A Fully General Operational Semantics for UML 2.0 Sequence Diagrams with Potential and Mandatory Choice*

Mass Soldal Lund[1,2] and Ketil Stølen[1,2]

[1] University of Oslo, Norway
[2] SINTEF Information and Communication Technology, Norway
{msl, kst}@sintef.no

Abstract. UML sequence diagrams is a specification language that has proved itself to be of great value in system development. When put to applications such as simulation, testing and other kinds of automated analysis there is a need for formal semantics. Such methods of automated analysis are by nature operational, and this motivates formalizing an operational semantics. In this paper we present an operational semantics for UML 2.0 sequence diagrams that we believe gives a solid starting point for developing methods for automated analysis. The operational semantics has been proved to be sound and complete with respect to a denotational semantics for the same language. It handles negative behavior as well as potential and mandatory choice. We are not aware of any other operational semantics for sequence diagrams of this strength.

1 Introduction

Unified Modeling Language (UML) sequence diagrams [1] and their predecessor Message Sequence Charts (MSC) [2] are specification languages that have proved themselves to be of great practical value in system development. When sequence diagrams are used to get a better understanding of the system through modeling, as system documentation or as means of communication between stakeholders of the system, it is important that the precise meaning of the diagrams is understood; in other words, there is need for a well-defined semantics. Sequence diagrams may also be put to further applications, such as simulation, testing and other kinds of automated analysis. This further increases the need for a formalized semantics; not only must the people who make and read diagrams have a common understanding of their meaning, but also the makers of methods and tools for analyzing the diagrams must share this understanding.

Methods of analysis like simulation and testing are in their nature operational; they are used for investigating what will happen when a system is executing.

* The work of this paper was conducted within and funded by the Basic ICT Research project SARDAS (15295/431) under the Research Council of Norway. We would like to thank Rolv Bræk, Manfred Broy, Kathrin Greiner, Øystein Haugen, Birger Møller-Pedersen, Atle Refsdal, Ragnhild Kobro Runde, Ina Schieferdecker and Thomas Weigert for useful comments on this work.

J. Misra, T. Nipkow, and E. Sekerinski (Eds.): FM 2006, LNCS 4085, pp. 380–395, 2006.

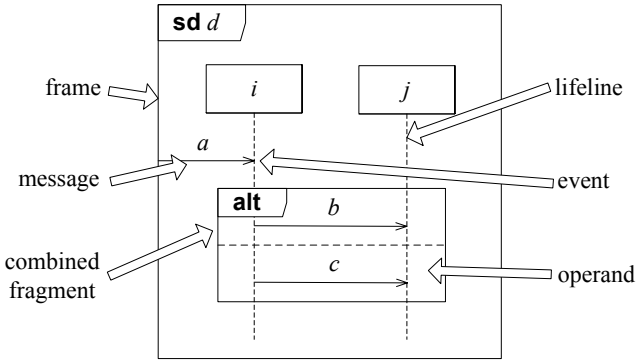

Fig. 1. Sequence diagram

When developing techniques for such analysis, not only do we need to understand the precise meaning of a specification, we also need to understand precisely the executions that are specified. This motivates formalization of semantics in an operational style. In this paper we present an operational semantics for UML sequence diagrams that we believe gives a solid starting point for developing such methods of analysis.

Sequence diagrams is a graphical specification language defined in the UML 2.0 standard [1]. The standard defines the graphical notation, but also an abstract syntax for the diagrams. Further the UML standard provides an informal semantics of the diagrams. Figure 1 shows a sequence diagram d in the graphical notation. A sequence diagram consists of a *frame*, representing the environment of the specified system, and one or more *lifelines*, representing components of the system. Arrows represent *messages* sent between lifelines or between a lifeline and the environment, and if the beginning or end of an arrow is at a lifeline this represents an *event*. *Combined fragments* are operators, like the choice operator alt, and each combined fragment has one or more *operands*.

The semantics of UML 2.0 sequence diagrams is trace based. The standard states that the semantics of a sequence diagram is a pair of traces (p, n) such that p is interpreted as valid (positive) traces and n is interpreted as invalid (negative) traces. Further, the union of p and n need not exhaust the trace universe.

Several properties of sequence diagrams prevent us from adopting a simple automata or process algebra approach to defining the formal semantics. First of all, sequence diagrams are partial specifications, and invalid behavior is specified explicitily by an operator neg. This means we cannot treat valid and invalid behavior as complementary sets. Further, communication between lifelines is asynchronous and lifelines are non-synchronizing, but choices, like the alt operator, are global. This means that sequence diagrams have a semi-global nature. Finally, the choice operator alt is ambiguous, and may be interpreted as either potential choice or mandatory choice. In our approach, this ambiguity is resolved by interpreting alt as potential choice and introducing a new operator xalt to do the job as mandatory choice.

In [3,4,5] a denotational semantics for sequence diagrams is formalized. We refer to this as the STAIRS semantics. STAIRS has a more general semantic model; the semantics of a diagram is a set of pairs $\{(p_1, n_1), (p_2, n_2), \ldots, (p_m, n_m)\}$. A pair (p_i, n_i) is referred to as an *interaction obligation*. The word "obligation" is used in order to emphasize that an implementation of a specification is required to fulfill every interaction obligation of the specification. This semantic model makes it possible to distinguish between potential and mandatory choice.

A trace is a (finite or infinite) sequence of events $\langle e_1, e_2, \ldots, e_i, \ldots \rangle$. We let $t_1 \frown t_2$ denote concatenation of the traces t_1 and t_2 and $\langle\rangle$ denote the empty trace. Let $\mathcal{H}$ be the trace universe. For each interaction obligation (p_i, n_i) we have that $p_i \cup n_i \subseteq \mathcal{H}$. All interaction obligations are independent of each other, and an interaction obligation is allowed to be inconsistent (i.e., we allow $p_i \cap n_i \neq \emptyset$).

The contribution of this paper is an operational semantics for UML 2.0 sequence diagrams. Obviously, choices must be made where the UML standard is ambiguous, but as far as possible the semantics is faithful to the standard. The semantics is easy to extend and modify. This allows us to give a "default" or "standard" interpretation, but also to experiment with the semantics and make variations on points unspecified by the standard. Specifically it has a formalized meta-level which allows definition of different execution strategies. It is not based on transformations to other formalisms, which makes it easy to work with. Further it has been proved to be sound and complete with respect to the STAIRS semantics.

The structure of this paper is as follows: In Sect. 2 we present the syntax over which the semantics is defined and in Sect. 3 the operational semantics. Soundness and completeness is treated in Sect. 4. In Sect. 5 we present related work and, finally, in Sect. 6 conclusions are provided. A short presentation of the denotational semantics of STAIRS is provided in Appendix A.

2 Syntax

The graphical notation of sequence diagrams is not suited as a basis for defining semantics, and the abstract syntax of the UML standard contains more information than we need for the task. Our operational semantics is defined over a simpler abstract syntax defined in [4, 5]. This is an event centric syntax in which the weak sequential operator `seq` is employed as the basic construct for combining diagram fragments.

The atom of a sequence diagram is the *event*. An event consists of a *message* and a *kind* where the kind decides whether it is the *transmit* or the *receive* event of the message. A message is a *signal*, which represents the contents of the message, together with the addresses of the transmitter and the receiver. Formally a signal is a label, and we let $\mathcal{S}$ denote the set of all signals. The transmitters and receivers are lifelines. Let $\mathcal{L}$ denote the set of all lifelines. A message m is defined as a triple $(s, t, r) \in \mathcal{S} \times \mathcal{L} \times \mathcal{L}$ with signal s, transmitter

t and receiver r. $\mathcal{M}$ denotes the set of all messages. On messages we define a transmitter function $tr.\_ \in \mathcal{M} \to \mathcal{L}$ and a receiver function $re.\_ \in \mathcal{M} \to \mathcal{L}$:

$$tr.(s,t,r) \stackrel{\mathsf{def}}{=} t \qquad re.(s,t,r) \stackrel{\mathsf{def}}{=} r$$

We let $\mathcal{K} = \{!, ?\}$ be the set of kinds, where ! represents transmit and ? represents receive. An event e is then a pair of a kind and a message: $(k,m) \in \mathcal{K} \times \mathcal{M}$. $\mathcal{E}$ denotes the set of all events. On events we define a kind function $k.\_ \in \mathcal{E} \to \mathcal{K}$ and a message function $m.\_ \in \mathcal{E} \to \mathcal{M}$:

$$k.(k,m) \stackrel{\mathsf{def}}{=} k \qquad m.(k,m) \stackrel{\mathsf{def}}{=} m$$

We let the transmitter and receiver functions also range over events, $tr.\_, re.\_ \in \mathcal{E} \to \mathcal{L}$, and define a lifeline function $l.\_ \in \mathcal{E} \to \mathcal{L}$ that returns the lifeline of an event:

$$tr.(k,m) \stackrel{\mathsf{def}}{=} tr.m \qquad re.(k,m) \stackrel{\mathsf{def}}{=} re.m \qquad l.e \stackrel{\mathsf{def}}{=} \begin{cases} tr.e \textbf{ if } k.e = ! \\ re.e \textbf{ if } k.e = ? \end{cases}$$

A sequence diagram is built out of events, the binary operators seq, par, alt and xalt, and the unary operators neg and loop. Related to the graphical syntax, the operators represent combined fragments and their arguments the operands. In addition we let skip represent the empty sequence diagram. Let $\mathcal{D}$ be the set of all syntactically correct sequence diagrams. $\mathcal{D}$ is defined recursively as follows:

$$\begin{array}{ll} & \mathsf{skip} \in \mathcal{D} \\ e \in \mathcal{E} & \Rightarrow e \in \mathcal{D} \\ d_1, d_2 \in \mathcal{D} & \Rightarrow d_1 \ \mathsf{seq}\ d_2 \in \mathcal{D} \wedge d_1 \ \mathsf{par}\ d_2 \in \mathcal{D} \wedge \\ & \quad d_1 \ \mathsf{alt}\ d_2 \in \mathcal{D} \wedge d_1 \ \mathsf{xalt}\ d_2 \in \mathcal{D} \\ d \in \mathcal{D} & \Rightarrow \mathsf{neg}\ d \in \mathcal{D} \\ d \in \mathcal{D} \wedge I \subseteq \mathbb{N}_\infty & \Rightarrow \mathsf{loop}\ I\ d \in \mathcal{D} \\ d \in \mathcal{D} \wedge n \in \mathbb{N}_\infty & \Rightarrow \mathsf{loop}\langle n \rangle\ d \in \mathcal{D} \end{array}$$

In the definitions of the two loops we have $\mathbb{N}_\infty \stackrel{\mathsf{def}}{=} \mathbb{N} \cup \{\infty\}$, where ∞ is a number greater than all other numbers and has the property $\infty - 1 = \infty$. The intention behind $\mathsf{loop}\ I\ d$ is that d should be looped any number $n \in I$ times. The UML standard describes two loops $\mathsf{loop}(n)$ and $\mathsf{loop}(n,m)$, where n is the minimum number and m the maximum number of iterations. We may define these as:

$$\mathsf{loop}(n)\ d \stackrel{\mathsf{def}}{=} \mathsf{loop}\ [n..\infty]\ d \qquad \mathsf{loop}(n,m)\ d \stackrel{\mathsf{def}}{=} \mathsf{loop}\ [n..m]\ d$$

As can be expected, we have associativity of seq, par, alt and xalt. We also have commutativity of par, alt and xalt. Furthermore the empty sequence diagram skip is the identity element of seq and par. The combination of skip and loop is discussed in Sect. 3.2.

In this abstract syntax the diagram of Fig. 1 is expressed as:[1]

$$d = (?,(a,env,i))\ \mathsf{seq}\ ((!,(b,i,j))\ \mathsf{seq}\ (?,(b,i,j))\ \mathsf{alt}\ (!,(c,i,j))\ \mathsf{seq}\ (?,(c,i,j)))$$

[1] Here we let env denote the environment of the diagram. Formally this is a gate, but gates are outside the scope of this paper. Also note that seq binds stronger than alt.

3 Operational Semantics

An operational semantics of a language defines an interpreter for the language. In our case the input to the interpreter is a sequence diagram represented in the abstract syntax defined above. The output of the interpreter is a trace of events representing an execution. It is defined as the combination of two transition systems, which we refer to as the *execution system* and the *projection system*. The execution system is a transition system over

$$[\_, \_] \in \mathcal{B} \times \mathcal{D}$$

where $\mathcal{B}$ represents the set of all states of the communication medium and $\mathcal{D}$ the set of all syntactically correct sequence diagrams. The projection system is a transition system over

$$\Pi(\_, \_, \_) \in \mathbb{P}(\mathcal{L}) \times \mathcal{B} \times \mathcal{D}$$

where $\mathbb{P}(\mathcal{L})$ is the powerset of the set of all lifelines. The projection system is used for finding enabled events at each stage of the execution and is defined recursively.

These two systems work together in such a way that for each step in the execution, the execution system updates the projection system by passing on the current state of the communication medium, and the projection system updates the execution system by selecting the event to execute and returning the state of the diagram after the execution of the event.

We also formalize a meta-level that encloses the execution system. At this meta-level we may define several meta-strategies that guide the execution and that are used for formalizing our notions of negative, potential and mandatory behavior.

3.1 The Execution System

The execution system has two rules. The first rule represents the execution of a single event and uses the projection system to find an enabled event to execute. It is defined as

$$[\beta, d] \xrightarrow{e} [update(\beta, e), d'] \textbf{ if } \Pi(ll.d, \beta, d) \xrightarrow{e} \Pi(ll.d, \beta, d') \wedge e \in \mathcal{E} \qquad (1)$$

where e is an event and $ll.d$ is a function returning the set of lifelines in d.

In general we assume the structure of the communication medium, i.e. the means of communication, to be underspecified. The only requirement is that the following functions are defined:

- $add \in \mathcal{B} \times \mathcal{M} \to \mathcal{B}$: Adds a message.
- $rm \in \mathcal{B} \times \mathcal{M} \to \mathcal{B}$: Removes a message.
- $ready \in \mathcal{B} \times \mathcal{M} \to \mathbb{B}$: Returns **true** if the communication medium is in a state where it can deliver the message and **false** otherwise.

The function $update \in \mathcal{B} \times \mathcal{E} \to \mathcal{B}$ is defined as:

$$update(\beta, e) \stackrel{\text{def}}{=} \begin{cases} add(\beta, m.e) \textbf{ if } k.e = ! \\ rm(\beta, m.e) \textbf{ if } k.e = ? \end{cases}$$

Since receiver information is embedded into the messages, these functions are sufficient. In this paper we assume the most general communication model, i.e.: no ordering on the messages. This means that, e.g., message overtaking is possible. Formally then, $\mathcal{B}$ may be defined as the set of all multisets over $\mathcal{M}$, *add* as multiset union, *rm* as multiset minus and *ready* as multiset containment.

The second rule of the execution system executes silent events. The rules of the projection system handle the sequence diagram operators alt, xalt, neg and loop. Resolving these operators, such as choosing the branch of an alt are considered silent events. We define the set of silent events to be

$$\mathcal{T} = \{\tau_{alt}, \tau_{xalt}, \tau_{neg}, \tau_{pos}, \tau_{loop}\}$$

with $\mathcal{E} \cap \mathcal{T} = \emptyset$. The reason for introducing all these different silent events is that they give high flexibility in defining execution strategies by making the silent events and their kinds available at the meta-level. The rule is simple:

$$[\beta, d] \xrightarrow{\tau} [\beta, d'] \textbf{ if } \Pi(ll.d, \beta, d) \xrightarrow{\tau} \Pi(ll.d, \beta, d') \wedge \tau \in \mathcal{T} \tag{2}$$

The empty diagram skip cannot be rewritten, but we assert that it produces the empty trace, i.e.:

$$[\beta, \mathsf{skip}] \xrightarrow{\langle\rangle} [\beta, \mathsf{skip}] \tag{3}$$

This also means that execution terminates when skip is reached.

3.2 The Projection System

The Empty Diagram. It is not possible to rewrite $\Pi(L, \beta, \mathsf{skip})$. skip being the identity element of seq and par, skip seq d, d seq skip, skip par d and d par skip are treated as identical to d.

loop$\langle\infty\rangle$ skip is more problematic. Seen as a program this construct is similar to the java fragment `while(true) { }`, i.e., a program that produces nothing and never terminates. When related to the denotational semantics, however, the semantics of loop$\langle\infty\rangle$ skip should be the empty trace $\langle\rangle$, since the denotational semantics characterize observation after infinite time. A simple solution would be to syntactically disallow the construct all together. Because we do not want to make too many syntactic constraints, and because we want to stay close to the denotational semantics we choose to let loop$\langle\infty\rangle$ skip reduce to skip, even though this may be seen as counter-intuitive from an operational point of view.

Event. The simplest case is the diagram consisting of only one event e. In this case the system delivers the event if the event is enabled given the set of lifelines and the state of the communication medium. This means firstly that the event must belong to one of the lifelines, and secondly that either the event must be a

transmit event or its message must be available in the communication medium. The need for L will be evident in the definition of rules for seq below.

$$\Pi(L,\beta,e) \xrightarrow{e} \Pi(L,\beta,\mathsf{skip}) \textbf{ if } l.e \in L \wedge (k.e = ! \vee ready(\beta, m.e)) \tag{4}$$

Weak Sequencing. The weak sequencing operator seq defines a partial order on the events in a diagram; the ordering of events on each lifeline and between the transmit and receive of a message is preserved, but all other ordering of events is arbitrary. Because of this, there may be enabled events in both the left and the right argument of a seq if there are lifelines represented in the right argument of the operator that are not represented in the left argument. This leads to two rules for the seq operator.

If there is an overlap between the given set of lifelines and the lifelines of the left hand side of the seq, this means that the lifelines in this intersection may have enabled events on the left hand side only. Hence, with respect to these lifelines, the system must look for enabled events in the left operand.

$$\begin{array}{l} \Pi(L,\beta,d_1 \ \mathsf{seq} \ d_2) \xrightarrow{e} \Pi(L,\beta,d_1' \ \mathsf{seq} \ d_2) \\ \quad \textbf{if } ll.d_1 \cap L \neq \emptyset \wedge \Pi(ll.d_1 \cap L,\beta,d_1) \xrightarrow{e} \Pi(ll.d_1 \cap L,\beta,d_1') \end{array} \tag{5}$$

If the lifelines of the left hand side do not exhaust the given set of lifelines, this means there are lifelines only represented on the right hand side, and that there may be enabled events on the right hand side of the operator. This means the system may look for enabled events at the right hand side of the seq, but only with respect to the lifelines not represented on the left hand side.

$$\begin{array}{l} \Pi(L,\beta,d_1 \ \mathsf{seq} \ d_2) \xrightarrow{e} \Pi(L,\beta,d_1 \ \mathsf{seq} \ d_2') \\ \quad \textbf{if } L \setminus ll.d_1 \neq \emptyset \wedge \Pi(L \setminus ll.d_1,\beta,d_2) \xrightarrow{e} \Pi(L \setminus ll.d_1,\beta,d_2') \end{array} \tag{6}$$

Note that the two conditions $ll.d_1 \cap L \neq \emptyset$ and $ll.d_1 \setminus L \neq \emptyset$ are not mutually exclusive. If both these condition are true at the same time there may be enabled events at both sides of the seq operator. These events are then interleaved arbitrarily. In such a case the rules may be applied in arbitrary order.

Because the transitions of the system are used as conditions in the recursion of these rules, the rules will not be applied unless an enabled event is found deeper in the recursion. Because of this the system will always be able to return an enabled event if enabled events exist.

Interleaving. The parallel operator par specifies interleaving of the events from each of its arguments; in other words parallel merge of the executions of each of the arguments. The rules of par are similar to the rules of seq, but simpler since we do not have to preserve any order between the two operands. One of the operands is chosen arbitrarily. As with the seq rules, the use of transitions as the conditions of the rules ensures that an enabled event is found if enabled events exist.

$$\begin{array}{l} \Pi(L,\beta,d_1 \ \mathsf{par} \ d_2) \xrightarrow{e} \Pi(L,\beta,d_1' \ \mathsf{par} \ d_2) \\ \quad \textbf{if } \Pi(ll.d_1 \cap L,\beta,d_1) \xrightarrow{e} \Pi(ll.d_1 \cap L,\beta,d_1') \end{array} \tag{7}$$

$$\begin{array}{l} \Pi(L,\beta,d_1 \text{ par } d_2) \xrightarrow{e} \Pi(L,\beta,d_1 \text{ par } d_2') \\ \quad \textbf{if } \Pi(ll.d_2 \cap L,\beta,d_2) \xrightarrow{e} \Pi(ll.d_2 \cap L,\beta,d_2') \end{array} \tag{8}$$

Choice. The rules for choices end the recursion; the choice is resolved and a silent event is produced. By resolving the choice instead of looking for events deeper down, we ensure that the same choice is made for all the lifelines covered by the choice operator.

$$\Pi(L,\beta,d_1 \text{ alt } d_2) \xrightarrow{\tau_{alt}} \Pi(L,\beta,d_k), \textbf{ for } k \in \{1,2\} \tag{9}$$

$$\Pi(L,\beta,d_1 \text{ xalt } d_2) \xrightarrow{\tau_{xalt}} \Pi(L,\beta,d_k), \textbf{ for } k \in \{1,2\} \tag{10}$$

The rules for alt and xalt are identical except for the kind of event they produce. This reflects the fact that the operators are indistinguishable at the execution level, but since they produce different events, the kind of the choice is available at the meta-level and this will be used in the definition of meta-strategies. Because we have that $ll.(d_1 \text{ alt } d_2) \cap L \neq \emptyset$ and $ll.(d_1 \text{ xalt } d_2) \cap L \neq \emptyset$ no conditions or restrictions on L are needed.

Negative. The operator neg is treated as a choice with one negative branch and one empty branch. Silent events are used to flag which branch is chosen, and hence the choice is made available at the meta-level.

$$\Pi(L,\beta,\text{neg } d) \xrightarrow{\tau_{pos}} \Pi(L,\beta,\text{skip}) \tag{11}$$

$$\Pi(L,\beta,\text{neg } d) \xrightarrow{\tau_{neg}} \Pi(L,\beta,d) \tag{12}$$

Similar to the choice rules, we have that $ll.(\text{neg } d) \cap L = ll.d \cap L \neq \emptyset$.

Iteration. Informally, in loop I d there is a non-deterministic choice between the numbers of I. If $n \in I$ is picked, d should be iterated n times. This is formalized by a rule that chooses which number to use:

$$\Pi(L,\beta,\text{loop } I\ d) \xrightarrow{\tau_{alt}} \Pi(L,\beta,\text{loop}\langle n\rangle\ d) \textbf{ if } n \in I \tag{13}$$

$\text{loop}\langle n\rangle\ d$ is a loop with a counter. In the rule the counter is decreased by one for each iteration. We also produce a silent event to represent the iteration of a loop. Even though iteration of a loop in itself is not the most relevant information at the meta-level, it may be useful for defining execution strategies, for example if we want to give iteration of the loop low priority.

$$\Pi(L,\beta,\text{loop}\langle n\rangle\ d) \xrightarrow{\tau_{loop}} \Pi(L,\beta,d \text{ seq loop}\langle n-1\rangle\ d) \tag{14}$$

Also here we have the property that $ll.(\text{loop}\langle n\rangle\ d) \cap L = ll.d \cap L \neq \emptyset$. Since we have that $\infty - 1 = \infty$, $\text{loop}\langle\infty\rangle\ d$ specifies an infinite loop. Further we assert that $\text{loop}\langle 0\rangle\ d$ is equal to skip, i.e., $\text{loop}\langle 0\rangle\ d \stackrel{\text{def}}{=} \text{skip}$, so we do not need a special rule for this situation.

3.3 Meta-strategies

There are several strategies we may choose when executing a sequence diagram and generating the histories of its possible executions. Examples of this may be generating one or a specific number of random traces, all traces, all prefixes of a certain length, etc. We wish to have the possibility of varying the execution strategy. The way to do this is to define different meta-strategies for executing the diagrams with the operational semantics. Two examples are given below. In both we make use of a meta-system over

$$\{\!|\_, \_|\!\} \in \mathcal{H} \times \mathcal{EX}$$

where $\mathcal{H}$ is the set of all traces and $\mathcal{EX}$ denotes the set of states of the execution system. The first place of this pair is a "container" for a trace and the second place is the current state of the execution system.

One Random Trace. The strategy may be defined by the means of two rules, one rule for normal events and one rule for silent events:

$$\{\!|t, V|\!\} \longrightarrow \{\!|t^\frown\langle e\rangle, V'|\!\} \textbf{ if } V \xrightarrow{e} V' \wedge e \in \mathcal{E} \tag{15}$$

$$\{\!|t, V|\!\} \longrightarrow \{\!|t, V'|\!\} \textbf{ if } V \xrightarrow{\tau} V' \wedge \tau \in \mathcal{T} \tag{16}$$

The initial state for execution of a sequence diagram d is:

$$\{\!|\langle\rangle, [\emptyset, d]|\!\}$$

All Traces. With this strategy we want to generate all possible traces of a diagram d and place them in the correct semantic structure of STAIRS. As explained in Sect. 1, the semantic model of STAIRS is a set of interaction obligations $\{(p_1, n_1), \ldots, (p_m, n_m)\}$. For each interaction obligation (p_i, n_i), p_i is a set of positive traces and n_i is a set of negative traces.

Instead of sets of traces we will use "interaction obligations" of sets of positive and negative executions, i.e. meta-system states. Initially we have a set consisting of a single interaction obligation with the initial state of d as the only positive element and no negative elements:

$$\{(\{\{\!|\langle\rangle, [\emptyset, d]|\!\}\}, \emptyset)\}$$

In the following we define rules that for each execution state deduce the next steps to be made, and in executing these steps rewrite the whole structure. To make the rules more readable, we only show as much of the context, the surrounding structure, as is needed for defining the rules.

If we want all traces, we need to make a branch in the execution every time there is a possibility of more than one event occurring first. The rule for executing events asserts that for a given state, the generation must branch for every enabled event:

$$T \cup \{\{\!|t, V|\!\}\} \longrightarrow T \cup \{\{\!|t^\frown\langle e\rangle, V'|\!\} \mid V \xrightarrow{e} V' \wedge e \in \mathcal{E}\} \tag{17}$$

The rule for resolving an alt is similar. For each branch of the alt, the execution must branch:

$$T \cup \{\{\!|t, V|\!\}\} \longrightarrow T \cup \{\{\!|t, V'|\!\} \mid V \xrightarrow{\tau_{alt}} V'\} \tag{18}$$

The rule for iteration of loop is defined in the same fashion:

$$T \cup \{\{\!|t, V|\!\}\} \longrightarrow T \cup \{\{\!|t, V'|\!\} \mid V \xrightarrow{\tau_{loop}} V'\} \tag{19}$$

The rules for resolving a neg are more complicated since they concern an interaction obligation and not only one of the sets in an interaction obligation. Let P, P', N, N' be sets. The rule for resolving a neg in a valid execution is then:

$$(P \cup \{\{\!|t, V|\!\}\}, N) \longrightarrow (P', N') \tag{20}$$

where

$$P' = P \cup \{\{\!|t, V'|\!\} \mid V \xrightarrow{\tau_{pos}} V'\}$$
$$N' = N \cup \{\{\!|t, V'|\!\} \mid V \xrightarrow{\tau_{neg}} V'\}$$

In an already invalid execution, there is no difference between choosing the positive or negative branch:

$$(P, N \cup \{\{\!|t, V|\!\}\}) \longrightarrow (P, N') \tag{21}$$

where

$$N' = N \cup \{\{\!|t, V'|\!\} \mid V \xrightarrow{\tau_{pos}} V' \vee V \xrightarrow{\tau_{neg}} V'\}$$

Resolving an xalt involves splitting an interaction obligation, and hence, the rules for xalt need even more context:

$$O \cup \{(P \cup \{\{\!|t, V|\!\}\}, N)\} \longrightarrow O \cup \{(P \cup \{\{\!|t, V'|\!\}\}, N) \mid V \xrightarrow{\tau_{xalt}} V'\} \tag{22}$$

$$O \cup \{(P, N \cup \{\{\!|t, V|\!\}\})\} \longrightarrow O \cup \{(P, N \cup \{\{\!|t, V'|\!\}\}) \mid V \xrightarrow{\tau_{xalt}} V'\} \tag{23}$$

Using these rules will in some cases give a result that differs from the denotational semantics. For example, consider the diagram

$$d = (e_1 \ \mathsf{alt} \ e_2) \ \mathsf{seq} \ (e_3 \ \mathsf{xalt} \ e_4)$$

where (for simplicity) the events e_1, e_2, e_3, e_4 are all on the same lifeline. The denotation of d is:

$$[\![\, d \,]\!] = \{(\{\langle e_1, e_3\rangle, \langle e_2, e_3\rangle\}, \emptyset), (\{\langle e_1, e_4\rangle, \langle e_2, e_4\rangle\}, \emptyset)\}$$

The operational semantics gives us executions:

$$[\beta_0, (e_1 \ \mathsf{alt} \ e_2) \ \mathsf{seq} \ (e_3 \ \mathsf{xalt} \ e_4)] \xrightarrow{\tau_{alt}} [\beta_0, e_i \ \mathsf{seq} \ (e_3 \ \mathsf{xalt} \ e_4)]$$
$$\xrightarrow{e_i} [\beta_1, e_3 \ \mathsf{xalt} \ e_4] \xrightarrow{\tau_{xalt}} [\beta_1, e_j] \xrightarrow{e_j} [\beta_2, \mathsf{skip}]$$

with $i \in \{1,2\}$ and $j \in \{3,4\}$. With the above strategy the execution first branches because of the alt, and then the xalt splits the interaction obligation for each of these executions. Because of this we get four interaction obligations:

$$\{(\{\langle e_1,e_3\rangle,\langle e_2,e_3\rangle\},\emptyset),(\{\langle e_1,e_4\rangle,\langle e_2,e_4\rangle\},\emptyset),\\ (\{\langle e_1,e_3\rangle,\langle e_2,e_4\rangle\},\emptyset),(\{\langle e_1,e_4\rangle,\langle e_2,e_3\rangle\},\emptyset)\}$$

To deal with this we need to give priority resolving xalts over resolving the other kinds of choices. We define a special rule allowing xalts on the right hand side of a seq being resolved regardless of the lifeline constraints:

$$\mathit{\Pi}(L,\beta,d_1 \mathsf{\ seq\ } d_2) \xrightarrow{\tau_{xalt}} \mathit{\Pi}(L,\beta,d_1 \mathsf{\ seq\ } d_2') \textbf{ if } \mathit{\Pi}(L,\beta,d_2) \xrightarrow{\tau_{xalt}} \mathit{\Pi}(L,\beta,d_2') \quad (24)$$

In addition (22) and (23) are given priority over (18)-(21). The execution strategy then gives the same interaction obligations as the denotational semantics.

4 Soundness and Completeness

The operational semantics is sound and complete with respect to the denotational semantics presented in [3,4,5]. Informally, the *soundness* property means that if the operational semantics produces a trace from a given diagram, this trace should be included in the denotational semantics of that diagram. By *completeness* we mean that all traces in the denotational semantics of a diagram should be producible applying the operational semantics on that diagram. In this section we state our soundness and completeness results and provide sketches of the proofs. The full proofs are found in [6].

Let $\mathcal{O}$ be the set of all interaction obligations. $[\![d]\!] \in \mathbb{P}(\mathcal{O})$ is the denotation of d (the formal definition is found in Appendix A). We write $t \in [\![d]\!]$ for $t \in \bigcup_{(p,n)\in[\![d]\!]}(p \cup n)$. $E \circledS t$ denotes the trace t with all events not in E filtered away. $env^!_{\mathcal{M}}.d$ is the multiset of messages m such that the receive event but not the transmit event of m is present in d.

Theorem 1 (Termination). *Given a diagram $d \in \mathcal{D}$ without infinite loop. Then execution of $[env^!_{\mathcal{M}}.d, d]$ will terminate.*

Proof. Define a function $w \in \mathcal{D} \to \mathbb{N}$ such that $w(\mathsf{skip}) \stackrel{\mathsf{def}}{=} 0$, $w(e) \stackrel{\mathsf{def}}{=} 1$, $w(d_1 \mathsf{\ seq\ } d_2) = w(d_1 \mathsf{\ par\ } d_2) \stackrel{\mathsf{def}}{=} w(d_1)+w(d_2)+1$, $w(d_1 \mathsf{\ alt\ } d_2) = w(d_1 \mathsf{\ xalt\ } d_2) \stackrel{\mathsf{def}}{=} \max(w(d_1), w(d_2)) + 1$, $w(\mathsf{neg}\ d) \stackrel{\mathsf{def}}{=} w(d) + 1$, $w(\mathsf{loop}\langle n\rangle\ d) \stackrel{\mathsf{def}}{=} n(w(d) + 2)$ and $w(\mathsf{loop}\ I\ d) \stackrel{\mathsf{def}}{=} \max(I)(w(d) + 2) + 1$. It is easy to see that for all $d \in \mathcal{D}$, $w(d) \geq 0$, and that for every execution step $[\beta, d] \xrightarrow{e} [\beta', d']$ or $[\beta, d] \xrightarrow{\tau} [\beta, d']$, $w(d) > w(d')$. Thus, execution of $[env^!_{\mathcal{M}}.d, d]$ must terminate. □

Theorem 2 (Soundness). *Given a diagram $d \in \mathcal{D}$ without infinite loop. For all $t \in (\mathcal{E} \cup \mathcal{T})^*$, if there exists $\beta \in \mathcal{B}$ such that $[env^!_{\mathcal{M}}.d, d] \xrightarrow{t} [\beta, \mathsf{skip}]$ then $\mathcal{E} \circledS t \in [\![d]\!]$.*

Proof. We show this by induction on the structure of d. The induction start $d = \mathsf{skip}$ or $d = e$ is trivial. As induction hypothesis, we assume that the theorem holds for d_1 and d_2. There are seven cases to consider. We start with $d = d_1 \ \mathsf{seq} \ d_2$. Assume that $[env^!_{\mathcal{M}}.d, d_1 \ \mathsf{seq} \ d_2] \xrightarrow{t} [\beta, \mathsf{skip}]$, then by (1), (2), (5), (6), t is obtained by executing d_1 and d_2 in an alternate fashion. This means we have $[env^!_{\mathcal{M}}.d_k, d_k] \xrightarrow{t_k} [\beta_k, \mathsf{skip}]$ such that t is a merge of t_1 and t_2. By the induction hypothesis $\mathcal{E} \circledS t_k \in [\![d_k]\!]$ so we must show that the merge preserves the causality of messages and ordering of events on each lifeline. The first is assured by (1) and (4), and the second by (5) and (6). The proof for $d = d_1 \ \mathsf{par} \ d_2$ is similar except we do not have to think about preserving the ordering along lifelines. For $d = d_1 \ \mathsf{alt} \ d_2$ we must show that $\mathcal{E} \circledS t \in [\![d_k]\!]$ for $k = 1$ or $k = 2$. We observe that, by (2) and (9), $[env^!_{\mathcal{M}}.d, d] \xrightarrow{\tau_{alt}} [env^!_{\mathcal{M}}.d, d_k] \xrightarrow{t} [\beta, \mathsf{skip}]$, so by induction hypothesis $\mathcal{E} \circledS t \in [\![d_k]\!]$. Because $\tau_{alt} \notin \mathcal{E}$ it is sufficient simply to choose the right k. The case of $d = d_1 \ \mathsf{xalt} \ d_2$ is identical. So is $d = \mathsf{neg} \ d_1$ by observing that this is a choice between d_1 and skip. $d = \mathsf{loop} \ I \ d_1$ is treated in the same way as alt, and $d = \mathsf{loop}\langle n \rangle \ d_1$ as n consecutive seqs. □

Theorem 3 (Completeness). *Given a diagram $d \in \mathcal{D}$ without infinite loop. For all $t \in \mathcal{E}^*$, if $t \in [\![d]\!]$ then there exist trace $t' \in (\mathcal{E} \cup \mathcal{T})^*$ and $\beta \in \mathcal{B}$ such that $[env^!_{\mathcal{M}}.d, d] \xrightarrow{t'} [\beta, \mathsf{skip}]$ and $\mathcal{E} \circledS t' = t$.*

Proof. By induction on the structure of d. The induction start $d = \mathsf{skip}$ or $d = e$ is trivial. We assume that the theorem holds for d_1 and d_2 as the induction hypothesis. There are seven cases to consider. Assume $t \in [\![d_1 \ \mathsf{seq} \ d_2]\!]$. Then $t_k \in [\![d_k]\!]$ must exist such that t is a merge of t_1 and t_2, but in such a way that (a) the causality of messages is preserved and (b) for all lifelines l, the events on lifeline l in t_1 precede the events on lifeline l in t_2. By the induction hypothesis $[env^!_{\mathcal{M}}.d_k, d_k] \xrightarrow{t'_k} [\beta_k, \mathsf{skip}]$ and $\mathcal{E} \circledS t'_k = t_k$. This means we may obtain t' such that $t' = \mathcal{E} \circledS t$ by executing d_1 and d_2 in an alternating fashion. (a) ensures that this execution never is blocked by (4) and (b) ensures that execution is never blocked by (6). The case for $d = d_1 \ \mathsf{par} \ d_2$ is similar, but we do not have to take (b) into consideration. For $t \in [\![d_1 \ \mathsf{alt} \ d_2]\!]$ we must have that $t \in [\![d_k]\!]$ for $k = 1$ or $k = 2$. By the induction hypothesis we have t'' such that $[env^!_{\mathcal{M}}.d_k, d_k] \xrightarrow{t''} [\beta_k, \mathsf{skip}]$ and $\mathcal{E} \circledS t'' = t$. By choosing the appropriate k, and letting $t' = \langle \tau_{alt} \rangle^\frown t''$ we easily see that $[env^!_{\mathcal{M}}.d, d] \xrightarrow{t'} [\beta, \mathsf{skip}]$ (by (2) and (9)) and $\mathcal{E} \circledS t' = t$ (because $\tau_{alt} \notin \mathcal{E}$). As above, xalt, neg and loop I are treated in the same way as alt, and $\mathsf{loop}\langle n \rangle$ is treated as n consecutive seqs. □

With respect to diagrams that contain infinite loop, we must assume weak fairness between diagram fragments for the operational semantics to be sound. This means that an arbitrary diagram fragment may not be reachable by the projection system for infinitely many consecutive execution steps without being executed. With this assumption we avoid situations where some part of a diagram only is executed finitely often even though it is inside an infinite loop.

5 Related Work

Several approaches to defining operational semantics for UML sequence diagrams and MSC have been made. The MSC semantics presented in [7] is similar to our execution system, but lacks the formal meta-level. In [8] semantics for the MSC variant Live Sequence Charts (LSC) is defined. This semantics has a meta-level, formalized by pseudo-code, which is used for assigning meaning to invalid executions. In both [9] and [10] LSC semantics is applied to UML sequence diagrams, but none of them conform to the intended UML semantics. In [11] safety and liveness properties are used for distinguishing valid from invalid behavior, but the approach is based on a large amount of transformation and diagrams are not composed by weak sequencing. The MSC semantics presented in [12] has some of the same variability and extendibility that we are aiming at in our semantics, but is heavily based on synchronization of lifelines. The UML semantics of [13] is similar to ours in that it is defined by rewrite rules operating directly on a syntactical representation of sequence diagrams, but treats invalid behavior as the complement of valid behavior.

On inspection of these and other approaches to operational semantics for sequence diagrams and MSCs, like [14, 15, 16, 17, 18, 19, 20], we find that they differ from our semantics in one or more of the following:

- Non-conformance with the intended semantics of UML.
- No notion of explicit negative behavior and no distinction between negative behavior and unspecified behavior
- No distinction between potential and mandatory choice.
- Lack of a proper meta-level that may be used for assigning meaning to negative and potential/mandatory behavior.
- Lack of possibility and freedom in defining and formalizing a meta-level.
- Lack of modifiability and extensibility, e.g., with respect to the communication model.
- Requiring transformations from the textual syntax into the formalism of the approach.

Our aim has been to stay close to the UML standard in both syntax and semantics. Further we have aimed to facilitate ease of extension and modification when adapting the semantics to different interpretations and applications of sequence diagrams.

6 Conclusions

In this paper we have presented an operational semantics for UML 2.0 sequence diagrams. We are not aware of any other operational semantics for UML 2.0 sequence diagrams or MSCs with the same strength and generality as ours. Several approaches have been made, but all with significant shortcomings.

Our operational semantics for UML 2.0 sequence diagrams is simple and is defined with extensibility and variation in mind. It does not involve any translation or transformation of the diagrams into other formalisms, which makes it easy

to use and understand. It is sound and complete with respect to a reasonable denotational formalization of the UML standard.

The operational semantics have a formalized meta-level for defining execution strategies. This meta-level is used for distinguishing valid from invalid traces, and for distinguishing between traces of different interaction obligations. Further it may be used for defining different meta-strategies that guide the execution. We have shown two examples: generating a single trace and generating all traces with a white box view of the diagram. Other examples may be to generate a specific number of traces or prefixes of a specific length. It is also possible to define strategies that take a black box view of the diagram.

The semantics is implemented in the term rewriting language Maude [21], and forms the basis of a tool for analysis of sequence diagrams currently under development. Recent work includes test generation from sequence diagrams; see [22] for more details.

References

1. Object Management Group: Unified Modeling Language: Superstructure, version 2.0. (2005) OMG Document: formal/2005-07-04.
2. International Telecommunication Union: Message Sequence Chart (MSC), ITU-T Recommendation Z.120. (1999)
3. Haugen, Ø., Husa, K.E., Runde, R.K., Stølen, K.: STAIRS towards formal design with sequence diagrams. Software and Systems Modeling **4** (2005) 355–367
4. Haugen, Ø., Husa, K.E., Runde, R.K., Stølen, K.: Why timed sequence diagrams require three-event semantics. In: Scenarios: Models, transformations and tools. International Workshop, Dagstuhl Castle, Germany, September 2003. Revised selected papers. Number 3466 in LNCS. Springer (2005) 1–25
5. Haugen, Ø., Husa, K.E., Runde, R.K., Stølen, K.: Why timed sequence diagrams require three-event semantics. Research report 309, Department of Informatics, University of Oslo (2004) Revised June 2005.
6. Lund, M.S., Stølen, K.: A fully general operational semantics for UML sequence diagrams with potential and mandatory choice. Research report 330, Department of Informatics, University of Oslo (2006)
7. Jonsson, B., Padilla, G.: An execution semantics for MSC-2000. In: 10th International SDL Forum: Meeting UML (SDL'01). Number 2078 in LNCS, Springer (2001) 365–378
8. Harel, D., Marelly, R.: Come, let's play: Scenario-based programming using LSCs and the Play-Engine. Springer (2003)
9. Cavarra, A., Küster-Filipe, J.: Formalizing liveness-enriched sequence diagrams using ASMs. In: 11th International Workshop on Abstract State Machines 2004, Advances in Theory and Practice (ASM'04). Number 3052 in LNCS, Springer (2004) 67–77
10. Harel, D., Maoz, S.: Assert and negate revisited: Modal semantics for UML sequence diagrams. In: 5th International Workshop on Scenarios and State Machines: Models, Algorithms, and Tools (SCESM'06), ACM Press (2006) 13–19
11. Grosu, R., Smolka, S.A.: Safety-liveness semantics for UML 2.0 sequence diagrams. In: 5th International Conference on Application of Concurrency to System Design (ACSD'05), IEEE Computer Society (2005) 6–14

12. Letichevsky, A., Kapitonova, J., Kotlyarov, V., Volkov, V., Letichevsky, A., Weigert, T.: Semantics of Message Sequence Charts. In: 12th International SDL Forum: Model Driven Systems Design (SDL'05). Number 3530 in LNCS, Springer (2005) 117–132
13. Cengarle, M.V., Knapp, A.: Operational semantics of UML 2.0 interactions. Technical report TUM-I0505, Technische Universität München (2005)
14. Alur, R., Etessami, K., Yannakakis, M.: Inference of Message Sequence Charts. IEEE Transactions on Software Engineering **29** (2003) 623–633
15. Alur, R., Etessami, K., Yannakakis, M.: Realizability and verification of MSC graphs. Theoretical Computer Science **331** (2005) 97–114
16. Mauw, S., Reniers, M.A.: Operational semantics for MSC'96. Computer Networks **31** (1999) 1785–1799
17. Mauw, S., Reniers, M.A.: High-level Message Sequence Charts. In: 8th International SDL Forum: Time for Testing, SDL, MSC and Trends (SDL'97), Elsevier (1997) 291–306
18. International Telecommunication Union: Message Sequence Chart (MSC), ITU-T Recommendation Z.120, Annex B: Formal semantics of Message Sequence Charts. (1998)
19. Uchitel, S., Kramer, J., Magee, J.: Incremental elaboration of scenario-based specification and behavior models using implied scenarios. ACM Transactions on Software Engineering and Methodology **13** (2004) 37–85
20. Kosiuczenko, P., Wirsing, M.: Towards an integration of Message Sequence Charts and Timed Maude. Journal of Integrated Design & Process Science **5** (2001) 23–44
21. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: Maude Manual (Version 2.2). SRI International, Menlo Park. (2005)
22. Lund, M.S., Stølen, K.: Deriving tests from UML 2.0 sequence diagrams with neg and assert. In: 1st International Workshop on Automation of Software Test (AST'06), ACM Press (2006) 22–28

A Denotational Semantics

On diagrams we have the constraints that a given message should syntactically occur only once, and if both the transmitter and the receiver lifelines of the message are present in the diagram, then both the transmit event and receive event of that message must be in the diagram. In each trace, a transmit event should always be ordered before the corresponding receive event. We let $\mathcal{H}$ denote the set of all traces that complies with this requirement.

$\mathcal{O}$ is the set of interaction obligations. The semantics of a diagram is defined by a function $[\![\_]\!] \in \mathcal{D} \to \mathbb{P}(\mathcal{O})$. For the empty diagram and the diagram consisting of a single event, the semantics is given by:

$$[\![\mathsf{skip}]\!] \stackrel{\mathsf{def}}{=} \{(\{\langle\rangle\}, \emptyset)\} \qquad [\![e]\!] \stackrel{\mathsf{def}}{=} \{(\{\langle e\rangle\}, \emptyset)\}$$

We define weak sequencing of trace sets:

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

where $e.l$ denotes the set of events that may take place on the lifeline l. The seq construct is defined as:

$$[\![d_1 \,\mathsf{seq}\, d_2]\!] \stackrel{\text{def}}{=} \{o_1 \gtrsim o_2 \mid o_1 \in [\![d_1]\!] \wedge o_2 \in [\![d_2]\!]\}$$

where weak sequencing of interaction obligations is defined as:

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

In order to define par, we first define parallel execution on trace sets:

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

In this definition, the oracle p resolves the non-determinism in the interleaving. π_2 is a projection operator returning the second element of a pair, and $\circledT$ is an operator for filtering pairs of traces (see [6] for formal definitions). The par construct itself is defined as:

$$[\![d_1 \,\mathsf{par}\, d_2]\!] \stackrel{\text{def}}{=} \{o_1 \parallel o_2 \mid o_1 \in [\![d_1]\!] \wedge o_2 \in [\![d_2]\!]\}$$

where parallel execution of interaction obligations is defined as:

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

The semantics of alt is the inner union of the interaction obligations:

$$[\![d_1 \,\mathsf{alt}\, d_2]\!] \stackrel{\text{def}}{=} \{(p_1 \cup p_2, n_1 \cup n_2) \mid (p_1, n_1) \in [\![d_1]\!] \wedge (p_2, n_2) \in [\![d_2]\!]\}$$

The xalt is defined as the union of interaction obligations:

$$[\![d_1 \,\mathsf{xalt}\, d_2]\!] \stackrel{\text{def}}{=} [\![d_1]\!] \cup [\![d_2]\!]$$

The neg construct defines negative traces:

$$[\![\mathsf{neg}\, d]\!] \stackrel{\text{def}}{=} \{(\{\langle\rangle\}, p \cup n) \mid (p, n) \in [\![d]\!]\}$$

The semantics of loop is defined by a semantic loop construct μ_n, where n is the number of times the loop should be iterated. Let $\uplus$ be a generalization of potential choice (inner union of interaction obligation). loop is then defined as:

$$[\![\mathsf{loop}\ I\ d]\!] \stackrel{\text{def}}{=} \biguplus_{i \in I} \mu_i\ [\![d]\!]$$

For $n \in \mathbb{N}$ (finite loop), μ_n is defined as

$$\mu_0\ O \stackrel{\text{def}}{=} \{(\{\langle\rangle\}, \emptyset)\} \qquad \mu_n\ O \stackrel{\text{def}}{=} O \gtrsim \mu_{n-1}\ O \quad \textbf{if } n > 0$$

For a treatment of infinite loop, see [5] or [6].