trace $\sigma''_{[1..2n]} \in (\Sigma_{seq})^*$.

Sub-formula $\vartheta_{n+1}$ 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, $\vartheta_{n+1}$ codifies the semantics of Message $m_{n+1}$ in $\sigma_{seq_{n+1}}$. In $\sigma'_{[1..2n+2]}$, $s_{n+1}$ 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 $\sigma'_{[1..2n+2]}$, where $|\Sigma_{seq}| = 2n$, and both of $s_{n+1}$ and $r_{n+1}$ occur only once. Thus, we can deduct that $\varepsilon_{seq_{n+1}}$ 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 $\varepsilon_{seq_{n+1}, m_{n+1}}$ 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 51a); or two OSs of $m_{n+1}$ locate on one new Lifeline, $L_{k+1}$ (see figure 51b).

$$\varepsilon_{seq_{n+1}, m_{n+1}} = \alpha_{seq_{L_{k+1}}} \land \alpha_{seq_{L_{k+2}}}$$

Sub-formula $\varepsilon_{seq_{n+1}, m_{n+1}}$ is a conjunction of $\alpha_{seq_{L_{k+1}}}$ and $\alpha_{seq_{L_{k+2}}}$. It returns $true$ only if none or at most one OS locates on each Lifeline. Therefore only one OS located on $L_{k+1}$ and $L_{k+2}$ respectively. $\varepsilon_{seq_{n+1}, m_{n+1}}$ 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, $\varepsilon_{seq_{n+1}, m_{n+1}}$ codifies the order between the OSs of $seq_n$ and the OSs of $m_{n+1}$ in $\sigma_{seq_{n+1}}$. In $\sigma'_{[1..2n+2]}$, the OSs of $seq_n$ and the OSs of $m_{n+1}$ respect the order defined by the semantic aspects of $seq_{n+1}$.

- 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 51c).

$$\varepsilon_{seq_{n+1}, m_{n+1}} = (\neg r_{n+1} \land \tilde{U} OS_{pre}) \land \alpha_{seq_{L_{k+1}}}$$

Sub-formula $\varepsilon_{seq_{n+1}, m_{n+1}}$ defines that $r_{n+1}$ 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. $\alpha_{seq_{L_{k+1}}}$ returns $true$. It denotes that only $s_{n+1}$ locates on $L_{k+1}$. Thus, $\varepsilon_{seq_{n+1}, m_{n+1}}$ codifies the order between the OSs of $seq_n$ and the OSs of $m_{n+1}$ in $\sigma_{seq_{n+1}}$. In $\sigma'_{[1..2n+2]}$, the OSs of $seq_n$ and the OSs of $m_{n+1}$ respect the order defined by the semantic aspects of $seq_{n+1}$.

- 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 51d). or two OSs of $m_{n+1}$ locate on an existing Lifeline $L_i$ ($i \leq k$) (see figure 51e).

$$\varepsilon_{seq_{n+1}, m_{n+1}} = (\neg s_{n+1} \land \tilde{U} OS_{pre}) \land \alpha_{seq_{L_{k+1}}}$$

Similarly, sub-formula $\varepsilon_{seq_{n+1}, m_{n+1}}$ 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. $\alpha_{seq_{L_{k+1}}}$ 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, $\varepsilon_{seq_{n+1}, m_{n+1}}$ codifies the order between the OSs of $seq_n$ and the OSs of $m_{n+1}$ in $\sigma_{seq_{n+1}}$. In $\sigma'_{[1..2n+2]}$, the OSs of $seq_n$ and the OSs of $m_{n+1}$ respect the order defined by the semantic aspects of $seq_{n+1}$.

- 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 51f).

$$\varepsilon_{seq_{n+1}, m_{n+1}} = (\neg s_{n+1} \land \tilde{U} OS_{pre}) \land (\neg r_{n+1} \land \tilde{U} OS_{pre})$$

Sub-formula $\varepsilon_{seq_{n+1}, m_{n+1}}$ defines that $s_{n+1}$ cannot happen until $OS_{pre}$ has taken place, where $OS_{pre}$ is the OS occurring right before $s_{n+1}$ on Lifeline $L_i$, and $r_{n+1}$ cannot happen until $OS_{pre}$ has taken place, where $OS_{pre}$ is the OS occurring right before $r_{n+1}$ on Lifeline $L_j$. As the semantic aspect of $seq_n$ defined, $s_{n+1}$ should locate right below $OS_{pre}$ on Lifeline $L_i$, and $r_{n+1}$ should locate right below $OS_{pre}$ on Lifeline $L_j$. OSs execute in their graphical order along each Lifeline. Thus, $\varepsilon_{seq_{n+1}, m_{n+1}}$ codifies the order between the OSs of $seq_n$ and the OSs of $m_{n+1}$ in $\sigma_{seq_{n+1}}$. In $\sigma'_{[1..2n+2]}$, the OSs of $seq_n$ and the OSs of $m_{n+1}$ respect the order defined by the semantic aspects of $seq_{n+1}$.

Now we have proven that $\sigma'_{[1..2n+2]}$ respects all the semantic aspects of $seq_{n+1}$, i.e., $\sigma'_{[1..2n+2]} \in (\Sigma_{seq})^*$.

To conclude, $\forall \sigma'. \sigma' \in \Sigma_\omega$, if $\sigma' \in (\Sigma_{seq_{n+1}})^\omega$, then $\sigma'_{[1..2n+2]} \in (\Sigma_{seq})^\omega$.

\[ \square \]

**APPENDIX B**

**PROOF OF THEOREM 2 AND THEOREM 3**

**Theorem 2.** $(\Sigma_{seq})^*$ and $PRE_{2k+2p}(\Sigma_{seq})$ 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, $c_{i1}$. ($r = 1$)

  - **Case 1.** Sequence Diagram $seq_0$ contains no CF. ($r = 0$)
The proof follows the one for basic Sequence Diagram.

- Case 2. Sequence Diagram seq1 contains only one CF, cf1. \((r = 1)\)

\[
\hat{\Pi}_{seq1} = \left( \bigwedge_{i \in LN(seq1)} \tilde{\alpha}_g \right) \land \left( \bigwedge_{j \in MSG(seq1)} \rho_j \right) \land \left( \bigwedge_{j \in MSG(seq1)} \beta_j \right) \land \Phi_{seq1} \land \varepsilon_{seq1}
\]

Case 2.1 We assume that cf1 has a Operands whose Interaction Constraint evaluate to False. The bth Operand contains \(q_b\) Messages, where \(1 \leq b \leq a\).

\[
\Phi_{cf1} = \Phi_{seq1} = \bigwedge_{i \in LN(cf1)} \bigwedge_{OS_{post} \in post(cf1_1)} \left( \Phi_{OS_{post}} \right)
\]

First, we consider the semantic aspects of the OSs directly enclosed in seq1. We project seq1 onto each of its covered Lifelines to obtain a EU. We also project cf1 onto each of its covered Lifeline to obtain a CEU. Therefore, each EU of seq1 may contain a CEU of cf1 and BEUs grouped by the OSs directly enclosed in the EU. Similar to the semantics of an BEU within a basic Sequence Diagram, the semantics of any BEU directly enclosed in the EU of seq1 specifies that OSs are ordered as their graphical order. If \(v \in \{\Sigma_{seq1}\}^*\), we can easily infer that \(v \cdot \tau^v \models \bigwedge_{i \in LN(seq1)} \tilde{\alpha}_g\). The semantics of each Message directly enclosed in seq1 specifies that its receiving OS cannot happen before the sending OS, and both OS can occur once only once. Accordingly, we can easily infer that \(v \cdot \tau^v \models \bigwedge_{j \in MSG(seq1)} \rho_j\), and \(v \cdot \tau^v \models \bigwedge_{j \in MSG(seq1)} \beta_j\).

Then, we consider the semantics of cf1. It defines that cf1 does not execute when the Constraints of all the Operands evaluate to False. cf1’s preceding Interaction Fragments and succeeding Interaction Fragments are ordered by Weak Sequencing. In this case, cf1’s preceding OS must happen before its succeeding OS on each Lifeline. We use LTL formula \(\eta_{cf1}^{f1}\) to capture cf1’s semantics. \(\eta_{cf1}^{f1}\) does not specify the order of OSs within Operands because the Operands whose Constraints evaluate to False are excluded. We assume that if \(v \cdot \tau^v\) does not satisfy \(\eta_{cf1}^{f1}\), then \(\eta_{cf1}^{f1}\) specifies that, on Lifeline \(i\), cf1’s preceding OS, \(OS_{pre}\) occurs after cf1’s succeeding OS, \(OS_{post}\). However, \(\eta_{cf1}^{f1}\) specifies that, on each Lifeline covered by cf1, its succeeding OS cannot happen until its preceding OS finishes execution. Functions \(pre(cf1 \uparrow_i)\) and \(post(cf1 \uparrow_i)\) return the set of OSs which may happen right before and after CEU \(cf1 \uparrow_i\). 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 \(v \cdot \tau^v \models \eta_{cf1}^{f1}\).

Finally, we consider the interleaving semantics of seq1. No OS in cf1 can executes, so only the OSs directly enclosed in seq1 can be enabled to execute. We can prove that \(v \cdot \tau^v \models \varepsilon_{seq1}\). The proof follows the one for basic Sequence Diagram.

Now we have proven that if \(v \in (\Sigma_{seq1})^*\), then \(v \cdot \tau^v \models \Pi_{seq1}\).

(b) We wish to prove that, \(\forall \upsilon, \sigma \in \omega, \sigma \in \{\Sigma_{seq1}\}^* \land \sigma \in \{\Sigma_{seq1}\}^*\), then \(v \cdot \tau^v \models \Pi_{seq1}\). In \(\Sigma_{LT_L}\), no OS within cf1 is enabled to execute because all the Constraints of cf1’s Operand evaluate to False. If \(\sigma \in (\Sigma_{seq1})^*\), then \(\sigma = \sigma_{[1..2h]} \cdot \tau^\omega\), which follows Lemma 2. We wish to prove that \(\sigma_{[1..2h]} \cdot \tau^\omega\) respects all the semantics of seq1, \(\sigma \models \Pi_{seq1}\), so \(\sigma\) satisfies all sub-formulas of \(\Pi_{seq1}\). We prove that the sub-formulas capture the semantic aspects as below.

First, we discuss the sub-formulas \(\tilde{\alpha}_g\), \(\rho_j\), \(\beta_j\) for seq1. Function \(TBEU(seq1 \uparrow_i)\) returns the BEUs directly enclosed in seq1 on Lifeline \(i\). These BEUs, which are separated using CEU of cf1 on Lifeline \(i\), are formed by the OSs directly enclosed in seq1. Function \(MSG(cf1)\) returns the set of Messages directly enclosed in cf1. We can prove that these sub-formulas capture the semantics of OSs directly enclosed in seq1. The proof follows the one for OS within basic Sequence Diagram.

Next, we discuss the sub-formula \(\eta_{cf1}^{f1}\). It defines that, on Lifeline \(i\), OSs in \(post(cf1 \uparrow_i)\) cannot happen until OSs in \(pre(cf1 \uparrow_i)\) finish execution. We assume that, if \(\eta_{cf1}^{f1}\) does not capture the semantics of cf1, then on a Lifeline, \(i\), the preceding OS of cf1, OS_{pre}, happens after the succeeding OS of cf1, OS_{post}. However, the semantics of \(\eta_{cf1}^{f1}\) defines the Weak Sequencing between cf1’s preceding OSs and succeeding OSs, i.e., its preceding OSs must happen before its succeeding OS on the same Lifeline. Therefore, OS_{pre} must happen before OS_{post}, which contradicts our assumption. In this way, we can prove that \(\eta_{cf1}^{f1}\) captures the semantics of cf1.

Finally, we discuss the sub-formula \(\varepsilon_{seq1}\). 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. We can prove that \(\varepsilon_{seq_1}\) captures the interleaving semantics of \(seq_1\) by following the proof for basic Sequence Diagram.

Now we have proven that \(\forall \sigma, \sigma \in \Sigma^*, \text{if } \sigma \in (\Sigma_{LTL})^\omega, \) it respects all the semantic aspects of \(seq_1, \text{i.e., } \sigma[1..2h] \in (\Sigma_{sem})^\star.\)

To conclude, \(\forall \sigma, \sigma \in \Sigma^*, \text{if } \sigma \in (\Sigma_{sem})^\star, \) then \(\sigma \cdot \tau^\omega \models \Pi_{seq_1}, \) and \(\forall \sigma, \sigma \in \Sigma^*, \text{if } \sigma \in (\Sigma_{LTL})^\omega, \) then \(\sigma[1..2h] \in (\Sigma_{sem})^\star.\)

- Case 2.2 We assume that \(cf_1\) has at least one Operand whose Constraint evaluates to \(cf_{\sigma \nu}\) have executed. In this case, function \(seq\) captures the interleaving semantics of \(seq_1\) by following the proof for basic Sequence Diagram.

Now we have proven that \(\forall \sigma, \sigma \in \Sigma^*, \text{if } \sigma \in (\Sigma_{LTL})^\omega, \) it respects all the semantic aspects of \(seq_1, \text{i.e., } \sigma[1..2h] \in (\Sigma_{sem})^\star.\)

To conclude, \(\forall \sigma, \sigma \in \Sigma^*, \text{if } \sigma \in (\Sigma_{sem})^\star, \) then \(\sigma \cdot \tau^\omega \models \Pi_{seq_1}, \) and \(\forall \sigma, \sigma \in \Sigma^*, \text{if } \sigma \in (\Sigma_{LTL})^\omega, \) then \(\sigma[1..2h] \in (\Sigma_{sem})^\star.\)

* 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 52, where \(cond1\) evaluate to \(true\), and \(cond2\) evaluates to \(false\)).

(a) We wish to prove that, \(\forall \sigma, \sigma \in \Sigma^*, \text{if } \sigma \in (\Sigma_{seq})^\star, \) then \(\sigma \cdot \tau^\omega \models \Pi_{seq_1}.\)

First, we consider the semantic aspects of the OSs directly enclosed in \(seq_1\). We can prove that \(\sigma \cdot \tau^\omega\) satisfies \(\bigwedge_{i \in \text{LN}(cf_1)} \tilde{\alpha}_{g_i}, \bigwedge_{j \in \text{MSG}(seq_1)} \beta_j, \) and \(\bigwedge_{j \in \text{MSG}(seq_1)} \beta_j.\) The proof follows the same as in case 2.1.

Then, 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 \(\tilde{\phi}_{cf_1}\) describes the semantics of the Operands whose Constraints evaluate to \(true\) using function \(TOP(cf_1),\) where the formula for each Operand follows the formula for a basic Sequence Diagram, i.e., a conjunction of \(\tilde{\alpha}^s, \beta^s, \) and \(\rho_j^s.\) Therefore, we can prove that \(\sigma \cdot \tau^\omega \models \tilde{\phi}_{cf_1}\) 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 \(i(1 \leq i \leq j), cf_1 \text{’s preceding OSs must execute before its CEU’s execution, and } cf_1 \text{’s succeeding OSs must execute afterwards. Function } \text{pre}(cf_1, \tau_i)\) returns the set of OSs which may happen right before CEU \(cf_1 \uparrow_i.\) The semantics aspect of \(seq_1\) defines that, for Lifeline \(i(1 \leq i \leq j),\) every OS within \(cf_1 \uparrow_i\) cannot execute until all OSs in \(\text{pre}(cf_1, \tau_i)\) finish execution. We wish to prove that the semantic aspect is captured by the first conjunct of sub-formula \(\tilde{\phi}_{cf_1}.\) We assume that, if \(\sigma \cdot \tau^\omega\) does not satisfy the first conjunct of \(\tilde{\phi}_{cf_1},\) then \(\tilde{\phi}_{cf_1}\) defines that, on Lifeline \(i,\) at least one OS, \(r_{c+d}(see figure 52),\) occurs before \(OS_{pre}.\) \(OS_{pre}\) is an OS in \(\text{pre}(cf_1, \tau_i).\) The first conjunct of \(\tilde{\phi}_{cf_1}\) specifies that any OS within \(cf_1\) on Lifeline \(i\) cannot execute until the OSs in \(\text{pre}(cf_1, \tau_i)\) finish execution, so \(OS_{pre}\) must happen before \(r_{c+d}\), which contradicts our assumption. In this way, we can prove that \(\sigma \cdot \tau^\omega\) satisfies the first conjunct of \(\tilde{\phi}_{cf_1}.\) Similarly, we can also prove that \(\sigma \cdot \tau^\omega\) satisfies the second conjunct of \(\tilde{\phi}_{cf_1}.\) Hence, \(\sigma \cdot \tau^\omega \models \tilde{\phi}_{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, 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 \(\sigma \in (\Sigma_{seq})^\star,\) we can deduce that \(|\sigma| = |AOS(seq_1)| = 2h + 2p.\) It is easy to infer that \(\sigma \cdot \tau^\omega \models \varepsilon_{seq_1}.\)

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

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

If \(\sigma \in (\Sigma_{seq})^\star, \) then \(\sigma \in (\Sigma_{seq})^\star \cdot \tau^\omega,\) 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 \(\tilde{\alpha}_g, \beta, \) and \(\rho_j\) for \(seq_1.\) We can prove that these sub-formulas capture the semantics of OSs directly enclosed in \(seq_1.\) The proof follows
Fig. 52. Example of Sequence Diagram with CF

the one in case 2.1.
Then, we discuss the sub-formula $\hat{\theta}^{c_{f_1}}$. Function $\bigwedge_{op \in \text{TOP}(c_{f_1})}$ returns the set of Operands whose Constraints evaluate to True within $c_{f_1}$. Hence, $\hat{\theta}^{c_{f_1}}$ only captures the semantics of Operands whose Constraints evaluate to True. It is consistent with the semantic aspect of $c_{f_1}$, which excludes the Operands whose Constraints evaluate to False. For each Operand whose Constraints evaluate to True, we wish to prove that sub-formulas $\hat{\alpha}_i, \rho_j, \hat{\beta}_j$ capture its semantics. $c_{f_1}$ contains no other CFs, so $\text{ABEU}(op \uparrow_i)$ 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 $\hat{\theta}^{c_{f_1}}$ captures the semantics of Combined Fragment $c_{f_1}$.

Next, we discuss the sub-formula $\hat{\gamma}_i^{c_{f_1}}$ for Lifeline $i$. We wish to prove that it captures the order of CEU $c_{f_1} \uparrow_i$ and its preceding/succeeding OSs on Lifeline $i$. The first conjunct of $\hat{\gamma}_i^{c_{f_1}}$ defines that any OS in CEU $c_{f_1} \uparrow_i$ cannot happen before all OSs in $\text{pre}(c_{f_1} \uparrow_i)$ finish execution. If it does not capture the semantic aspect, then we assume that at least an OS in $\text{pre}(c_{f_1} \uparrow_i)$, $\text{OS}_{pre}$, occurs after an OS in $c_{f_1} \uparrow_i$, $r_{c+d}$. Function $\text{pre}(c_{f_1} \uparrow_i)$ returns the set of OSs which may happen right before CEU $c_{f_1} \uparrow_i$. The semantics defines that all OS in $\text{pre}(c_{f_1} \uparrow_i)$ must happen before all OS within CEU $c_{f_1} \uparrow_i$. Thus, $\text{OS}_{pre}$ must occur before $r_{c+d}$, which contradicts our assumption. In this way, we have proven that the first conjunct of $\hat{\gamma}_i^{c_{f_1}}$ captures the order of CEU $c_{f_1} \uparrow_i$ and its preceding OSs on Lifeline $i$. Similarly, we can prove that the second conjunct of $\hat{\gamma}_i^{c_{f_1}}$ captures the order of CEU $c_{f_1} \uparrow_i$ and its succeeding OSs on Lifeline $i$. Therefore, we have proven that $\hat{\gamma}_i^{c_{f_1}}$ captures the order of CEU $c_{f_1} \uparrow_i$ and its preceding/succeeding OSs on Lifeline $i$.

Finally, we discuss the sub-formula $\varepsilon_{seq_1}$. It represents that only one OS in $|\text{AOS}(seq_1)|$ executes at a time, or all OSs in $|\text{AOS}(seq_1)|$ have executed. Function $|\text{AOS}(seq_1)|$ returns the set of OSs which can be enabled to execute in $seq_1$, i.e., it returns a set which includes the OSs directly enclosed in $seq_1$ and the OSs within $c_{f_1}$’s Operand whose Constraint evaluates to True. In $seq_1$, $|\text{AOS}(seq_1)| = 2h + 2p$. From lemma 2, if $\sigma \models \varepsilon_{seq_1}$, then $\sigma = \sigma_{[1 \ldots 2h+2p]} \cdot \tau^\omega$. Therefore, $\varepsilon_{seq_1}$ captures the semantic aspect, which enforces that only one object can execute an OS at a time and all
enabled OSs of seq_1 execute uninterrupted. Now we have proven that ∀σ, σ ∈ Σ^ω, if σ ∈ (Σ_{LTTL}^ω), respects all the semantic aspects of seq_1, i.e., σ[1..2h+2p] ∈ (Σ_{sem}^seq_1)^+. 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, ∀v, u ∈ Σ^∗, if v ∈ (Σ_{seq_1}^seq_1)^+, then v · τ^ω |= π_{seq_1}, and ∀σ, σ ∈ Σ^ω, if σ ∈ (Σ_{LTTL}^ω), then σ[1..2h+2p] ∈ (Σ_{seq_1}^seq_1)^+.

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 ω_{CF}. 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^strict, has two Operands whose Interaction Constraints evaluate to True. The first Operand contains p_1 Messages, and the second Operand contains p Messages. cf_1^strict covers i Lifelines. (a) We wish to prove that, ∀v, u ∈ Σ^∗, if v ∈ (Σ_{seq_1}^seq_1)^+, then v · τ^ω |= π_{seq_1}.

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 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. In this case, preEU(u) returns the last OS in EU u. The semantic aspect of Strict Sequencing can be considered as that, any OS in Operand k cannot happen until the OSs in all preEU(u), where u is the EU of Operand k on Lifeline j (1 ≤ j ≤ i), finish execution. We introduce sub-formula χ_k to capture the semantics of Operand k. We assume that, if v · τ^ω does not sat-
To conclude, $\forall u,v \in \Sigma^*$, if $v \in (\Sigma^{\text{seq1}}_{\text{sem}})^*$, then $v \cdot \tau^\omega \models \Pi_{\text{seq1}}$, and $\forall \sigma, \sigma \in \Sigma^\omega$, if $\sigma \in (\Sigma^{\text{seq1}}_{\text{seq}})^*$, then $\sigma_{[1,2h+2p]} \in (\Sigma^{\text{seq}}_{\text{sem}})^*$. 

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 $\Psi_{\text{CF}}^\text{alt}$ to capture the semantics of Alternatives. Sub-formulas $\overline{\theta}_{m}^{\text{CF}}$ and $\overline{z}_{i,m}^{\text{CF}}$ can be rewritten into $\overline{\theta}_{m}^{\text{CF}}$ and $\overline{z}_{i,m}^{\text{CF}}$ by following the same procedures of rewriting sub-formulas $\theta_{m}^{\text{CF}}$ and $\gamma_{i,m}^{\text{CF}}$. The LTL formula of Alternatives, $\Psi_{\text{CF}}^\text{alt}$, with rewritten sub-formulas is shown in figure 53. We assume that, a given Alternatives, $\Psi_{\text{CF}}^{\text{alt}}$, has two Operands whose Interaction Constraints evaluate to True. The first Operand contains $p_1$ Messages, and the second Operand contains $p_2$ Messages. $\Psi_{\text{CF}}^{\text{alt}}$ covers $i$ Lifelines. 

$$\Phi_{\text{CF}}^{f_1} = \Psi_{\text{alt}}^{f_1}$$

(a) We wish to prove that, $\forall u,v \in \Sigma^*$, if $v \in (\Sigma^{\text{seq1}}_{\text{sem}})^*$, then $v \cdot \tau^\omega \models \Pi_{\text{seq1}}$. 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 $\Psi_{\text{alt}}^{f_1}$ defines the semantics of Operand $m$ whose Constraint evaluates to True. If $m$ is chosen, its semantics is captured by sub-formula $\overline{\theta}_{m}^{f_1}$ and $\overline{z}_{i,m}^{f_1}$. Otherwise, $\Psi_{\text{alt}}^{f_1}$ evaluates to True, denoting that $m$ is excluded. We can prove that $\overline{\theta}_{m}^{f_1}$ describes the order among OSs within $m$ by following the proof for sub-formula $\overline{\theta}_{m}^{f_1}$. Similarly, we can prove that $\overline{z}_{i,m}^{f_1}$ describes the order among OSs within $m$ and the Alternatives’ adjacent OSs by following the proof for sub-formula $\overline{z}_{i,m}^{f_1}$. Therefore, $v \cdot \tau^\omega$ satisfies $\Psi_{\text{alt}}^{f_1}$. We have proven that $v \cdot \tau^\omega$ satisfies $\alpha_g, \rho_j$, and $\beta_j$ for seq1 in case 2.1.2(1). For sub-formula $\varepsilon_{\text{seq1}}$, function $AOS(seq1)$ returns the enabled and chosen OSs, i.e., for Alternatives, only the OSs within the chosen Operand are returned. We can prove that $v \cdot \tau^\omega$ satisfies $\varepsilon_{\text{seq1}}$ by following the proof in case 2.1.2(1). Hence, we can prove that $v \cdot \tau^\omega \models \Pi_{\text{seq1}}$.

(b) We wish to prove that, $\forall u,v \in \Sigma^*$, if $\sigma \in (\Sigma^{\text{seq2}}_{\text{LT}})^\omega$, $\sigma_{[1,2h+2p]} \in (\Sigma^{\text{seq}}_{\text{sem}})^*(m$ is the chosen Operand of $\Psi_{\text{CF}}^{f_1}$). If $\sigma \in (\Sigma^{\text{seq2}}_{\text{LT}})^\omega$, then $\sigma = \sigma_{[1,2h+2p]} \cdot \tau^\omega$, which follows Lemma 2. We wish to prove that $\sigma = \sigma_{[1,2h+2p]}$ respects all the semantics of seq1, $\sigma \models \Pi_{\text{seq1}}$, so $\sigma$ satisfies all sub-formulas of $\Pi_{\text{seq1}}$. We have proven that the sub-formulas $\delta_g, \rho_j$, and $\beta_j$ capture the semantics of OS directly enclosed in seq1; sub-formula $\varepsilon_{\text{seq1}}$ captures the interleaving semantics of seq1 (see case 2.1.2(1)). We need to prove that sub-formula $\Psi_{\text{alt}}^{f_1}$ captures the semantics of Alternatives.

Sub-formula $\Psi_{\text{alt}}^{f_1}$ is a conjunction of sub-formula $\Psi_{\text{alt}}^{m}$, where $m$ is an Alternatives’Operand whose Constraint evaluates to True. Therefore, the Operands whose Constraints evaluate to False are excluded. $\Psi_{\text{alt}}^{m}$ evaluates to False if $m$ is unchosen, which captures the semantics that the unchosen Operands are excluded. $\Psi_{\text{alt}}^{m}$ is a conjunction of sub-formulas $\overline{\theta}_{m}^{f_1}$ and $\overline{z}_{i,m}^{f_1}$ when $m$ is the chosen Operand. We can prove that sub-formula $\overline{z}_{i,m}^{f_1}$ captures the order among OSs within $m$ by following the proof of $\overline{\theta}_{m}^{f_1}$. We can also prove that sub-formula $\overline{z}_{i,m}^{f_1}$ captures the order between OSs within $m$ and the Alternatives’adjacent OSs by following the proof of $\overline{\theta}_{m}^{f_1}$. In this way, we can prove that sub-formula $\Psi_{\text{alt}}^{f_1}$ captures the semantics of Alternatives. Hence, we can prove that $\sigma_{[1,2h+2p]} \in (\Sigma^{\text{seq1}}_{\text{sem}})^*$. To conclude, $\forall u,v \in \Sigma^*$, if $v \in (\Sigma^{\text{seq1}}_{\text{seq}})^*$, then $v \cdot \tau^\omega \models \Pi_{\text{seq1}}$, and $\forall \sigma, \sigma \in \Sigma^\omega$, if $\sigma \in (\Sigma^{\text{seq2}}_{\text{LT}})^\omega$, then $\sigma_{[1,2h+2p]} \in (\Sigma^{\text{seq}}_{\text{sem}})^*$.

Case 2.4 The Loop represents the iterations of its sole Operand. We capture the semantics of Loop using LTL formula $\Psi_{\text{loop}}^{f_1}$ 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 $\alpha_g, \rho_j$, and $\beta_j$ as proven. We need to prove that sub-formula $\bigwedge_{k \in \text{NFTOP}(CF)} \chi_k$ of Strict Sequencing. The proof is quite similar as the proof for sub-formula $\bigwedge_{k \in \text{NFTOP}(CF)} \chi_k$.
Sub-formulas $\tilde{\gamma}^{CF_{n+1}}$, $\tilde{\gamma}^{CF_{n+1}}$, 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, $\nu^{\prime} \cdot \tau^{\omega}$ satisfies $\tilde{\gamma}^{CF_{n+1}}$ and the additional sub-formulas for each Operator. Sub-formula $\tilde{\gamma}^{CF_{n+1}}$ still specifies the Weak Sequencing between $cf_{n+1}$ and its preceding/succeeding Interaction Fragments. Comparing to base case, $cf_{n+1}$’s preceding/succeeding Interaction Fragments can be OSs or CFs. We wish to prove that our algorithms for calculating $pre(cf_{n+1} \uparrow i)$ and $post(cf_{n+1} \uparrow i)$ are correct.

Function $pre(cf_{n+1} \uparrow i)$ returns the set of OSs which happen right before CEU $cf_{n+1} \uparrow i$. We focus on the CEU or EU $v$ prior to $cf_{n+1} \uparrow i$ on Lifeline $i$. The EUs whose Constraints evaluate to False are excluded. Therefore, $v$ should be a CEU containing at least one EU whose Constraint evaluates to True or an EU whose Constraint evaluates to True. We start from the CEU or EU prior to $cf_{n+1} \uparrow i$, and check the CEUs and EUs until we find $v$. If $v$ does not exist, we define that the first conjunct of $\tilde{\gamma}^{CF_{n+1}}$ evaluates to True. Otherwise, we discuss the return value of the function by different cases.

1. **Case i.** If $v$ is a BEU, function returns the OS in the bottom of $v$, $OS_v$. We assume that if the function returns another OS, $OS_s$, then $OS_s$ should happen after $OS_v$. However, the semantics defines that OSs are ordered graphically in a BEU. $OS_s$ is the last one to execute in $v$, which contradicts our assumption. Thus, we can prove that the function returns the OS in the bottom of $v$.

2. **Case ii.** If $v$ is a CEU with one BEU whose Constraint evaluates to True, function returns the OS in the bottom of the BEU as we proven in case 1.

3. **Case iii.** If $v$ is a CEU with multiple BEUs whose Constraints evaluate to True. (1) $v$ with Operator “par” returns a set containing the last OS of each BEU, as defined by the semantics of Parallel (We have proven in base case 2.2.1); (2) $v$ with Operator “alt” returns a set containing the last OS of the chosen BEU, as defined by the semantics of Parallel (Fig. 53. Rewriting LTL formula for Alternatives).
Alternatives (We have proven in base case 2.3); (3) \( v \) with Operator “weak” or “strict” returns a set containing the last OS of the last BEU, as defined by the semantics of Strict Sequencing (We have proven in base case 2.2.2).

- Case iv. If \( v \) is a CEU with nested CEUs. (1) If \( v \) directly contains only one EU whose Constraint evaluates to True, we find the EU’s last CEU or EU, \( w \), and recursively apply case 1 to 4 to prove it. (2) If \( v \) directly contains multiple EU’s whose Constraint evaluates to True, we recursively apply case 1 to 4 to (a) each EU to prove it (\( v \)’s Operator is “par”); (b) the chosen EU to prove it (\( v \)’s Operator is “alt”); (c) the last EU (\( v \)’s Operator is “weak” or “strict”) to prove it.

The proof of the algorithm for calculating \( \text{post}(c_{f_{n+1}} \uparrow_1) \) follows the one of the algorithm for calculating \( \text{pre}(c_{f_{n+1}} \uparrow_1) \). Hence, \( v \cdot \tau^\omega = \approx c_{f_{n+1}} \).

Finally, we consider the semantic aspect for the \( seq_{n+1} \). Function \( AOS(seq_{n+1}) \) returns the set of chosen and enabled OSs within \( seq_{n+1} \). The semantic aspect specifies that only one enabled OS can execute at a time, and all enabled OSs should execute uninterrupted. If \( v' \in (\Sigma_{sem}^{seq_{n+1}})^* \), we can deduce that \( |v'| = |AOS(seq_{n+1})| = 2h + 2p_n + 2p_{n+1} \). It is easy to infer that \( v' \cdot \tau^\omega = \approx v_{seq_{n+1}} \).

Now we have proven that if \( v' \in (\Sigma_{sem}^{seq_{n+1}})^* \), then \( v' \cdot \tau^\omega = \approx \Pi_{seq_{n+1}} \).

(b) We wish to prove that, \( \forall \sigma' \cdot \sigma' \in \Sigma^\omega \), if \( \sigma' \in (\Sigma_{LTL}^{seq_{n+1}})^\omega \), then \( \sigma'_{[1, 2h+2p_n+2p_{n+1}]} \in (\Sigma_{sem}^{seq_{n+1}})^* \).

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

First, we discuss the sub-formulas \( \sigma_{\mathfrak{g}} \), \( \rho_{\mathfrak{j}} \), and \( \beta_{\mathfrak{j}} \) 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 \( seq_{n} \), the sub-formula captures the semantics of \( n \) CFs. In \( seq_{n+1} \), adding \( c_{f_{n+1}} \) does not change the semantics of the existing CFs. It is easy to infer that, sub-formula \( \phi CF \) still captures the semantics of the CFs except for \( c_{f_{n+1}} \).

Next, we discuss the sub-formula formula \( \phi CF_{n+1} \), which is a conjunction of sub-formulas \( \theta CF_{f_{n+1}} \), \( \zeta CF_{f_{n+1}} \), and the additional one for each Operator. With the proof of base case, \( \theta CF_{f_{n+1}} \) captures the semantics of \( c_{f_{n+1}} \)’s Operators, while the additional sub-formula captures the semantics of \( c_{f_{n+1}} \)’s Operator. Sub-formula \( \zeta CF_{f_{n+1}} \) may be different from the base case, since the preceding/succeeding Interaction Fragment of \( c_{f_{n+1}} \) can be other CFs. On Lifeline \( i \), functions \( \text{pre}(c_{f_{n+1}} \uparrow_1) \) and \( \text{post}(c_{f_{n+1}} \uparrow_1) \) return the set of OSs which may happen right before and after CEU \( c_{f_{n+1}} \uparrow_1 \) respectively. We have proven that our algorithms can calculate \( \text{pre}(c_{f_{n+1}} \uparrow_1) \) and \( \text{post}(c_{f_{n+1}} \uparrow_1) \) for all the cases. Thus, we can infer that \( \bigwedge_{i \in \mathcal{L}(CF)} \theta CF_{f_{n+1}} \) still captures the Weak Sequencing between \( c_{f_{n+1}} \) and its preceding/succeeding Interaction Fragments.

Finally, we discuss the sub-formula \( \varepsilon_{seq_{n+1}} \). 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 \( seq_{n+1} \), \( |AOS(seq_{n+1})| = 2h + 2p_n + 2p_{n+1} \). From lemma 2, if \( \sigma = \varepsilon_{seq_{n+1}} \), then \( \sigma = \sigma_{[1, 2h+2p_n+2p_{n+1}]} \cdot \tau^\omega \). Therefore, \( \varepsilon_{seq_{n+1}} \) captures the interleaving semantics of \( seq_{n+1} \).

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

To conclude, \( \forall v' \cdot v' \in \Sigma^\omega \), if \( v' \in (\Sigma_{sem}^{seq_{n+1}})^* \), then \( v' \cdot \tau^\omega = \approx \Pi_{seq_{n+1}} \) and \( \forall \sigma' \cdot \sigma' \in \Sigma^\omega \), if \( \sigma' \in (\Sigma_{LTL}^{seq_{n+1}})^\omega \), then \( \sigma'_{[1, 2h+2p_n+2p_{n+1}]} \in (\Sigma_{sem}^{seq_{n+1}})^* \).

Fig. 54. Example of Sequence Diagram with nested CF

If a Sequence Diagram contains nested CFs, the semantics of nested CFs are independent. For instance, if \( c_{f_1} \) whose Operand is \( op_1 \) contains \( c_{f_2} \) whose Operand is \( op_2 \) (see figure 54), the semantic rules of \( c_{f_1} \) do not constrain the semantic rules of \( c_{f_2} \).

Theorem 3. \( (\Sigma_{sem}^{seq_{n+1}})^* \) and \( \phi CF_{n+2} \cdot \phi CF_{n+2} \cdot (\Sigma_{LTL}^{seq_{n+1}})^\omega \) are equal.

Proof: We use mathematical induction, which is based on the maximal layer of CE, \( l \), within \( seq_{n+1} \). The proof follows the one for theorem 2.
Inductive step. seq_{nested}^{n+1} directly contains r CFs. We assume that cf\_u, which is a CF directly enclosed in seq_{nested}^{n+1}, contains cf\_u, which is a CF with the maximal layer within seq_{nested}^{n+1}. The maximal layer of CF within seq_{nested}^{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 \( \forall u,v \in (\Sigma_\text{seq}^{n+1})^*, v \cdot \tau \omega = \Pi_\text{seq}^{n+1}, \forall \sigma, \sigma' \in (\Sigma_\text{LTL}^{n+1})^\omega \), then \( \sigma[1..2n+2p_n] \in (\Sigma_\text{seq}^{n+1})^* \). (i = n)

We add a CF cf\_u in seq_{nested}^{n+1} to form a new Sequence Diagram, seq_{nested}^{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_{nested}^{n+1}. In seq_{nested}^{n+1}, \( p_n \) Messages are chosen and enabled in Operands whose Interaction Constraints evaluate to True. We wish to prove that, \( \forall u', v' \in \Sigma^n \), if \( u' \in (\Sigma_\text{seq}^{n+1})^* \), then \( u' \cdot \tau \omega = \Pi_\text{seq}^{n+1}, \forall \sigma, \sigma' \in (\Sigma_\text{LTL}^{n+1})^\omega \), then \( \sigma'[1..2n+2p_n] \in (\Sigma_\text{seq}^{n+1})^* \).

When we add cf\_u into seq_{nested}^{n+1}, then order of the OSs directly enclosed in seq_{nested}^{n+1} keep unchanged. Thus, the semantics of the OSs directly enclosed in seq_{nested}^{n+1} can still be captured using the corresponding sub-formulas of \( \Pi_\text{seq}^{n+1} \). The LTL templates \( \tilde{\Pi}_\text{seq}^{n+1} \) and \( \tilde{\Pi}_\text{seq}^{n+1} \) are shown as,

\[
\tilde{\Pi}_\text{seq}^{n+1} = (\bigwedge_{i \in L(N(\text{seq}^{n+1}))} \tilde{\alpha}_g) \land (\bigwedge_{j \in M(S(\text{seq}^{n+1}))} \rho_j) \land (\bigwedge_{j \in M(S(\text{seq}^{n+1}))} \beta_j) \land (\bigwedge_{i \in L(N(\text{seq}^{n+1}))} \phi CF).
\]

Next, we consider CF cf\_u, whose semantics is captured using \( \Phi CF \). We discuss sub-formula \( \Phi CF \) using three cases. (1) If all the Constraints of cf\_u’s Operands evaluate to False, \( \Phi CF = \emptyset \). We can prove that \( u' \cdot \tau \omega \) satisfies \( \Phi CF \). 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, \( \Phi CF = \Psi CF \land \Phi CF \). The semantics of cf\_u is not altered by adding cf\_u. Hence, we can infer that \( u' \cdot \tau \omega \) satisfies \( \Phi CF \). We can prove that \( u' \cdot \tau \omega \) satisfies \( \tilde{\gamma}_i CF \) and \( \bigwedge_{i \in L(N(cf_u))} \tilde{\gamma}_i CF \).

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

Now we have proven that if \( u' \in (\Sigma_\text{seq}^{n+1})^* \), then \( u' \cdot \tau \omega = \tilde{\Pi}_\text{seq}^{n+1} \).

(b) We wish to prove that, \( \forall u', \sigma' \in \Sigma^n \), if \( \sigma' \in (\Sigma_\text{LTL}^{n+1})^\omega \), then \( \sigma'[1..2n+2p_n+1] \in (\Sigma_\text{seq}^{n+1})^* \).

If \( \sigma' \in (\Sigma_\text{LTL}^{n+1})^\omega \), then \( \sigma' = \sigma[1..2n+2p_n+1] \), \( \tau \omega \), which follows Lemma 2. We wish to prove that \( \sigma'[1..2n+2p_n+1] \) respects all the semantics of seq_{nested}^{n+1}, \( \sigma' \vdash \tilde{\Pi}_\text{seq}^{n+1}, \) so \( \sigma' \) satisfies all sub-formulas of seq_{nested}^{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_{nested}^{n+1}. These sub-formulas are not changed, so they still capture the semantics of OSs directly enclosed in seq_{nested}^{n+1}. We can also infer that these sub-formulas capture the semantics of OSs directly enclosed in seq_{nested}^{n+1}.

Then, we discuss the sub-formula \( \bigwedge_{i \in L(N(seq_{nested}^{n+1}))} \tilde{\alpha}_g \land g \in \text{TBE} \) seq_{nested}^{n+1} for seq_{nested}^{n+1}. The sub-formula \( \bigwedge_{i \in L(N(seq_{nested}^{n+1}))} \tilde{\alpha}_g \land g \in \text{TBE} \) seq_{nested}^{n+1} captures the semantics of the CFs (except for cf\_u) directly enclosed in it. In seq_{nested}^{n+1}, adding cf\_u does not change the semantics of the CFs except for cf\_u. It is easy to infer that the sub-formula still captures the
semantics of the CFs except for $cf_v$.

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

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

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

To conclude, $\forall \sigma', \sigma' \in \Sigma^*$, if $\sigma' \in (\Sigma^{seq_{\text{nested}}}_{\text{seq}+1} \cdot \Sigma^{\text{nested}}_{\text{seq}+1})^*$, then $\sigma' \cdot \tau^\omega = \Pi_{\text{seq}_{\text{nested}}}^{n+1}$, and $\forall \sigma', \sigma' \in \Sigma^*$, if $\sigma' \in (\Sigma^{seq_{\text{nested}}}_{\text{LTL}} \cdot \Sigma^{\text{nested}}_{\text{LTL}})^*$, then $\sigma'_{[1..2h+2p_{n+1}]} \in (\Sigma_{\text{seq}+1})^*$. \hfill $\blacksquare$
Fig. 58. Paragraph 164.506(c)(1)

Fig. 59. Paragraph 164.506(c)(2)

Fig. 60. Paragraph 164.506(c)(3)

Fig. 61. Paragraph 164.506(c)(5)

Fig. 62. Paragraph 164.506(c)

Fig. 63. Paragraph 164.506
Fig. 64. Paragraph 164.508(a)(2)

Fig. 65. Paragraph 164.508(a)(3)(i)

Fig. 66. Paragraph 164.508(b)(5)

Fig. 67. Paragraph 164.508(c)(4)

Fig. 68. Paragraph 164.510(a)(2)
Fig. 75. Paragraph 164.512(a)(1)

Fig. 76. Paragraph 164.512(a)(2)

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

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

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

Fig. 80. Paragraph 164.512(b)(1)(iv)
Fig. 87. Paragraph 164.512(e)(1)(i)

Fig. 88. Paragraph 164.512(e)(1)(ii)

Fig. 89. Paragraph 164.512(e)(1)(iv)

Fig. 90. Paragraph 164.512(e)
Fig. 91. Paragraph 164.512(f)(1)(i)

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

Fig. 93. Paragraph 164.512(f)(1)

Fig. 94. Paragraph 164.512(f)(2)

Fig. 95. Paragraph 164.512(f)(3)(1)
Fig. 96. Paragraph 164.512(f)(3)(ii)

Fig. 98. Paragraph 164.512(f)(4)

Fig. 97. Paragraph 164.512(f)(3)

Fig. 99. Paragraph 164.512(f)(5)
Fig. 100. Paragraph 164.512(f)(6)

Fig. 101. Paragraph 164.512(f)

Fig. 102. Paragraph 164.512(g)(1)

Fig. 103. Paragraph 164.512(g)(2)

Fig. 104. Paragraph 164.512(g)
Fig. 105. Paragraph 164.512(h)

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

Fig. 107. Paragraph 164.512(i)(1)(ii)
Fig. 108. Paragraph 164.512(i)(iii)
Fig. 115. Paragraph 164.512(k)(1)(iii)

Fig. 116. Paragraph 164.512(k)(1)(iv)

Fig. 117. Paragraph 164.512(k)(1)

Fig. 118. Paragraph 164.512(k)(2)

Fig. 119. Paragraph 164.512(k)(3)

Fig. 120. Paragraph 164.512
Fig. 121. Paragraph 164.520(a)(1)

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

Fig. 123. Paragraph 164.520(a)(2)(ii)

Fig. 124. Paragraph 164.520(a)(2)(iii)
Fig. 147. Paragraph 164.526(a)(1)

Fig. 148. Paragraph 164.526(a)(2)(i)

Fig. 149. Paragraph 164.526(a)(2)(ii)

Fig. 150. Paragraph 164.526(a)(2)(iii)

Fig. 151. Paragraph 164.526(a)(2)(iv)

Fig. 152. Paragraph 164.526(b)(1)

Fig. 153. Paragraph 164.526(b)(2)(i)

Fig. 154. Paragraph 164.526(b)(2)(ii)

Fig. 155. Paragraph 164.526(c)(2)
Fig. 156. Paragraph 164.526(c)(3)(i)

Fig. 157. Paragraph 164.526(c)(3)(ii)

Fig. 158. Paragraph 164.526(d)(2)

Fig. 159. Paragraph 164.526(d)(3)

Fig. 160. Paragraph 164.526(d)(5)(i)

Fig. 161. Paragraph 164.526(d)(5)(ii)

Fig. 162. Paragraph 164.526(d)(5)(iii)

Fig. 163. Paragraph 164.526(c)
Fig. 164. Paragraph 164.528(a)(1)

Fig. 165. Paragraph 164.528(a)(2)(i)

Fig. 166. Paragraph 164.528(a)(2)(ii)

Fig. 167. Paragraph 164.528(a)(3)

Fig. 168. Paragraph 164.528(c)(1)(i)

Fig. 169. Paragraph 164.528(c)(1)(ii)

Fig. 170. Paragraph 164.528(c)(2)