Under consideration for publication in Formal Aspects of Computing

Relating Computer Systems to Sequence Diagrams — The Impact of Underspecification and Inherent Nondeterminism

Ragnhild Kobro Runde[1], Atle Refsdal[1,2] and Ketil Stølen[1,2]

[1]Department of Informatics, University of Oslo, Norway
[2]SINTEF ICT, Norway

Abstract.
Having a sequence diagram specification and a computer system, we need to answer the question: Is the system compliant with the sequence diagram specification in the desired way? We present a procedure for answering this question for sequence diagrams with underspecification and inherent nondeterminism. The procedure is independent of any concrete technology, and relies only on the execution traces that may be produced by the system. If all traces are known, the procedure results in either "compliant" or "not compliant". If only a subset of the traces is known, the conclusion may also be "likely compliant" or "likely not compliant".

Keywords: sequence diagrams; computer systems; refinement; implementation; compliance; denotational trace semantics

1. Introduction

Having a sequence diagram specification and a computer system, we need to answer the question: *Is the system compliant with the specification in the desired way?* Intuitively, a system is compliant with a specification if the behaviours of the system are as described by the specification. The system should potentially be able to perform every behaviour that the specification requires it to offer, and it should do nothing that the specification disallows.

The question of compliance is essential every time a computer system is built from a specification. Even so, the relationship between sequence diagrams and computer systems is surprisingly unclear. An important

Correspondence and offprint requests to: Ragnhild Kobro Runde, Department of Informatics, PO Box 1080 Blindern, N-0316 Oslo, Norway. e-mail: ragnhild.runde@ifi.uio.no

reason for this, is that sequence diagrams (in contrast to most other techniques for specifying dynamic behaviour) give only a partial view of the behaviour. Also, sequence diagrams are used for specifying computer systems within a broad range of application domains, and they are used for different methodological purposes including requirements capture, illustrating example runs, test scenario specification and risk scenario documentation. The various usages of sequence diagrams differ in the expressiveness required, and in how the partiality of sequence diagrams should be understood.

In this paper we investigate compliance with respect to two classes of sequence diagrams: Sequence diagrams with underspecification and sequence diagrams with both inherent nondeterminism and underspecification. The first class may be used to capture trace properties, i.e., properties that can be falsified by a single trace. Examples of trace properties include safety and liveness properties [AS85]. The second class is also able to capture trace-set properties, which are properties that can only be falsified by sets of traces. Trace-set properties include many information flow security properties as well as permissions in the setting of policy rules [SSS09]. In fact, in the case of information flow security properties, being able to distinguish between inherent nondeterminism and underspecification is necessary in order to avoid the refinement anomaly [Jac89, Ros95, Jür01, SS06].

Underspecification means that certain aspects of the system behaviour are left open. Typically, underspecification is a consequence of abstraction and a desire to focus on the essential behaviour of the system. Underspecification implies a kind of nondeterminism, since the specification allows those responsible for implementing or further refining the specification to choose between alternative ways of performing a task.

Inherent nondeterminism, on the other hand, means that the system must be able to produce all of the described alternatives. For instance, when sequence diagrams are used to describe example runs of the system, each of these example runs is required to be mirrored in the final implementation. Each example run then constitutes a trace-set property. Another example is when specifying a gambling machine or similar, where it is necessary to ensure that both winning and losing outcomes can be produced by the system.

In this paper, we define a set of compliance relations taking into account what sequence diagram class is used, as well as different interpretations of the partiality of sequence diagrams. We propose a general procedure for checking compliance of computer systems with respect to sequence diagrams, parameterized with the compliance relation to be employed. The procedure is independent of the concrete implementation technologies used, as this is not prescribed by a sequence diagram. Instead, we represent the system by the set of execution traces that the system is able to produce. An execution trace is a sequence of events such as transmission and reception of messages to and from the entities in the system. The set of execution traces may be established or estimated for instance by source code inspection, or by testing.

In practice, the system may be able to produce infinitely many traces, and traces may be infinitely long. If we do not have access to the source code, and the execution traces are found by e.g. testing, it will not be possible to establish infinitely long traces or infinitely many traces by observation alone. Instead, the best that can be achieved is an estimate based on a finite number of finitely long observations. If a system is observed for a long period of time and nothing happens, it may be assumed that nothing more will happen no matter how long one waits. Similarly, if the same output has been transmitted continuously for a long time, then it may be assumed that an infinite loop has been entered. In practice this kind of assumptions and estimates are unavoidable. Moreover, obtaining the full set of traces is not very realistic for systems of some size. In the field of testing, this problem can be addressed by defining selection hypotheses under which a verdict can be reached from a finite test set [Gau95]. In a similar manner, our compliance checking procedure is designed to help in deciding whether compliance holds or not also in cases where only a subset of the execution traces is known.

Based on the compliance relations, we derive a set of corresponding refinement relations. By refinement, we mean adding more detail to the specification while preserving the requirements from the original specification. Any system compliant with the refined sequence diagram should also be compliant with the original diagram. The refinement and compliance relations given in this paper all support a stepwise and compositional development process.

To summarize, in the general case and in many situations encountered in practice, it is not possible to automatically check whether a system complies with a sequence diagram. Even in such situations, however, or more correctly, in such situations in particular, we need clear definitions of what it means to comply with a sequence diagram from an intuitive point of view and also methodological advice for how to check compliance. These definitions and methodological advice in relation to sequence diagrams with underspecification and inherent nondeterminism are the main contributions of this paper. We are not aware of similar contributions in the literature.

The rest of this paper is organized as follows: Section 2 gives an overview of the compliance checking procedure. In Section 3 we introduce sequence diagrams with underspecification and their semantic model. In Section 4 we define compliance relations for sequence diagrams with underspecification, and derive the corresponding refinement relations. Section 5 extends sequence diagrams with inherent nondeterminism, while compliance and refinement for such sequence diagrams are defined in Section 6.

In Sections 4 and 6, we assume that the complete set of execution traces for the system is known. In Section 7, we characterize the conditions under which the procedure may arrive at a definitive conclusion when only a subset of the execution traces is known, and give guidelines for what should be done when these conditions do not hold. We present related work in Section 8 before concluding in Section 9.

2. The Compliance Checking Procedure

An overview of the compliance checking procedure is given in Figure 1. As can be seen from the figure, the procedure takes a sequence diagram D with semantics $[\![D]\!]$, a computer system S whose set of execution traces is characterized by $traces(S)$, and a compliance relation $\mapsto_\rho$ as input (where ρ is a parameter representing the exact compliance relation to be used), and reaches one of four different conclusions. If the complete set of execution traces is known, the conclusion will always be either *compliant* or *not compliant*. On the other hand, with only a subset of the execution traces available, the conclusion may also be one of *likely compliant* or *likely not compliant*.

The first step of the procedure is to obtain a subset T of the execution traces for the system S. How this is achieved is not prescribed by the procedure, but typical alternatives include source code inspection and testing. Preferably, T should be the complete set of execution traces, but this is often not possible as explained in the introduction.

After having obtained (a subset of) the execution traces, the procedure continues in step 2 by transforming this set T into a mathematical representation $\langle T \rangle_D$ of the system. This representation uses the same semantic model as $[\![D]\!]$, and is further described in Sections 4.1 and 6.1.

Next, step 3 is to check whether the given compliance relation $\mapsto_\rho$ holds between the semantics $[\![D]\!]$ of the sequence diagram and the system representation $\langle T \rangle_D$. Depending on the result, the procedure continues with one of the two symmetrical branches in Figure 1.

If the compliance relation holds, the left branch of the figure is followed, starting with a new check as step 4a. If T is the complete set of execution traces, then it may be concluded that the system S is compliant with the sequence diagram D according to the compliance relation $\mapsto_\rho$. There are also cases where the same positive conclusion may be reached even when T only contains a subset of the execution traces. These cases are described precisely by the predicate P^ρ_{pos}, defined in Section 7.

In the cases where $[\![D]\!] \mapsto_\rho \langle T \rangle_D$ holds, but T contains only a subset of the execution traces and the predicate P^ρ_{pos} does not hold, one may try to obtain a more complete estimate for T. Section 7 gives guidelines for how to do this. These guidelines describe what kind of traces one should look for in the system in order to contradict the positive verdict from step 3. If such traces are found, another iteration of the procedure is performed, starting at step 2. However, if no such traces may be found, then the procedure concludes that although it is impossible to give a definitive verdict, the system S is *likely* to be compliant with D according to $\mapsto_\rho$.

The right branch of Figure 1 is symmetrical, describing the steps to be taken when $[\![D]\!] \mapsto_\rho \langle T \rangle_D$ is found not to hold in step 3.

We now continue with describing each of the ingredients of the procedure in more detail, starting with a short introduction to sequence diagrams with underspecification (but not inherent nondeterminism) and their semantic model.

3. Sequence Diagrams with Underspecification

This section provides a general introduction to sequence diagrams with underspecification (but not inherent nondeterminism, which is treated in Section 5), and their semantic model as defined in STAIRS [HHRS05]. For further details of the STAIRS semantics of sequence diagrams, we refer to [HHRS05, RHS05b] and the summary in Appendix A. This formal semantics is compliant with the semi-formal descriptions given in the UML 2.x standard [OMG10].

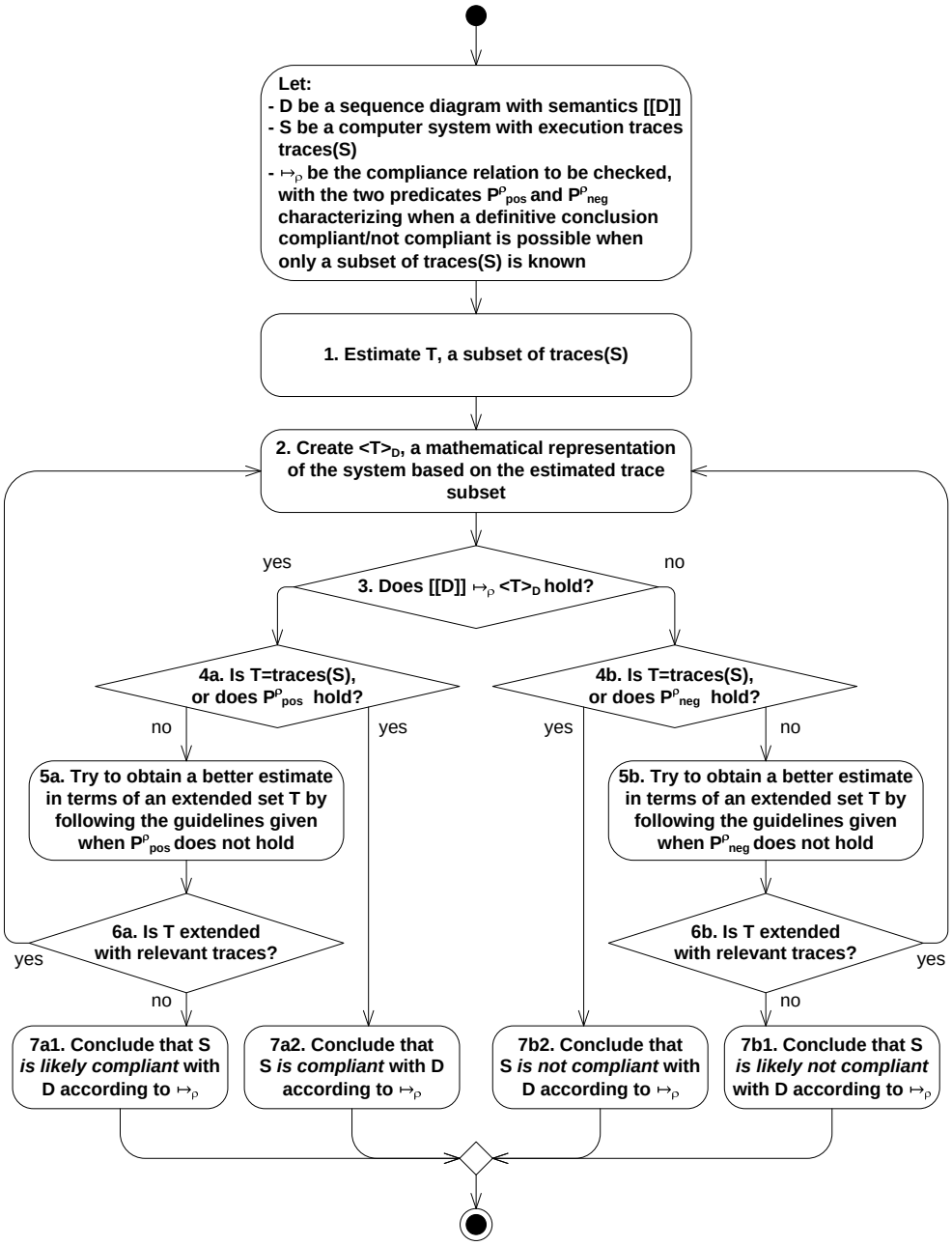

Fig. 1. Overview of the compliance checking procedure

We use the simple sequence diagram in Figure 2 to introduce some terminology. D is the name of the sequence diagram, A and B are lifelines (corresponding to e.g. components or objects), while x and y are messages from B to A. In this paper we only consider sequence diagrams where both the transmitter and the receiver lifelines are present for all messages. We say that the diagram in Figure 2 includes four events, the sending of x (denoted $!x$), the reception of x (denoted $?x$), and the sending and reception of y. A sequence diagram defines a number of traces representing system runs. For each lifeline, the events are ordered from

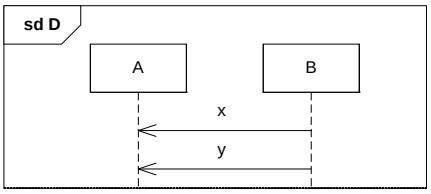

Fig. 2. Simple sequence diagram

top to bottom. In addition, a send event must occur before the corresponding receive event. The sequence diagram D defines two traces: $\langle !x, ?x, !y, ?y\rangle$ and $\langle !x, !y, ?x, ?y\rangle$.

As indicated above, the semantics of D is denoted $[\![D]\!]$. For sequence diagrams containing underspecification, but not inherent nondeterminism, the semantics is given as an *interaction obligation* (p, n) where p is a set of positive (i.e., valid) traces and n is a set of negative (i.e., invalid) traces. Positive traces represent desired or acceptable behaviour, while negative traces represent undesired or unacceptable behaviour. Traces not in the diagram are called inconclusive, and may be introduced as positive or negative by later refinement steps. Letting $\mathcal{H}$ denote the universe of all well-formed traces, the traces $\mathcal{H} \setminus (p \cup n)$ are inconclusive in the interaction obligation (p, n).

The positive traces of an interaction obligation constitute underspecification, i.e., alternative traces that those implementing the specification may choose between. This underspecification may be the result of weak sequencing of messages as in the example above (and formally defined in Appendix A.1). Underspecification may also be specified using the alt operator, formally defined by:

$$[\![D_1 \text{ alt } D_2]\!] \stackrel{\text{def}}{=} [\![D_1]\!] \uplus [\![D_2]\!] \tag{1}$$

where $\uplus$ is inner union of interaction obligations, defined by:

$$(p_1, n_1) \uplus (p_2, n_2) \stackrel{\text{def}}{=} (p_1 \cup p_2, n_1 \cup n_2) \tag{2}$$

As can be seen from these definitions, the alt operator can be used both to introduce more underspecification by combining sets of positive traces, and also to impose more restrictions by combining negative trace-sets. By taking the union also of the negative traces, the alt operator can be used to merge alternatives that are considered to be similar, both at the positive and the negative level.

Definitions of some other central composition operators may be found in Appendix A.1.

4. Relating Computer Systems to Sequence Diagrams with Underspecification

In this section we define and explain the compliance relations for relating computer systems to sequence diagrams with underspecification. First, in Section 4.1 we define how to represent a computer system in the semantic model used for sequence diagrams with underspecification as described in Section 3. This definition is used in step 2 of the compliance checking procedure in Figure 1. Section 4.2 then defines the compliance relations, while Section 4.3 presents an example of using these definitions together with the compliance checking procedure.

The refinement relations corresponding to the compliance relations in Section 4.2 are derived in Section 4.4. Section 4.5 presents important theoretical results with respect to the compliance and refinement relations in this section. Finally, Section 4.6 expands on the example from Section 4.3 to illustrate refinement and the theoretical results from Section 4.5.

In this section, we assume that the complete set of execution traces for the system is known. The additional definitions and guidelines used in the procedure when this is not the case are presented in Section 7.

4.1. System Representation

In order to check a computer system S represented by its complete set of execution traces, denoted $traces(S)$, against a sequence diagram specification containing underspecification (but not inherent nondeterminism),

$traces(S)$ must be transformed into an interaction obligation $\langle S \rangle_D$. In this interaction obligation, the traces in $traces(S)$ are the only positive ones.

For all traces, either the trace may be produced by the system or it may not. Therefore, a computer system cannot have inconclusive traces, and all relevant traces that are not in $traces(S)$ are regarded as negative. We consider the primary scope of a sequence diagram D to be the set of all lifelines in D, denoted $ll(D)$. Therefore, when checking compliance with respect to D, the relevant traces for $\langle S \rangle_D$ is taken as the set $\mathcal{H}^{ll(D)}$ (formally defined in Appendix A.1) of all well-formed traces consisting only of events taking place on the lifelines in the sequence diagram D.

This leads to the following definition of $\langle S \rangle_D$:

$$\langle S \rangle_D \stackrel{\mathsf{def}}{=} (traces(S), \mathcal{H}^{ll(D)} \setminus traces(S)) \tag{3}$$

When considering subsets of $traces(S)$, definition (3) is overloaded to trace-sets in the obvious manner, i.e.:

$$\langle T \rangle_D \stackrel{\mathsf{def}}{=} (T, \mathcal{H}^{ll(D)} \setminus T) \tag{4}$$

for T a set of well-formed traces.

4.2. Compliance

Even though sequence diagrams are partial specifications, any compliance relation for sequence diagrams must at least relate all of the traces of the sequence diagram to the execution traces of the system. As explained in Section 3, the negative traces of an interaction obligation represent undesired system behaviour, implying that they should be negative also in the system representation.

The positive traces of the interaction obligation represent underspecification, meaning that some of them may be positive and some of them may be negative in the system representation. The partial nature of sequence diagrams leads to three natural compliance relations, differing in whether the positive trace-set in the system representation is required to include an arbitrary number (zero not excluded), at least one, or nothing but positive traces of the sequence diagram.

Basic compliance 1: This is the most flexible of the compliance relations, where the execution traces of the system may contain any number of inconclusive and positive traces from the specification, without any further restrictions:

$$[\![D]\!] \mapsto_{b1} \langle S \rangle_D \stackrel{\mathsf{def}}{=} neg.[\![D]\!] \subseteq neg.\langle S \rangle_D \ \wedge \ pos.[\![D]\!] \subseteq pos.\langle S \rangle_D \cup neg.\langle S \rangle_D \tag{5}$$

where pos and neg are functions selecting the positive and negative trace-set of an interaction obligation, respectively.

Basic compliance 2: With this compliance relation, the execution traces of the system should contain at least one of the positive traces from the sequence diagram, but it may also contain an arbitrary number of inconclusive traces:

$$[\![D]\!] \mapsto_{b2} \langle S \rangle_D \stackrel{\mathsf{def}}{=} [\![D]\!] \mapsto_{b1} \langle S \rangle_D \ \wedge \ pos.[\![D]\!] \cap pos.\langle S \rangle_D \neq \emptyset \tag{6}$$

Basic compliance 3: This is the least flexible of the three compliance relations, requiring that the set of execution traces of the system includes only (some of the) positive traces, but none of the inconclusive traces, of the sequence diagram:

$$[\![D]\!] \mapsto_{b3} \langle S \rangle_D \stackrel{\mathsf{def}}{=} [\![D]\!] \mapsto_{b2} \langle S \rangle_D \ \wedge \ pos.\langle S \rangle_D \subseteq pos.[\![D]\!] \tag{7}$$

We consider the three compliance relations above to be the only reasonable ones for sequence diagrams containing underspecification and not inherent nondeterminism. A fourth interpretation of the positive traces is also feasible, requiring that *all* positive traces are to be produced by the system. However, this corresponds to what we refer to as inherent nondeterminism, and will be treated in Section 6.

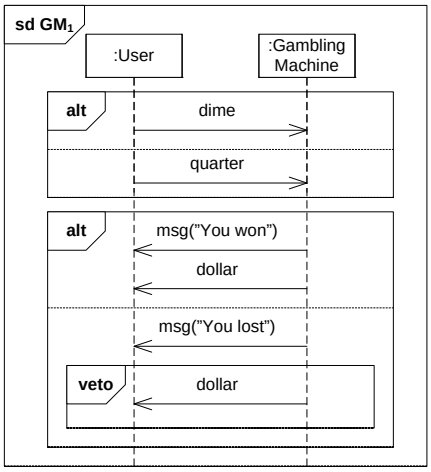

Fig. 3. Sequence diagram with underspecification (alt)

4.3. Compliance Example

As a simple example, consider the specification of a gambling machine in Figure 3. First, the machine receives either a dime or a quarter. As a result, the machine either sends the message "You won" together with a dollar, or the message "You lost".[1] The veto operator is a high-level operator for specifying negative behaviour, formally defined in Appendix A.1. In this example, veto is used to specify that the message "You lost" should *not* be followed by a dollar.

According to the definitions in Section 3 and Appendix A.1, the semantics of the sequence diagram GM_1 given in Figure 3 is an interaction obligation with six positive and four negative traces. The first operand of the second alt operator has two traces due to weak sequencing of the two messages (i.e., no ordering between the reception of "You won" and the sending of a dollar), while the second operand has only one positive trace consisting of the sending and reception of "You lost". Combined with the two traces of the first alt operator, this gives a total of four "winning" and two "losing" traces in the positive trace-set. Similarly, the weak sequencing of the two messages in the second operand of the second alt operator gives two negative traces, and a total of four negative traces when combined with the two traces of the first alt operator.

Shortening each message to only a few letters, the semantics $[\![GM_1]\!]$ may be written as:

$$
\begin{array}{l}
(\{\langle !di, ?di, !m(yw), ?m(yw), !do, ?do\rangle, \langle !qu, ?qu, !m(yw), ?m(yw), !do, ?do\rangle, \\
\quad \langle !di, ?di, !m(yw), !do, ?m(yw), ?do\rangle, \langle !qu, ?qu, !m(yw), !do, ?m(yw), ?do\rangle, \\
\quad \langle !di, ?di, !m(yl), ?m(yl)\rangle, \langle !qu, ?qu, !m(yl), ?m(yl)\rangle\} , \\
\quad \{\langle !di, ?di, !m(yl), ?m(yl), !do, ?do\rangle, \langle !qu, ?qu, !m(yl), ?m(yl), !do, ?do\rangle, \\
\quad \langle !di, ?di, !m(yl), !do, ?m(yl), ?do\rangle, \langle !qu, ?qu, !m(yl), !do, ?m(yl), ?do\rangle\})
\end{array}
$$

A possible way to implement this specification would be to build a system S_1 where the gambling machine may only receive a dime, after which it responds with a "You lost" message and then nothing more happens. This would correspond to the trace-set $traces(S_1) = \{\langle !di, ?di, !m(yl), ?m(yl)\rangle\}$.

We may now use the procedure outlined in Section 2 in order to check compliance of such a computer system S_1 to the sequence diagram GM_1.

1. In this section we assume complete knowledge of the execution traces, i.e., $T = traces(S_1) = \{\langle !di, ?di, !m(yl), ?m(yl)\rangle\}$.
2. The use of definition (4) then gives $\langle T\rangle_{GM_1} = (\{\langle !di, ?di, !m(yl), ?m(yl)\rangle\}, \mathcal{H}^{ll(GM_1)} \setminus \{\langle !di, ?di, !m(yl), ?m(yl)\rangle\})$.
3. $[\![GM_1]\!] \mapsto_\rho \langle T\rangle_{GM_1}$ holds for all three compliance relations defined in Section 4.2, as the single execution

[1] As we will come back to in Section 5, alt is not the best operator to use between these two last alternatives.

trace $\langle !di, ?di, !m(yl), ?m(yl)\rangle$ is positive in $[\![GM_1]\!]$, and the negative traces of $[\![GM_1]\!]$ are also negative in $\langle T\rangle_{GM_1}$.

4. As noted above, we have $T = traces(S_1)$, and may conclude (step 7a2 of the compliance checking procedure in Figure 1) that S_1 is compliant with GM_1 according to both $\mapsto_{b1}$, $\mapsto_{b2}$ and $\mapsto_{b3}$.

This example demonstrates that for alternatives specified as underspecification (e.g. using the alt operator), the computer system is not required to produce more than one of these. To require the system to produce all alternatives, we need the xalt operator that will be described in Section 5.

4.4. Refinement

Usually, a system is not made directly from the initial specification, but intermediate specifications are needed. Such intermediate specifications, gradually adding more information to the specification in order to bring it closer to a full description of the system, may be related by refinement relations.[2] For each of the three compliance relations in Section 4.2, we now derive a corresponding refinement relation. An important requirement is that any valid system that is compliant with the refined specification should also be compliant with the original specification.

Basic refinement 1: Basic refinement 1 of interaction obligations is derived directly from the relation basic compliance 1:

$$(p, n) \rightsquigarrow_{b1} (p', n') \stackrel{\text{def}}{=} n \subseteq n' \wedge p \subseteq p' \cup n' \tag{8}$$

As can be seen from the definition, a refinement step may add more positive and/or negative behaviours to the specification, hence reducing the set of inconclusive traces. Also, a refinement step may reduce underspecification, i.e., redefine positive traces as negative. Negative traces always remain negative.

Basic refinement 2: For basic compliance 2, the additional conjunct (compared to basic compliance 1) required a non-empty intersection between the positive traces of the sequence diagram and the positive traces of the system representation. This requirement is not directly transferable to the corresponding refinement relation, as a system in compliance with the refinement would not necessarily be in compliance with the original sequence diagram. As an example of this, assume that $[\![D]\!] = (\{t_1, t_2\}, \emptyset)$, $[\![D']\!] = (\{t_2, t_3\}, \emptyset)$ and $traces(S) = \{t_3\}$. With $p \cap p' \neq \emptyset$ as the additional requirement for basic refinement 2, we would in this example have that S complies with D' (with t_3 as the common trace) and D' refines D (with t_2 as the common trace), but S does not comply with D (as the single execution trace t_3 is not included in $[\![D]\!]$). Hence, the requirement $p \cap p' \neq \emptyset$ is not strong enough for basic refinement 2.
For $p \cap traces(S)$ to be non-empty when $p' \cap traces(S)$ is non-empty, we must instead have $p' \subseteq p$, i.e., basic refinement 2 may reduce the set of positive traces, but not redefine inconclusive traces as positive:

$$(p, n) \rightsquigarrow_{b2} (p', n') \stackrel{\text{def}}{=} (p, n) \rightsquigarrow_{b1} (p', n') \wedge p' \subseteq p \tag{9}$$

Basic refinement 3: As for basic refinement 1, basic refinement 3 may be derived directly from the definition of basic compliance 3. Note, however, that the additional requirement for basic compliance 3 is already captured by the definition of basic refinement 2 above. Hence, basic refinement 2 and 3 reduce to the same relation. For easier reference, basic refinement 3 is defined as a separate relation:

$$(p, n) \rightsquigarrow_{b3} (p', n') \stackrel{\text{def}}{=} (p, n) \rightsquigarrow_{b2} (p', n') \tag{10}$$

4.5. Theoretical Results

In this section we present a number of essential properties that are fulfilled by the refinement and compliance relations defined above. The proofs of these theorems may be found in [RRS11].

The composition operators defined in Section 3 and Appendix A.1 are all monotonic with respect to

[2] The specification may also be changed due to e.g. error correction or changed requirements. However, such changes will typically not be considered as refinements.

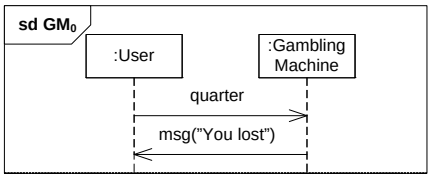

Fig. 4. Initial example run for the gambling machine

the three refinement relations above, thus ensuring compositionality in the sense that the various parts of a sequence diagram may be refined separately.

Theorem 4.1. (Monotonicity with respect to basic refinement.) For the refinement relation $\rightsquigarrow_{bx}$, with $x \in \{1, 2, 3\}$:

$$[\![D_1]\!] \rightsquigarrow_{bx} [\![D_1']\!] \wedge [\![D_2]\!] \rightsquigarrow_{bx} [\![D_2']\!] \Rightarrow [\![\mathsf{op}_1\ D_1]\!] \rightsquigarrow_{bx} [\![\mathsf{op}_1\ D_1']\!] \ \wedge \ [\![D_1\ \mathsf{op}_2\ D_2]\!] \rightsquigarrow_{bx} [\![D_1'\ \mathsf{op}_2\ D_2']\!]$$

where op_1 is any of the unary operators (e.g. veto) and op_2 any of the binary operators (e.g. alt or seq) defined in Section 3 and Appendix A.1.

If different refinement relations are used for refining different parts of the sequence diagram, the resulting diagram will at least be a basic refinement 1 of the original diagram, as basic refinement 2 (and 3) is a special case of basic refinement 1.

All refinement relations are also transitive, ensuring that the result of successive refinement steps is a valid refinement of the original sequence diagram. Again, using different refinement relations in the various steps means that at least basic refinement 1 will hold between the last sequence diagram in the refinement chain and the original sequence diagram.

Theorem 4.2. (Transitivity of basic refinement.) For the refinement relation $\rightsquigarrow_{bx}$, with $x \in \{1, 2, 3\}$:

$$[\![D]\!] \rightsquigarrow_{bx} [\![D']\!] \wedge [\![D']\!] \rightsquigarrow_{bx} [\![D'']\!] \Rightarrow [\![D]\!] \rightsquigarrow_{bx} [\![D'']\!]$$

Finally, we also have transitivity between refinement and compliance.

Theorem 4.3. (Transitivity between basic refinement and basic compliance.) For the refinement relation $\rightsquigarrow_{bx}$ and compliance relation $\mapsto_{bx}$ with $x \in \{1, 2, 3\}$:

$$[\![D]\!] \rightsquigarrow_{bx} [\![D']\!] \wedge [\![D']\!] \mapsto_{bx} \langle S \rangle_{D'} \Rightarrow [\![D]\!] \mapsto_{bx} \langle S \rangle_{D}$$

The two transitivity theorems (Theorems 4.2 and 4.3) are important as they ensure that a computer system that complies with a sequence diagram resulting from a series of refinement steps, will also comply with the original sequence diagram.

4.6. Refinement Example

Consider the sequence diagram GM_0 in Figure 4, describing an initial example run for the gambling machine, in which the machine receives a quarter and then replies with a "You lost" message. The gambling machine GM_1 in Figure 3 is a valid refinement of GM_0 according to basic refinement 1, but not according to basic refinement 2 (and 3), as can be seen by splitting the diagram in two parts and considering each part separately.

First, the quarter message in Figure 4 leads to an interaction obligation with only one positive trace (the sending and reception of the quarter), and no negative traces. In the first alt operand in Figure 3, this interaction obligation is extended with one new positive trace (the sending and reception of a dime), which was inconclusive in Figure 4. This extension is allowed by basic refinement 1, but not by basic refinement 2 (and 3).

Similarly, the "You lost" message in Figure 4 leads to an interaction obligation with one positive and no negative traces, which is extended in the second alt operand in Figure 3 with two positive and two negative traces. Again, this is allowed by basic refinement 1, but not by basic refinement 2 (and 3).

Hence, the two operands of the implicit weak sequencing operator (between the two messages) in Figure 4 are refined separately according to basic refinement 1. By the monotonicity theorem (Theorem 4.1), we may

then conclude that the sequence diagram in Figure 3 is a valid refinement of the diagram in Figure 4 by basic refinement 1 (i.e., $GM_0 \rightsquigarrow_{b1} GM_1$).

In Section 4.3, the compliance checking procedure was used to conclude that the system S_1 (with only one trace where the gambling machine receives a dime and responds with "You lost") was in compliance with the sequence diagram GM_1 in Figure 3 according to all three basic compliance relations, including $\mapsto_{b1}$ (i.e., $GM_1 \mapsto_{b1} \langle S_1 \rangle_{GM_1}$). From $GM_0 \rightsquigarrow_{b1} GM_1$ and $GM_1 \mapsto_{b1} \langle S_1 \rangle_{GM_1}$ we may conclude $GM_0 \mapsto_{b1} \langle S_1 \rangle_{GM_0}$ by Theorem 4.3, without having to use the compliance checking procedure again.

The fact that S_1 is in basic compliance 1 with GM_0 may seem a surprising result at first, as the gambling machine receives a dime in the system S_1, but a quarter in the sequence diagram specification GM_0. However, this is a natural consequence of GM_0 being only a partial specification of the gambling machine, and of the use of underspecification in the form of the first alt operator in GM_1, stating that a dime and a quarter are considered equally good alternatives as input to the gambling machine. (On the other hand, using the less flexible relations basic compliance 2 or 3, would give the result that S_1 is *not* in compliance with GM_0, as both of these relations would require the quarter-alternative from GM_0 to be reflected in the system.)

5. Sequence Diagrams with Inherent Nondeterminism

In the case of underspecification (and no inherent nondeterminism) investigated above, a system may comply with a given sequence diagram even if the system is able to produce only one of the positive traces in the diagram, and nothing else. In most cases this is of course not satisfactory, as one would like to specify a *set* of behaviours that should all be reflected in the implementation in one way or another. One example is the gambling machine from Section 4.3, where the sequence diagram allowed a system where the only possible outcome was the user losing his money. A realistic specification would be to require that both winning and losing should be possible outcomes. Also, the choice between the two should be performed nondeterministically (or at least appear so to the user of the gambling machine).

For specifying inherent nondeterminism, or alternatives that must all be reflected in the specified system, we use the xalt operator first introduced in [HS03]. The semantics of a sequence diagram D that may contain inherent nondeterminism in addition to underspecification, is no longer a single interaction obligation as in Section 3, but instead defined as a *set* of any number of interaction obligations [HHRS05]. The idea is that each interaction obligation gives a requirement that must be fulfilled by any system that should be in compliance with the sequence diagram, while the positive traces within an interaction obligation represent underspecification as before.

Formally, the semantics of the xalt operator is defined by:

$$[\![D_1 \text{ xalt } D_2]\!] \stackrel{\text{def}}{=} [\![D_1]\!] \cup [\![D_2]\!] \tag{11}$$

Hence, the composition of D_1 and D_2 by xalt requires all the inherent nondeterminism specified by D_1 in addition to all the inherent nondeterminism specified by D_2. In other words, the result of xalt-composition is the union of the sets of interaction obligations capturing the semantics of the two operands. This means that a trace may be positive in one interaction obligation and negative in another, as will be illustrated by the example in Section 6.3.

The generalized definitions for the other composition operators, including alt, may be found in Appendix A.2.

6. Relating Computer Systems to Sequence Diagrams with Inherent Nondeterminism and Underspecification

In this section we discuss how to relate computer systems to sequence diagrams with both alt and xalt, similar to what we did for sequence diagrams with only alt in Section 4. As in Section 4, we assume that the complete set of execution traces for the system is known, and leave the additional definitions and guidelines when this is not the case to Section 7.

6.1. System Representation

In order to characterize compliance between a system S and a sequence diagram D with inherent nondeterminism (as well as underspecification), we redefine $\langle S \rangle_D$ to consist of one interaction obligation for each trace in $traces(S)$:

$$\langle S \rangle_D \stackrel{\mathsf{def}}{=} \{(\{t\}, \mathcal{H}^{ll(D)} \setminus \{t\}) \mid t \in traces(S)\} \tag{12}$$

The idea is that there is no underspecification in a system. Hence, there is no need to have interaction obligations with more than one positive trace, and the system representation consists of one interaction obligation for each one of the execution traces.

When considering subsets of $traces(S)$, definition (12) is overloaded to trace-sets in the obvious manner, i.e.:

$$\langle T \rangle_D \stackrel{\mathsf{def}}{=} \{(\{t\}, \mathcal{H}^{ll(D)} \setminus \{t\}) \mid t \in T\} \tag{13}$$

for T a set of well-formed traces.

6.2. Compliance

As stated in Section 5, each interaction obligation in the semantics of a sequence diagram represents a requirement to be reflected in any system compliant with the diagram. For sequence diagrams with inherent nondeterminism, there are two natural interpretations with respect to compliance, differing in whether the system representation is required to reflect nothing but the interaction obligations of the sequence diagram, or if additional behaviours are also allowed.

From Section 4.2, we have three alternative basic compliance relations that may be used for pairwise comparison between the interaction obligations of the sequence diagram and the system representation. In principle, this leads to a total of six different compliance relations for sequence diagrams with both underspecification and inherent nondeterminism. However, with the system representation containing only one positive trace in each interaction obligation, there is no difference between basic compliance 2 and 3. Thus, we only use basic compliance 1 and 2 in the definitions below.

General compliance 1 and 2: With general compliance, every interaction obligation from the specification should be reflected in at least one of the interaction obligations in the system representation, but there is no restriction on additional interaction obligations in the system representation:

$$[\![D]\!] \mapsto_{gx} \langle S \rangle_D \stackrel{\mathsf{def}}{=} \forall o \in [\![D]\!] : \exists o' \in \langle S \rangle_D : o \mapsto_{bx} o' \tag{14}$$

for $x \in \{1, 2\}$.

Limited compliance 1 and 2: Limited compliance requires not only that all interaction obligations in the specification are reflected in the system, but also that every interaction obligation in the system representation complies with at least one interaction obligation from the specification. This limits the possibilities for additional behaviours in the system:

$$[\![D]\!] \mapsto_{lx} \langle S \rangle_D \stackrel{\mathsf{def}}{=} [\![D]\!] \mapsto_{gx} \langle S \rangle_D \ \wedge \ \forall o' \in \langle S \rangle_D : \exists o \in [\![D]\!] : o \mapsto_{bx} o' \tag{15}$$

for $x \in \{1, 2\}$.

6.3. Compliance Example

Figure 5 is a revised specification of the gambling machine from Figure 3, replacing the second alt operator with xalt and adding some more negative behaviours. The ref-construct may be understood as a syntactical shorthand for the contents of the referenced sequence diagram.

The semantics $[\![GM_2]\!]$ of this sequence diagram is a set of two interaction obligations, one for each of

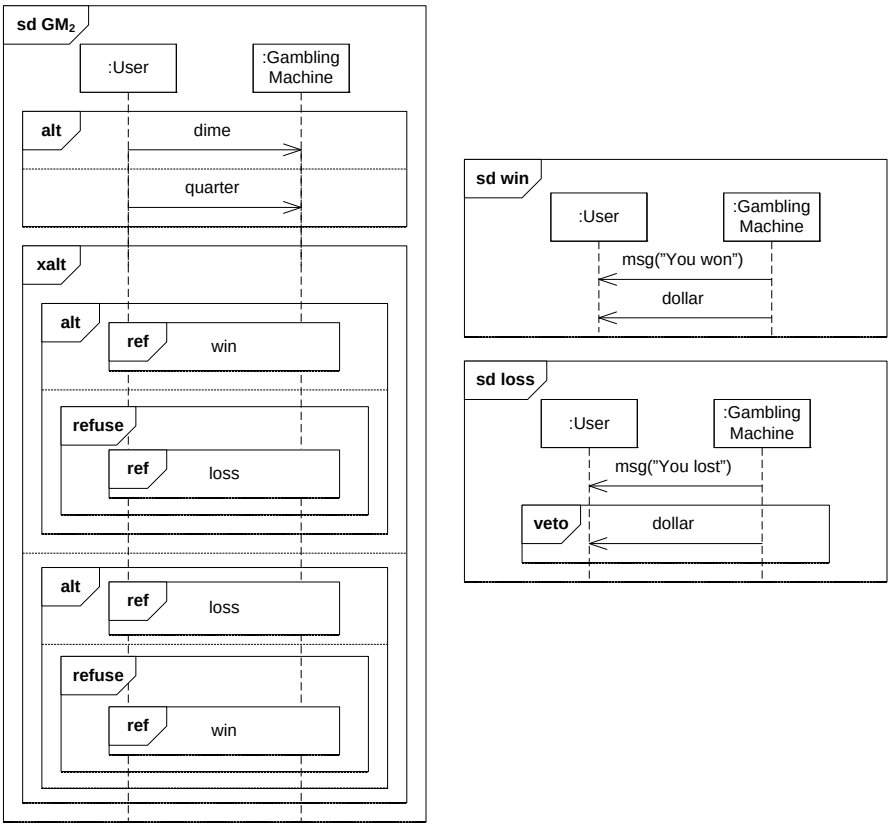

Fig. 5. Sequence diagram with inherent nondeterminism (xalt)

the two xalt operands:

$$\{\ (\ \{\langle !di, ?di, !m(yw), ?m(yw), !do, ?do\rangle, \langle !qu, ?qu, !m(yw), ?m(yw), !do, ?do\rangle,$$
$$\langle !di, ?di, !m(yw), !do, ?m(yw), ?do\rangle, \langle !qu, ?qu, !m(yw), !do, ?m(yw), ?do\rangle\}\ ,$$
$$\{\langle !di, ?di, !m(yl), ?m(yl)\rangle, \langle !qu, ?qu, !m(yl), ?m(yl)\rangle,$$
$$\langle !di, ?di, !m(yl), ?m(yl), !do, ?do\rangle, \langle !qu, ?qu, !m(yl), ?m(yl), !do, ?do\rangle,$$
$$\langle !di, ?di, !m(yl), !do, ?m(yl), ?do\rangle, \langle !qu, ?qu, !m(yl), !do, ?m(yl), ?do\rangle\}\ )\ ,$$

$$(\ \{\langle !di, ?di, !m(yl), ?m(yl)\rangle, \langle !qu, ?qu, !m(yl), ?m(yl)\rangle\}\ ,$$
$$\{\langle !di, ?di, !m(yw), ?m(yw), !do, ?do\rangle, \langle !qu, ?qu, !m(yw), ?m(yw), !do, ?do\rangle,$$
$$\langle !di, ?di, !m(yw), !do, ?m(yw), ?do\rangle, \langle !qu, ?qu, !m(yw), !do, ?m(yw), ?do\rangle,$$
$$\langle !di, ?di, !m(yl), ?m(yl), !do, ?do\rangle, \langle !qu, ?qu, !m(yl), ?m(yl), !do, ?do\rangle,$$
$$\langle !di, ?di, !m(yl), !do, ?m(yl), ?do\rangle, \langle !qu, ?qu, !m(yl), !do, ?m(yl), ?do\rangle\}\ )\ \}$$

The first of these interaction obligations comes from the first xalt operand (combined by weak sequencing with the two traces of the alt operand on top), containing four traces representing alternative "winning" outcomes, while the six traces representing "losing" outcomes are considered negative in this interaction obligation. The second interaction obligation above comes from the second xalt operand and contains two positive traces representing possible "losing" outcomes. The four "winning" traces are negative in this interaction obligation, together with the four traces representing the erroneous situation where the user loses but still gets a dollar.

The system S_1 given in Section 4.3 is not in compliance with GM_2, as can be seen by using the compliance checking procedure as follows:

1. $T = traces(S_1) = \{\langle !di, ?di, !m(yl), ?m(yl)\rangle\}$ (as in Section 4.3).

2. The use of definition (12) then gives
$\langle T \rangle_{GM_2} = \{(\{\langle !di, ?di, !m(yl), ?m(yl) \rangle\}, \mathcal{H}^{ll(GM_2)} \setminus \{\langle !di, ?di, !m(yl), ?m(yl) \rangle\})\}$.
3. $[\![\ GM_2\ ]\!] \mapsto_\rho \langle T \rangle_{GM_2}$ does not hold for any of the four compliance relations defined in Section 6.2. The single interaction obligation in $\langle T \rangle_{GM_2}$ is not in basic compliance (neither 1 nor 2) with the first interaction obligation in $[\![\ GM_2\ ]\!]$, as the given execution trace is negative in that interaction obligation. Hence, the first interaction obligation in $[\![\ GM_2\ ]\!]$ is not reflected in the system representation as required by both general and limited compliance.
4. As we have $T = traces(S_1)$, we may conclude (step 7b2 of the compliance checking procedure in Figure 1) that S_1 is not in compliance with GM_2 according to any of the four compliance relations $\mapsto_{g1}$, $\mapsto_{g2}$, $\mapsto_{l1}$ and $\mapsto_{l2}$.

Now assume a system S_2 that is similar to S_1, except that after receiving a dime, S_2 may also reply with the message "You won" and a dollar. S_2 would then be in compliance with GM_2 according to all of the compliance relations defined in this section. Again, we demonstrate the use of the compliance checking procedure for relating S_2 to GM_2:

1. The informal description of S_2 corresponds to the trace-set $traces(S_2) = \{t_1, t_2, t_3\}$, where $t_1 = \langle !di, ?di, !m(yw), ?m(yw), !do, ?do \rangle$, $t_2 = \langle !di, ?di, !m(yw), !do, ?m(yw), ?do \rangle$ and $t_3 = \langle !di, ?di, !m(yl), ?m(yl) \rangle$. As we assume complete knowledge of the execution traces, we have $T = traces(S_2)$.
2. The use of definition (12) then gives
$\langle T \rangle_{GM_2} = \{(\{t_1\}, \mathcal{H}^{ll(GM_2)} \setminus \{t_1\}), (\{t_2\}, \mathcal{H}^{ll(GM_2)} \setminus \{t_2\}), (\{t_3\}, \mathcal{H}^{ll(GM_2)} \setminus \{t_3\})\}$.
3. $[\![\ GM_2\ ]\!] \mapsto_\rho \langle T \rangle_{GM_2}$ holds for all four compliance relations defined in Section 6.2. The interaction obligation $(\{t_1\}, \mathcal{H}^{ll(GM_2)} \setminus \{t_1\})$ complies with the first interaction obligation in $[\![\ GM_2\ ]\!]$ according to both basic compliance 1 and 2. The same is the case for the interaction obligation $(\{t_2\}, \mathcal{H}^{ll(GM_2)} \setminus \{t_2\})$. Similarly, the interaction obligation $(\{t_3\}, \mathcal{H}^{ll(GM_2)} \setminus \{t_3\})$ complies with the second interaction obligation in $[\![\ GM_2\ ]\!]$ according to both basic compliance 1 and 2. Hence, both interaction obligations in $[\![\ GM_2\ ]\!]$ are reflected in an interaction obligation in $\langle T \rangle_{GM_2}$, as required by general compliance. Also, each of the three interaction obligations in $\langle T \rangle_{GM_2}$ complies with an interaction obligation in $[\![\ GM_2\ ]\!]$ as required by limited compliance.
4. As we have $T = traces(S_2)$, we may conclude (step 7a2 of the compliance checking procedure in Figure 1) that S_2 is compliant with GM_2 according to both $\mapsto_{g1}$, $\mapsto_{g2}$, $\mapsto_{l1}$ and $\mapsto_{l2}$.

6.4. Refinement

Similar to what we did for basic compliance in Section 4, we now derive four refinement relations corresponding to the four compliance relations in Section 6.2.

General refinement 1 and 2: General refinement 1 and 2 are derived directly from general compliance 1 and 2, i.e., each interaction obligation in the original sequence diagram should be refined by one of the interaction obligations in the refined sequence diagram:

$$[\![\ D\ ]\!] \rightsquigarrow_{gx} [\![\ D'\ ]\!] \stackrel{\mathsf{def}}{=} \forall o \in [\![\ D\ ]\!] : \exists o' \in [\![\ D'\ ]\!] : o \rightsquigarrow_{bx} o' \tag{16}$$

for $x \in \{1, 2\}$.
As can be seen from the definition, general refinement 1 and 2 both allow a refinement to introduce new interaction obligations that are not refinements of any interaction obligation in the original specification, possibly increasing the inherent nondeterminism required of the system. Note also that for general refinement 2 to hold, basic refinement 2 should be used for refining *all* of the interaction obligations of the original specification.

Limited refinement 1 and 2: As for general refinement, limited refinement 1 and 2 are derived directly from the definition of limited compliance 1 and 2:

$$[\![\ D\ ]\!] \rightsquigarrow_{lx} [\![\ D'\ ]\!] \stackrel{\mathsf{def}}{=} [\![\ D\ ]\!] \rightsquigarrow_{gx} [\![\ D'\ ]\!] \ \wedge\ \forall o' \in [\![\ D'\ ]\!] : \exists o \in [\![\ D\ ]\!] : o \rightsquigarrow_{bx} o'$$

for $x \in \{1, 2\}$.

Again, limited refinement 2 only holds if general refinement 2 holds and each of the interaction obligations for the refined sequence diagram is a basic refinement 2 of at least one of the interaction obligations for the original specification.

6.5. Theoretical results

For the refinement and compliance relations defined above, we have monotonicity and transitivity theorems, corresponding to Theorems 4.1–4.3 in Section 4. Again, all proofs may be found in [RRS11].

Theorem 6.1. (Monotonicity with respect to general and limited refinement.) For the refinement relation $\rightsquigarrow_\rho$, with $\rho \in \{g1, g2, l1, l2\}$:

$$[\![D_1]\!] \rightsquigarrow_\rho [\![D_1']\!] \wedge [\![D_2]\!] \rightsquigarrow_\rho [\![D_2']\!] \Rightarrow [\![\mathsf{op}_1\ D_1]\!] \rightsquigarrow_\rho [\![\mathsf{op}_1\ D_1']\!] \quad \wedge \quad [\![D_1\ \mathsf{op}_2\ D_2]\!] \rightsquigarrow_\rho [\![D_1'\ \mathsf{op}_2\ D_2']\!]$$

where op_1 is any of the unary operators (e.g. veto) and op_2 any of the binary operators (e.g. alt, xalt or seq) defined in Section 5 and Appendix A.2.

If different refinement relations are used for refining different parts of the sequence diagram, the resulting diagram will at least be a general refinement 1 of the original diagram, as general refinement 2 and limited refinement 1 and 2 are all special cases of general refinement 1.

Theorem 6.2. (Transitivity of general and limited refinement.) For the refinement relation $\rightsquigarrow_\rho$, with $\rho \in g1, g2, l1, l2$:

$$[\![D]\!] \rightsquigarrow_\rho [\![D']\!] \wedge [\![D']\!] \rightsquigarrow_\rho [\![D'']\!] \Rightarrow [\![D]\!] \rightsquigarrow_\rho [\![D'']\!]$$

Theorem 6.3. (Transitivity between general/limited refinement and general/limited compliance.) For the refinement relation $\rightsquigarrow_\rho$ and compliance relation $\mapsto_\rho$ with $\rho \in \{g1, g2, l1, l2\}$:

$$[\![D]\!] \rightsquigarrow_\rho [\![D']\!] \wedge [\![D']\!] \mapsto_\rho \langle S \rangle_{D'} \Rightarrow [\![D]\!] \mapsto_\rho \langle S \rangle_D$$

Finally, the following Theorem 6.4 states that for sequence diagrams without any xalt operator (i.e., without inherent nondeterminism), we have the natural correspondences between the compliance relations in Section 4 for sequence diagrams with underspecification, and the compliance relations in Section 6 for sequence diagrams that also allows inherent nondeterminism.

Until now we have overloaded the notation for the semantic representation of diagrams and computer systems in order to enhance readability. It is now necessary to introduce the full notation. Let $[\![D]\!]^b$ and $[\![D]\!]^g$ denote the semantics of the sequence diagram D when interpreted according to the definitions in Section 3 and Section 5, respectively. Similarly, for a system S we use $\langle S \rangle_D^b$ and $\langle S \rangle_D^g$ to denote its semantic representation with respect to D according to definition (3) and definition (12), respectively.

Theorem 6.4. (Correspondence.) For a sequence diagram D without xalt:

$$[\![D]\!]^b \mapsto_{b1} \langle S \rangle_D^b \Leftrightarrow [\![D]\!]^g \mapsto_{l1} \langle S \rangle_D^g \tag{a}$$

$$\wedge$$

$$[\![D]\!]^b \mapsto_{b3} \langle S \rangle_D^b \Leftrightarrow [\![D]\!]^g \mapsto_{l2} \langle S \rangle_D^g \tag{b}$$

Theorem 6.4 states that for sequence diagrams without xalt, basic compliance 1 and limited compliance 1 are always in accordance with each other, as are basic compliance 3 and limited compliance 2.

From Theorem 6.4 and the definitions of basic compliance, it follows that basic compliance 2 is positioned in between limited compliance 1 and 2 with respect to how much it restricts the system. Basic compliance 2 allows the system representation to include traces that are inconclusive in the sequence diagram, something that is not allowed by limited compliance 2. On the other hand, limited compliance 1 allows the system representation to include *only* traces that are inconclusive in the sequence diagram, while basic compliance 2 requires the system representation to also include at least one positive trace from the sequence diagram.

Also, general compliance allows more implementations than basic compliance. This is because general compliance interprets the partiality of sequence diagrams very flexible, allowing the system representation to contain additional interaction obligations that are not refinements of any of the interaction obligations for

the sequence diagram. In particular, the system representation may include an interaction obligation with a trace t as positive, even if t is negative in all of the original interaction obligations. However, implementing a negative trace is not allowed by basic compliance, where a single interaction obligation is the semantic model used for representing both the sequence diagram and the system.

6.6. Refinement and Correspondence Example

The gambling machine specification GM_2 in Figure 5 is a valid refinement of the specification GM_1 in Figure 3 according to all four refinement relations defined in Section 6.4, as all of the positive behaviours of GM_2 are positive also for GM_1, the remaining positive behaviours of GM_1 are negative in both interaction obligations for GM_2, and the negative behaviours of GM_1 remain negative in both interaction obligations for GM_2.

In Section 6.3, the compliance checking procedure was used to conclude that the system S_2 (which receives a dime and then responds with either "You lost" or with "You won" and a dollar) was in compliance with the sequence diagram GM_2 in Figure 5 according to all four compliance relations in Section 6.2. By transitivity between refinement and compliance (Theorem 6.3), we may then conclude that S_2 is also in compliance with GM_1 according to all four compliance relations $\mapsto_{g1}$, $\mapsto_{g2}$, $\mapsto_{l1}$ and $\mapsto_{l2}$, .

As GM_1 does not include any xalt operators, the correspondence results in Theorem 6.4 may be used to establish that S_2 is in compliance with GM_1 also according to the three basic compliance relations in Section 4.2, without having to use the compliance checking procedure again. $\mapsto_{b1}$ and $\mapsto_{b3}$ follows directly from the theorem, while $\mapsto_{b2}$ follows from Theorem 6.4(b) and the definition of $\mapsto_{b3}$.

7. Exploiting the Theory in Practice

In the previous sections, we have assumed that the complete set of execution traces for the system is known. However, as explained in the introduction, one will often be in the situation where this is not the case and only a finite subset of the traces is available. When the compliance checking in step 3 of the procedure in Figure 1 is based on an incomplete system representation, the result is not automatically valid for the system itself. A system may comply with the specification even if compliance does not hold for the incomplete system representation, and vice versa.

In this section we define the predicates to be used in step 4 of the compliance checking procedure in order to determine whether a definitive conclusion may be reached although only a subset of the execution traces is known. We also present the guidelines used in step 5 of the procedure when trying to extend the set of known execution traces. These are traces trying to contradict the verdict given in step 3 of the procedure. Even in cases where a definitive answer in practice cannot be given because one needs to know the complete set of traces, an inability to find such traces as described in the relevant guidelines should at least result in an increased confidence in the procedure verdict.

For the compliance relations defined in Sections 4.2 and 6.2, the concrete predicates and guidelines are presented in Sections 7.2 and 7.3, respectively. But first, in Section 7.1, we explain the formal role of the predicates in more detail.

7.1. Soundness and Completeness Criteria

Step 4 of the compliance checking procedure uses one of two different predicates, P_{pos} or P_{neg}, depending on the result of the compliance checking in step 3. For each compliance relation $\mapsto_\rho$, the predicate $P^\rho_{pos}(D,T)$ is defined so that it ensures soundness in the sense that when $\mapsto_\rho$ holds between the semantics of the sequence diagram D and the system representation based on the trace-(sub)set T, then it is sufficient to show that $P^\rho_{pos}(D,T)$ is true in order to conclude that any system that may produce all of the traces in T will indeed comply with the sequence diagram. Formally, this means that $P^\rho_{pos}(D,T)$ fulfils the following criterion:

$$[\![\ D\ ]\!]^x \mapsto_\rho \langle T\rangle^x_D \wedge P^\rho_{pos}(D,T) \Rightarrow \forall S : (T \subseteq \mathit{traces}(S) \Rightarrow [\![\ D\ ]\!]^x \mapsto_\rho \langle S\rangle^x_D) \tag{17}$$

where $x = b$ and $\rho \in \{b1, b2, b3\}$, or $x = g$ and $\rho \in \{g1, g2, l1, l2\}$.

Also, $P^\rho_{pos}(D,T)$ ensures completeness in the sense that if $P^\rho_{pos}(D,T)$ is false when $\mapsto_\rho$ holds, then there

exists at least one system that is not compliant with the sequence diagram even though the system is able to produce all of the traces in T. I.e., $P^{\rho}_{pos}(D,T)$ fulfils the following criterion:

$$[\![\ D\ ]\!]^x \mapsto_\rho \langle T\rangle^x_D \wedge \neg P^{\rho}_{pos}(D,T) \Rightarrow \exists S : (T \subseteq \mathit{traces}(S) \wedge [\![\ D\ ]\!]^x \not\mapsto_\rho \langle S\rangle^x_D) \tag{18}$$

where $x = b$ and $\rho \in \{b1, b2, b3\}$, or $x = g$ and $\rho \in \{g1, g2, l1, l2\}$.

Similarly, P^{ρ}_{neg} is defined to ensure soundness and completeness such that a negative verdict for $\mapsto_\rho$ in step 3 is guaranteed to be correct if and only if $P^{\rho}_{neg}(D,T)$ is true:

$$[\![\ D\ ]\!]^x \not\mapsto_\rho \langle T\rangle^x_D \wedge P^{\rho}_{neg}(D,T) \Rightarrow \forall S : (T \subseteq \mathit{traces}(S) \Rightarrow [\![\ D\ ]\!]^x \not\mapsto_\rho \langle S\rangle^x_D) \tag{19}$$

$$[\![\ D\ ]\!]^x \not\mapsto_\rho \langle T\rangle^x_D \wedge \neg P^{\rho}_{neg}(D,T) \Rightarrow \exists S : (T \subseteq \mathit{traces}(S) \wedge [\![\ D\ ]\!]^x \mapsto_\rho \langle S\rangle^x_D) \tag{20}$$

where $x = b$ and $\rho \in \{b1, b2, b3\}$, or $x = g$ and $\rho \in \{g1, g2, l1, l2\}$.

In the following sections, we go through each of the compliance relations defined in this paper and give theorems stating exactly the conditions that $\mathit{traces}(S)$ must fulfil in order to satisfy each relation. For each relation $\mapsto_\rho$, the predicates $P^{\rho}_{pos}(D,T)$ and $P^{\rho}_{neg}(D,T)$ are derived from the corresponding theorems. Proofs that these predicates satisfy criteria (17)–(20), as well as the proofs of the theorems, may be found in [RRS11].

7.2. Predicates and Guidelines for Sequence Diagrams with Underspecification

In this section, we present the predicates and corresponding guidelines to be used when working with the basic compliance relations defined in Section 4.2, i.e., for relating computer systems to sequence diagrams containing underspecification but not inherent nondeterminism.

7.2.1. Predicates and Guidelines for Basic Compliance 1

Theorem 7.1 (Condition for $\mapsto_{b1}$).

$$\mathit{neg}.[\![\ D\ ]\!]^b \cap \mathit{traces}(S) = \emptyset \Leftrightarrow [\![\ D\ ]\!]^b \mapsto_{b1} \langle S\rangle^b_D$$

Theorem 7.1 tells us that S complies with D according to $\mapsto_{b1}$ if and only if the set of execution traces (i.e., $\mathit{traces}(S)$) does not include any of the traces specified as negative (i.e., in $\mathit{neg}.[\![\ D\ ]\!]^b$). This has two important consequences. Firstly, with only a subset of $\mathit{traces}(S)$ at hand, this requirement can be established with certainty only if the specification has no negative traces. With an empty set of negative traces, $\mathit{neg}.[\![\ D\ ]\!]^b \cap \mathit{traces}(S) = \emptyset$ is trivially true and basic compliance 1 holds for any system S. Hence:

$$P^{b1}_{pos}(D,T) \stackrel{\mathsf{def}}{=} \mathit{neg}.[\![\ D\ ]\!]^b = \emptyset \tag{21}$$

Secondly, Theorem 7.1 implies that for $\mapsto_{b1}$, compliance is broken if a trace specified as negative by D is found among the execution traces. As a consequence, if basic compliance 1 holds for the subset T of execution traces, but P^{b1}_{pos} does not hold, then one should try to extend T with traces that are specified as negative. If it is revealed that the system may produce one such trace, basic compliance 1 does not hold. On the other hand, if no negative trace may be found in the system, compliance can be assumed with high confidence (although it can never be guaranteed without knowledge about the complete set of execution traces).

Theorem 7.2 (Certainty of negative verdicts using $\mapsto_{b1}$).

$$T \subseteq \mathit{traces}(S) \wedge [\![\ D\ ]\!]^b \not\mapsto_{b1} \langle T\rangle^b_D \Rightarrow [\![\ D\ ]\!]^b \not\mapsto_{b1} \langle S\rangle^b_D$$

Theorem 7.2 states that if basic compliance 1 does not hold for a subset of the execution traces, then no additional trace is able to change this into a positive verdict. This is an example of the ideal situation, where the procedure verdict in step 3 is guaranteed to be correct without any additional conditions. Consequently, no additional guidelines for step 5 are needed, and we get:

$$P^{b1}_{neg}(D,T) \stackrel{\mathsf{def}}{=} \mathit{true} \tag{22}$$

	Positive verdict in procedure step 3	**Negative verdict in procedure step 3**
Formal predicate	P^{b1}_{pos}: $neg.[\![D]\!]^b = \emptyset$ P^{b2}_{pos}: $neg.[\![D]\!]^b = \emptyset$ P^{b3}_{pos}: $pos.[\![D]\!]^b \setminus neg.[\![D]\!]^b = \mathcal{H}$	P^{b1}_{neg}: *true* P^{b2}_{neg}: $[\![D]\!]^b \not\mapsto_{b1} \langle T \rangle^b_D \ \vee \ pos.[\![D]\!]^b \setminus neg.[\![D]\!]^b = \emptyset$ P^{b3}_{neg}: $pos.[\![D]\!]^b \setminus neg.[\![D]\!]^b = \emptyset \ \vee \ T \neq \emptyset$
Informal explanation of predicate	P^{b1}_{pos}: Definitive if D specifies no negative traces. P^{b2}_{pos} : Definitive if D specifies no negative traces. P^{b3}_{pos}: Definitive if D specifies all possible traces as truly positive.	P^{b1}_{neg}: Definitive. P^{b2}_{neg} : Definitive if $\mapsto_{b1}$ does not hold or D specifies no truly positive traces. P^{b3}_{neg}: Definitive if D specifies no truly positive traces or at least one execution trace is known.
Guidelines for procedure step 5	P^{b1}_{pos}: Look for behaviour that is specified as negative by D. P^{b2}_{pos}: Look for traces that are specified as negative by D. P^{b3}_{pos}: Look for traces that are specified as negative or inconclusive by D.	P^{b1}_{neg}: *(No guideline needed.)* P^{b2}_{neg}: Look for traces that are specified as truly positive by D. P^{b3}_{neg}: Look for any execution trace.

Table 1. Overview of predicates and guidelines for basic compliance 1, 2 and 3

7.2.2. Predicates and Guidelines for Basic Compliance 2

Theorem 7.3 (Condition for $\mapsto_{b2}$).

$$pos.[\![D]\!]^b \cap traces(S) \neq \emptyset \wedge neg.[\![D]\!]^b \cap traces(S) = \emptyset \Leftrightarrow [\![D]\!]^b \mapsto_{b2} \langle S \rangle^b_D$$

Theorem 7.3 states that S complies with D according to basic compliance 2 if and only if the following two requirements are fulfilled:

1. the set of execution traces (i.e., $traces(S)$) includes at least one trace specified as positive by D (i.e., in $pos.[\![D]\!]^b$).
2. the set of execution traces does not include any trace specified as negative by D (this is the same requirement as for $\mapsto_{b1}$).

If the first requirement holds for a subset of the execution traces, then obviously it will also hold for the complete set of traces. This means that a positive procedure verdict for $\mapsto_{b2}$ in step 3 is guaranteed to be correct for the system S in exactly the same cases as for $\mapsto_{b1}$, i.e.:

$$P^{b2}_{pos}(D,T) \ \stackrel{\mathsf{def}}{=} \ neg.[\![D]\!]^b = \emptyset \tag{23}$$

Theorem 7.3 also tells us that there are two ways in which S may *not* be in basic compliance 2 with D. The first possibility is that the set of execution traces includes at least one trace specified as negative by D (i.e., $neg.[\![D]\!]^b \cap traces(S) \neq \emptyset$). From Section 7.2.1, we know that in this case, basic compliance 1 will not hold either, and the procedure verdict in step 3 is guaranteed to be correct.

The second possibility of non-compliance with respect to $\mapsto_{b2}$ is if the set of execution traces does not include any of the traces specified as positive by D. However, without access to the complete set of execution traces, the system may include a trace specified as positive by D even if such a trace is not found in the given subset T. As a result, if basic compliance 2 does not hold for T, then this verdict is guaranteed to be correct for the system S only if D specifies no truly positive traces (i.e., traces that are positive without also being negative), in which case $\mapsto_{b2}$ cannot possibly hold for any system at all. As a result, we get:

$$P^{b2}_{neg}(D,T) \ \stackrel{\mathsf{def}}{=} \ [\![D]\!]^b \not\mapsto_{b1} \langle T \rangle^b_D \ \vee \ pos.[\![D]\!]^b \setminus neg.[\![D]\!]^b = \emptyset \tag{24}$$

If P^{b2}_{neg} does not hold, one should try to extend T with traces specified as truly positive by D. If no such trace is found, then basic compliance 2 most likely does not hold. On the other hand, if one truly positive trace is revealed, the procedure verdict in step 3 will change to positive. The situation will then be as described above, where one should look for traces specified as negative by D in order to obtain increased confidence in the positive verdict.

7.2.3. Predicates and Guidelines for Basic Compliance 3

Theorem 7.4 (Condition for $\mapsto_{b3}$).

$$traces(S) \subseteq pos.[\![D]\!]^b \setminus neg.[\![D]\!]^b \Leftrightarrow [\![D]\!]^b \mapsto_{b3} \langle S \rangle_D^b$$

Theorem 7.4 states that S complies with D according to basic compliance 3 if and only if all execution traces (i.e., $traces(S)$) are specified as truly positive by D. With only a subset of $traces(S)$ at hand, this requirement can only be established with certainty if every possible trace (i.e., all traces in $\mathcal{H}$) is specified as truly positive by D, in which case $traces(S) \subseteq pos.[\![D]\!]^b \setminus neg.[\![D]\!]^b$ will be trivially true for all systems S. Hence:

$$P_{pos}^{b3}(D,T) \stackrel{\mathsf{def}}{=} pos.[\![D]\!]^b \setminus neg.[\![D]\!]^b = \mathcal{H} \tag{25}$$

If P_{pos}^{b3} does not hold, one should try to extend T with traces that are not specified as truly positive by D, i.e., traces that D specifies as negative or inconclusive. If no such trace is found, then basic compliance 3 can be assumed with high confidence. On the other hand, if one negative or inconclusive trace is found, the procedure verdict in step 3 will change to negative. As will be described next, this verdict is guaranteed to be correct, and the system S will not be in basic compliance 3 with D.

Theorem 7.4 implies that if basic compliance 3 does not hold, this will be because the set of execution traces is not a subset of the traces specified as truly positive by D. This will always be the case if no traces are specified as truly positive by D, in which case $traces(S) \not\subseteq (pos.[\![D]\!]^b \setminus neg.[\![D]\!]^b)$ will be trivially true as any real system S has at least one trace. Also, if basic compliance 3 is found not to hold for a non-empty subset of the execution traces, this will be because at least one of those traces are not specified as truly positive by D. Hence, the system S is known to include a trace that is not allowed by D, and basic compliance 3 cannot hold. Together, this gives:

$$P_{neg}^{b3}(D,T) \stackrel{\mathsf{def}}{=} pos.[\![D]\!]^b \setminus neg.[\![D]\!]^b = \emptyset \vee T \neq \emptyset \tag{26}$$

In practice, P_{neg}^{b3} will always hold as the second disjunct ($T \neq \emptyset$) gives that it is sufficient to know at least one of the execution traces to conclude that a negative procedure verdict is correct also for the complete system. As a result, if step 3 of the compliance checking procedure gives a negative verdict for basic compliance 3, the conclusion will in practice always be that the system S is not in basic compliance 3 with D.

7.3. Predicates and Guidelines for Sequence Diagrams with Inherent Nondeterminism and Underspecification

Similar to what we did for basic compliance in the previous section, we now present the predicates and guidelines to be used when working with the general and limited compliance relations defined in Section 6.2, i.e., the compliance relations that may be used for relating computer systems to sequence diagrams containing inherent nondeterminism in addition to underspecification.

7.3.1. Predicates and Guidelines for General Compliance 1

Theorem 7.5 (Certainty of positive verdicts using $\mapsto_{g1}$).

$$T \subseteq traces(S) \wedge [\![D]\!]^g \mapsto_{g1} \langle T \rangle_D^g \Rightarrow [\![D]\!]^g \mapsto_{g1} \langle S \rangle_D^g$$

Theorem 7.5 states that in order to show that S complies with D according to general compliance 1, it is sufficient to find a subset T of $traces(S)$ such that this subset complies with D. This means that with a positive verdict for $\mapsto_{g1}$ in step 3 of the procedure, we are again in an ideal situation where this verdict is guaranteed to be correct even when only a subset of the execution traces is known. This is due to the fact that general compliance 1 interprets the partiality of sequence diagrams to mean that as long as S reflects every interaction obligation in D, any additional behaviour is also allowed. Thus, no additional guidelines for step 5 are needed, and we get:

$$P_{pos}^{g1}(D,T) \stackrel{\mathsf{def}}{=} true \tag{27}$$

	Positive verdict in procedure step 3	**Negative verdict in procedure step 3**
Formal predicate	P^{g1}_{pos}: *true* P^{g2}_{pos}: *true*	P^{g1}_{neg}: $\exists(p,n) \in [\![\ D\ ]\!]^g : n = \mathcal{H}$ P^{g2}_{neg}: $\exists(p,n) \in [\![\ D\ ]\!]^g : p \setminus n = \emptyset$
Informal explanation of predicate	P^{g1}_{pos}: Definitive. P^{g2}_{pos}: Definitive.	P^{g1}_{neg}: Definitive if there exists an interaction obligation for D where all traces are specified as negative. P^{g2}_{neg}: Definitive if there exists an interaction obligation for D where no traces are specified as truly positive.
Guidelines for procedure step 5	P^{g1}_{pos}: *(No guideline needed.)* P^{g2}_{pos}: *(No guideline needed.)*	P^{g1}_{neg}: For each interaction obligation for D, look for traces that are specified as non-negative. P^{g2}_{neg}: For each interaction obligation for D, look for traces that are specified as truly positive.

Table 2. Overview of predicates and guidelines for general compliance 1 and 2

Theorem 7.6 (Condition for $\not\mapsto_{g1}$).

$$\exists(p,n) \in [\![\ D\ ]\!]^g : \mathit{traces}(S) \subseteq n \Leftrightarrow [\![\ D\ ]\!]^g \not\mapsto_{g1} \langle S \rangle^g_D$$

Theorem 7.6 states that in order to show that S does *not* comply with D according to general compliance 1, it is necessary and sufficient to show that there exists an interaction obligation (p,n) in $[\![\ D\ ]\!]^g$ where all of the execution traces (i.e., $\mathit{traces}(S)$) are specified as negative. However, this can only be established from a subset of the execution traces if there exists an interaction obligation where every possible trace is specified as negative. If such an interaction obligation exists, $\mathit{traces}(S) \subseteq n$ will be trivially true for any system S and general compliance 1 can never hold. Hence:

$$P^{g1}_{neg}(D,T) \stackrel{\mathsf{def}}{=} \exists(p,n) \in [\![\ D\ ]\!]^g : n = \mathcal{H} \tag{28}$$

When P^{g1}_{neg} does not hold, one should focus on each of the interaction obligations for the specification, one at a time, and try to extend T with traces that are non-negative in that obligation. If it is possible to find one non-negative trace for each interaction obligation in the specification, the verdict in step 3 of the procedure will change to positive, which by Theorem 7.5 is guaranteed to be correct for any system S having T among its set of execution traces.

7.3.2. Predicates and Guidelines for General Compliance 2

Theorem 7.7 (Certainty of positive verdicts using $\mapsto_{g2}$).

$$T \subseteq \mathit{traces}(S) \wedge [\![\ D\ ]\!]^g \mapsto_{g2} \langle T \rangle^g_D \Rightarrow [\![\ D\ ]\!]^g \mapsto_{g2} \langle S \rangle^g_D$$

As in the case of general compliance 1, general compliance 2 requires the system to be able to produce some specific traces, but any other execution trace is allowed as well. This is captured in Theorem 7.7, which states that if a subset of the execution traces results in a positive verdict for $\mapsto_{g2}$ in step 3 of the procedure, then the existence of additional execution traces is of no importance, the conclusion will always be that S is in general compliance 2 with D. Again we are in the ideal situation, where no additional guideline is needed and we get:

$$P^{g2}_{pos}(D,T) \stackrel{\mathsf{def}}{=} \mathit{true} \tag{29}$$

Theorem 7.8 (Condition for $\not\mapsto_{g2}$).

$$\exists(p,n) \in [\![\ D\ ]\!]^g : \mathit{traces}(S) \cap (p \setminus n) = \emptyset \Leftrightarrow [\![\ D\ ]\!]^g \not\mapsto_{g2} \langle S \rangle^g_D$$

From Theorem 7.8 we get that in order to show that S does *not* comply with D according to general compliance 2, it is necessary and sufficient to find an interaction obligation in $[\![\ D\ ]\!]^g$ where none of the execution traces are specified as truly positive (i.e., positive without also being negative). With only a subset of the execution traces available, this can only be established with certainty if the obligation specifies no truly positive traces (i.e., $p \setminus n = \emptyset$), in which case none of the additional execution traces will ever be able

to contradict $traces(S) \cap (p \setminus n) = \emptyset$. (This situation is similar to the second possibility of non-compliance for basic compliance 2 in Section 7.2.2). Hence:

$$P^{g2}_{neg}(D,T) \stackrel{\mathsf{def}}{=} \exists (p,n) \in [\![D]\!]^g : p \setminus n = \emptyset \tag{30}$$

When P^{g2}_{neg} does not hold, one should focus on the interaction obligations for D, one at a time, and look for traces that are specified as truly positive in that obligation. If it is revealed that the execution traces include at least one truly positive trace from each of the interaction obligations in D, the verdict for $\mapsto_{g2}$ in step 3 will change to positive. By Theorem 7.7, this verdict will then be assuredly correct even if the complete set of execution traces is still not known.

7.3.3. Predicates and Guidelines for Limited Compliance 1

Theorem 7.9 (Condition for $\mapsto_{l1}$).

$$[\![D]\!]^g \mapsto_{g1} \langle S \rangle^g_D \wedge traces(S) \cap \bigcap_{(p,n) \in [\![D]\!]^g} n = \emptyset \Leftrightarrow [\![D]\!]^g \mapsto_{l1} \langle S \rangle^g_D$$

Theorem 7.9 states that S complies with D according to limited compliance 1 if and only if the following two requirements are fulfilled:

1. S complies with D according to general compliance 1.
2. the set of execution traces does not include any trace specified by D as globally negative (i.e., negative in all interaction obligations).

From Section 7.3.1 we know that if general compliance 1 holds for a subset of the execution traces, it will also hold for the system S. For the second requirement above, this can only be established from a proper subset of the execution traces if D specifies no globally negative traces, in which case $traces(S) \cap \bigcap_{(p,n) \in [\![D]\!]^g} n = \emptyset$ will be trivially true. Consequently:

$$P^{l1}_{pos}(D,T) \stackrel{\mathsf{def}}{=} \bigcap_{(p,n) \in [\![D]\!]^g} n = \emptyset \tag{31}$$

If P^{l1}_{pos} does not hold, one should try to extend T with traces that are specified as globally negative by D. If it is revealed that the system S may produce one such trace, the verdict in step 3 will change to negative, and in this case, the negative verdict is guaranteed to be correct for S as explained below.

Theorem 7.9 also tells us that there are two cases that result in a negative procedure verdict for limited compliance 1. The first case is when general compliance 1 does not hold either. In this situation, we know from Section 7.3.1 that the verdict in step 3 is guaranteed to be correct only if there exists an interaction obligation for the specification where all possible traces are specified as negative. The second case leading to a negative procedure verdict for $\mapsto_{l1}$ is when one of the known execution traces is specified as globally negative by D. Together, this gives:

$$P^{l1}_{neg}(D,T) \stackrel{\mathsf{def}}{=} T \cap \bigcap_{(p,n) \in [\![D]\!]^g} n \neq \emptyset \vee \exists (p,n) \in [\![D]\!]^g : n = \mathcal{H} \tag{32}$$

If P^{l1}_{neg} does not hold, and the verdict in step 3 is negative because general compliance 1 does not hold for the given subset of execution traces, the guideline provided for a negative verdict for $\mapsto_{g1}$ in Table 2 should be used in order to increase the confidence in the negative verdict or, alternatively, achieve a positive verdict for $\mapsto_{g1}$ instead (which we know from Section 7.3.1 is guaranteed to be correct).

If, on the other hand, P^{l1}_{neg} does not hold and the verdict in step 3 is negative due to the second case above (regardless of whether $\mapsto_{g1}$ holds or not), any system S containing that particular globally negative trace as one of its execution traces will be non-compliant with the specification according to $\mapsto_{l1}$. In this situation, then, no additional guideline is needed.

	Positive verdict in procedure step 3	**Negative verdict in procedure step 3**
Formal predicate	P^{l1}_{pos}: $\bigcap_{(p,n)\in[\![\ D\ ]\!]^g} n = \emptyset$ P^{l2}_{pos}: $\bigcup_{(p,n)\in[\![\ D\ ]\!]^g} (p \setminus n) = \mathcal{H}$	P^{l1}_{neg}: $T \cap \bigcap_{(p,n)\in[\![\ D\ ]\!]^g} n \neq \emptyset \ \vee\ \exists (p,n) \in [\![\ D\ ]\!]^g : n = \mathcal{H}$ P^{l2}_{neg}: $T \not\subseteq \bigcup_{(p,n)\in[\![\ D\ ]\!]^g} (p \setminus n) \ \vee\ \exists (p,n) \in [\![\ D\ ]\!]^g : p \setminus n = \emptyset$
Informal explanation of predicate	P^{l1}_{pos}: Definitive if D specifies no globally negative traces. P^{l2}_{pos}: Definitive if every possible trace is specified as truly positive in at least one interaction obligation for D.	P^{l1}_{neg}: Definitive if the known subset T contains a trace specified by D as globally negative, or if there exists an interaction obligation for D where all traces are specified as negative. P^{l2}_{neg}: Definitive if the known subset T contains a trace that is not truly positive in any interaction obligation for D, or there exists an obligation where no traces are truly positive.
Guidelines for procedure step 5	P^{l1}_{pos}: Look for traces that are specified as globally negative by D. P^{l2}_{pos}: Look for traces that are not specified as positive in any interaction obligation for D.	P^{l1}_{neg}: For each interaction obligation for D, look for traces that are non-negative. P^{l2}_{neg}: For each interaction obligation for D, look for traces that are specified as truly positive.

Table 3. Overview of predicates and guidelines for limited compliance 1 and 2

7.3.4. Predicates and Guidelines for Limited Compliance 2

Theorem 7.10 (Condition for $\mapsto_{l2}$).

$$[\![\ D\ ]\!]^g \mapsto_{g2} \langle S \rangle^g_D \wedge \mathit{traces}(S) \subseteq \bigcup_{(p,n)\in[\![\ D\ ]\!]^g} (p \setminus n) \Leftrightarrow [\![\ D\ ]\!]^g \mapsto_{l2} \langle S \rangle^g_D$$

Theorem 7.10 states that S complies with D according to limited compliance 2 if and only if the following two requirements are fulfilled:

1. S complies with D according to general compliance 2.
2. each of the execution traces are specified by D as truly positive in at least one interaction obligation (not necessarily the same).

From Section 7.3.2 we know that general compliance 2 is guaranteed to hold for the system S if it holds for a subset of the execution traces. However, without knowing the complete set of execution traces, the second requirement above can only be established if every possible trace is specified by D as positive in at least one interaction obligation. Consequently:

$$P^{l2}_{pos}(D,T) \stackrel{\mathsf{def}}{=} \bigcup_{(p,n)\in[\![\ D\ ]\!]^g} (p \setminus n) = \mathcal{H} \tag{33}$$

If P^{l2}_{pos} does not hold, one should try to extend T with traces that are not specified by D as truly positive in any of the interaction obligations. If it is revealed that the system is able to produce one such trace, the verdict in step 3 will change to negative, which in this case is guaranteed to be correct (see below).

Theorem 7.10 also tells us that there are two cases that result in a negative verdict for limited compliance 2 in step 3 of the procedure. The first case is when general compliance 2 also results in a negative verdict in step 3, which we know from Section 7.3.2 is guaranteed to be correct only if there exists an interaction obligation for D where no traces are specified as truly positive. The second case leading to a negative verdict for $\mapsto_{l2}$ is when one of the known execution traces is not specified as truly positive in any of the interaction obligations in D. Together, this gives:

$$P^{l2}_{neg}(D,T) \stackrel{\mathsf{def}}{=} T \not\subseteq \bigcup_{(p,n)\in[\![\ D\ ]\!]^g} (p \setminus n) \vee \exists (p,n) \in [\![\ D\ ]\!]^g : p \setminus n = \emptyset \tag{34}$$

.

In the same manner as for limited compliance 1, if P^{l2}_{neg} does not hold and the verdict for limited compliance 2 in step 3 is negative due to the fact that general compliance 2 does not hold for the given subset of execution traces, the guideline provided for a negative verdict for $\mapsto_{g2}$ in Table 2 should be used.

If P^{l2}_{neg} does not hold and the verdict is negative due to one of the subset traces not being specified as truly positive in any of the interaction obligations for D (regardless of whether $\mapsto_{g2}$ holds or not), the system S is guaranteed to be non-compliant with D according to $\mapsto_{l2}$ and no additional guideline is needed.

8. Related Work

To the best of our knowledge, there is no other paper treating the relationship between computer systems and sequence diagrams as thoroughly as we do in this paper. The closest is the work by Cengarle and Knapp in [CK04], which defines a notion of implementation that has inspired our notion of basic compliance 2. In contrast to all of the refinement notions given in this paper, [CK04] allows positive traces to become inconclusive. However, this refinement notion does not give monotonicity for all of the composition operators. Therefore, a restricted refinement notion is defined where the set of positive traces is kept unchanged (i.e., the only possible refinement step is supplementing traces as negative). With this restriction, monotonicity is achieved.

Inherent nondeterminism is not treated in [CK04]. In fact, the distinction between inherent nondeterminism and underspecification that allows us to abstract trace-set properties and plays such a fundamental role in our approach is hardly reflected in most other approaches. There are however some notable exceptions. We believe our work on compliance is also of significance for these. We now go through these approaches in more detail.

In the work by Krüger [Krü00], a variant of Message Sequence Charts (MSC) is given a formal semantics and provided a formal notion of refinement. Four different interpretations of MSC are proposed, referred to as the existential, universal, exact, and negative interpretation. The existential interpretation requires the fulfilment of the MSC in question by at least one system execution, while the universal interpretation requires the fulfilment of the MSC in all executions. The exact interpretation is a strengthening of the universal interpretation by explicitly prohibiting behaviours other than the ones specified by the MSC. Finally, the negative interpretation requires that no execution is allowed to fulfil the MSC. Three notions of refinement are defined, referred to as property, message, and structural refinement. Property refinement corresponds to the classical notion of refinement as reverse set inclusion, i.e., the removal of underspecification by reducing the possible behaviour of the system. Message refinement is to replace a single message with a sequence of messages, while structural refinement means to decompose an instance (i.e., a lifeline), with a set of instances.

The exact interpretation is similar to our notion of basic compliance 3, while the existential interpretation is particularly interesting in our context as it may capture trace-set properties in much the same way as we do with inherent nondeterminism and general compliance. As shown in [Krü00], a system that fulfils an MSC specification under the universal, exact, or negative interpretation also fulfils specifications that are property refinements of the original specification. This is, however, not the case for the existential interpretation, which means that trace-set properties are not preserved under refinement.

Uchitel et al. [UBC07] present a technique for generating Modal Transition Systems (MTSs) from a specification given as a combination of system properties and scenario specifications. System properties, such as safety or liveness properties, are universal as they impose requirements on all of the execution traces, and are in [UBC07] specified in Fluent Linear Temporal Logic. Scenario specifications are existential as they provide examples of intended system behaviour, and are specified in the form of MSC. An MTS is a behaviour model with the expressiveness to distinguish between required, possible, and proscribed behaviour. The method for generating MTSs from properties and scenarios ensures that all system behaviours satisfy the properties while the system may potentially fulfil all the scenarios. Furthermore, both composition and refinement of behaviour models preserve the original properties and scenarios. With this approach, MSC can be utilized to capture trace-set properties and preserve these under refinement. However, as opposed to our approach, [UBC07] is not a pure interaction based development process, and the use of MSC is restricted to the existential interpretation.

Live sequence charts (LSC) [DH01, HM03] is an extension of MSC that particularly address the issue of expressing liveness properties. LSC support the specification of two types of diagrams, referred to as existential and universal diagrams. An existential diagram describes an example scenario that must be satisfied by at least one execution trace, whereas the scenario described by a universal diagram must be satisfied by all execution traces. Universal charts can furthermore be specified as conditional scenarios by the specification of a prechart that, if successfully fulfilled by an execution trace, requires the fulfilment of the scenario described in the chart body. The universal/existential distinction is a distinction between

mandatory and provisional behaviour, respectively. Such a distinction is also made between elements of a single LSC by characterizing these as hot or cold, where a hot element is mandatory and a cold element is provisional.

LSC seem to have the expressiveness needed for trace-set properties by the use of existential diagrams. Obviously, falsification of requirements expressed by an existential diagram cannot be done by a single execution trace. However, a system development in LSC is intended to undergo a shift from an existential view in the initial phases to a universal view in later stages as knowledge of the system evolves. Such a development process with LSC will generally not preserve trace-set properties. Moving from an existential view to a universal view can be understood as a form of refinement, but LSC is not supported by a formal notion of refinement.

Modal sequence diagrams (MSD) [HM08] is defined as a UML 2.x profile. The notation is an extension of the UML sequence diagram notation based on the universal/existential distinction of LSC. The main motivation for the development of the MSD language is the problematic definitions of the assert and negate constructs of UML sequence diagrams. The authors observe that the UML 2.x specification is contradictory in the definition of these constructs, and also claim that the UML trace semantics of valid and invalid traces is inadequate for properly supporting an effective use of the constructs.

The semantics for MSDs is basically the same as for LSC. The main difference is that the LSC prechart construct is left out. Instead, a more general approach is adopted where cold fragments inside universal diagrams serve the purpose of a prechart. A cold fragment is not required to be satisfied by all execution traces, but if it is satisfied, it requires the satisfaction of the subsequent hot fragment. With respect to capturing trace-set properties and preserving these under refinement, the situation is the same as for LSC. It is in [HM08] indicated that existential diagrams should be kept in the specification as the development process evolves and that universal diagrams are gradually added. With such an approach, trace-set properties might be preserved.

In [CF04, CF05], Cavarra and Küster-FIlipe present an operational semantics for UML 2.x sequence diagrams inspired by LSC. Liveness properties are expressed by adding OCL (Object Constraint Language [OMG06]) constraints to the sequence diagrams. This makes it possible to distinguish between may and must behaviour, and between universal and existential diagrams. The semantics is formalized using abstract state machines. Choices are essentially treated as nested if-then-else statements, and may not be used for underspecification or inherent nondeterminism.

A continuation of this work may be found in [Fil06, Bow06], where Küster-Filipe gives an LSC inspired denotational semantics of UML 2.x sequence diagrams based on partially ordered sets. These are used to build event structures, and modal logic constraints over the event structures are used to express negative behaviour, as well as must and may behaviour. The only notion of refinement discussed in these approaches is referencing, i.e., structural decomposition.

Triggered message sequence charts (TMSC) [SC06] is an approach in the family of MSC that is related to our approach in several ways. The development of TMSC is motivated by the fact that MSC does not have the expressiveness to define conditional scenarios, i.e., that one interaction (the triggering scenario) requires the execution of another (the action scenario), that MSC does not formally define a notion of refinement, and that MSC lacks structuring mechanisms that properly define ways of grouping scenarios together.

The triggering scenarios of TMSC are closely related to the precharts of LSC. An important semantic difference, however, is that whereas LSC synchronizes at the beginning of precharts and main charts, TMSC is based on weak sequencing in the spirit of MSC. TMSC includes composition operators for sequential composition, recursion (similar to loop), and parallel composition. The most interesting composition operators in the context of this paper, however, are delayed choice and internal choice. Both operators take a set of diagrams as operands and define a choice between these. Internal choice is an operator similar to the alt operator and defines underspecification. An implementation that can execute only one of the operands is a correct implementation. Delayed choice, on the other hand, is an operator somewhat related to our xalt operator where a correct implementation must provide all the choices defined by the specification. However, as opposed to inherent non-determinism captured by the xalt operator, a delayed choice is not made until a given execution forces it to be made.

It is observed in [SC06] that the simple trace-set semantics of MSC does not have the expressiveness to distinguish between optional and required behaviour, which means that a heterogeneous mix of these in the specification is not supported. Such a mix is supported in our approach, syntactically by the alt operator for underspecification and the xalt operator for inherent non-determinism, semantically by the set of interaction obligations. The semantics of TMSC is defined in terms of so-called acceptance trees. Acceptance trees record

the traces that are defined by a specification, but also distinguish between required and optional behaviour. Importantly, the semantics of TMSC supports a notion of refinement that preserves the required behaviour and gives freedom with respect to optional behaviour.

[SC06] stress that refinement should preserve properties such as safety and liveness. These properties are trace properties, and the issue of capturing trace-set properties and preserving these under refinement is not discussed. However, by the semantics of the delayed choice operator and the fact that this type of choice is preserved under refinement, TMSC seem to have the expressiveness to capture and preserve trace-set properties. Like the refinement relations given in this paper, refinement of TMSC is compositional, i.e., composition of specifications is monotonic with respect to refinement. An important difference between our approach and TMSC is that the latter does not support the specification of negative behaviour. This also means that with TMSC there is no notion of inconclusive behaviour; a system specification defines a set of valid traces, and all other traces are invalid.

Grosu and Smolka [GS05] interpret positive and negative sequence diagrams as specifying liveness and safety properties, respectively. This is a much stronger interpretation than the traditional use of sequence diagrams for illustrating example runs. Based on several transformation steps, the semantics of sequence diagrams is defined as two Büchi automata, one for the positive and one for the negative behaviour. The approach is based on composing simple sequence diagrams without composition operators into high-level sequence diagrams, i.e., interaction overview diagrams. A valid trace is allowed to have any suffix, and events not explicitly mentioned in the diagram may be interleaved with the specified events. Refinement is defined as language inclusion, and the most common composition operators are shown to be monotonic with respect to this refinement notion.

Another operational semantics for sequence diagrams is given by Knapp and Wuttke in [KW07]. In contrast to [GS05], a single interaction automaton is created for the entire diagram. To simplify this process, negation and loop may only be applied to simple diagrams, and a simpler variation of loop is introduced, where loop iterations are sequenced by strict rather than weak sequencing. Synchronization is also used between lifelines before entering a choice. No notion of refinement is included in [KW07], neither is there a distinction between underspecification and inherent nondeterminism.

Störrle [Stö03] is one of the few approaches where alternatives are interpreted as must, similar to our xalt operator for specifying inherent nondeterminism. Refinement means adding new traces to the specification. Reducing underspecification is not an issue in [Stö03], as there is no treatment of underspecification in the semantics. However, refinement also includes adding new events to the traces.

There exists also other works on the semantics of sequence diagrams. However, to the best of our knowledge none of these include a distinction between underspecification and inherent nondeterminism or work on refinement and the relationship between computer systems and sequence diagrams. In general, the semantics is also far from the informal description given in the UML 2.x semantics. Overviews of different semantics of sequence diagrams may be found in [LRS10] and [MW10], including the works discussed above and also others.

Underspecification and nondeterminism have been important issues also in other specification formalisms. In the setting of algebraic specifications, Walicki and Meldal [WM01] argue for using explicit nondeterminism in cases where underspecification might in fact lead to overspecification. However, the resulting system may still be deterministic. In [LAMB89], Larsen et al made a similar distinction in VDM-SL when interpreting looseness in specifications, i.e., specifications allowing alternative behaviours. Looseness in function definitions is interpreted as underdeterminedness (i.e., underspecification), meaning that the exact definition is chosen at implementation time. Looseness in operations is interpreted as nondeterminism where the choice may be delayed until run-time, meaning that the final system may be either deterministic or nondeterministic.

In CSP [Hoa85, Ros98], there are two different operators for nondeterminism, referred to as internal and external nondeterminism, respectively. With internal nondeterminism, the system is free to choose whether it should offer all alternatives or only one (or some) of them. The choice may be performed at run-time, making the system nondeterministic, but the choice may also be made by the implementer, resulting in a deterministic system. For external nondeterminism (also called environmental choice), the behaviour is determined by the environment and the system must be able to perform all alternatives.

In the refinement calculus [BvW98], Back and von Wright make a similar distinction between angelic and demonic choice. An angelic choice is a choice that is made by the system with the goal of establishing a given postcondition. This means that if the behaviours are similar up to some point, the choice between them may be deferred as long as possible in order to increase the chances of obtaining the desired end result. On the other hand, a demonic choice is assumed to be resolved by an environment with another goal. Hence,

the system may only guarantee the given postcondition if that condition may be established for *all* of the demonic alternatives.

Steen et al [SBDB97] extend the process algebraic language LOTOS [ISO89] with a disjunction operators for specifying implementation freedom (i.e., underspecification), leaving the traditional LOTOS choice operator to be used for inherent nondeterminism. With this new disjunction operator, exactly one of the alternatives may be implemented, in contrast to the usual interpretation of underspecification that also allows implementations with several of the alternative behaviours.

Looking outside the scope of specifications expressed as sequence diagrams, there are also numerous works that address testing related to formal specifications, and we now present a few of these. In [Gau95], Gaudel introduces a theory of testing based on algebraic specifications and briefly reports on some case studies and experiments related to the theory. The theory defines formally concepts like test, (successful) test experiment, unbias (meaning that correct programs are not rejected) and validity (meaning that only correct programs are accepted). To deal with the fact that an exhaustive test set is generally infinite, the theory allows selection hypotheses to be defined under which a verdict can be reached from a finite test set. The choice of selection hypothesis determines what tests should be included in the test set and should be guided by the specification. Weakening the selection hypothesis implies increasing the test set and corresponds to extending the set of known traces in our approach. Hence, our guidelines for extending the set of known traces – which are also guided by the specification – correspond to guidelines for choice of selection hypothesis.

Lee and Yannakakis [LY96] present a survey of methods for testing finite state machines. Five different fundamental problems of testing related to state machines are presented, but the one of most interest in our context is conformance testing. Here, the task is to test whether an implementation represented by a black box, i.e., an unknown finite state machine where only I/O behaviour can be observed, conforms to a specification state machine. This amounts to finding a so-called checking sequence for the specification state machine. A checking sequence is an input sequence that under certain assumptions (for example about the number of states of the implementation) distinguishes the specification from all other state machines. The paper presents and discusses a number of different methods and heuristics for effectively finding checking sequences.

In [Tre99], Tretmans presents a framework for testing based on formal specifications. The goal is to link the informal world of actual implementations and tests to the formal world of specifications and models. Unlike [Gau95], [LY96] and the approach presented in this paper, the framework in [Tre99] is generic in the sense that it is not tied to any particular specification language. Concepts like conformance, testing, sound/exhaustive test suites and test derivation are defined at a generic level. After presenting the generic framework, Tretmans shows how it can be instantiated for labelled transition system specifications, and presents an algorithm for derivation of test cases for such specifications that are sound in the sense that a conforming implementation will never get a negative verdict. An interesting exercise would be to try to apply the framework to our approach, although this is beyond the scope of this paper. Lund [LS06, Lun08] has made use of Tretmans' framework to test refinement in the setting of STAIRS.

The efficient selection of test cases is investigated from a practical and empirical viewpoint in [JMV04] by Juristo et al. Although this paper does not really focus on the role of formal specifications with respect to testing (and some of the approaches even require knowledge of the source code of the implementation), it is interesting because it aims to compile the existing knowledge and results about testing techniques that have been tried out, either in the laboratory or in the field, and to evaluate and compare the results. Unfortunately, the authors conclude that knowledge about testing techniques based on empirical investigations is very limited. Hopefully, in the future we will see more empirical research on the application of testing based on formal specifications.

9. Conclusions

If sequence diagrams are to be used as formal specifications, it is important to know how to evaluate whether a given computer system is compliant with the sequence diagram or not. To the best of our knowledge, our work is the first to address this in detail. Sequence diagrams are different from most other techniques for specifying dynamic behaviour in that they usually give only a partial view. When relating computer systems to sequence diagram specifications, it is crucial to understand exactly how this partiality should be interpreted.

In this paper, we have proposed a general procedure for checking compliance with respect to two classes

of sequence diagrams: Sequence diagrams with underspecification, which may be used to capture trace properties, and sequence diagrams with both inherent nondeterminism and underspecification, which may also be used to capture trace-set properties. To this end, we have introduced a set of natural compliance relations, differing with respect to how the partiality of sequence diagrams should be understood. From these compliance relations, we have derived a set of corresponding refinement relations, and shown that these relations satisfy the mathematical properties of transitivity and monotonicity that are necessary to support a stepwise and compositional development process.

The approach is faithful to the UML 2.x standard, both with respect to the underlying semantic model using sets of positive and negative traces, and with respect to the semantics given for each concrete operator. In particular, all of our definitions take into account the partial nature of sequence diagrams. The compliance checking procedure is independent of any particular programming language or paradigm. All we require, is that there exists some means to obtain (a subset of) the execution traces of the system. Depending on the known execution traces and the specification, the verdict from applying the procedure may be definitive or only an indication of whether compliance holds or not. We have defined predicates distinguishing precisely between these two cases, and given methodological advice for how to proceed in order to obtain increased confidence in the verdict.

In this paper we have only considered sequence diagrams without external input and output. Our results may be generalized to handle also sequence diagrams with such external communication by in each case defining an adversary representing the environment of the system, and then checking compliance under the assumption of this adversary.

We plan to generalize our approach to handle probabilistic choice, where the different alternatives are provided with a set of probabilities specifying how often that alternative should be chosen. See [RRS07] for a first step in this direction.

To make practical use of the theoretical work presented in this paper, tool support is obviously needed. We have plans to extend the Escalator tool developed by Lund [Lun08, Lun09] to facilitate testing according to our procedure and guidelines. This will furthermore allow us to investigate the feasibility and usefulness of the proposed procedure in empirical studies.

Acknowledgement

The research on which this paper reports has been partly carried out within the context of the IKT-2010 project SARDAS (15295/431) and the IKT SOS project ENFORCE (164382/V30), both funded by the Research Council of Norway. We thank the other members of the SARDAS project for useful discussions related to this work.

References

[AS85] Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, 1985.

[Bow06] Juliana Küster Filipe Bowles. Decomposing interactions. In *Algebraic Methodology and Software Technology (AMAST 2006)*, volume 4019 of *LNCS*, pages 189–203. Springer, 2006.

[BS01] Manfred Broy and Ketil Stølen. *Specification and Development of Interactive Systems:* Focus *on Streams, Interfaces, and Refinement*. Springer, 2001.

[BvW98] Ralph-Johan Back and Joakim von Wright. *Refinement Calculus: A Systematic Introduction*. Springer, 1998.

[CF04] Alessandra Cavarra and Juliana Küster Filipe. Formalizing liveness-enriched sequence diagrams using ASMs. In *Abstract State Machines (ASM 2004)*, volume 3052 of *LNCS*, pages 62–77. Springer, 2004.

[CF05] Alessandra Cavarra and Juliana Küster Filipe. Combining sequence diagrams and OCL for liveness. *Electronic Notes in Theoretical Computer Science*, 115:19–38, 2005.

[CK04] María Victoria Cengarle and Alexander Knapp. UML 2.0 interactions: Semantics and refinement. In *Proceedings 3rd International Workshop on Critical Systems Development with UML (CSDUML'04)*, Technical report TUM-I0415, pages 85–99. Institut für Informatik, Technische Universität München, 2004.

[DH01] Werner Damm and David Harel. LSC's: Breathing life into message sequence charts. *Formal Methods in System Design*, 19(1):45–80, 2001.

[Fil06] Juliana Küster Filipe. Modelling concurrent interactions. *Theoretical Computer Science*, 351(2):203–220, 2006.

[Gau95] Marie-Claude Gaudel. Testing can be formal, too. In *Theory And Practice of Software Development (TAPSOFT'95)*, volume 915 of *LNCS*, pages 82–96. Springer, 1995.

[GS05] Radu Grosu and Scott A. Smolka. Safety-liveness semantics for UML 2.0 sequence diagrams. In *Proceedings Applications of Concurrency to System Design (ACSD'05)*, pages 6–14. IEEE Computer Society, 2005.

[HHRS05] Øystein Haugen, Knut Eilif Husa, Ragnhild Kobro Runde, and Ketil Stølen. STAIRS towards formal design with sequence diagrams. *Software and Systems Modeling*, 4(4):349–458, 2005.

[HM03] David Harel and Rami Marelly. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer, 2003.

[HM08] David Harel and Shahar Maoz. Assert and negate revisited: modal semantics for UML sequence diagrams. *Software and Systems Modeling*, 7(2):237–252, 2008.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[HS03] Øystein Haugen and Ketil Stølen. STAIRS — Steps to analyze interactions with refinement semantics. In *The Unified Modeling Language. Modeling Languages and Applications (UML 2003)*, volume 2863 of *LNCS*, pages 388–402. Springer, 2003.

[ISO89] International Standards Organization. *Information Processing Systems – Open Systems Interconnection – LOTOS – a Formal Description Technique Based on the Temporal Ordering of Observational Behaviour – ISO 8807*, 1989.

[Jac89] Jeremy Jacob. On the derivation of secure components. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 242–247, 1989.

[JMV04] Natalia Juristo, Ana María Moreno, and Sira Vegas. Reviewing 25 years of testing technique experiments. *Empirical Software Engineering*, 9(1–2):7–44, 2004.

[Jür01] Jan Jürjens. Secrecy-preserving refinement. In *Formal Methods for Increasing Software Productivity (FME 2001)*, volume 2021 of *LNCS*, pages 135–152. Springer, 2001.

[Krü00] Ingolf Heiko Krüger. *Distributed System Design with Message Sequence Charts*. PhD thesis, Technische Universität München, 2000.

[KW07] Alexander Knapp and Jochen Wuttke. Model checking of UML 2.0 interactions. In *Models in Software Engineering*, volume 4364 of *LNCS*, pages 42–51. Springer, 2007.

[LAMB89] Peter Gorm Larsen, Michael Meincke Arentoft, Brian Q. Monahan, and Stephen Bear. Towards a formal semantics of the BSI/VDM specification language. In *Information processing 89: Proceedings IFIP 11th World Computer Congress*, pages 95–100. Elsevier, 1989.

[LRS10] Mass Soldal Lund, Atle Refsdal, and Ketil Stølen. Semantics of UML models for dynamic behavior. A survey of different approaches. In *Model-Based Engineering of Embedded Real-Time Systems*, volume 6100 of *LNCS*, pages 77–103. Springer, 2010.

[LS06] Mass Soldal Lund and Ketil Stølen. Deriving tests from UML 2.0 sequence diagrams with neg and assert. In *Proceedings 1st International Workshop on Automation of Software Test (AST'06)*, pages 22–28. ACM Press, 2006.

[Lun08] Mass Soldal Lund. *Operational analysis of sequence diagram specifications*. PhD thesis, University of Oslo, 2008.

[Lun09] Mass Soldal Lund. Model-based testing with the Escalator tool. *Telektronikk*, 105(1):117–125, 2009.

[LY96] David Lee and Mihalis Yannakakis. Principles and methods of testing finite state machines – a survey. *Proc. IEEE*, 84(8):1090–1123, 1996.

[MW10] Zoltán Micskei and Hélène Waeselynck. The many meanings of UML 2 sequence diagrams: a survey. *Software and Systems Modeling*, Online First:1–26, 2010.

[OMG06] Object Management Group. *Object Constraint Language 2.0*, document: formal/2006-05-01 edition, 2006.

[OMG10] Object Management Group. *UML 2.3 Superstructure Specification*, document: formal/2010-05-05 edition, 2010.

[RHS05a] Ragnhild Kobro Runde, Øystein Haugen, and Ketil Stølen. How to transform UML neg into a useful construct. In *Proceedings Norsk Informatikkonferanse (NIK 2005)*, pages 55–66. Tapir, 2005.

[RHS05b] Ragnhild Kobro Runde, Øystein Haugen, and Ketil Stølen. Refining UML interactions with underspecification and nondeterminism. *Nordic Journal of Computing*, 12(2):157–188, 2005.

[Ros95] Bill Roscoe. CSP and determinism in security modelling. In *Proceedings 1995 IEEE Symposium on Security and Privacy*, pages 114–127. IEEE Computer Society Press, 1995.

[Ros98] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice-Hall, 1998.

[RRS07] Ragnhild Kobro Runde, Atle Refsdal, and Ketil Stølen. Relating computer systems to sequence diagrams with underspecification, inherent nondeterminism and probabilistic choice. Part 2: Probabilistic choice. Technical Report 347, Department of Informatics, University of Oslo, 2007.

[RRS11] Ragnhild Kobro Runde, Atle Refsdal, and Ketil Stølen. Relating computer systems to sequence diagrams — the impact of underspecification and inherent nondeterminism. Technical Report 410, Department of Informatics, University of Oslo, 2011.

[SBDB97] Maarten Steen, Howard Bowman, John Derrick, and Eerke Boiten. Disjunction of LOTOS specifications. In *Formal Description Techniques and Protocol Specification, Testing and Verification (FORTE X / PSTV XVII '97)*, pages 177–192. Chapman & Hall, 1997.

[SC06] Bikram Sengupta and Rance Cleaveland. Triggered message sequence charts. *IEEE Transaction on Software Engineering*, 32(8):587–607, 2006.

[SS06] Fredrik Seehusen and Ketil Stølen. Information flow property preserving transformation of UML interaction diagrams. In *Proceedings Symposium on Access Control Models and Technologies (SACMAT 2006)*, pages 150–159. ACM, 2006.

[SSS09] Fredrik Seehusen, Bjørnar Solhaug, and Ketil Stølen. Adherence preserving refinement of trace-set properties in STAIRS: exemplified for information flow properties and policies. *Software and Systems Modeling*, 8(1):45–65, 2009.

[Stö03] Harald Störrle. Assert, negate and refinement in UML-2 interactions. In *Proceedings 2nd International Workshop on Critical Systems Development with UML (CSDUML'03)*, Technical report TUM-I0317, pages 79–93. Institut für Informatik, Technische Universität München, 2003.

[Tre99] Jan Tretmans. Testing concurrent systems: a formal approach. In *Proceedings 10th International Conference on Concurrency Theory (CONCUR'99)*, volume 1664 of *LNCS*, pages 46–65. Springer, 1999.

[UBC07] Sebastián Uchitel, Greg Brunet, and Marsha Chechik. Behaviour model synthesis from properties and scenarios. In *Proceedings 29th Interational Conference in Software Engineering (ISCE'07)*, pages 34–43. IEEE Computer Society, 2007.

[WM01] M. Walicki and S. Meldal. Nondeterminism vs. underspecification. In *Proceedings Systemics, Cybernetics and Informatics (ISAS-SCI 2001)*, pages 551–555. IIIS, 2001.

A. Summary of the STAIRS Semantics for UML 2.x Sequence Diagrams

Formally, a message is a triple (s, tr, re) consisting of a signal s, a transmitter lifeline tr and a receiver lifeline re. $\mathcal{M}$ and $\mathcal{L}$ denotes the set of all messages and lifelines, respectively. $\mathcal{E}$ denotes the set of all events. Formally, an event is a pair (k, m) consisting of a kind k (either ! or ?) and a message m. We define the functions

$$k.\_ \in \mathcal{E} \to \{!, ?\}, \quad tr.\_ , re.\_ \in \mathcal{E} \to \mathcal{L}$$

to yield the kind, transmitter and receiver of an event, respectively.

A.1. Sequence Diagrams with Underspecification

As explained in Section 3, the semantic model of the class of sequence diagrams containing only underspecification and not inherent nondeterminsim, is an interaction obligation (p, n) where p is the set of positive and n the set of negative traces.

Here, we give the definitions of the most central basic composition operators for sequence diagrams in addition to alt (which was defined in Section 3), namely par, seq and refuse. The operators par and seq are used for parallel and (weak) sequential composition, respectively. For specifying negative behaviours, we follow [RHS05a] and use the operator refuse instead of the neg operator used in UML 2.x. Definition of other operators such as loop and assert may be found in e.g. [RHS05b].

Parallel composition ($\parallel$) of two trace sets corresponds to point-wise interleaving of their individual traces. The ordering of events within each trace is maintained in the result. For sequential composition ($\gtrsim$) we require in addition that for events on the same lifeline, all events from the first trace are ordered before the events from the second trace. Formally:

$$s_1 \parallel s_2 \stackrel{\text{def}}{=} \{h \in \mathcal{H} \mid \exists p \in \{1,2\}^\infty : \pi_2((\{1\} \times \mathcal{E}) \circledcirc (p, h)) \in s_1 \wedge \pi_2((\{2\} \times \mathcal{E}) \circledcirc (p, h)) \in s_2\} \tag{35}$$

$$s_1 \gtrsim s_2 \stackrel{\text{def}}{=} \{h \in \mathcal{H} \mid \exists h_1 \in s_1, h_2 \in s_2 : \forall l \in \mathcal{L} : e.l \circledS h = e.l \circledS h_1 \frown e.l \circledS h_2\} \tag{36}$$

where $\mathcal{E}$ is the set of all events; π_2 is a projection operator returning the second element of a pair; $\frown$ is the concatenation operator for sequences; and $e.l$ is the set of events that may take place on the lifeline l, formally defined by:

$$e.l \stackrel{\text{def}}{=} \{e \in \mathcal{E} \mid (k.e =! \wedge tr.e = l) \vee (k.e =? \wedge re.e = l)\} \tag{37}$$

$\circledS$ and $\circledcirc$ are filtering operators for traces and pairs of traces, respectively. $E \circledS h$ is the trace obtained from the trace h by removing from h all events that are not in the set of events E. For instance, we have that

$$\{e_1, e_3\} \circledS \langle e_1, e_1, e_2, e_1, e_3, e_2\rangle = \langle e_1, e_1, e_1, e_3\rangle$$

The operator $\circledcirc$ is a generalization of $\circledS$ filtering pairs of traces with respect to pairs of elements such that for instance

$$\{(1, e_1), (1, e_2)\} \circledS (\langle 1, 1, 2, 1, 2\rangle, \langle e_1, e_1, e_1, e_2, e_2\rangle) = (\langle 1, 1, 1\rangle, \langle e_1, e_1, e_2\rangle)$$

For formal definitions of $\circledS$ and $\circledcirc$, see [BS01].

For interaction obligations, parallel composition ($\parallel$), sequential composition ($\gtrsim$) and refusal ($\dagger$) are defined

by:

$$(p_1, n_1) \parallel (p_2, n_2) \stackrel{\text{def}}{=} (p_1 \parallel p_2, (n_1 \parallel p_2) \cup (n_1 \parallel n_2) \cup (p_1 \parallel n_2)) \tag{38}$$

$$(p_1, n_1) \gtrsim (p_2, n_2) \stackrel{\text{def}}{=} (p_1 \gtrsim p_2, (n_1 \gtrsim p_2) \cup (n_1 \gtrsim n_2) \cup (p_1 \gtrsim n_2)) \tag{39}$$

$$\dagger(p_1, n_1) \stackrel{\text{def}}{=} (\emptyset, p_1 \cup n_1) \tag{40}$$

Notice that composing a positive and a negative trace always yields a negative trace, while the result of composing an inconclusive trace with a positive or negative trace is always inconclusive.

The sequence diagram operators for parallel composition (par), sequential composition (seq) and negative behaviour (refuse) are then defined by:

$$[\![D_1 \text{ par } D_2]\!] \stackrel{\text{def}}{=} [\![D_1]\!] \parallel [\![D_2]\!] \tag{41}$$

$$[\![D_1 \text{ seq } D_2]\!] \stackrel{\text{def}}{=} [\![D_1]\!] \gtrsim [\![D_2]\!] \tag{42}$$

$$[\![\text{refuse } D_1]\!] \stackrel{\text{def}}{=} \dagger [\![D_1]\!] \tag{43}$$

In addition to the operators above, the example in Section 4.3 uses the high-level operator veto which is defined by:

$$\text{veto } D \stackrel{\text{def}}{=} (\text{skip}) \text{ alt } (\text{refuse } D) \tag{44}$$

where skip is the empty diagram with semantics defined by:

$$[\![\text{skip}]\!] \stackrel{\text{def}}{=} (\{\langle\rangle\}, \emptyset) \tag{45}$$

Finally, the set $\mathcal{H}^{ll(D)}$ used in the definition of the system representation in Section 4.1 is formally defined by:

$$\mathcal{H}^{ll(D)} \stackrel{\text{def}}{=} \{h \in \mathcal{H} \mid \forall i \in \mathbb{N} : i > \#h \vee \exists l \in ll(D) : h[i] \in e.l\} \tag{46}$$

where $ll(D)$ denotes all lifelines in the sequence diagram D, $\#h$ is the length of the sequence h, and $h[i]$ is the i'th element of this sequence.

A.2. Sequence Diagrams with Inherent Nondeterminism and Underspecification

As explained in Section 5, the semantic model of the class of sequence diagrams containing both inherent nondeterminism and underspecification is a set of interaction obligations.

With this generalized semantic model, we need to lift the definitions for parallel composition ($\parallel$), sequential composition ($\gtrsim$), inner union ($\uplus$) and refusal ($\dagger$) to sets of interaction obligations:

$$O_1 \parallel O_2 \stackrel{\text{def}}{=} \{o_1 \parallel o_2 \mid o_1 \in O_1 \wedge o_2 \in O_2\} \tag{47}$$

$$O_1 \gtrsim O_2 \stackrel{\text{def}}{=} \{o_1 \gtrsim o_2 \mid o_1 \in O_1 \wedge o_2 \in O_2\} \tag{48}$$

$$O_1 \uplus O_2 \stackrel{\text{def}}{=} \{o_1 \uplus o_2 \mid o_1 \in O_1 \wedge o_2 \in O_2\} \tag{49}$$

$$\dagger O_1 \stackrel{\text{def}}{=} \{\dagger o_1 \mid o_1 \in O_1\} \tag{50}$$

With these definitions for sets of interaction obligations, the main definitions of the sequence diagram operators par, seq, and refuse (definitions (41)–(43)) and alt (definition (1)) are unchanged, while the semantics of skip is redefined as:

$$[\![\text{skip}]\!] \stackrel{\text{def}}{=} \{(\{\langle\rangle\}, \emptyset)\} \tag{51}$$