Equivalence of denotational and operational semantics for interaction languages

Erwan Mahe, Christophe Gaston, Pascale Le Gall

Equivalence of Denotational and Operational Semantics for Interaction Languages

Erwan Mahe[1], Christophe Gaston[1], and Pascale Le Gall[2]

[1] Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France
[2] Université Paris-Saclay, CentraleSupélec, F-91192, Gif-sur-Yvette, France

Abstract. Message Sequence Charts (MSC) and Sequence Diagrams (SD) are graphical models representing the behaviours of distributed and concurrent systems via the scheduling of discrete emission and reception events. So as to exploit them in formal methods, a mathematical semantics is required. In the literature, different kinds of semantics are proposed: denotational semantics, well suited to reason about algebraic properties and operational semantics, well suited to establish verification algorithms. We define an algebraic language to specify so-called interactions, similar to the MSC and SD models. It is equipped with a denotational semantics associating sets of traces (sequences of observed events) to interactions. We then define a structural operational semantics in the style of process algebras and prove the equivalence of the two semantics.

Keywords: interactions, sequence diagrams, distributed & concurrent systems, formal language, denotational semantics, operational semantics

1 Introduction

Modelling asynchronous communications between concurrent processes is possible under a variety of formalisms, such as process algebras [22], Petri Nets [4], series-parallel languages [11], distributed automata [1], or formalisms derived from Message Sequence Charts (MSC) [16]. MSCs are graphical models representing information exchanges between sub-systems. Various offshoots of MSCs, including UML Sequence Diagrams (UML-SD) [18], have been proposed and we call languages from that family "Interaction Languages" (IL). Interactions are interesting due to their graphical nature and ease of understanding. IL make it possible to describe scenarios using intuitions that are very easy to share: a) a vertical line per sub-system, called a lifeline, which from top to bottom describes the succession of events as perceived by the sub-system, b) exchanges of messages inducing causality relations between the lifelines, c) high-level operators such as parallel composition to structure simple scenarios.

So as to use interactions in formal methods, they have to be fitted with formal semantics. A major hurdle in defining those lies in the treatment of weak sequencing. Weak sequencing allows events taking place on different lifelines to occur in any order while strictly ordering those that take place on the same lifeline. The survey [17] provides an overview of solutions found in the literature.

The most direct ones consist in defining semantics by translation: interactions are translated into models of other formalisms provided with formal semantics. Typical examples are Petri Nets [3], automata [10] or process algebra [8]. The main advantage of such approaches is that those formalisms are equipped with tools such as Model-Checker or Model-Based Testing tools. However, a notable drawback is that the target formalisms are defined on concepts (states, transitions, places...) that are quite different from the ones handled in ILs. Then it is difficult to know whether or not the objects resulting from the translation preserve the meaning associated to the original interaction. For example, in [8], the authors propose to translate UML-SD into Communicating Sequential Processes (CSP) [6]. UML-SD operators such as weak sequencing are encoded in a non-trivial manner, using sequence, parallel composition etc. as CSP does not introduce any similar operator, and as the translation is not presented exhaustively, it is not clear if the asynchronous nature of executions of UML-SD is fully reflected. Other approaches treat IL by equipping them with direct mathematical semantics, either denotational or operational. Denotational semantics either rely on partial order sets [23,15] or on algebraic operators [9]. Operational semantics [16] are given in the form of production rules similar to process algebra. Denotational semantics, based on sets of accepted execution traces, are close to intuition. They are well adapted to reason on and prove various properties about interactions. Operational semantics are closer to executable semantics and are well suited to prove the correctness of algorithms realizing formal analysis.

In this paper we set the basis of a framework to deal with interactions via two kinds of semantics (denotational and operational). While in [15,14] we formulated semantics for interactions by identifying the positions of communication actions using the Dewey notation, we now abstract away positions, to define a denotational semantics in an algebraic style as in [9] and an operational semantics in the style of Plotkin. Those new formulations enables us to prove the equivalence of the semantics (with an automated Coq proof available in [13]). To our knowledge, there are no similar equivalence results in the literature. In particular, in [15], there were only some tooled experiments hinting towards their equivalence. Our IL extends the one in [15] with additional loop mechanisms. Our denotational semantics can be seen as an extension of the one in [9] with repetition operators in the form of variants of the algebraic Kleene closure. We define a structural operational semantics in the fashion of process algebras [2]. It adopts some of the ideas introduced in [16,21] but is closer to usual structural operational semantics than the one in [16,21] which includes maps between sent and received messages or negative application rule conditions.

This paper is organized as follows: Section 2 introduces the concepts of interactions and traces. Section 3 presents the syntax of our IL and defines a trace semantics in denotational-style. In Section 4 a structural operational semantics is defined in the style of process calculi and we demonstrate its equivalence to the former in Section 5. Finally, in Sections 6 and 7 we discuss some related works and we conclude. A formalisation using Coq of the main demonstrations is available in [13].

2 Basic interactions & intuition of their meaning

Interactions describe the behavior of distributed and concurrent systems based on their internal and external communications. They are defined over a signature $\Omega = (L, M)$ where L is a set of lifelines and M is a set of messages.

2.1 Preliminaries

The executions of systems are characterized by sequences of events called communication actions (actions for short) which are of two kinds: either the emission of a message $m \in M$ from a lifeline $l \in L$, denoted by $l!m$, or the reception of $m \in M$ by $l \in L$, denoted by $l?m$. $\mathbb{A}_\Omega$ denotes the set of actions over Ω. For any such action a, $\theta(a)$ denotes the lifeline on which a occurs.

Sequences of actions, called traces, are words in $\mathbb{A}_\Omega^*$, with "." denoting the concatenation operation and ϵ being the empty trace. We denote by $\mathbb{T}_\Omega = \mathbb{A}_\Omega^*$ the set of traces. Thus, for any two traces t_1 and t_2, $t_1.t_2$ is the trace composed of the sequence of actions of t_1 followed by the sequence of actions of t_2. We introduce operators to compose (sets of) traces, modeling different notions of scheduling: the strict sequencing (;), the interleaving ($||$) and the weak sequencing ($\between$).

The set $t_1;t_2$ of strict sequencing of traces t_1 and t_2 is defined as $\{t_1.t_2\}$. By choosing ";" for denoting the extension of "." to sets of traces, we adopt the same notation as in [9] for the strict sequencing operator.

Interleaving allows elements of distinct traces to be reordered w.r.t. one another while preserving the order that is specific to each trace. The set $t_1||t_2$ of interleavings of traces t_1 and t_2 is defined by:

$$\begin{array}{cc} \epsilon||t_2 = \{t_2\} & t_1||\epsilon = \{t_1\} \\ (a_1.t_1)||(a_2.t_2) = \{a_1.t \mid t \in t_1||(a_2.t_2)\} \cup \{a_2.t \mid t \in (a_1.t_1)||t_2\} & \end{array}$$

By contrast, weak sequencing only allows such permutations when actions do not occur on the same lifeline. We define a predicate $t \between l$ meaning that the trace t contains an action on the lifeline l (we say t has conflicts w.r.t. l):

$$\epsilon \between l = \bot \qquad \text{and} \qquad (a.t) \between l = (\theta(a) = l) \vee (t \between l)$$

By overloading $\between$, the set $t_1 \between t_2$ of weak sequencing of t_1 and t_2 is defined by:

$$\begin{array}{cc} \epsilon \between t_2 = \{t_2\} & t_1 \between \epsilon = \{t_1\} \\ (a_1.t_1) \between (a_2.t_2) = \{a_1.t \mid t \in t_1 \between (a_2.t_2)\} \cup \{a_2.t \mid t \in (a_1.t_1) \between t_2,\ \neg(a_1.t_1 \between \theta(a_2))\} & \end{array}$$

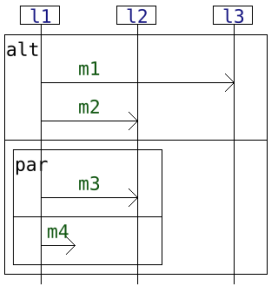
When defining $t_1' \between t_2'$, the order of the actions in each trace is preserved and actions in t_2' can only precede those in t_1' that do not occur on the same lifeline. This explains the two subsets constituting $(a_1.t_1) \between (a_2.t_2)$: the first one contains all traces whose first action is a_1 and tail belongs to $t_1 \between (a_2.t_2)$ and the second one is empty if lifeline of a_2 occurs in $a_1.t_1$ (i.e. $\neg(a_1.t_1 \between \theta(a_2))$), and contains all traces whose first action is a_2 and tail belongs to $(a_1.t_1) \between t_2$ otherwise.

The previous binary operators (";", "$\between$" and "||") defined on traces are canonically extended to sets of traces as follows: with $\diamond \in \{;, \between, ||\}$, $T_1 \diamond T_2$ is the union

of all the sets $t_1 \diamond t_2$ with $t_1 \in T_1$ and $t_2 \in T_2$. The use of the strict sequencing (";"), weak sequencing ("⨳") and interleaving ("||") operators will be illustrated with Fig.1 in Section 2.2.

2.2 Basic interactions

An example of interaction is given in the left of Fig.1. Lifelines l_1, l_2 and l_3 are drawn as vertical lines. Emission and reception actions are drawn as horizontal arrows carrying the transmitted messages m_1, m_2, m_3 and m_4 and which respectively exit the emitting lifeline or point towards the receiving lifeline. When a direct emission-reception causality occurs, we draw both actions as a single arrow from the emitter towards the receiver.



$$= \left(\begin{array}{c} (\{l_1!m_1\};\{l_3?m_1\}) \\ ⨳(\{l_1!m_2\};\{l_2?m_2\}) \end{array} \right) \cup \left(\begin{array}{c} (\{l_1!m_3\};\{l_2?m_3\}) \\ ||\{l_1!m_4\} \end{array} \right)$$

$$= \left(\begin{array}{c} \{l_1!m_1.l_3?m_1\} \\ ⨳\{l_1!m_2.l_2?m_2\} \end{array} \right) \cup \left(\begin{array}{c} \{l_1!m_3.l_2?m_3\} \\ ||\{l_1!m_4\} \end{array} \right)$$

$$= \left\{ \begin{array}{ll} l_1!m_1.l_3?m_1.l_1!m_2.l_2?m_2, & l_1!m_3.l_2?m_3.l_1!m_4, \\ l_1!m_1.l_1!m_2.l_3?m_1.l_2?m_2, & l_1!m_3.l_1!m_4.l_2?m_3, \\ l_1!m_1.l_1!m_2.l_2?m_2.l_3?m_1 & l_1!m_4.l_1!m_3.l_2?m_3 \end{array} \right\}$$

Fig. 1: Example of a basic interaction & its trace semantics

The top to bottom direction relates to time passing. An action (arrow) drawn above another one generally occurs beforehand. This scheduling of actions corresponds to the weak sequencing operator. By contrast, strict sequencing may be used to enforce precedence relations between actions occurring on different lifelines. These two scheduling operators will be respectively denoted by the keywords *seq* and *strict*. Other keywords (*alt*, *par*, *loop*) will be also used for denoting other scheduling mechanisms. In Fig.1, the arrow carrying m_1 and specifying its passing between l_1 and l_3 is modelled by the interaction $strict(l_1!m_1, l_3?m_1)$. Using the *strict* operator here obliges $l_3?m_1$ to occur after $l_1!m_1$, which reflects the causality of the passing of message m_1 between l_1 and l_3. The fact that this arrow stands above that carrying m_2 can be modelled using the weak sequencing operator: $seq(strict(l_1!m_1, l_3?m_1), strict(l_1!m_2, l_2?m_2))$. Using *seq* here instead of *strict* allows for instance $l_2?m_2$ to occur before $l_3?m_1$ even though the latter is drawn above. However $l_1!m_2$ cannot occur before $l_1!m_1$ because they both occur on l_1. Note that, in contrast to *strict*, the *seq* operator has no graphical representation in diagrams, as it corresponds to the default scheduling operator.

Parallel and alternative compositions can also be used. On Fig.1, the passing of m_3 and the emission of m_4 are scheduled using parallel composition. In the diagram representation this corresponds to the box labelled with "par", modelled

by the term $par(strict(l_1!m_3, l_2?m_3), l_1!m_4)$. Actions scheduled with *par* can occur in any order w.r.t. one another. Here, $l_1!m_4$ can occur before $l_1!m_3$, after $l_2?m_3$ or in between those two actions. Alternative composition is an exclusive non-deterministic choice between behaviors. Like *par*, *alt* is drawn as a box labelled with "alt". The global term describing the left of Fig.1 is given by:

$$alt(seq(strict(l_1!m_1, l_3?m_1), strict(l_1!m_2, l_2?m_2)), par(strict(l_1!m_3, l_2?m_3), l_1!m_4))$$

2.3 Repetition operators on sets of traces

Scheduling operators define compositions of traces obtained from enabling or forbidding the reordering of actions according to some scheduling policy. All three are associative (in addition, $||$ is commutative) and admit $\{\epsilon\}$ as a neutral element. We define (Kleene) closures of those operators to specify repetitions[3]:

Definition 1 (Kleene closures). *For any $\diamond \in \{;, \between, ||\}$ and any $T \in \mathcal{P}(\mathbb{T}_\Omega)$, the Kleene closure $T^{\diamond *}$ of T is defined by: $T^{\diamond *} = \bigcup_{j\in\mathbb{N}} T^{\diamond j}$ with $T^{\diamond 0} = \{\epsilon\}$ and $T^{\diamond j} = T \diamond T^{\diamond(j-1)}$ for $j > 0$.*

$$\left\{ \begin{array}{l} l_1!m_1.l_2!m_2, \\ l_2?m_1 \end{array} \right\}^{\between *} = \left\{ \begin{array}{ll} \epsilon, & l_1!m_1.l_2!m_2.l_1!m_1.l_2!m_2, \\ l_1!m_1.l_2!m_2, & l_1!m_1.l_1!m_1.l_2!m_2.l_2!m_2, \\ l_2?m_1, & l_1!m_1.l_2!m_2.l_2?m_1, \\ & l_2?m_1.l_2?m_1, \\ & l_2?m_1.l_1!m_1.l_2!m_2, \\ & l_1!m_1.l_2?m_1.l_2!m_2, \end{array} \quad \cdots \right\}$$

Fig. 2: Example illustrating the weak Kleene closure

The three Kleene closures $^{;*}$, $^{\between *}$ and $^{||*}$ are respectively called, strict, weak and interleaving Kleene closures. Within the K-closure $T^{\diamond *}$ we can find traces obtained from the repetition (using $\diamond$ as a scheduler) of any number of traces of T. In the example from Fig.2 we consider a set T containing two traces $l_1!m_1.l_2!m_2$ and $l_2?m_1$. The first 3 powersets of T (i.e. $T^{\between 0} \cup T^{\between 1} \cup T^{\between 2}$) are displayed, the rest of the weak K-closure of T (i.e. $T^{\between *}$) is represented by the $\cdots$.

We have the property $T \diamond T^{\diamond *} = T^{\diamond *}$, analogous to the one defining replication $!P$ (i.e. $!P = P|!P$) in the family of process calculi (e.g. see [19]), expressing an unbounded number of copies of P along the parallel composition "$|$". For $\diamond \in \{;, \between, ||\}$ whenever $a.t \in T_1 \diamond T_2$ (with a and t any action and trace and T_1 and T_2 any sets of traces), it may be so that action a is taken from a trace $a.t'$ that belongs to either T_1 or T_2. Definition 2 introduces restricted versions of the scheduling operators so as to impose action a to be taken from T_1.

[3] For a set E, $\mathcal{P}(E)$ is the set of all subsets of E.

Definition 2 (Restricted scheduling operators). *For any* $\diamond \in \{;,\ \between,\ ||\}$, *we define the operator* $\diamond^{\curvearrowleft}$ *such that for any sets of traces* T_1 *and* T_2 *we have:*

$$T_1 \diamond^{\curvearrowleft} T_2 = \{\ t \in T_1 \diamond T_2 \mid (t = a.t') \Rightarrow (\exists\ t_1 \in \mathbb{T}_\Omega,\ s.t.\ (a.t_1 \in T_1)\ \wedge\ (t' \in \{t_1\} \diamond T_2))\ \}$$

As an example, given $T_1 = \{l_1!m.l_1?m\}$ and $T_2 = \{l_2!m\}$, we have:

$$T_1 \between T_2 = \left\{ \begin{array}{l} l_1!m.l_1?m.l_2!m, \\ l_1!m.l_2!m.l_1?m, \\ l_2!m.l_1!m.l_1?m \end{array} \right\} \quad \text{and} \quad T_1 \between^{\curvearrowleft} T_2 = \left\{ \begin{array}{l} l_1!m.l_1?m.l_2!m, \\ l_1!m.l_2!m.l_1?m \end{array} \right\}$$

We now define Head-First closures (abbr. HF-closure) of scheduling operators.

Definition 3 (Head-first closures). *For any* $\diamond \in \{;,\ \between,\ ||\}$, *we define the Head-First closure of* $\diamond$ *as* ${}^{\diamond^{\curvearrowleft}*}$ *i.e. the Kleene closure of the restricted* $\diamond^{\curvearrowleft}$ *operator.*

In the following we will show that HF-closure and K-closure are equivalent for ; and || but that this is not the case for $\between$.

Lemma 1. *For any* $\diamond \in \{;,\ ||\}$, $T \in \mathcal{P}(\mathbb{T}_\Omega)$, t *in* $\mathbb{T}_\Omega$ *and* $a \in \mathbb{A}_\Omega$ *we have:*

$$(a.t \in T^{\diamond *}) \Rightarrow (\exists\ t' \in \mathbb{T}_\Omega\ s.t.\ (a.t' \in T) \wedge\ (t \in \{t'\} \diamond T^{\diamond *}))$$

Proof. By induction on j given $a.t \in T^{\diamond j}$. For ||, we use its commutativity. □

Lemma 2 (Equivalence of HF & K closures for ; & ||). *For any set of traces* T, *we have* ${T^{;}}^{;^{\curvearrowleft}*} = T^{;*}$ *and* ${T^{||}}^{||^{\curvearrowleft}*} = T^{||*}$.

Proof. By induction on a member trace t. □

Let us detail a counter example showing that the weak K-closure ${}^{\between *}$ and the weak HF-closure ${}^{\between^{\curvearrowleft}*}$ are not equivalent. Given $T = \{l_1!m_1.l_2?m_1,\ l_2!m_2\}$, let us consider the powerset $T^{\between 2}$ of T. By definition, $\{l_2!m_2\} \between \{l_1!m_1.l_2?m_1\} \subset T^{\between 2}$. Here we can choose to take $l_1!m_1$ as a first action and therefore $t = l_1!m_1.l_2!m_2.l_2?m_1 \in T^{\between 2}$. However, $t \notin T \between^{\curvearrowleft} T = T^{\between^{\curvearrowleft} 2}$ and more generally, for any j smaller or greater that 2, $t \notin T^{\between^{\curvearrowleft} j}$. Hence, $T^{\between *} \not\subseteq T^{\between^{\curvearrowleft}*}$.

3 Syntax & denotational semantics

As noted earlier in Section 2.2, interactions terms are defined inductively. Basic building blocks include the empty interaction $\varnothing$ which specifies the empty behavior ϵ (observation of no action) and any atomic action a of $\mathbb{A}_\Omega$, which specifies the single-element trace a. More complex behavior can then be specified inductively using the binary constructors *strict*, *seq*, *par* and *alt* and the unary constructors $loop_S$ (strict loop), $loop_H$ (head loop up to $\between$), $loop_W$ (weak loop) and $loop_P$ (parallel loop).

Definition 4 (Interaction Language). *We denote by $\mathbb{I}_\Omega$ the set of terms inductively defined by the set of operation symbols $\mathcal{F} = \mathcal{F}_0 \cup \mathcal{F}_1 \cup \mathcal{F}_2$ s.t.:*

- *symbols or arity 0 (constants) are $\mathcal{F}_0 = \{\varnothing\} \cup \mathbb{A}_\Omega$*
- *symbols of arity 1 are $\mathcal{F}_1 = \{loop_S,\ loop_H,\ loop_W,\ loop_P\}$*
- *symbols of arity 2 are $\mathcal{F}_2 = \{strict,\ seq,\ par,\ alt\}$*

In Definition 4 we define our Interaction Language (IL) as a set of terms $\mathbb{I}_\Omega$ inductively defined from the set of symbols $\mathcal{F}$ with arity in $\mathbb{N}$. The set $\mathcal{P}(\mathbb{T}_\Omega)$ of sets of traces admits the structure of a $\mathcal{F}$-algebra using operators introduced in section 2. The denotational semantics of interactions is then defined in Definition 5 using the initial homomorphism associated to this $\mathcal{F}$-algebra.

Definition 5 (Denotational semantics). *$\mathcal{A} = (\mathcal{P}(\mathbb{T}_\Omega),\ \{f^{\mathcal{A}} \mid f \in \mathcal{F}\})$ is the $\mathcal{F}$-algebra defined by the following interpretations of the operation symbols in $\mathcal{F}$:*

$$\begin{array}{lll} & strict^{\mathcal{A}} = ; & loop_S^{\mathcal{A}} = {}^{;*} \\ \varnothing^{\mathcal{A}} = \{\epsilon\} & seq^{\mathcal{A}} = \between & loop_H^{\mathcal{A}} = {}^{\between\urcorner *} \\ a^{\mathcal{A}} = \{a\} & par^{\mathcal{A}} = \| & loop_W^{\mathcal{A}} = {}^{\between *} \\ & alt^{\mathcal{A}} = \cup & loop_P^{\mathcal{A}} = {}^{\|*} \end{array}$$

The denotational semantics σ_d of $\mathbb{I}_\Omega$ is the unique $\mathcal{F}$-homomorphism $\sigma_d : \mathbb{I}_\Omega \to \mathcal{P}(\mathbb{T}_\Omega)$ between the free term $\mathcal{F}$-algebra[4] $\mathcal{T}_\mathcal{F}$ and $\mathcal{A}$.

The semantics of constants $\varnothing$ and $a \in \mathbb{A}_\Omega$ are sets containing a single element being respectively $\{\epsilon\}$ and $\{a\}$. The *strict*, *seq*, *par* and *alt* symbols are respectively associated to the ;, $\between$, $\|$ and union $\cup$ operators on sets of traces. The use of the strict sequencing (";"), weak sequencing ("$\between$") and interleaving ("||") operators is illustrated on the right of Fig.1 so as to compute the semantics of the interaction example given on the left of Fig.1. For instance, from the second line to the third line, using distinct colours to better visualise the differences between scheduling operators, weak sequencing $\{l_1!m_1.l_3?m_1\} \between \{l_1!m_2.l_2?m_2\}$ allows $l_1!m_2$ to be reordered before $l_3?m_1$ but not before $l_1!m_1$ while interleaving $\{l_1!m_3.l_2?m_3\} \| \{l_1!m_4\}$ allows $l_1!m_4$ to be placed anywhere w.r.t. $l_1!m_3$ and $l_2?m_3$. The resulting set of traces is given on the bottom right.

From a system designer perspective, using $loop_S$, $loop_H$, $loop_W$ or $loop_P$ is motivated by different goals:

- With $loop_S(i)$, each instance of a repeatable behavior must be executed entirely before any other can start. We can use $loop_S$ to specify some critical repeatable behavior of which there can only exist one instance at a time.
- With $loop_P(i)$, all existing instances can be executed concurrently w.r.t. one another, and, at any given moment, new instances can be created. $loop_P$ can therefore be used to specify protocols in which any number of new sessions can be created and run in parallel.

[4] The free term $\mathcal{F}$-algebra is defined by interpreting symbols of $\mathcal{F}$ as constructors of new terms: for $f \in \mathcal{F}$ of arity j, for $t_1, \ldots t_j \in \mathbb{I}_\Omega$, $f(t_1, \ldots t_j)$ is interpreted as itself.

4 A structural operational semantics

Now we present our structural operational semantics. It relies on the definition (by structural induction) of two predicates: "$i \downarrow$" (the termination predicate) indicates that the interaction i accepts the empty trace and "$i \xrightarrow{a} i'$" (the execution relation) indicates that traces $a.t$ such that t is accepted by i' are accepted by i. The relation $\rightarrow$ allows the determination, for any interaction i, of which actions a can be immediately executed, and, of potential follow-up interactions i' which express continuations t of traces $a.t$ accepted by i. Defining an execution relation $\rightarrow$ is a staple of process calculus [2]. We will pay particular attention to the weak sequencing operator in Section 4.2 before defining $\rightarrow$ in Section 4.3.

4.1 Termination

By reasoning on the structure of an interaction term i, we can determine whether or not the empty trace ϵ belongs to its semantics. When this holds, we say that i terminates and use the notation $i \downarrow$ as in [2,16].

Definition 6 (Termination). *The predicate* $\downarrow \subset \mathbb{I}_\Omega$ *is such that for any* i_1 *and* i_2 *from* $\mathbb{I}_\Omega$*, any* $f \in \{strict, seq, par\}$ *and any* $k \in \{S, H, W, P\}$ *we have:*

$$\frac{}{\varnothing \downarrow} \qquad \frac{i_1 \downarrow}{alt(i_1, i_2) \downarrow} \qquad \frac{i_2 \downarrow}{alt(i_1, i_2) \downarrow} \qquad \frac{i_1 \downarrow \quad i_2 \downarrow}{f(i_1, i_2) \downarrow} \qquad \frac{}{loop_k(i_1) \downarrow}$$

All rules of Definition 6 are evident. The empty interaction $\varnothing$ only accepts ϵ, and thus terminates. An interaction with a loop at its root terminates because it is possible to repeat zero times its content. As $alt(i_1, i_2)$ specifies a choice, it terminates iff either i_1 or i_2 terminates. An interaction of the form $f(i_1, i_2)$, with f being a scheduling constructor, terminates iff both i_1 and i_2 terminate. The rules are consistent with the denotational semantics:

Lemma 3 (Termination w.r.t. σ_d). *For any* $i \in \mathbb{I}_\Omega$*,* $(i \downarrow) \Leftrightarrow (\epsilon \in \sigma_d(i))$

Proof. By induction on the term structure of interactions. □

In summary, $i \downarrow$ means that i may terminate immediately, but because of non-determinism, depending on the nature of i, i may allow arbitrary long traces.

4.2 Dealing with weak-sequencing using evasion & pruning

Weak sequencing only allows interleavings between actions that occur on different lifelines. As a result, within an interaction of the form $i = seq(i_1, i_2)$, some actions that can be executed in i_2 (i.e. such that $i_2 \xrightarrow{a} i_2'$) may also be executed in $seq(i_1, i_2)$, i.e. such that $seq(i_1, i_2) \xrightarrow{a} i'$. In other words, given a trace $a.t \in \sigma_d(i)$, action a might correspond to an action expressed by i_2. This is however conditioned by the ability of i_1 to express traces that have no conflict w.r.t. a so that a may be placed in front of what is expressed by i_1 when recomposing $a.t$.

We define the evasion predicate as a weaker notion than the termination predicate $\downarrow$. The evasion predicate "$\downarrow^{\not\ast}$" can be described as a form of local termination. For a lifeline l, we say that i evades l, denoted by $i \downarrow^{\not\ast} l$ if i accepts at least one trace that does not contain actions occurring on l.

Definition 7 (Evasion). *The predicate $\downarrow^{\not\ast} \subset \mathbb{I}_\Omega \times L$ is such that for i_1 and i_2 in $\mathbb{I}_\Omega$, $l \in L$, $a \in \mathbb{A}_\Omega$, $f \in \{strict, seq, par\}$ and $k \in \{S, H, W, P\}$ we have:*

$$\frac{}{\varnothing \downarrow^{\not\ast} l} \qquad \frac{\theta(a) \neq l}{a \downarrow^{\not\ast} l}$$

$$\frac{i_1 \downarrow^{\not\ast} l}{alt(i_1, i_2) \downarrow^{\not\ast} l} \qquad \frac{i_2 \downarrow^{\not\ast} l}{alt(i_1, i_2) \downarrow^{\not\ast} l} \qquad \frac{i_1 \downarrow^{\not\ast} l \qquad i_2 \downarrow^{\not\ast} l}{f(i_1, i_2) \downarrow^{\not\ast} l} \qquad \frac{}{loop_k(i_1) \downarrow^{\not\ast} l}$$

The empty interaction $\varnothing$ evades any lifeline as ϵ contains no action. An interaction reduced to a single action a evades l iff a does not occur on l. As for termination, an interaction with a loop at its root evades any lifeline because it accepts ϵ. Choice and scheduling operators are also handled in the same manner as for the termination predicate. Moreover, we consider the collision predicate $\not\downarrow^{\not\ast}$ by considering dual structural rules w.r.t. those defining the evasion predicate $\downarrow^{\not\ast}$ so that we have: $i \not\downarrow^{\not\ast} l$ iff $\neg(i \downarrow^{\not\ast} l)$).

Lemma 4 (Evasion w.r.t. σ_d).
For any $l \in L$ and $i \in \mathbb{I}_\Omega$,$(i \downarrow^{\not\ast} l) \Leftrightarrow (\exists\ t \in \sigma_d(i), \neg(t \not\ast l))$

Proof. By induction on the term structure of interaction. □

Let us remark that, for any $i \in \mathbb{I}_\Omega$, if $i \downarrow$ then $\forall\ l \in L$, $i \downarrow^{\not\ast} l$. Indeed, ϵ has no conflict w.r.t. any l. The opposite does not hold: it suffices to consider $i = alt(l_1!m, l_2!m)$ and observe that $\forall\ l \in \{l_1, l_2\}$, $i \downarrow^{\not\ast} l$ holds while $i \downarrow$ does not.

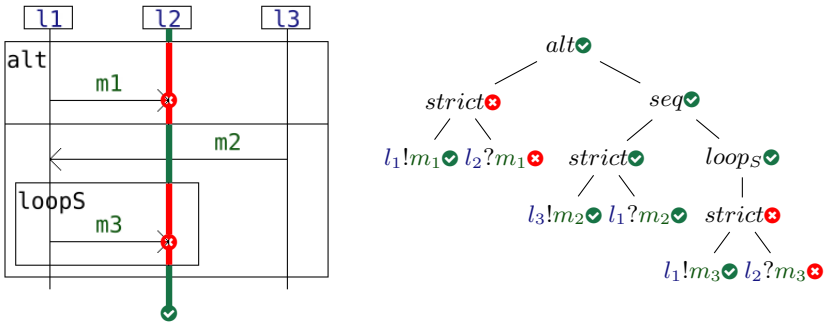


Fig. 3: Illustration of the evasion predicate (here w.r.t. lifeline l_2)

The application of the evasion predicate (w.r.t. lifeline l_2) is illustrated on Fig.3. On the right is represented the syntactic structure of an interaction i, and,

on the left, the corresponding drawing as a sequence diagram. On the syntax tree, the nodes are decorated with symbols ✔ (resp. ✖) to signify that the sub-interaction underneath that node evades (resp. collides with) l_2. Starting from the leaves we can decorate all nodes and conclude once the root is reached. By taking the right branch of the alternative and by choosing not to instantiate the loop, we can see that i accepts some traces that have no conflict w.r.t. lifeline l_2 (in our case, only the trace $l_3!m_2.l_1?m_2$). As a result the interaction i verifies $i \downarrow^{\divideontimes} l_2$. On the diagram representation, evasion can be illustrated by drawing a line over l_2 the lifeline of interest. This line can be decomposed into several areas that are colored either in green or in red. The coloration depends on whether the sub-interaction corresponding to the operand evades or collides with l_2.

Provided that $i_1 \downarrow^{\divideontimes} \theta(a)$, an action a that is executable in i_2 i.e. s.t. $i_2 \xrightarrow{a} i'_2$ is also executable in $i = seq(i_1, i_2)$. However, this is not enough to define a rule $seq(i_1, i_2) \xrightarrow{a} i'$ compatible with the semantics σ_d. i' must specify continuations t s.t. $a.t \in \sigma_d(i)$. Continuation traces t are built from traces $t_1 \in \sigma_d(i_1)$ and t_2 such that $\neg(t_1 \divideontimes \theta(a))$ and $a.t_2 \in \sigma_d(i_2)$. By defining i'_1 as the interaction which expresses exactly traces t_1 s.t. $\neg(t_1 \divideontimes \theta(a))$ we may produce a rule $seq(i_1, i_2) \xrightarrow{a} seq(i'_1, i'_2)$. The computation of i'_1 is called *pruning* and is defined as an inductive relation $\divideontimes\!\!\longrightarrow$ s.t. $i \overset{l}{\divideontimes\!\!\rightarrow} i'$ indicates that the pruning of $i \in \mathbb{I}_\Omega$ w.r.t. $l \in L$ yields $i' \in \mathbb{I}_\Omega$. Pruning is defined so that $\sigma_d(i') \subseteq \sigma_d(i)$ is the maximum subset of $\sigma_d(i)$ that contains no trace conflicting with l (see Lemma 6).

Definition 8 (Pruning). *The pruning relation* $\divideontimes\!\!\longrightarrow \subset \mathbb{I}_\Omega \times L \times \mathbb{I}_\Omega$ *is s.t. for any* $l \in L$*, any* $f \in \{strict, seq, par\}$ *and any* $k \in \{S, H, W, P\}$*:*

$$\frac{}{\varnothing \overset{l}{\divideontimes\!\!\rightarrow} \varnothing} \qquad \frac{}{a \overset{l}{\divideontimes\!\!\rightarrow} a}\ \theta(a) \neq l \qquad \frac{i_1 \overset{l}{\divideontimes\!\!\rightarrow} i'_1 \qquad i_2 \overset{l}{\divideontimes\!\!\rightarrow} i'_2}{f(i_1, i_2) \overset{l}{\divideontimes\!\!\rightarrow} f(i'_1, i'_2)}$$

$$\frac{i_1 \overset{l}{\divideontimes\!\!\rightarrow} i'_1 \qquad i_2 \overset{l}{\divideontimes\!\!\rightarrow} i'_2}{alt(i_1, i_2) \overset{l}{\divideontimes\!\!\rightarrow} alt(i'_1, i'_2)} \qquad \frac{i_1 \overset{l}{\divideontimes\!\!\rightarrow} i'_1}{alt(i_1, i_2) \overset{l}{\divideontimes\!\!\rightarrow} i'_1}\ i_2 \not\downarrow^{\divideontimes} l \qquad \frac{i_2 \overset{l}{\divideontimes\!\!\rightarrow} i'_2}{alt(i_1, i_2) \overset{l}{\divideontimes\!\!\rightarrow} i'_2}\ i_1 \not\downarrow^{\divideontimes} l$$

$$\frac{i_1 \overset{l}{\divideontimes\!\!\rightarrow} i'_1}{loop_k(i_1) \overset{l}{\divideontimes\!\!\rightarrow} loop_k(i'_1)} \qquad \frac{}{loop_k(i_1) \overset{l}{\divideontimes\!\!\rightarrow} \varnothing}\ i_1 \not\downarrow^{\divideontimes} l$$

Evasion and pruning are intertwined notions. Indeed, as per Lemma 5 evasion is equivalent to the existence and unicity of a pruned interaction.

Lemma 5 (Conditional existence & unicity for pruning).
For any $i \in \mathbb{I}_\Omega$ *and any* $l \in L$, $(i \downarrow^{\divideontimes} l) \Leftrightarrow (\exists!\ i' \in \mathbb{I}_\Omega \text{ s.t. } i \overset{l}{\divideontimes\!\!\rightarrow} i')$

Proof. By induction on the term structure of interactions. □

Let us comment on the rules defining the pruning relation. We have $\varnothing \overset{l}{\divideontimes\!\!\rightarrow} \varnothing$ because the semantics of $\varnothing$ being $\{\epsilon\}$, there are no conflicts w.r.t. l. Any action $a \in \mathbb{A}_\Omega$ is prunable iff $\theta(a) \neq l$. In such a case, a needs not be eliminated and thus $a \overset{l}{\divideontimes\!\!\rightarrow} a$. For $i = alt(i_1, i_2)$ to be prunable we must have either or both of $i_1 \downarrow^{\divideontimes} l$

or $i_2 \downarrow^{\divideontimes} l$. If both branches evade l they can be pruned and kept as alternatives in the new interaction term. If only a single one does, we only keep the pruned version of this single branch. For any scheduling constructor f, if $i = f(i_1, i_2)$, in order to have $i \downarrow^{\divideontimes} l$ we must have both $i_1 \downarrow^{\divideontimes} l$ and $i_2 \downarrow^{\divideontimes} l$. In that case the unique interaction i' such that $i \overset{l}{\divideontimes\!\!\rightarrow} i'$ is defined as the scheduling, using f, of the pruned versions of i_1 and i_2. For loops $i = loop_k(i_1)$ with $k \in \{S, H, W, P\}$, we distinguish two cases: (a) if $i_1 \not\downarrow^{\divideontimes} l$ then any execution of i_1 will yield a trace conflicting l and repetitions should be forbidden; (b) if $i_1 \downarrow^{\divideontimes} l$ repetitions are kept, given that i_1 can be pruned as $i_1 \overset{l}{\divideontimes\!\!\rightarrow} i'_1$. This being the modification which preserves a maximum amount of traces, we have $loop_k(i_1) \overset{l}{\divideontimes\!\!\rightarrow} loop_k(i'_1)$.

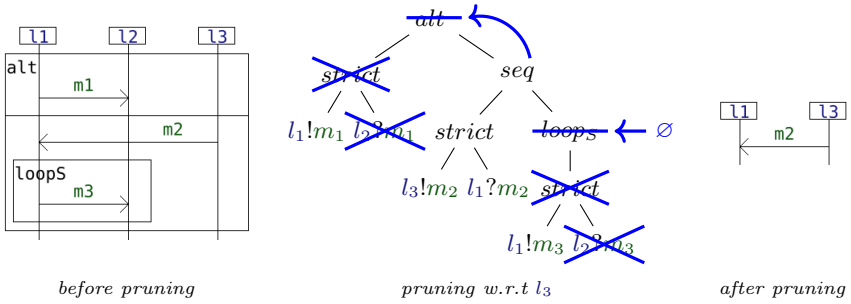


Fig. 4: Illustration of pruning

We have seen that the interaction i of Fig.3 satisfies $i \downarrow^{\divideontimes} l_2$. Therefore Lemma 5 implies the existence of a unique i' s.t. $i \overset{l_2}{\divideontimes\!\!\rightarrow} i'$. Fig.4 illustrates the computation of i'. The blue lines represent the modifications in the syntax of i that occur during its pruning into i'. On Fig.3 we decorated sub-interactions of i with ⛔ whenever they did not evade l_2. During pruning, those sub-interactions must be eliminated given that the resulting term must not express actions occurring on l_2. Hence, on Fig.4, we have crossed in blue the problematic sub-interactions. The root node is an *alt*. Let us note $i = alt(i_1, i_2)$. On Fig.3 we have seen that we have $i_1 \not\downarrow^{\divideontimes} l_2$ and $i_2 \downarrow^{\divideontimes} l_2$. Therefore we have $i \overset{l_2}{\divideontimes\!\!\rightarrow} i'_2$ with i'_2 being such that $i_2 \overset{l_2}{\divideontimes\!\!\rightarrow} i'_2$. This selection of the right branch of the *alt* is illustrated on Fig.4 by the curved arrow which "replaces" the *alt* by the *seq* on its bottom right. There remains to determine i'_2 s.t. $i_2 \overset{l_2}{\divideontimes\!\!\rightarrow} i'_2$. At the root of i_2 we have a *seq*. Let us note $i_2 = seq(i_A, i_B)$. As per Fig.3 we have both $i_A \downarrow^{\divideontimes} l_2$ and $i_B \downarrow^{\divideontimes} l_2$ and therefore $i'_2 = seq(i'_A, i'_B)$ such that $i_A \overset{l_2}{\divideontimes\!\!\rightarrow} i'_A$ and $i_B \overset{l_2}{\divideontimes\!\!\rightarrow} i'_B$. Underneath i_A, no actions occur on l_2 and hence $i'_A = i_A$. At the root of i_B we have a $loop_S$. Let us note $i_B = loop_S(i_C)$. As per Fig.3 we have $i_C \not\downarrow^{\divideontimes} l_2$ and therefore $i'_B = \varnothing$ which is illustrated on Fig.4 by the $\leftarrow \varnothing$ in blue, which "replaces" the $loop_S$ by $\varnothing$. Finally there remain $i' = seq(i_A, \varnothing)$, which is drawn on the right of Fig.4.

Lemma 6 states that given $i \overset{l}{\nleftrightarrow} i'$, the pruned interaction i' exactly specifies all the executions of i that do not involve l.

Lemma 6 (Pruning w.r.t. σ_d). *For any $l \in L$ and any i and i' from $\mathbb{I}_\Omega$:*

$$(i \overset{l}{\nleftrightarrow} i') \Rightarrow (\sigma_d(i') = \{t \in \sigma_d(i) \mid \neg(t \between l)\})$$

Proof. By induction on the term structure of interactions. □

4.3 Execution relation & operational semantics

A structural operational semantic in the style of Plotkin [20] allows determining traces $t = a_1.\cdots.a_n$ through the assertion of a succession of predicates of the form $i_j \xrightarrow{a_j} i_{j+1}$ representing the evolution of the system. By expressing action a_j, the system goes from being modelled by i_j to being modelled by i_{j+1}.

Definition 9 (Execution relation).
The execution relation $\rightarrow \subset \mathbb{I}_\Omega \times \mathbb{A}_\Omega \times \mathbb{I}_\Omega$ is s.t.:

$$\frac{}{a \xrightarrow{a} \varnothing} \qquad \frac{i_1 \xrightarrow{a} i_1'}{alt(i_1,i_2) \xrightarrow{a} i_1'} \qquad \frac{i_2 \xrightarrow{a} i_2'}{alt(i_1,i_2) \xrightarrow{a} i_2'}$$

$$\frac{i_1 \xrightarrow{a} i_1'}{par(i_1,i_2) \xrightarrow{a} par(i_1',i_2)} \qquad \frac{i_2 \xrightarrow{a} i_2'}{par(i_1,i_2) \xrightarrow{a} par(i_1,i_2')}$$

$$\frac{i_1 \xrightarrow{a} i_1'}{strict(i_1,i_2) \xrightarrow{a} strict(i_1',i_2)} \qquad \frac{i_2 \xrightarrow{a} i_2'}{strict(i_1,i_2) \xrightarrow{a} i_2'}\ i_1 \downarrow$$

$$\frac{i_1 \xrightarrow{a} i_1'}{seq(i_1,i_2) \xrightarrow{a} seq(i_1',i_2)} \qquad \frac{i_1 \overset{\theta(a)}{\nleftrightarrow} i_1' \qquad i_2 \xrightarrow{a} i_2'}{seq(i_1,i_2) \xrightarrow{a} seq(i_1',i_2')}$$

$$\frac{i_1 \xrightarrow{a} i_1'}{loop_S(i_1) \xrightarrow{a} strict(i_1', loop_S(i_1))} \qquad \frac{i_1 \xrightarrow{a} i_1'}{loop_H(i_1) \xrightarrow{a} seq(i_1', loop_H(i_1))}$$

$$\frac{i_1 \xrightarrow{a} i_1' \qquad loop_W(i_1) \overset{\theta(a)}{\nleftrightarrow} i'}{loop_W(i_1) \xrightarrow{a} seq(i', seq(i_1', loop_W(i_1)))} \qquad \frac{i_1 \xrightarrow{a} i_1'}{loop_P(i_1) \xrightarrow{a} par(i_1', loop_P(i_1))}$$

Many of the rules are directly similar to those in use for process algebras. In an interaction reduced to an action a, a may be executed with $\varnothing$ as remaining interaction. If within $i = alt(i_1, i_2)$, action a can be executed in either i_1 or i_2 with either $i_1 \xrightarrow{a} i_1'$ or $i_2 \xrightarrow{a} i_2'$ then it may be executed in i and the resulting interaction is either i_1' or i_2'. For $i = par(i_1, i_2)$, if we have either $i_1 \xrightarrow{a} i_1'$ or $i_2 \xrightarrow{a} i_2'$ then a may be executed in i and the resulting interaction naturally is either $par(i_1', i_2)$ or $par(i_1, i_2')$. Executing actions on the left of either a *strict* or a *seq* follows the same rule as in the case of a *par* because no precedence relations are enforced on the left-hand side. However, an action a may be executed on the right of $i = strict(i_1, i_2)$ only if i_1 terminates. Indeed, in that case i_1 may

express the empty trace ϵ as per Lemma 3 and nothing prevents a to be the first action to be executed. The resulting interaction is then i'_2 given that we force i_1 to express ϵ. Likewise, within $i = seq(i_1, i_2)$ there is a condition for executing an action a on the right. This condition is that $i_1 \downarrow^{\divideontimes} \theta(a)$, which, as per Lemma 5 is implied by the condition $i_1 \divideontimes \xrightarrow{\theta(a)} i'_1$. Finally, we obtain $i \xrightarrow{a} seq(i'_1, i'_2)$ given that $i_2 \xrightarrow{a} i'_2$ and that the pruning of i_1 up $\theta(a)$ yields i'_1.

Let us look at rules for loop operators $loop_S$, $loop_H$ and $loop_P$ which look the same, i.e. $loop_k(i) \xrightarrow{a} f(i', loop_k(i))$ under the condition $i \xrightarrow{a} i'$ and using the notation $(k, \diamond, f) \in \{(S, ;, strict), (H, \divideontimes, seq), (P, ||, par)\}$.

Any $t \in \sigma_d(f(i', loop_k(i)))$ verifies $t \in \{t_1\} \diamond \sigma_d(loop_k(i))$ for a certain $t_1 \in \sigma_d(i')$. If action a comes from the first iteration of the loop i.e. $a.t \in \{a.t_1\} \diamond^{\urcorner} \sigma_d(loop_k(i)) \subset \sigma_d(i) \diamond^{\urcorner} \sigma_d(loop_k(i))$, it coincides with using the restricted operator $\diamond^{\urcorner}$ as a scheduler. It turns out that $loop_H$ is explicitly associated to ${}^{\divideontimes\urcorner}*$ and thus the formulation of its rule is self-evident. In the case of $loop_S$ and $loop_P$ it is the fact that the HF and K-closures of ; and $||$ are equivalent (as per Lemma 2) which enables their respective rules to be formulated in this manner.

The rule for $loop_W$ allows for the first action to be taken from a later iteration of the loop. Let us consider $i = loop_W(i_1)$ and $a.t \in \sigma_d(i)$. The rule is formulated such that $t \in \sigma_d(seq(i', seq(i'_1, i)))$ with $i \divideontimes \xrightarrow{\theta(a)} i'$ and $i_1 \xrightarrow{a} i'_1$. Given that i is a loop, it is always prunable (Lemma 5) so there exists i' s.t. $i \divideontimes \xrightarrow{\theta(a)} i'$. The fact that $t \in \sigma_d(seq(i', seq(i'_1, i)))$ translates into having $t \in \sigma_d(i') \divideontimes \sigma_d(i'_1) \divideontimes \sigma_d(i)$. Then, if a is taken from the first iteration of the loop, then, given that $\epsilon \in \sigma_d(i')$ (Lemma 6) we have $t \in \{\epsilon\} \divideontimes \{t'_1\} \divideontimes \sigma_d(i)$ with $t_1 = a.t'_1 \in \sigma_d(i_1)$. If a is taken from the second iteration of the loop, let us consider $t_1 \in \sigma_d(i_1)$ the first iteration and $t_2 = a.t'_2 \in \sigma_d(i_1)$ the second one (from which a is taken and hence $t'_2 \in \sigma_d(i'_1)$). We have $t \in \{t_1\} \divideontimes \{t'_2\} \divideontimes \sigma_d(i)$ and the condition $\neg(t_1 \divideontimes \theta(a))$. This condition implies, as per Lemma 6 that $\{t_1\} \subset \sigma_d(i')$. The reasoning is the same when a is taken from later instances. Let us consider $a.t \in \{t_1\} \divideontimes \cdots \divideontimes \{t_{n-1}\} \divideontimes \{t'_n\} \divideontimes \sigma_d(i)$. We then have $\{t_1\} \divideontimes \cdots \divideontimes \{t_{n-1}\} \subset \sigma_d(i')$ because i' is either a loop (and therefore absorbing) or $\varnothing$ (all the t_j are then ϵ). Hence the rule indeed allows a to be taken from any iteration.

The predicates $\downarrow$ and $\rightarrow$ ground the operational semantics σ_o given below:

Definition 10 (Operational semantics). *$\sigma_o : \mathbb{I}_\Omega \rightarrow \mathcal{P}(\mathbb{T}_\Omega)$ is s.t.:*

$$\frac{i \downarrow}{\epsilon \in \sigma_o(i)} \qquad \frac{t \in \sigma_o(i') \qquad i \xrightarrow{a} i'}{a.t \in \sigma_o(i)}$$

4.4 Illustrative example

On Fig.5 we illustrate both the operational semantics and an example showcasing the difference between $loop_H$ and $loop_W$. Execution trees are drawn with the help of the HIBOU tool described in [14]. We consider repetitions of $i = alt(strict(l_1!m_1, l_2?m_1), l_2!m_2)$. On the first row, we illustrate the construction of a trace accepted by $seq(i, i)$ where i is repeated twice using weak sequencing. Here, the second occurrence of action $l_1!m_1$ (at the bottom) is immediately

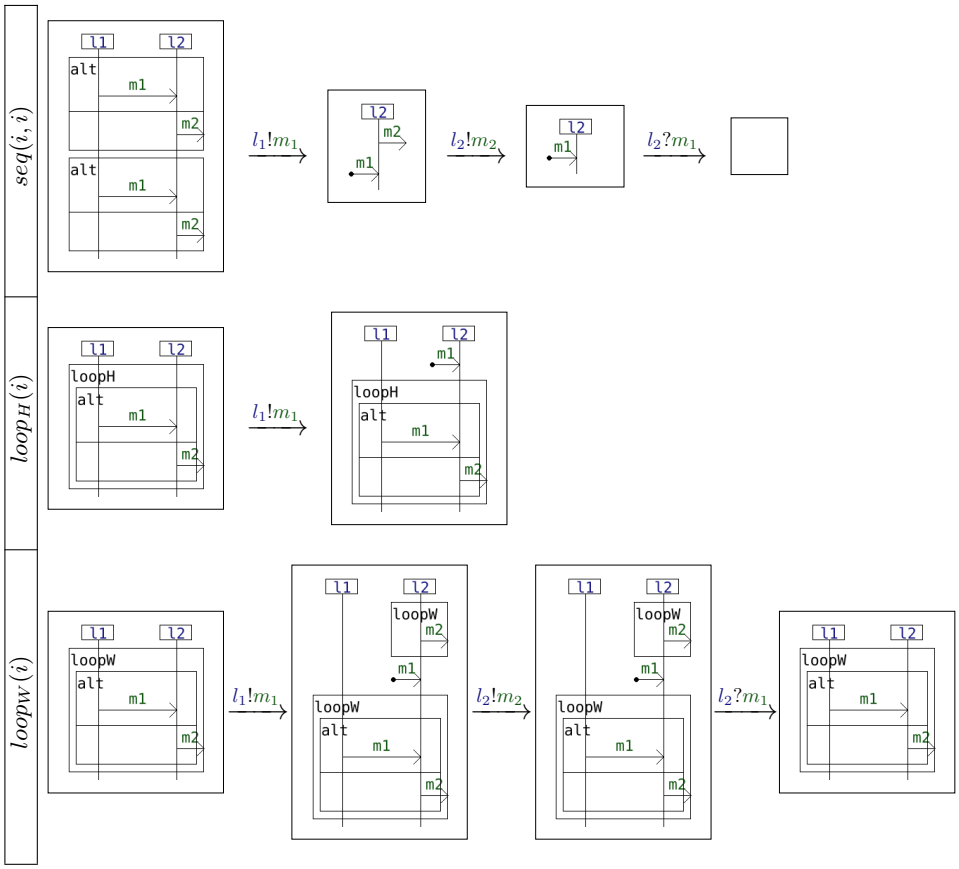


Fig. 5: Illustration of the operational semantics & of the counter-example from Sec.2.3

executable because, with pruning, we can force the choice of the right branch of the first alternative which evades l_1. At the end, the trace $t = l_1!m_1.l_2!m_2.l_2?m_1$ is expressed by $seq(i, i)$. Now, if we consider $loop_H(i)$, we get what is illustrated on the second row of Fig.5. We can manage to execute the first action $l_1!m_1$ but from that point, the second action of t which is $l_2!m_2$ is not executable. Indeed, the presence of $l_2?m_1$ at the top of the diagram prevents it to be executed. As $loop_H$ is associated to the weak HF-closure $^{*\circlearrowleft *}$ and not to the K-closure **, it is therefore expected that t could not be accepted by $loop_H(i)$ in this example. However, on the third row of Fig.5, $loop_W(i)$ can recognize t. The addition of the pruned version of the loop allows one to delay the determination of the instance as part of which the initial $l_1!m_1$ is executed.

5 Proving the equivalence of both semantics

In the following we prove the equivalence of σ_o and σ_d. A formalisation of the proofs using Coq is available in [13].

Let us at first prove that for any interaction i we have $\sigma_o(i) \subseteq \sigma_d(i)$. The first step to do so is to characterize the execution relation "$\rightarrow$" w.r.t. σ_d:

Lemma 7 (Property 1 of $\rightarrow$ w.r.t. σ_d). *For any $a \in \mathbb{A}_\Omega$, $t \in \mathbb{T}_\Omega$ and i and i' from $\mathbb{I}_\Omega$, $\big((i \xrightarrow{a} i') \wedge (t \in \sigma_d(i'))\big) \Rightarrow (a.t \in \sigma_d(i))$*

Proof. By induction on cases that make $i \xrightarrow{a} i'$ possible. □

Lemma 7 and Lemma 3 state that the σ_d semantics accepts the same two construction rules (that for the empty trace ϵ and that for non empty traces of the form $a.t$) as those that define σ_o inductively. As a result any trace that is accepted according to σ_o is also be accepted according to σ_d:

Theorem 1 (Inclusion of σ_o in σ_d). *For any $i \in \mathbb{I}_\Omega$ we have $\sigma_o(i) \subseteq \sigma_d(i)$*

Proof. By induction on a member trace t. □

Let us now prove the reciprocate, i.e. that for any interaction i, $\sigma_d(i) \subseteq \sigma_o(i)$. We provide, with Lemma 8, a second characterization of "$\rightarrow$" w.r.t. σ_d.

Lemma 8 (Property 2 of $\rightarrow$ w.r.t. σ_d). *For any $a \in \mathbb{A}_\Omega$, $t \in \mathbb{T}_\Omega$ and $i \in \mathbb{I}_\Omega$, $(a.t \in \sigma_d(i)) \Rightarrow \big(\exists\ i' \in \mathbb{I}_\Omega,\ \ (i \xrightarrow{a} i') \wedge (t \in \sigma_d(i'))\big)$*

Proof. By induction on the term structure of interactions. □

Thanks to Lemma 8 and Lemma 3 we conclude with Theorem 2:

Theorem 2 (Inclusion of σ_d in σ_o). *For any $i \in \mathbb{I}_\Omega$ we have $\sigma_d(i) \subseteq \sigma_o(i)$*

Proof. By induction on a member trace t. □

We have therefore proven both inclusion and can conclude that the operational semantics σ_o is indeed equivalent to the denotational-style semantics σ_d.

6 Related works

Unlike some other works (e.g. [9]), we do not have a dedicated construction for the passing of a message from a lifeline to another. We formulate this in the form $strict(a!m, b?m)$, expressing that the emission of message m on lifeline a precedes its reception on b. In [17] a survey of formal semantics associated to UML-SDs is proposed. It is notable that UML-SDs are described semi-formally in the norm [18]. This allows for a rich language with operators such as *assert* or *negate* [5] which are not covered in our IL. However a full formalisation proves difficult, as explained in [17,5]. Most formal approaches rely on translations towards other formalisms [3] or consist in denotational semantics [23] that are most often based

on partial order sets. The extent to which UML-SDs are formalised may vary [17]: some works formalize loops [12], others do not [9], and some only allow finitely many iterations [23]. In all cases where there are loops, only one loop operator is proposed and may correspond to either $loop_H$ or $loop_W$.

Our denotational semantics is inspired from [9]. We have completed their definitions dealing with loop operators. [16] introduces an operational semantics for MSC using a termination predicate and an execution relation. Similarities between [16] and our work include the use of pruning which, in [16], relates to a "permission relation". In [16], loops are not handled and there is no *strict* constructor: direct causal relations between actions occurring on different lifelines (e.g. emission-reception of a message) are handled by maps, updated during executions. Moreover, rules involve negative conditions such as $i \overset{a}{\not\rightarrow}$ expressing that it is not possible to find an interaction i' verifying $i \overset{a}{\rightarrow} i'$. This way of doing reduces the set of rules to be considered, but does not give clear access to reasoning about the rule system itself, in particular reasoning about semantics equivalence. In [21], a loop construction for weak sequencing composition is considered in addition to the constructions discussed in [16]. Rules in [21] include two rules similar to our rules for $loop_H$ and $loop_W$ so that the semantics includes the two ways of dealing with composition according to the weak sequencing $\diamond$.

Earlier works of ours [15,14] focused on the static analysis of traces against interactions. We have proposed an algorithmicised semantics allowing us, given an interaction i and an action a, to compute the follow-up interaction i' whose traces are licit extensions of a with respect to i. This previous semantics was defined in a functional style by identifying the actions likely to start a trace by means of their position in the interaction term. The use of positions makes the semantics less readable and understandable than a structural operational semantics *à la* process algebra, and hampers reasoning about the semantics. Novel contributions in this paper w.r.t. [15,14] consist in the distinction of $loop_W$ & $loop_H$, the formulation of a denotational semantics in an algebraic style rather than using precedence relations, the formulation of a structural operational semantics and primarily, a proof of equivalence between both semantics.

7 Conclusion & further work

In this paper we define an IL including weak and strict sequencing, parallel and alternative composition as well as four distinct loop operators to specify different kinds of repetition. We formulate the semantics of this IL: (1) in denotational-style, making use of composition & algebraic operators and (2) in operational-style by reconstructing accepted traces via the succession of atomic executions. The equivalence of both formulations is proven (Coq proof in [13]). We currently investigate how to enrich our language with some form of value passing i.e. instead of exchanging abstract messages we may interpret them concretely or symbolically with typed data. This last point is notably addressed in some process calculi frameworks [7].

References

1. Akshay, S., Bollig, B., Gastin, P., Mukund, M., Narayan Kumar, K.: Distributed timed automata with independently evolving clocks. In: van Breugel, F., Chechik, M. (eds.) 19th Conf. on Concurrency Theory (CONCUR). pp. 82–97. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
2. Baeten, J.: Process algebra with explicit termination. Computing science reports, Technische Universiteit Eindhoven (2000)
3. Eichner, C., Fleischhack, H., Meyer, R., Schrimpf, U., Stehno, C.: Compositional semantics for uml 2.0 sequence diagrams using petri nets. In: Prinz, A., Reed, R., Reed, J. (eds.) SDL 2005: Model Driven. pp. 133–148. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
4. Haddad, S., Khmelnitsky, I.: Dynamic recursive petri nets. In: Janicki, R., Sidorova, N., Chatain, T. (eds.) Application and Theory of Petri Nets and Concurrency. pp. 345–366. Springer International Publishing, Cham (2020)
5. Harel, D., Maoz, S.: Assert and negate revisited: Modal semantics for UML sequence diagrams. Software and Systems Modeling **7**(2), 237–252 (2008)
6. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall (1985)
7. Ingólfsdóttir, A., Lin, H.: A symbolic approach to value-passing processes. In: Bergstra, J., Ponse, A., Smolka, S. (eds.) Handbook of Process Algebra, pp. 427–478. Elsevier Science, Amsterdam (2001). https://doi.org/10.1016/B978-044482830-9/50025-4
8. Jacobs, J., Simpson, A.C.: On a process algebraic representation of sequence diagrams. In: Canal, C., Idani, A. (eds.) Software Engineering and Formal Methods - SEFM 2014 Collocated Workshops: HOFM, SAFOME, OpenCert, MoKMaSD, WS-FMDS, Grenoble, France, September 1-2, 2014, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8938, pp. 71–85. Springer (2014). https://doi.org/10.1007/978-3-319-15201-1_5, https://doi.org/10.1007/978-3-319-15201-1_5
9. Knapp, A., Mossakowski, T.: UML Interactions Meet State Machines - An Institutional Approach. In: 7th Conf. on Algebra and Coalgebra in Computer Science (CALCO). Leibniz International Proceedings in Informatics (LIPIcs), vol. 72 (2017)
10. Knapp, A., Wuttke, J.: Model checking of uml 2.0 interactions. In: Kühne, T. (ed.) Models in Software Engineering. pp. 42–51. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
11. Lodaya, K., Weil, P.: Series-parallel languages and the bounded-width property. Theor. Comput. Sci. **237**(1–2), 347–380 (Apr 2000). https://doi.org/10.1016/S0304-3975(00)00031-1
12. Lu, L., Kim, D.K.: Required behavior of sequence diagrams: Semantics and conformance. ACM Trans. Softw. Eng. Methodol. **23**(2) (Apr 2014). https://doi.org/10.1145/2523108
13. Mahe, E.: Coq proof for the equivalence of the semantics. erwanm974.github.io/coq_hibou_label_semantics_equivalence/, accessed: 2021-10-14
14. Mahe, E., Bannour, B., Gaston, C., Lapitre, A., Le Gall, P.: A small-step approach to multi-trace checking against interactions. In: Proceedings of the 36th Annual ACM Symposium on Applied Computing. p. 1815–1822. SAC '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3412841.3442054

15. Mahe, E., Gaston, C., Le Gall, P.: Revisiting semantics of interactions for trace validity analysis. In: Wehrheim, H., Cabot, J. (eds.) Fundamental Approaches to Software Engineering. pp. 482–501. Springer International Publishing, Cham (2020)
16. Mauw, S., Reniers, M.A.: Operational semantics for msc'96. Computer Networks **31**(17), 1785–1799 (1999). https://doi.org/10.1016/S1389-1286(99)00060-2
17. Micskei, Z., Waeselynck, H.: The many meanings of uml 2 sequence diagrams: a survey. Software & Systems Modeling **10**(4), 489–514 (2011)
18. OMG: Unified Modeling Language v2.5.1. omg.org/spec/UML/2.5.1/PDF (12 2017)
19. Parrow, J.: An introduction to the π-calculus. In: Bergstra, J.A., Ponse, A., Smolka, S.A. (eds.) Handbook of Process Algebra, pp. 479–543. North-Holland / Elsevier (2001)
20. Plotkin, G.: A structural approach to operational semantics. The Journal of Logic and Algebraic Programming **60-61**, 17–139 (07 2004). https://doi.org/10.1016/j.jlap.2004.05.001
21. Reniers, M.: Message sequence chart : syntax and semantics. Ph.D. thesis, Mathematics and Computer Science (1999). https://doi.org/10.6100/IR524323
22. Rensink, A., Wehrheim, H.: Weak sequential composition in process algebras. In: Jonsson, B., Parrow, J. (eds.) 5th Conf. on Concurrency Theory (CONCUR). pp. 226–241. Lecture Notes in Computer Science, Springer (1994). https://doi.org/10.1007/BFb0015012
23. Störrle, H.: Semantics of interactions in uml 2.0. In: IEEE Symposium on Human Centric Computing Languages and Environments, 2003. Proceedings. 2003. pp. 129–136 (10 2003). https://doi.org/10.1109/HCC.2003.1260216