Denotational and operational semantics for interaction languages : application to trace analysis

Erwan Mahe, Christophe Gaston, Pascale Le Gall

Denotational and Operational Semantics for Interaction Languages : application to trace analysis

Erwan Mahe [a], Christophe Gaston [a], Pascale Le Gall [b]

[a] *Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France*
[b] *Université Paris-Saclay, CentraleSupélec, F-91192, Gif-sur-Yvette, France*

Abstract

Graphical depictions of distributed systems' behaviors in the form of Sequence Diagrams (SD) are widely used, with formalisms such as Message Sequence Charts (MSC) or UML-SD. Yet, only restricted subsets of these languages are associated to formal semantics, most of which are given by translation towards other formalisms. These translational approaches are the only ones enabling formal verification thanks to the ecosystem and tools associated to the target formalism. However, traceability of SD concepts is lost and the translation of some operators, in particular the weakly sequential loop, is problematic.

In this paper, we define an Interaction Language to encode SD and ground it formally by associating it to three different semantics which we prove to be equivalent. A "denotational" semantics, relying on composing operators over sets of traces (sequences of atomic actions) allows one to reason on algebraic properties of SD. A structural "operational" semantics apprehends SD as executable objects which can express traces one action after the other. It also bridges the gap between the two other semantics and enables proving their equivalence. A functional style "execution" semantics separates two concerns intertwined in the operational semantics that is: identifying immediately executable actions (frontier actions) and computing follow-up SD which specify continuations of behaviors. The use of positions to identify frontier actions resolves non-determinism as every distinct occurrence of these actions have unique positions and are associated to a unique follow-up SD. Additionally, this enables visualizing frontier actions in SD as well as the execution of SD.

These three semantics constitute a framework which enable using SD directly for formal verification. Traceability of SD concepts and executed actions is preserved and the weakly sequential loops are treated as any other operator.

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

1. Introduction

In this work, we use the terminology *Interaction Languages* to refer to a family of languages that stem from the original Message Sequence Charts

(MSC) [13]. It includes various offshoots of MSC such as Live Sequence Charts (LSC) [6], and the widely used UML Sequence Diagrams (UML-SD) [34]. *Interactions* are models whose central purpose is to describe communication flow between different actors. They are particularly adapted to describe Distributed Systems (DS) defined here as systems composed of several sub-systems deployed on different machines and communicating via message passing. An interaction defines the intended behaviors of a system and its graphical counterpart allows the visualization of these behaviors as follows: **(1)** each sub-system is associated with a vertical line, called a lifeline, which from top to bottom describes the succession of events as perceived by the sub-system, **(2)** message passing is depicted as arrows between lifelines and **(3)** high-level operators enable, among other, the scheduling or repeating of more simple scenarios.

The intuitive aspect of the graphical representation of interactions makes them popular for purposes such as documentation. Interaction Languages such as MSC and UML-SD have a standardized syntax and tool support which democratized their use. However, and this is particularly the case for UML-SD, their semantics are not necessarily formally defined (see the survey [33]). Attempts to formalize MSC or UML-SD have led to a rich literature. The survey [33] provides an overview of such attempts for UML-SD. Roughly speaking, two kinds of approaches are mostly found: on the one hand, semantics defined directly in relation to the syntax of the language and on the other hand, semantics obtained by systematic translations towards formalisms which are themselves associated with formal semantics.

Semantics derived from the syntax of formal languages can be defined in various manners (denotational, operational with big or small steps, axiomatic, etc.). In the literature, such semantics for interaction languages are mostly either denotational or operational. In [40, 29] a denotational semantics of interactions is given using partial order sets and in [16] it relies on algebraic operators. These denotational semantics directly manipulate behaviors specified by interactions in the form of partial orders on events or sequences of events. As a result, they are well adapted to reason on and prove various properties about interactions themselves. For instance, they can be used to identify behavioral equivalence (syntactically distinct interactions which specify the same set of behaviors) or inclusion (an interaction which specifies a subset of the behaviors specified by another). In [32] an operational semantics of interactions is given in the form of production rules similar to process algebra. By contrast, an operational semantics lies closer to having executable models and is better suited to implement and prove the correctness of algorithms realizing (among others) Runtime Verification. However, to the best of our knowledge, none of these approaches were further developed to include more expressive interactions and to exploit them in a tooled formal setting (for instance for formal verification).

By contrast, most approaches to exploit interactions in this manner provide formal semantics by means of a translation of the interactions (with their standard syntax but informal semantics) towards another formalism which has a well-established formal semantics and tool support (e.g., Petri Nets [8], automata [17, 1, 39], sets of communicating automata [4] or process algebra [14]).

The main advantage of such approaches is that those formalisms are equipped with tools such as model-checkers or model-based testing tools. These translations consist in associating syntactic constructs of the interaction language with syntactic constructs of the target language. However, the target formalisms may be defined on concepts (states, transitions, places, etc.) that are quite different from the ones handled in interaction languages (lifelines, operators such as weak sequencing and associated loop operators, etc.). This impairs the traceability of concepts and limits the explainability of subsequent formal techniques which exploit the translated objects.

Contributions about equivalent semantics and traceability of executed actions. In this paper, we extend [30], define three semantics of interactions, and prove their equivalence to one another. These three semantics are, for the first, a denotational semantics in the style of [16], for the second, an operational semantics in the style of [31] and, for the last, an execution semantics formulated in the style of functional programming languages.

This third semantics is easily implementable and separates two concerns found in structural operational semantics [36], which are determining which actions are immediately executable and computing the term resulting from the derivations corresponding to the execution of said actions. This enables us to execute interactions and visualize their executions as in [29]. Moreover, by using unique positions to distinguish distinct instances of the same action, this approach resolves non-determinism. Indeed, for any immediately executable action characterized by its position there is only a single follow-up interaction which characterizes the remainder of behaviors of the initial interaction that start with the execution of that action. Also, via this position in the term, we can unambiguously visualize the position of the executed action in the diagrammatic representation of the interaction.

By contrast, the denotational semantics allows us to reason and prove some properties on interactions (for instance properties related to compositionality). It also enables us, as we do in this paper, to characterize groups of semantically equivalent interactions and, for each such group, to determine a unique representative. As a result, within our framework of three equivalent semantics, we can leverage the advantages of each and use them in combination in the spirit of the Unified Theories of Programming (UTP) [11]. For instance, in [27] we analyze the conformity of partially observed behaviors against interactions (which serve as specifications) using the execution semantics for re-enacting observed parts of the behaviors and a compositionality result proved using the denotational semantics to handle partial observation. In [26], we also combine these two semantics to generate reduced Non-deterministic Finite Automata (NFA) from interactions, where each state of the NFA corresponds to a group of semantically equivalent terms and the transitions correspond to the execution of actions identified using the execution semantics.

Contributions about weak sequencing. Formalizing interactions is not trivial [33] and one of the major difficulties relies on dealing with operators such as *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. Weak sequencing is, therefore, a key operator to specify distributed system behaviors, where by default, events occurring on different subsystems cannot be ordered temporally, as there is no global clock to refer to for that purpose.

The manner in which weak sequencing is handled in most translational approaches is unsatisfactory as it does not allow the unrestricted handling of weak sequential loops (i.e., unbounded repetition via weak sequencing). In [14], UML-SD is translated to CSP but only the strict sequential loop — in which a first instance of the repeated behavior must terminate (i.e., be entirely executed) before another instance can start being executed — is handled (CSP sequencing is used to schedule distinct instances of loops). In [39], NFA are built from UML-SD by composition but the UML-SD loop is matched to the Kleene star operation on NFA which yet again corresponds to a strictly sequential loop. In [1], graphs of basic MSCs (which is a particular kind of interaction language) are translated to NFA. If the edges of the graph are interpreted as weak sequencing then it is impossible to build the NFA (undecidability under the asynchronous interpretation [1]). However, if these edges are interpreted as strict sequencing then it is possible to build the NFA but it is impossible to represent a weak sequential loop. In [17], UML-SD are translated to specific automata in which counters are set to record how many times lifelines in loops are executed. This allows them to translate weak sequential loops but in a restricted manner because these loops can only contain a basic interaction (without high-level operators). In [4], UML-SD are translated into sets of communicating automata. To handle non-determinism (caused by alternatives, loops, etc.), each automaton keeps track of the region (UML-SD combined fragments) of the diagram in which it currently is. However, with the weak sequential loop, one such automaton may need to be in several regions simultaneously, which is not handled here.

As a result, our framework provides a unique take on weak sequencing, which allows handling weak sequential loops as any other operator (it can be nested and put anywhere above or under any other operator). Our interaction language contains four kinds of loop. Three of these are defined using the Kleene closure of distinct scheduling operators (strict sequencing, weak sequencing and interleaving). The different iterations of the repeated behavior are joined together using the corresponding scheduling operator. A peculiarity of weak sequencing allows for defining an additional loop, which uses a different kind of closure.

Outline. This paper is organized as follows. After some preliminaries in Section 2, we introduce scheduling and repetition operators on traces in Section 3. Section 4 introduces our interaction language, its associated trace semantics in denotational style and a rewrite system to compute "canonical forms of interactions" defined as unique representatives of groups of semantically equivalent interactions. In Section 5, a structural operational semantics is defined in the style of process calculi and is proven equivalent to the denotational semantics.

Section 6 introduces the execution semantics with the addition of optimizations brought by the use of semantically sound transformations defined in Section 4 and proves that it is equivalent to the operational one. Finally, in Sections 7 and 8 we discuss some related works and we conclude.

2. Preliminaries

In the following we present some preliminary notions and definitions that are used in the remainder of the paper. Notions of set theory are used throughout the paper. Behaviors specified by interaction diagrams are formalized as sequences of atomic actions. Hence, we introduce prerequisite notions of formal languages and words. Terms and term algebras are required for the definition of the syntax of our interaction language and its denotational semantics. Binary relations and notions of equational logic are required to be able to relate semantically equivalent interactions. By introducing term rewriting, we will be able to characterize unique representative of groups of semantically equivalent interactions. Class rewriting extends term rewriting in cases where some rewrite rules need to be used in both directions.

Sets and words. Given a set E, $\mathcal{P}(E)$ is the set of all subsets of E. A word of alphabet E is a finite sequence of elements of E. For example, $w = e_1 e_2 \cdots e_k$ is a word of length $k > 2$ with $\forall\ i \in [1,k]$, $e_i \in E$. We denote by E^* the set of words of alphabet E and by $|w|$ the length of a word $w \in E^*$. We denote by $\varepsilon \in E^*$ the empty word of length 0 and, each letter $e \in E$ is considered to be a word of length 1 i.e., $E \subset E^*$. The concatenation operator "." is such that for any two words $w \in E^*$ and $w' \in E^*$, the word denoted by $w.w'$ is such that:

- if $w' = \varepsilon$ then $w.\varepsilon = w$
- and if $w' = ew''$ with $e \in E$ and $w'' \in E^*$ then $w.ew'' = we.w''$.

Algebras. An algebra signature is a finite set of operation symbols $\mathcal{F} = \bigcup_{j \geq 0} \mathcal{F}_j$ such that for any integer $j \geq 0$, the set $\mathcal{F}_j$ is that of symbols of arity j. Symbols of arity 0 are constants. A $\mathcal{F}$-algebra $\mathcal{A} = (A, \{f^{\mathcal{A}} \mid f \in \mathcal{F}\})$ is defined by a set A of values and a family of applications $f^{\mathcal{A}} : A^j \to A$ for each f in $\mathcal{F}_j$.

Considering a set $\mathcal{X}$, the term algebra $\mathcal{T}_{\mathcal{F}}(\mathcal{X})$ with variables in $\mathcal{X}$ has for set of values (denoted simply as $\mathcal{T}_{\mathcal{F}}(\mathcal{X})$) the smallest set such that:

- $\mathcal{X} \cup \mathcal{F}_0 \subset \mathcal{T}_{\mathcal{F}}(\mathcal{X})$
- and for any symbol $f \in \mathcal{F}$ of arity $j > 0$ and terms $t_1, \cdots, t_j \in \mathcal{T}_{\mathcal{F}}(\mathcal{X})^j$, $f(t_1, \cdots, t_j) \in \mathcal{T}_{\mathcal{F}}(\mathcal{X})$.

The free term algebra $\mathcal{T}_{\mathcal{F}} = \mathcal{T}_{\mathcal{F}}(\emptyset)$ verifies that for any $\mathcal{F}$-algebra $\mathcal{A}$, there exists a unique homomorphism $\mathcal{T}_{\mathcal{F}} \to \mathcal{A}$ interpreting each ground term in $\mathcal{T}_{\mathcal{F}}$ according to $\mathcal{A}$.

For any term $t \in \mathcal{T}_{\mathcal{F}}(\mathcal{X})$, we denote by $var(t) \in \mathcal{P}(\mathcal{X})$ its set of variables such that:

- $\forall\ t \in \mathcal{X}$, $var(t) = \{t\}$,
- $\forall\ t \in \mathcal{F}_0$, $var(t) = \emptyset$
- and for all $t \in \mathcal{F}(\mathcal{X})$ of the form $f(t_1, \cdots, t_j)$ with $j \in \mathbb{N}^+$, $var(f(t_1, \cdots, t_j)) = \bigcup_{k\in[1,j]} var(t_k)$.

For any $t \in \mathcal{T}_{\mathcal{F}}(\mathcal{X})$, we denote by $pos(t) \in \mathcal{P}((\mathbb{N}^+)^*)$ its set of positions accordingly to the Dewey Decimal Notation [7]. Positions within terms are defined as words on integers as follows:

- $\forall\ t \in \mathcal{X} \cup \mathcal{F}^0$, $pos(t) = \{\varepsilon\}$,
- and for all t in $\mathcal{F}(\mathcal{X})$ of the form $f(t_1, \cdots, t_j)$ with $j \in \mathbb{N}^+$, $pos(f(t_1, \cdots, t_j)) = \{\varepsilon\} \cup \bigcup_{k\in[1,j]} \{k.p \mid p \in pos(t_k)\}$.

For any term t, and any position $p \in pos(t)$, we denote resp. by $t(p)$ and $t_{|p}$ the operation symbol and sub-term at position p within t, and, for any term s, $t[s]_p$ denotes the term obtained by substituting $t_{|p}$ with s within t.

A substitution $\phi : \mathcal{T}_{\mathcal{F}}(\mathcal{X}) \rightarrow \mathcal{T}_{\mathcal{F}}(\mathcal{X})$ extends a mapping $\phi^\dagger : \mathcal{X} \rightarrow \mathcal{T}_{\mathcal{F}}(\mathcal{X})$ s.t.:

- $\forall\ t \in \mathcal{X}$, $\phi(t) = \phi^\dagger(t)$,
- $\forall\ t \in \mathcal{F}_0$, $\phi(t) = t$
- and for all t in $\mathcal{F}(\mathcal{X})$ of the form $f(t_1, \cdots, t_j)$ with $j \in \mathbb{N}^+$, $\phi(f(t_1, \cdots, t_j)) = f(\phi(t_1), \cdots, \phi(t_n))$.

We denote by $\mathrm{Sub}(\mathcal{T}_{\mathcal{F}}(\mathcal{X}))$ the set of all such substitutions.

Binary relations. A binary relation $\rightsquigarrow$ on a set A is a subset of $A \times A$, commonly used with an infix notation. For any two relations $\rightsquigarrow$ and $\leadsto$:

- composition is s.t. $\rightsquigarrow \circ \leadsto = \{(x, z) \in A^2 \mid \exists\ y \in A, (x \rightsquigarrow y) \text{ and } (y \leadsto z)\}$,
- $\leftsquigarrow$ denotes the inverse relation $\{(y, x) | x \rightsquigarrow y\}$,
- $\leftrightsquigarrow$ denotes the symmetric closure $\rightsquigarrow \cup \leftsquigarrow$,
- $\overset{0}{\rightsquigarrow}$ denotes the identity relation $\{(x, x) \mid x \in A\}$,
- $\overset{1}{\rightsquigarrow}$ denotes the relation $\rightsquigarrow$ itself,
- for any $j > 1$, $\overset{j}{\rightsquigarrow}$ is the relation $\overset{j-1}{\rightsquigarrow} \circ \rightsquigarrow$
- and $\overset{*}{\rightsquigarrow}$ denotes the reflexive and transitive closure $\bigcup_{j=0}^{\infty} \overset{j}{\rightsquigarrow}$ of $\rightsquigarrow$.
- $\overset{*}{\leftrightsquigarrow}$ denotes its symmetric, reflexive and transitive closure (which is the reflexive and transitive closure of its symmetric closure).

Given signature $\mathcal{F}$ and variables $\mathcal{X}$, a binary relation $\rightsquigarrow$ on terms of $\mathcal{T}_{\mathcal{F}}(\mathcal{X})$ is:

- stable by substitution if, $\forall\ (x, y) \in \mathcal{T}_\mathcal{F}(\mathcal{X})^2$, $\forall\ \phi \in \text{Sub}(\mathcal{T}_\mathcal{F}(\mathcal{X}))$, we have $\phi(x) \rightsquigarrow \phi(y)$ whenever $x \rightsquigarrow y$
- $\mathcal{F}$-compatible if $\forall\ f \in \mathcal{F}_j$ with $j \in \mathbb{N}$, for all $(x_k, y_k)_{k\in[1,j]} \in \mathcal{T}_\mathcal{F}(\mathcal{X})^{2j}$, $f(x_1, \cdots, x_j) \rightsquigarrow f(y_1, \cdots, y_j)$ whenever $x_1 \rightsquigarrow y_1, \ldots$ and $x_j \rightsquigarrow y_j$ hold.

A relation on terms that is reflexive, symmetric, transitive, stable by substitution and $\mathcal{F}$-compatible is a *congruence relation*. Such a congruence relation $\approx$ partitions the set of terms into *congruence classes* $[\ ]_\approx$.

Equational logic and term rewriting. Given a signature $\mathcal{F}$ and a set of variables $\mathcal{X}$, an equation is a pair $(l, r) \in \mathcal{T}_\mathcal{F}(\mathcal{X})^2$ which we may denote by $l \sim r$. An axiom system E is a set of equations. The congruence relation $\approx_E$ is the reflexive, symmetric, transitive, stable by substitution and $\mathcal{F}$-compatible closure of E.

A *rewrite rule* is a pair $(l, r) \in \mathcal{T}_\mathcal{F}(\mathcal{X})^2$ s.t. $l \notin \mathcal{X}$, $var(r) \subseteq var(l)$ and which we may denote by $l \rightsquigarrow r$. A Term Rewriting System (TRS) is a set R of rewrite rules. The rewrite relation $\rightarrow_R$ is such that $x \rightarrow_R y$ holds iff there exists **(1)** a rule $l \rightsquigarrow r \in R$, **(2)** a position $p \in pos(x)$, and **(3)** a substitution $\phi \in \text{Sub}(\mathcal{T}_\mathcal{F}(\mathcal{X}))$ such that $x_{|p} = \phi(l)$ and $y = x[\phi(r)]_p$. A term t is said to be *irreducible* iff there are no t' s.t. $t \rightarrow_R t'$. We may write $t \rightarrow^!_R t'$ if $t \xrightarrow{*}_R t'$ and t' is irreducible. This defines a *normalizability* relation $\rightarrow^!_R$. A TRS R is *terminating* if there is no infinite series $(t_j)_{j\geq 0}$ of terms such that for any $j \in \mathbb{N}$, $t_j \rightarrow_R t_{j+1}$. A TRS R is *confluent* if $\xleftarrow{*}_R \circ \xrightarrow{*}_R \subseteq \xrightarrow{*}_R \circ \xleftarrow{*}_R$. A TRS R that is both terminating and confluent is said to be *convergent*. For such a rewrite system, any term t admits exactly one normal form, which we may denote by $R(t)$ following the notation from [7]. $R(t)$ is obtained using $t \rightarrow^!_R R(t)$. In the remainder of this paper, we may call this normal form the *canonical form* of t.

Class rewriting systems. A Class Rewriting Systems (CRS) is a pair (R, T) denoted as R/T where R is a TRS and T is an axiom system (often called a *theory* in this context[1]). A CRS R/T specifies a one-step rewrite relation $\rightarrow_{R/T} \in \mathcal{T}_\mathcal{F}(\mathcal{X})^2$ s.t. $\forall\ (x, z) \in \mathcal{T}_\mathcal{F}(\mathcal{X})^2$:

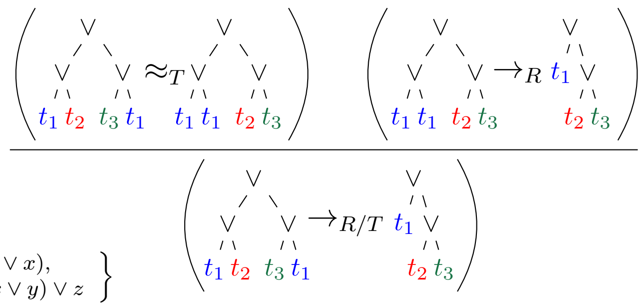
$$(x \rightarrow_{R/T} z) \Leftrightarrow \left(\begin{array}{l|l} \exists\ (l \rightsquigarrow r) \in R,\ \exists\ \phi \in \text{Sub}(\mathcal{T}_\mathcal{F}(\mathcal{X})), & (x \approx_T y[\phi(l)]_p) \\ \exists\ y \in \mathcal{T}_\mathcal{F}(\mathcal{X}),\ \exists\ p \in pos(y) & \wedge\ (y[\phi(r)]_p \approx_T z) \end{array} \right)$$

The example on Figure 1 illustrates the use of a CRS to simplify a Boolean expression involving the operation symbol $\vee$ of arity 2 used with an infix notation. This CRS consists of a theory T and a TRS R given on Figure 1a. An application of this CRS via a one-step rewriting operation is illustrated on Figure 1b. Let us denote by $x = (t_1 \vee t_2) \vee (t_3 \vee t_1)$, $y = (t_1 \vee t_1) \vee (t_2 \vee t_3)$ and $z = t_1 \vee ((t_2 \vee t_3)$ s.t. we have the pattern $x \approx_T y$, $y \rightarrow_R z$ and $x \rightarrow_{R/T} z$ from the previous definition. Here, we apply the rule $l \rightsquigarrow r = (t \vee t) \rightsquigarrow t$ with the substitution $\phi \in Sub(\mathcal{T}_\mathcal{F}(\mathcal{X}))$ s.t. $\phi(t) = t_1$ at position $p = 1$ in y.

[1]The term "rewriting modulo theory" is also employed.

$$R = (x \vee x) \rightsquigarrow x$$
$$T = \left\{ \begin{array}{c} (x \vee y) \sim (y \vee x), \\ x \vee (y \vee z) \sim (x \vee y) \vee z \end{array} \right\}$$

(a) Example CRS



(b) Example application

Figure 1: Example application of a CRS

In the particular case when the theory T only contains rules relative to the associativity or commutativity of operators (as in our example), such rewriting is called Associative-Commutative Rewriting (AC-R). Formulating a rewrite system as an AC-RS is advantageous as existing tools allow automated proofs of its termination [18] and confluence [41].

3. Scheduling and repetition operators for trace handling

Interactions describe the behavior of distributed and concurrent systems based on their internal and external communications. They are defined for a set L of lifelines and a set M of messages, representing respectively the sending and receiving locations and the messages exchanged between them. In the following, we assume that a signature $\Omega = (L, M)$ is given.

The behaviors of distributed 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:

$$\mathbb{A}_\Omega = \{l!m \mid m \in M, l \in L\} \cup \{l?m \mid m \in M, l \in L\}$$

For any action $a \in \mathbb{A}_\Omega$, we denote by $\theta(a)$ the lifeline on which a occurs (i.e., $\theta(a) = l$ if a is of the form $l!m$ or $l?m$).

Sequences of actions, called traces, are words in $\mathbb{A}_\Omega^*$. In the following, we denote by $\mathbb{T}_\Omega = \mathbb{A}_\Omega^*$ the set of traces over the signature Ω.

3.1. Binary composition operators

We introduce operators to compose traces, modeling different notions of scheduling:

- the strict sequencing (;),

- the interleaving ($||$),
- and the weak sequencing ($;_{\divideontimes}$).

These "scheduling" operators $\diamond \in \{;\, , ;_{\divideontimes}\, , \ ||\}$ are defined with the same profile: $\diamond : \mathbb{T}_\Omega \times \mathbb{T}_\Omega \to \mathcal{P}(\mathbb{T}_\Omega)$. They are canonically extended to sets of traces, i.e., to $\diamond : \mathcal{P}(\mathbb{T}_\Omega) \times \mathcal{P}(\mathbb{T}_\Omega) \to \mathcal{P}(\mathbb{T}_\Omega)$, as follows: for any two sets T_1 and T_2 of traces, $T_1 \diamond T_2$ 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 strict sequencing notation ($;$) extends the concatenation "." operator. The set $t_1;t_2$ of strict sequencing of traces t_1 and t_2 is defined as $\{t_1.t_2\}$. We choose this notation in line with [16]. The following example illustrates its use with elements $(a_k)_{k\in[1,8]}$ being atomic actions of $\mathbb{A}_\Omega$:

$$\left\{ \begin{array}{l} a_1.a_2, \\ a_3.a_4 \end{array} \right\} ; \left\{ \begin{array}{l} a_5.a_6, \\ a_7.a_8 \end{array} \right\} = \left\{ \begin{array}{l} a_1.a_2.a_5.a_6, \\ a_1.a_2.a_7.a_8, \\ a_3.a_4.a_5.a_6, \\ a_3.a_4.a_7.a_8 \end{array} \right\}$$

The colors are used to highlight the definition of the operator ";" together with sets. Strict sequencing is associative and admits $\{\varepsilon\}$ as a neutral element i.e., for any sets of traces T_1, T_2 and T_3, we have $T_1;(T_2;T_3) = (T_1;T_2);T_3$ and $T_1;\{\varepsilon\} = T_1 = \{\varepsilon\};T_1$.

Interleaving ($||$) allows elements of distinct traces to be reordered with regard to 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:

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

The following two examples illustrate the use of the interleaving operator:

$$\left\{ \begin{array}{l} a_1.a_2, \\ a_3 \end{array} \right\} ||\{a_4\} = \left\{ \begin{array}{l} a_1.a_2.a_4, \\ a_1.a_4.a_2, \\ a_4.a_1.a_2, \\ a_3.a_4, \\ a_4.a_3, \end{array} \right\} \qquad \{a_1.a_2\}||\{a_3.a_4\} = \left\{ \begin{array}{l} a_1.a_2.a_3.a_4, \\ a_1.a_3.a_2.a_4, \\ a_1.a_3.a_4.a_2, \\ a_3.a_1.a_2.a_4, \\ a_3.a_1.a_4.a_2, \\ a_3.a_4.a_1.a_2 \end{array} \right\}$$

Like strict sequencing, interleaving is associative and admits $\{\varepsilon\}$ as a neutral element. Moreover, it is also commutative i.e., for any sets of traces T_1 and T_2, we have $T_1||T_2 = T_2||T_1$.

Weak sequencing is a hybrid operator between strict sequencing and interleaving: it takes into account the location (lifeline) on which the actions it schedules take place. By contrast to the interleaving operator, weak sequencing only allows permutations of the order of actions when said actions do not occur on the same lifeline. In a similar fashion as [16], we define a "conflict" predicate $t \divideontimes l$ meaning that the trace t contains an action on the lifeline l:

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

If $t \divideontimes l = \top$, we say that the trace t has conflicts with the lifeline l. This enables us to define the set $t_1 ;_{\divideontimes} t_2$ of weak sequencing of t_1 and t_2 as follows:

$$\begin{array}{rcll} \varepsilon ;_{\divideontimes} t_2 & = & \{t_2\} & \\ t_1 ;_{\divideontimes} \varepsilon & = & \{t_1\} & \\ (a_1.t_1) ;_{\divideontimes} (a_2.t_2) & = & & \{a_1.t \mid t \in t_1 ;_{\divideontimes} (a_2.t_2)\} \\ & & \cup & \{a_2.t \mid t \in (a_1.t_1) ;_{\divideontimes} t_2,\ \neg(a_1.t_1 \divideontimes \theta(a_2))\} \end{array}$$

When defining $t'_1 ;_{\divideontimes} t'_2$, 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 $(a_1.t_1) ;_{\divideontimes} (a_2.t_2)$: the first one contains all traces whose first action is a_1 and tail belongs to $t_1 ;_{\divideontimes} (a_2.t_2)$ and the second one is empty if lifeline of a_2 occurs in $a_1.t_1$ (i.e. $a_1.t_1 \divideontimes \theta(a_2)$), and contains all traces whose first action is a_2 and tail belongs to $(a_1.t_1) ;_{\divideontimes} t_2$ otherwise.

In the following example, in order to illustrate the mechanisms of conflict with lifelines, the contents of actions are detailed: lifelines are chosen among l_1 and l_2 and messages among m_1, m_2 and m_3. Action $l_1!m_3$ can be scheduled before $l_2!m_2$ because they occur on different lifelines but $l_1!m_3$ cannot precede $l_1!m_1$ because they both occur on lifeline l_1.

$$\{l_1!m_1.l_2!m_2\} ;_{\divideontimes} \{l_1!m_3\} = \left\{ \begin{array}{l} l_1!m_1.l_2!m_2.l_1!m_3, \\ l_1!m_1.l_1!m_3.l_2!m_2 \end{array} \right\}$$

By contrast to interleaving, weak sequencing is not commutative. However it still is associative and admits $\{\varepsilon\}$ as a neutral element.

A trace, being a sequence of actions, represents a behavior expressed by a system. Hence a set of traces can be understood as a set of possible behaviors and the union operation ($\cup$) on sets can be interpreted as a choice between behaviors. This fourth binary operator is known to be associative and commutative but $\{\varnothing\}$ is not its neutral element.

3.2. Repetition operators

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 $\{\varepsilon\}$ as a neutral element. This allows us to define (Kleene) closures of those operators to specify repetitions.

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

The three Kleene closures $^{;*}$, $^{;_{\divideontimes}*}$ and $^{||*}$ are respectively called, strict, weak and interleaving Kleene closures (also called K-closure for short). 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. Let us remark the following properties:

- $\{\varepsilon\}^{\diamond *} = \{\varepsilon\}$ (i.e., $\{\varepsilon\}$ is a fixed point for $^{\diamond *}$),

- $\{\varepsilon\} \cup T^{\diamond *} = T^{\diamond *}$ because $\varepsilon \in T^{\diamond *}$
- $(T^{\diamond *})^{\diamond *} = T^{\diamond *}$ (idempotence)

This notion of repetition thus coincides with the notion of unbounded loop found in various languages. Repetitions with strict sequencing corresponds to the classical Kleene star (strictly sequential loop), while the one based on weak sequencing can be used to formalize a certain understanding of the UML-SD [33] loop (weakly sequential loop). By contrast, the interleaving Kleene closure (parallel loop) is more akin to the replication $!P$ (i.e., $!P = P|!P$) from process calculus (see [35]), expressing an unbounded number of copies of P along the parallel composition "$|$".

$$T = \left\{ \begin{array}{l} l_1!m.l_2!m_1.l_2!m_2, \\ l_2!m \end{array} \right\}$$

$$T^{;_{\ast}0} = \{\varepsilon\}$$

$$T^{;_{\ast}1} = T$$

$$T^{;_{\ast}2} = \left\{ \begin{array}{l} l_1!m.l_2!m_1.l_2!m_2.l_1!m.l_2!m_1.l_2!m_2, \\ l_1!m.l_2!m_1.l_1!m.l_2!m_2.l_2!m_1.l_2!m_2, \\ l_1!m.l_1!m.l_2!m_1.l_2!m_2.l_2!m_1.l_2!m_2, \\ l_1!m.l_2!m_1.l_2!m_2.l_2!m, \\ l_2!m.l_1!m.l_2!m_1.l_2!m_2, \\ l_1!m.l_2!m.l_2!m_1.l_2!m_2, \\ l_2!m.l_2!m \end{array} \right\}$$

Figure 2: Example illustrating the weak Kleene closure

Figure 2 illustrates an application of the weak Kleene closure. We consider a set T containing two traces $l_1!m.l_2!m_1.l_2!m_2$ and $l_2!m$. The first 3 powersets of T (i.e., $T^{;_{\ast}0} \cup T^{;_{\ast}1} \cup T^{;_{\ast}2}$) are displayed. Let us remark that the use of weak sequencing allows events occurring on l_2 to be reordered w.r.t. those occurring on l_1 but not w.r.t. other events occurring on l_2. As such weakly sequential repetition is distinct from both strictly sequential repetition (as demonstrated by $l_1!m.l_2!m.l_2!m_1.l_2!m_2 \in T^{;_{\ast}2} \setminus T^{;2}$ because $l_2!m$ cannot be reordered in this manner using strict sequencing) and parallel repetition (by $l_1!m.l_2!m_1.l_2!m.l_2!m_2 \in T^{||2} \setminus T^{;_{\ast}2}$ because with interleaving, nothing prevents $l_2!m$ to be reordered between $l_2!m_1$ and $.l_2!m_2$).

For $\diamond \in \{;, ;_{\ast}, ||\}$ 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), the 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, in our example case, that action a must be taken from T_1 and not from T_2.

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

$$T_1 \diamond^{\Lsh} T_2 = \left\{ t \in T_1 \diamond T_2 \;\middle|\; \begin{array}{c} \textbf{\textit{if}}\ \exists\, a \in \mathbb{A}_\Omega,\ t' \in \mathbb{T}_\Omega\ s.t.\ t = a.t' \\ \textbf{\textit{then}}\ \exists\, t_1 \in \mathbb{T}_\Omega,\ (a.t_1 \in T_1)\ \wedge\ (t' \in \{t_1\} \diamond T_2) \end{array} \right\}$$

In Definition 2, if the left-hand-side ($t = a.t'$) is true then the condition $\exists\, t_1 \in \mathbb{T}_\Omega,\ (a.t_1 \in T_1) \wedge (t' \in \{t_1\} \diamond T_2)$ applies as intended and the first action

a must be taken from a trace of T_1 and not from T_2. If the left-hand-side is false then the only form t can take is $t = \varepsilon$ and indeed the predicate becomes true and we have $\varepsilon \in T_1 \diamond^{\Lsh} T_2$ which works as intended because we also have $\varepsilon \in T_1 \diamond T_2$.

We use the restricted versions of the scheduling operators to define their Head-First closures (abbr. HF-closure) in Def.3.

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

In the following we show that HF-closure and K-closure are equivalent for the two operators $;$ and $||$ but that this is not the case for the weak sequencing operator $;_{\times}$. Intuitively, given $\diamond \in \{;\,, ||, ;_{\times}\}$ and a set of traces $T \subseteq \mathbb{T}$, any non empty trace $t \in T^{\diamond *}$ is such that there exists non-empty traces $t_1, \cdots, t_n \in T$ s.t. $t \in \{t_1\} \diamond \cdots \diamond \{t_n\}$. If the HF closure $\diamond^{\Lsh *}$ is equivalent to the Kleene closure $\diamond *$ then we must also have $t \in T^{\diamond^{\Lsh} *}$.

- For the strict sequencing we have $\{t_1\}; \cdots ; \{t_n\} = \{t_1\};^{\Lsh} \cdots ;^{\Lsh} \{t_n\}$ because in any case, an action from a trace t_i ($i \in [2, n]$) can only occur if all the actions from all the traces t_j with $j < i$ have already occurred.

- For the interleaving there exists a permutation $\eta : [1, n] \to [1, n]$ s.t. $\{t_1\}|| \cdots ||\{t_n\} = \{t_{\eta(1)}\}||^{\Lsh} \cdots ||^{\Lsh}\{t_{\eta(n)}\}$. Indeed, it suffices to reorder the traces t_i ($i \in [1, n]$) depending on the position in t where the first action a_i from t_i occurs. Then, because $\{a_i.t_i'\}||^{\Lsh}\{t_{i+1}\} = \{a_i\}; (\{t_i'\}||\{t_{i+1}\})$ and because $||$ allows any possible interleaving, the position of the t_i does not matter for the second and all the following actions. We can then conclude because by Definition 1, we have $\{t_{\eta(1)}\}||^{\Lsh} \cdots ||^{\Lsh}\{t_{\eta(n)}\} \subseteq T^{||^{\Lsh} *}$.

Reflecting the Coq proof [23], a first Lemma 1 is required to prove the equivalence for $;$ and $||$ in Lemma 2. Then a counter-example is produced to handle $;_{\times}$.

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:*

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

Proof. By definition of the K-closure, $a.t \in T^{\diamond *}$ implies the existence of $j \geq 0$ such that $a.t \in T^{\diamond j}$. We can then reason by induction on j. $j = 0$ is not possible because $T^{\diamond 0} = \{\varepsilon\}$. For $j = 1$, $a.t \in T^{\diamond 1} = T$ and $t \in \{t\} \diamond \{\varepsilon\} \subset \{t\} \diamond T^{\diamond *}$. For $j > 1$ the fact that $a.t \in T^{\diamond j} = T \diamond T^{\diamond (j-1)}$ implies the existence of $t'' \in T$ s.t. $a.t \in \{t''\} \diamond T^{\diamond (j-1)}$ then:

- if $\diamond =;$ this implies that either t'' is of the form $a.t'$ and $t \in \{t'\}; T^{;(j-1)}$ and therefore $t \in \{t'\}; T^{;*}$ or $t'' = \varepsilon$ and $a.t \in T^{;(j-1)}$ and we can use the induction hypothesis to conclude;

- if $\diamond = ||$ this implies that either t'' is of the form $a.t'$ and $t \in \{t'\}||T^{||(j-1)}$ and therefore $t \in \{t'\}||T^{||*}$ or there exists a certain t''' s.t. we have

$a.t''' \in T^{||(j-1)}$ and $t \in \{t''\}||\{t'''\}$. As $a.t''' \in T^{||(j-1)}$ we can apply the induction hypothesis to reveal t' such that $a.t' \in T$ and $t''' \in \{t'\}||T^{||*}$. We can conclude as follows:

$$\begin{array}{rll}
t \in \{t''\}||(\{t'\}||T^{||*}) & \Rightarrow \; t \in \{t''\}||(T^{||*}||\{t'\}) & \text{commutativity} \\
& \Rightarrow \; t \in (\{t''\}||T^{||*})||\{t'\} & \text{associativity} \\
& \Rightarrow \; t \in T^{||*}||\{t'\} & \text{property of Kleene closure} \\
& \Rightarrow \; t \in \{t'\}||T^{||*} & \text{commutativity}
\end{array}$$

□

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

Proof. Let us consider $\diamond \in \{;, \; ||\}$. By definition, we already have $T^{\diamond^{\Lsh}*} \subseteq T^{\diamond *}$. To prove $T^{\diamond *} \subseteq T^{\diamond^{\Lsh}*}$, let us reason by induction on the length of a member trace. For length 0, by definition $\varepsilon \in T^{\diamond^{\Lsh}*}$ and $\varepsilon \in T^{\diamond *}$. For a trace of the form $a.t$, if $a.t \in T^{\diamond *}$, then, as per Lemma 1 there exists a trace t' such that $a.t' \in T$ and $t \in \{t'\} \diamond T^{\diamond *}$. This in turn implies that given that action a is taken from $a.t'$, we have $a.t \in \{a.t'\} \diamond^{\Lsh} T^{\diamond *}$ and there exists $t'' \in T^{\diamond *}$ such that $t \in \{t'\} \diamond \{t''\}$. Given that $|t''| \leq |t| < |a.t|$, we have $t'' \in T^{\diamond^{\Lsh}*}$ by the induction hypothesis. Therefore we have that $a.t \in \{a.t'\} \diamond^{\Lsh} T^{\diamond^{\Lsh}*}$, and hence $a.t \in T^{\diamond^{\Lsh}*}$. □

$$T_1 = \{l_1!m_1.l_2?m_1\} \qquad T_2 ;_{\times} T_1 = \left\{ \begin{array}{l} l_2!m_2.l_1!m_1.l_2?m_1, \\ l_1!m_1.l_2!m_2.l_2?m_1 \end{array} \right\}$$

$$T_2 = \{l_2!m_2\} \qquad T_2 ;_{\times}^{\Lsh} T_1 = \{ \; l_2!m_2.l_1!m_1.l_2?m_1 \; \}$$

$$(T_1 \cup T_2)^{;_{\times}^{\Lsh} 2} = \left\{ \begin{array}{l} l_1!m_1.l_2?m_1.l_1!m_1.l_2?m_1, \\ l_1!m_1.l_1!m_1.l_2?m_1.l_2?m_1, \\ l_2!m_2.l_1!m_1.l_2?m_1, \\ l_1!m_1.l_2?m_1.l_2!m_2, \\ l_2!m_2.l_2!m_2 \end{array} \right\}$$

$$(T_1 \cup T_2)^{;_{\times} 2} = (T_1 \cup T_2)^{;_{\times}^{\Lsh} 2} \cup \{l_1!m_1.l_2!m_2.l_2?m_1\}$$

Figure 3: Example illustrating the weak Head-First closure

Figure 3 illustrates the use of the restricted weak sequencing operator $;_{\times}^{\Lsh}$ and its associated closure $^{;_{\times}^{\Lsh} 2*}$. The trace $t = l_1!m_1.l_2!m_2.l_2?m_1$ which is included in $T_2 ;_{\times} T_1$ is not in $T_2 ;_{\times}^{\Lsh} T_1$ because the first action must be taken from T_2. In the weak Head-First closure, we therefore have $t \notin (T_1 \cup T_2)^{;_{\times}^{\Lsh} 2}$ because it is neither in $T_2 ;_{\times}^{\Lsh} T_1$ nor in $T_1 ;_{\times} T_2$. This example demonstrates that the weak Head-First closure $^{;_{\times}{}^{\Lsh}*}$ is distinct from both the weak K-closure $^{;_{\times}*}$ and (more trivially) the strict K-closure $^{;*}$ (for instance consider $l_1!m_1.l_1!m_1.l_2?m_1.l_2?m_1$). It is therefore also a counter-example showing that Lemma 2 cannot be extended to include weak sequencing. This example will be further illustrated in Section 5.3 with the help of drawn interactions and their operational semantics.

Let us compare the scheduling operators $;$, $;^{\Lsh}_{\divideontimes}$, $;_{\divideontimes}$ and $||$ according to their strictness and permissiveness. This is formalized as follows: for any two sets T_1 and T_2 of traces,

$$T_1;T_2 \;\subseteq\; T_1;^{\Lsh}_{\divideontimes}T_2 \;\subseteq\; T_1;_{\divideontimes}T_2 \;\subseteq\; T_1||T_2$$

Let us then consider a total order $<$ on $\{; \;,\; ;^{\Lsh}_{\divideontimes} \;,\; ;_{\divideontimes} \;,\; ||\}$ such that

$$|| < ;_{\divideontimes} < ;^{\Lsh}_{\divideontimes} < ;$$

and let us denote by $min(\diamond_1, \diamond_2)$ the minimal (i.e., most permissive) operator between $\diamond_1$ and $\diamond_2$.

It then comes that, for any $\diamond_1 < \diamond_2$, and any set T of traces, we have $T^{\diamond_2 *} \subseteq T^{\diamond_1 *}$. This then implies the following property: for any two $\diamond_1$ and $\diamond_2$ and any set T of traces: $(T^{\diamond_1 *})^{\diamond_2 *} = (T^{\diamond_2 *})^{\diamond_1 *} = T^{min(\diamond_1,\diamond_2)*}$.

3.3. Summary of algebraic properties

property	**operators**	**equation**
associativity	$\diamond \in \{\cup, ;, ;_{\divideontimes}, \|\|\}$	$x \diamond (y \diamond z) = (x \diamond y) \diamond z$
commutativity	$\diamond \in \{\cup, \|\|\}$	$x \diamond y = y \diamond x$
$\{\varepsilon\}$-neutrality	$\diamond \in \{;, ;_{\divideontimes}, \|\|\}$	$\{\varepsilon\} \diamond x = x = x \diamond \{\varepsilon\}$
binary idempotence	$\cup$	$x \cup x = x$
$\{\varepsilon\}$-fixpoint	$\diamond \in \{;, ;^{\Lsh}_{\divideontimes}, ;_{\divideontimes}, \|\|\}$	$\{\varepsilon\}^{\diamond *} = \{\varepsilon\}$
$\{\varepsilon\}$-inclusion	$\diamond \in \{;, ;^{\Lsh}_{\divideontimes}, ;_{\divideontimes}, \|\|\}$	$\{\varepsilon\} \cup x^{\diamond *} = x^{\diamond *}$
repetition nesting	$\diamond_1, \diamond_2 \in \{;, ;^{\Lsh}_{\divideontimes}, ;_{\divideontimes}, \|\|\}$	$(x^{\diamond_1 *})^{\diamond_2 *} = x^{min(\diamond_1,\diamond_2)*}$

Figure 4: Algebraic properties of operators on sets of traces

In Section 3.1 and 3.2, we have defined a set of binary and unary operators over sets of traces. Those operators may then be used to define the semantics of an interaction diagram i as a set of *accepted* traces by mapping them to the corresponding syntactic elements i.

We have also characterized these operators by identifying some of their algebraic properties (which are summarized on Figure 4). This will then allow us to identify groups of semantically equivalent interactions and to compute normal forms of interactions (i.e., unique representatives of these groups).

4. Interaction language

In this section, we formalize our interaction language as a term algebra and associate it with a denotational-style trace semantics based on the previously defined composition operators on traces. We then characterize groups of semantically equivalent interactions via a set of equations and describe a process to compute canonical forms of interactions which are defined as unique representatives of classes of semantically equivalent interactions.

4.1. Syntax and denotational semantics

As sketched out so far, interactions will be defined as terms using composition and loop operators over traces. 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 behaviors can then be defined inductively using the binary constructors *strict*, *seq*, *par* and *alt* and the unary constructors $loop_S$ (strict loop), $loop_H$ (head loop up to $;_{\divideontimes}$), $loop_W$ (weak loop) and $loop_P$ (parallel loop).

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

- *symbols of 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 as a set of terms $\mathbb{I}_\Omega$ inductively defined from the set of symbols $\mathcal{F}$ with arity in $\mathbb{N}$. In the following we will exclusively use the notation $\mathcal{F}$ to designate the specific set of operation symbols from Definition 4.

The set $\mathcal{P}(\mathbb{T}_\Omega)$ of sets of traces admits the structure of a $\mathcal{F}$-algebra using composition and loop operators previously introduced. The denotational semantics of interactions is then given in Definition 5 using the initial homomorphism associated with 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 $\mathcal{F}$:*

$$
\begin{array}{lcl}
\varnothing^\mathcal{A} & = & \{\varepsilon\} \\
a^\mathcal{A} & = & \{a\}
\end{array}
\qquad
\begin{array}{lcl}
strict^\mathcal{A} & = & ; \\
seq^\mathcal{A} & = & ;_{\divideontimes} \\
par^\mathcal{A} & = & || \\
alt^\mathcal{A} & = & \cup
\end{array}
\qquad
\begin{array}{lcl}
loop_S^\mathcal{A} & = & ;^{*} \\
loop_H^\mathcal{A} & = & \overset{\Lsh}{;_{\divideontimes}}{}^{*} \\
loop_W^\mathcal{A} & = & ;_{\divideontimes}{}^{*} \\
loop_P^\mathcal{A} & = & ||^{*}
\end{array}
$$

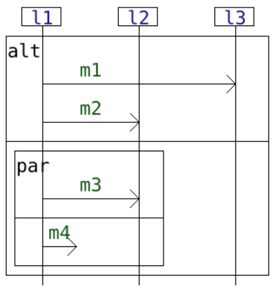
The denotational semantics σ_d of $\mathbb{I}_\Omega$ is the unique $\mathcal{F}$-homomorphism $\sigma_d : \mathbb{I}_\Omega \rightarrow \mathcal{P}(\mathbb{T}_\Omega)$ between the free term $\mathcal{F}$-algebra $\mathcal{T}_\mathcal{F}$ and $\mathcal{A}$.

Let us remark that our interaction language contains two kinds of loop associated to *seq* ($loop_H$ and $loop_W$), whereas the other two scheduling operators (*strict* and *par*) are each only associated to one loop (resp. $loop_S$ and $loop_P$). This is a direct consequence of Lemma 2 stating that Head-First closure and Kleene closure associated to resp. *strict* and *par* are equivalent. By contrast, this is not the case for *seq*, as demonstrated by the example from Figure 3.

4.2. A first example of interaction

An example of interaction is given on the left of Figure 5. 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, which exit the emitting lifeline or point towards the receiving lifeline. For example, in Figure 5, there is an emission ($l_1!m_4$) from the lifeline l_1 with the message m_4.

When a direct emission-reception causality occurs, we draw both actions as a single arrow from the emitter towards the receiver. We encode such direct causal relationships using strict sequencing, denoted by the keyword *strict*. For instance, in Figure 5, 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 causal order related to the passing of message m_1 between l_1 and l_3.



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

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

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

Figure 5: Example of a basic interaction & its trace semantics

The top to bottom direction relates to the passing of time. An action (arrow) drawn above another one generally occurs beforehand. This scheduling of actions corresponds to the weak sequencing operator, for which we use the *seq* keyword. This operator enforces precedence relations between actions occurring on the same lifeline. However, by contrast to strict sequencing, weak sequencing does not enforce precedence relations between actions occurring on different lifelines. In Figure 5, the term $seq(strict(l_1!m_1, l_3?m_1), strict(l_1!m_2, l_2?m_2))$ corresponds to the fact that the arrow carrying m_1 is drawn above that carrying 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. In contrast to *strict*, the *seq* operator has no graphical representation in diagrams, as it corresponds to the default scheduling operator.

Interleaving (also known as parallel) and alternative compositions can also be used to specify more complex behaviors. Those two operators correspond respectively to the *par* and *alt* keywords (displayed in Figure 5 in the top left corner of a box).

In Figure 5, 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. 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 Figure 5 is given by:

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

Here, either messages m_1 and m_2 are exchanged or (exclusively) messages m_3 and m_4 are exchanged.

The semantics of constants $\varnothing$ and $a \in \mathbb{A}_\Omega$ are sets containing a single element being respectively $\{\varepsilon\}$ and $\{a\}$. The $strict$, seq, par and alt symbols are respectively associated to the $;$, $;_{\divideontimes}$, $||$ and union $\cup$ operators on sets of traces. Their use is illustrated on the right of Figure 5 to compute the semantics of the interaction example given on the left of Figure 5. For instance, from the second line to the third line, using distinct colors for better visualization, weak sequencing $\{l_1!m_1.l_3?m_1\};_{\divideontimes}\{l_1!m_2.l_2?m_2\}$ 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 before, after or in-between $l_1!m_3$ and $l_2?m_3$. The resulting set of traces, which constitutes the semantics of the interaction drawn on the left, is given on the bottom right of Figure 5.

4.3. Some comments on the loop operators

From a system designer perspective, using $loop_S$, $loop_W$ or $loop_P$ applied to an arbitrary interaction i is motivated by different goals:

- With $loop_S(i)$, each instance of a repeatable behavior must be executed entirely before any other can start. Hence, we can use $loop_S$ to specify some repeatable critical behavior of which at most a single instance can exist at any time.

- With $loop_P(i)$, all existing instances can be executed concurrently with each other, and, at any given moment, new instances can be created. $loop_P$ can specify protocols in which any new sessions can be created and run in parallel.

- With $loop_W$, new instances can be created whenever the action triggering the instantiation occurs on a lifeline which is not occulted by previous instances. Roughly speaking, this is because:

  – even though on each lifeline, the lifeline must terminate the behavior assigned to it as part of the current instance of the loop before being able to start executing behaviors from a second instance,

- for any such instance, a lifeline might terminate before the others. Then it is allowed to start another instance while some other lifelines may not have yet finished the execution of the previous instance of the loop.

As a result, $loop_W$ can be used to specify repeatable behaviors that are sequential but that have no enforced synchronization mechanisms in-between lifelines.

The head loop $loop_H$ is associated with the weak HF-closure operator $^{\overset{\Lsh}{;_{\divideontimes}}*}$, which is an ad-hoc algebraic artefact and not a K-closure. We include it in our IL because it might correspond to a more intuitive understanding of sequential loops than $loop_W$, as illustrated in Section 5.3 and the survey [33].

4.4. *Identifying classes of semantically equivalent interactions*

From the algebraic properties of the $;$, $;_{\divideontimes}$, $||$, $\cup$, $^{;*}$, $^{\overset{\Lsh}{;_{\divideontimes}}*}$, $^{;_{\divideontimes}*}$ and $^{||*}$ operators given in Section 3 and summarized on Figure 4, one can infer the set of equations E from Definition 6. E induces a congruence relation $\approx_E$ on interaction terms, which, by construction, is sound (see Lemma 3) i.e., for any two terms i_1 and i_2, $i_1 \approx_E i_2$ implies that $\sigma_d(i_1) = \sigma_d(i_2)$. Thus, $\approx_E$ enables us to relate semantically equivalent but syntactically distinct interactions.

Definition 6. *Given signature $\mathcal{F}$ as introduced in Definition 4 and a set of variables $\mathcal{X}$ which includes $\{x, y, z\}$, let us consider the following axiom system E on the set $\mathcal{T}_{\mathcal{F}}(\mathcal{X})$ of interaction terms with variables:*

$$E = \begin{array}{rl} & \{f(f(x,y),z) \approx f(x,f(y,z)) \mid f \in \{strict, seq, par, alt\}\} \\ \cup & \{f(x,y) \approx f(y,x) \mid f \in \{par, alt\}\} \\ \cup & \{f(\varnothing, x) \approx x, \quad f(x,\varnothing) \approx x \mid f \in \{strict, seq, par\}\} \\ \cup & \{alt(x,x) \approx x\} \\ \cup & \bigcup_{k \in \{S,H,W,P\}} \left\{ \begin{array}{l} loop_k(\varnothing) \approx \varnothing, \\ alt(\varnothing, loop_k(x)) \approx loop_k(x) \end{array} \right\} \\ \cup & \bigcup_{(k_1,k_2) \in \{S,H,W,P\}^2} \{loop_{k_1}(loop_{k_2}(x)) \approx loop_{min(k_1,k_2)}(x)\} \end{array}$$

With $min(k_1, k_2)$ being defined[2] *given $P < W < H < S$.*

Lemma 3. *E is sound w.r.t. the $\mathcal{F}$-algebra $\mathcal{A}$ i.e., for any two terms t_1 and t_2 from $\mathcal{T}_{\mathcal{F}}$, if we have $t_1 \approx_E t_2$ then they have the same image $t_1^{\mathcal{A}} = t_2^{\mathcal{A}}$ (this implies interactions — i.e., ground terms — related by $\approx_E$ have the same σ_d semantics).*

Proof. The constituting equations are sound thanks to (in order): **(1)** the associativity of $;$, $;_{\divideontimes}$, $||$ and $\cup$, **(2)** the commutativity of $||$ and $\cup$, **(3)** $\{\varepsilon\}$ being a neutral element for the scheduling operators, **(4)** the idempotence of $\cup$, **(5)** properties of repetition operators including $(T^{\diamond_1 *})^{\diamond_2 *} = (T^{\diamond_2 *})^{\diamond_1 *} = T^{min(\diamond_1,\diamond_2)*}$. See Figure 4 for a summary of the algebraic properties and [22] for a Coq mechanisation. □

[2]reflecting the "repetition nesting" property of Figure 4

E being sound implies that the associated congruence $\approx_E$ puts in relation terms that are semantically equivalent. Let us remark however that E is not complete, which signifies that not all semantically equivalent interactions are related by $\approx_E$. Indeed, it suffices to consider that $seq(l!m_1, l!m_2)$ and $strict(l!m_1, l!m_2)$ have the same semantics but are not related by $\approx_E$. For the sake of simplicity and in order to limit the scope of this paper, we consider this simple axiom system E. Additional equations might be added to have a broader characterization of semantically equivalent interactions.

4.5. A rewrite system to characterize canonical interactions

Let us consider the following equational theory T, in which we gather all equations related to AC properties of our language.

$$T = \begin{array}{ll} & \{f(f(x,y),z) \approx f(x,f(y,z)) \mid f \in \{strict, seq, par, alt\}\} \\ \bigcup & \{f(x,y) \approx f(y,x) \mid f \in \{par, alt\}\} \end{array}$$

Using AC-R allows us to eliminate issues related to the directionality of rules related to associative and commutative properties. As a result, we define our rewrite system as the CRS $\rightarrow_{R/T}$ with the following rules:

$$R = \begin{array}{ll} & \{f(\varnothing,x) \rightsquigarrow x, \quad f(x,\varnothing) \rightsquigarrow x \mid f \in \{strict, seq, par\}\} \\ \cup & \{alt(x,x) \rightsquigarrow x\} \\ \cup & \bigcup_{k\in\{S,H,W,P\}} \left\{ \begin{array}{l} loop_k(\varnothing) \rightsquigarrow \varnothing, \\ alt(\varnothing, loop_k(x)) \rightsquigarrow loop_k(x) \end{array} \right\} \\ \cup & \bigcup_{(k_1,k_2)\in\{S,H,W,P\}^2} \{loop_{k_1}(loop_{k_2}(x)) \rightsquigarrow loop_{min(k_1,k_2)}(x)\} \end{array}$$

```
(VAR x y z)
(THEORY (AC alt) (AC par) (A strict) (A seq))
(RULES
  strict(x,∅) → x   seq(x,∅) → x   par(x,∅) → x
  strict(∅,x) → x   seq(∅,x) → x   par(∅,x) → x

  alt(x,x) → x

  loopS(∅) → ∅   loopH(∅) → ∅   loopW(∅) → ∅   loopP(∅) → ∅

  alt(∅,loopS(x)) → loopS(x)
  ...
  alt(∅,loopP(x)) → loopP(x)

  loopS(loopS(x)) → loopS(x)
  loopS(loopH(x)) → loopH(x)
  ...
  loopP(loopP(x)) → loopP(x)
)
```

Figure 6: Encoding (partial) of the CRS in WST format

The WST[3] format (international WorkShop on Termination) has been used to encode such rewrite systems, allowing verification of their properties to be

[3]termination-portal.org/wiki/WST

conducted automatically. On Figure 6, the encoding of our CRS is illustrated (with the $\cdots$ representing similar rules for the various loops). We used the TTT2 [18] tool to prove its termination and the CSI [41] tool to prove its confluence (see [24] for details).

Thus, we can define, for any interaction $i \in \mathbb{I}_\Omega$, a unique form $R(i)$ s.t. $i \rightarrow^{!}_{R/T} R(i)$. It is also self-evident that $\xrightarrow{*}_{R/T} = \approx_E$. Hence, the R form represents uniquely equivalence classes of $[\ ]_{\approx_E}$.

5. A structural operational semantics

In this section, we present a structural operational semantics for interactions in the style of process calculus [2]. It relies on the definition (by structural induction) of two predicates:

- $i \downarrow$ (the termination predicate), which indicates that the interaction i accepts the empty trace
- and $i \xrightarrow{a} i'$ (the execution relation) which 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 5.2 before defining $\rightarrow$ in Section 5.3.

5.1. Characterizing the termination of interactions

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, 32].

Definition 7 (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:*

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

Rules of Definition 7 are mostly trivial. 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 as stated by Lemma 4.

Lemma 4. *For any $i \in \mathbb{I}_\Omega$, $(i \downarrow) \Leftrightarrow (\varepsilon \in \sigma_d(i))$*

Proof. By induction on the term structure of interactions. See Appendix A. □

In summary, $i\downarrow$ signifies that i may terminate immediately, but because of non-determinism, depending on the nature of i, i can admit both empty and non-empty traces.

5.2. Dealing with weak-sequencing using evasion and 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 a that can be executed in i_2 (i.e., such that there exists a i'_2 satisfying $i_2 \xrightarrow{a} i'_2$) may also be executed in $seq(i_1, i_2)$ (i.e., such that we would have a i' satisfying $seq(i_1, i_2) \xrightarrow{a} i'$). 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 with $\theta(a)$ so that a can be placed in front of what is expressed by i_1 when recomposing $a.t$.

We define the evasion predicate $\downarrow^{\divideontimes}$ as a weaker notion than the termination predicate $\downarrow$, which corresponds to a form of local termination (up to a specific lifeline). For a lifeline l, we say that i evades l, denoted by $i \downarrow^{\divideontimes} l$ if i accepts at least one trace that does not contain actions occurring on l.

Definition 8 (Evasion)**.** *The predicate $\downarrow^{\divideontimes} \subset \mathbb{I}_\Omega \times L$ 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:*

$$\frac{}{\varnothing \downarrow^{\divideontimes} l} \qquad \frac{\theta(a) \neq l}{a \downarrow^{\divideontimes} l}$$

$$\frac{i_1 \downarrow^{\divideontimes} l}{alt(i_1, i_2) \downarrow^{\divideontimes} l} \qquad \frac{i_2 \downarrow^{\divideontimes} l}{alt(i_1, i_2) \downarrow^{\divideontimes} l} \qquad \frac{i_1 \downarrow^{\divideontimes} l \qquad i_2 \downarrow^{\divideontimes} l}{f(i_1, i_2) \downarrow^{\divideontimes} l} \qquad \frac{}{loop_k(i_1) \downarrow^{\divideontimes} l}$$

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 $\downarrow$ in Section 5.1.

Lemma 5. *For any $l \in L$ and $i \in \mathbb{I}_\Omega$,$(i \downarrow^{\divideontimes} l) \Leftrightarrow (\exists\ t \in \sigma_d(i), \neg(t \divideontimes l))$*

Proof. By induction on the term structure of interactions. See Appendix A. $\square$

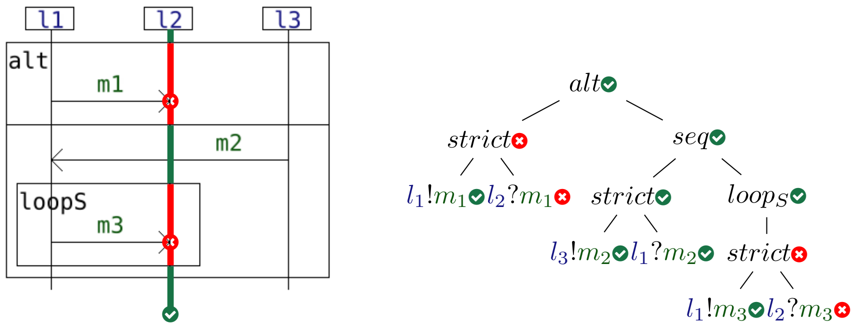
Let us remark that, for any $i \in \mathbb{I}_\Omega$, if $i\downarrow$ then for all l in L, $i \downarrow^{\divideontimes} l$. Indeed, if $i\downarrow$, then i can express the empty trace ε and ε has conflicts with no lifelines. The opposite does not hold: it suffices to consider $i = alt(l_1!m, l_2!m)$ and observe that for l in $\{l_1, l_2\}$, $i \downarrow^{\divideontimes} l$ holds while $i\downarrow$ does not.

Moreover, for ease of definitions and proofs in the following, we consider the collision predicate $\not\downarrow^{\divideontimes}$ by considering dual structural rules w.r.t. those defining the evasion predicate $\downarrow^{\divideontimes}$. Intuitively, $i \not\downarrow^{\divideontimes} l$ signifies that all traces of i have at least an action occurring on the lifeline l.

Definition 9 (Collision)**.** *The predicate $\not\downarrow^{\divideontimes} \subset \mathbb{I}_\Omega \times L$ is such that for i_1 and i_2 in $\mathbb{I}_\Omega$, $l \in L$, $a \in \mathbb{A}_\Omega$ and $f \in \{strict, seq, par\}$ we have:*

$$\frac{\theta(a) = l}{a \not\downarrow^{\divideontimes} l} \qquad \frac{i_1 \not\downarrow^{\divideontimes} l \qquad i_2 \not\downarrow^{\divideontimes} l}{alt(i_1, i_2) \not\downarrow^{\divideontimes} l} \qquad \frac{i_1 \not\downarrow^{\divideontimes} l}{f(i_1, i_2) \not\downarrow^{\divideontimes} l} \qquad \frac{i_2 \not\downarrow^{\divideontimes} l}{f(i_1, i_2) \not\downarrow^{\divideontimes} l}$$

As the rules defining the predicate $\not\downarrow^{\divideontimes}$ are defined by duality with respect to the ones defining the predicate $\downarrow^{\divideontimes}$, we have directly: $i \not\downarrow^{\divideontimes} l$ iff $\neg(i \downarrow^{\divideontimes} l))$.



(a) On diagram (b) On term

Figure 7: 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 in Figure 7. 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 choosing not to instantiate the loop, we can see that i accepts some traces that have no conflict with 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 of Figure 7, evasion is illustrated by drawing a line over l_2 which is the lifeline of interest. This line is decomposed into several areas that are colored either green or 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 $i = seq(i_1, i_2)$. This is only possible if the behavior of i_1 under consideration avoids the lifeline $\theta(a)$. As a consequence, 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 that 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 $\simeq^{\divideontimes}$ s.t. $i \simeq_l^{\divideontimes} i'$ 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 7).

Definition 10 (Pruning). *The pruning relation* $\simeq^{\divideontimes} \subset \mathbb{I}_\Omega \times L \times \mathbb{I}_\Omega$ *is s.t. for any* $l \in L$*, any* $f \in \{strict, seq, par\}$ *and any* $k \in \{S, H, W, P\}$*:*

$$\frac{}{\varnothing \simeq^{\divideontimes}_l \varnothing} \qquad \frac{}{a \simeq^{\divideontimes}_l a}\ \theta(a) \neq l \qquad \frac{i_1 \simeq^{\divideontimes}_l i'_1 \qquad i_2 \simeq^{\divideontimes}_l i'_2}{f(i_1, i_2) \simeq^{\divideontimes}_l f(i'_1, i'_2)}$$

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

$$\frac{i_1 \simeq^{\divideontimes}_l i'_1}{loop_k(i_1) \simeq^{\divideontimes}_l loop_k(i'_1)} \qquad \frac{}{loop_k(i_1) \simeq^{\divideontimes}_l \varnothing}\ i_1 \not\downarrow^{\divideontimes} l$$

Evasion (and its dual, collision) and pruning are intertwined notions. Indeed, as per Lemma 6 evasion is equivalent to the existence and unicity of a pruned interaction.

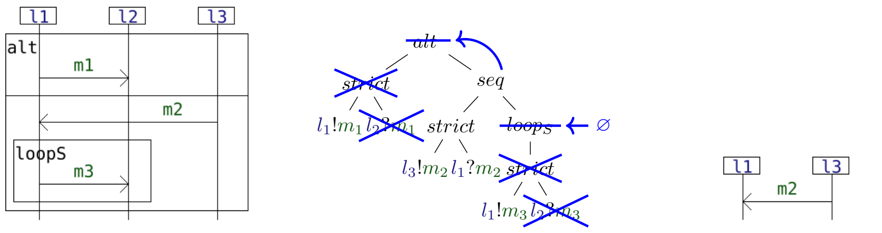
Lemma 6. *For any* $i \in \mathbb{I}_\Omega$ *and any* $l \in L$*,* $(i \downarrow^{\divideontimes} l) \Leftrightarrow (\exists!\ i' \in \mathbb{I}_\Omega$ *s.t.* $i \simeq^{\divideontimes}_l i')$

Proof. By induction on the term structure of interactions. See Appendix A. □

Let us comment on the rules defining the pruning relation.

- We have $\varnothing \simeq^{\divideontimes}_l \varnothing$ because the semantics of $\varnothing$ being $\{\varepsilon\}$, there are no conflicts with l;
- Any action $a \in \mathbb{A}_\Omega$ has conflicts with a lifeline l iff $\theta(a) = l$. If this is not the case, then a needs not be eliminated and thus $a \simeq^{\divideontimes}_l a$;
- 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 $i \simeq^{\divideontimes}_l i'$ 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 $i_1 \simeq^{\divideontimes}_l i'_1$. This being the modification which preserves a maximum amount of traces, we have $loop_k(i_1) \simeq^{\divideontimes}_l loop_k(i'_1)$.

We have seen that the interaction i of Figure 7 satisfies $i \downarrow^{\divideontimes} l_2$. Therefore Lemma 6 implies the existence of a unique i' s.t. $i \simeq^{\divideontimes}_{l_2} i'$. Figure 8 illustrates the computation of i'. The blue lines represent the modifications in the syntax of i that occur during its pruning into i'. On Figure 7 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



(a) Before pruning (b) Pruning w.r.t. l_2 (c) After pruning

Figure 8: Illustration of pruning

express actions occurring on l_2. Hence, in Figure 8, 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 Figure 7 we have seen that we have $i_1 \not\downarrow^{\divideontimes} l_2$ and $i_2 \downarrow^{\divideontimes} l_2$. Therefore we have $i \simeq^{\divideontimes}_{l_2} i'_2$ with i'_2 being such that $i_2 \simeq^{\divideontimes}_{l_2} i'_2$. This selection of the right branch of the *alt* is illustrated in Figure 8 by the curved arrow, which replaces the *alt* by the *seq* on its bottom right. There remains to determine i'_2 s.t. $i_2 \simeq^{\divideontimes}_{l_2} i'_2$. At the root of i_2, we have a *seq*. Let us note $i_2 = seq(i_A, i_B)$. As per Figure 7 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 $i_A \simeq^{\divideontimes}_{l_2} i'_A$ and $i_B \simeq^{\divideontimes}_{l_2} i'_B$. 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 Figure 7 we have $i_C \not\downarrow^{\divideontimes} l_2$ and therefore $i'_B = \varnothing$ which is illustrated on Figure 8 by the $\leftarrow \varnothing$ in blue, which replaces the $loop_S$ by $\varnothing$. Finally there remains $i' = seq(i_A, \varnothing)$, which is drawn on the right of Figure 8.

Lemma 7 states that given $i \simeq^{\divideontimes}_{l} i'$, the pruned interaction i' exactly specifies all the executions of i that do not involve l.

Lemma 7. *For any $l \in L$ and any i and i' from $\mathbb{I}_\Omega$ verifying $i \simeq^{\divideontimes}_{l} i'$:*

$$\sigma_d(i') = \{t \in \sigma_d(i) \mid \neg(t \divideontimes l)\}$$

Proof. By induction on the term structure of interactions. See Appendix A. □

5.3. Definition of the operational semantics

A structural operational semantics in the style of Plotkin [36] allows determining traces $t = a_1. \cdots .a_n$ belonging to the set of traces of an interaction through the assertion of a succession of predicates of the form $i_j \xrightarrow{a_j} i_{j+1}$ representing the evolution of the interaction. By expressing action a_j, the system goes from being modelled by i_j to being modelled by i_{j+1}.

Definition 11. *Given i_1, i_2, i'_1 and i'_2 interactions in $\mathbb{I}_\Omega$ and a an action in $\mathbb{A}_\Omega$, the execution relation $\rightarrow \subset \mathbb{I}_\Omega \times \mathbb{A}_\Omega \times \mathbb{I}_\Omega$ is s.t.:*

$$\frac{}{a \xrightarrow{a} \varnothing} \qquad \frac{i_1 \xrightarrow{a} i'_1}{alt(i_1, i_2) \xrightarrow{a} i'_1} \qquad \frac{i_2 \xrightarrow{a} i'_2}{alt(i_1, i_2) \xrightarrow{a} i'_2}$$

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

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

$$\frac{i_1 \xrightarrow{a} i_1'}{seq(i_1,i_2) \xrightarrow{a} seq(i_1',i_2)} \qquad \frac{i_1 \simeq^{\divideontimes}_{\theta(a)} i_1' \qquad i_2 \xrightarrow{a} i_2'}{seq(i_1,i_2) \xrightarrow{a} seq(i_1',i_2')}$$

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

$$\frac{i_1 \xrightarrow{a} i_1' \qquad loop_W(i_1) \simeq^{\divideontimes}_{\theta(a)} i'}{loop_W(i_1) \xrightarrow{a} seq(i',seq(i_1',loop_W(i_1)))} \qquad \frac{i_1 \xrightarrow{a} i_1'}{loop_P(i_1) \xrightarrow{a} par(i_1',loop_P(i_1))}$$

Most of these rules are similar to those found in the structural operational semantics of process algebras. Let us start by commenting on these relatively standard rules below. As the weak sequential loop $loop_W$ is a more original operator specific to interactions, the rule relating to $loop_W$ will be discussed in more detail later.

- In an interaction reduced to an action a, a can 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 can also be executed in i and the resulting interaction 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 can only be executed on the right of $i = strict(i_1,i_2)$ if i_1 terminates. Indeed, in that case, i_1 may express the empty trace ε as per Lemma 4, and nothing prevents a from being 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. But this rule has no direct analogue in the domain of process algebras. The condition is specific to interactions, without a counterpart in the context of process algebras. This condition is that i_1 evades the lifeline $\theta(a)$, i.e., $i_1 \downarrow^{\divideontimes} \theta(a)$, which, as per Lemma 7 is implied by the condition $i_1 \simeq^{\divideontimes}_{\theta(a)} i_1'$. 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 point out that this rule is sound since i'_1 precisely characterizes all traces of i_1 that avoid the lifeline $\theta(a)$.

- The rules for the loop operators $loop_S$, $loop_H$ and $loop_P$ are similar, 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., $a.t \in \{a.t_1\} \diamond^{\Lsh} \sigma_d(loop_k(i)) \subset \sigma_d(i) \diamond^{\Lsh} \sigma_d(loop_k(i))$, it coincides with using the restricted operator $\diamond^{\Lsh}$ as a scheduler. It turns out that $loop_H$ is explicitly associated with $;_{\divideontimes}^{\Lsh}{}^*$, and thus, the formulation of its rule is self-evident. In the case of $loop_S$ and $loop_P$, the HF and K-closures of ; and || are equivalent (as per Lemma 2) which enables their respective rules to be formulated in this manner.

In the following, we discuss the rule for $loop_W$ and give an intuition of its meaning from two perspectives, one operational, and one denotational.

Operational perspective on $loop_W$. A trace specified by $loop_W(i_1)$ is specified by the weak sequencing of a finite number $n \geq 0$ of iterations of the loop so that it belongs to the semantics of a certain $seq(x^1, seq(\cdots, seq(x^{n-1}, x^n)))$, where each $x^k = i_1$ with $k \in [1, n]$ corresponds to one iteration. For any iteration x^k with $k \in [1, n]$, let us call "ealier" iterations the x^j with $j < k$ and "later" iterations the x^j with $j > k$. Weak sequencing may allow the first action a that is executed to be taken from an iteration that is not the first iteration x^1. Let us suppose it is a certain x^k so that we have $x^k \xrightarrow{a} i'_1$. Then, the follow-up is defined by:

$$\begin{array}{l} seq(x^1, seq(\cdots, seq(x^k, seq(\cdots, x^n)))) \\ \qquad\qquad\qquad \xrightarrow{a} \\ \qquad\qquad\qquad\qquad seq(y^1, seq(\cdots, seq(y^{k-1}, seq(i'_1, seq(x^{k+1}, seq(\cdots, x^n)))))) \end{array}$$

where $\forall\ j \in [1, k-1],\ x^j \simeq^{\divideontimes}_{\theta(a)} y^j$. Indeed, it suffices to apply the right-hand-side rule for operator seq $k-1$ times and the left-hand-side rule once. Because the seq operator is associative, we can therefore rewrite this follow-up as $seq(Y, seq(i'_1, X))$ where:

$$\begin{array}{lcl} Y & = & seq(y^1, seq(\ldots, y^{k-1})) \\ X & = & seq(x^{k+1}, seq(\ldots, x^n)) \end{array}$$

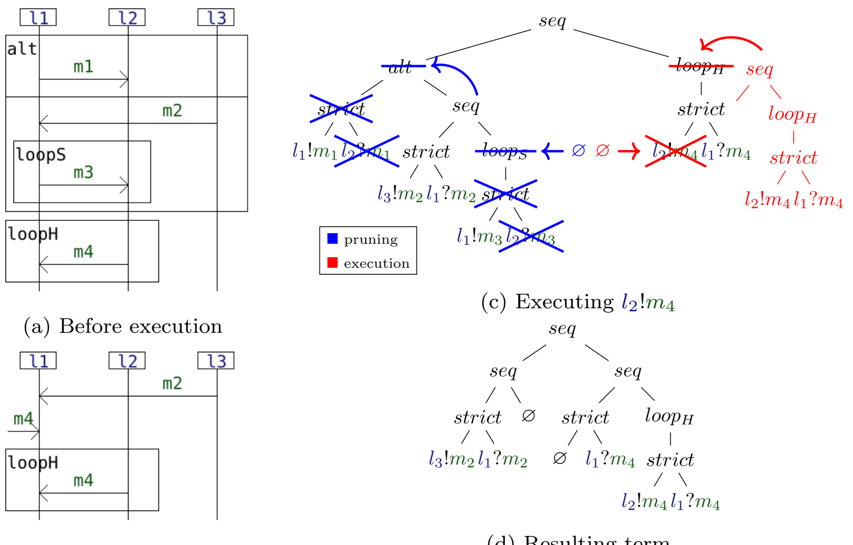
This structural pattern $seq(Y, seq(i'_1, X))$ matches that of $seq(i', seq(i'_1, loop_W(i_1)))$, the follow-up defined in the rule for $loop_W$. Moreover, we can see that X corresponds to the weak sequencing of a number of i_1 and therefore, its semantic is included in that of $loop_W(i)$. Similarly, because of the inductive definition of the $\simeq^{\divideontimes}$ relation, we have $seq(x^1, seq(\ldots, x^{k-1})) \simeq^{\divideontimes}_{\theta(a)} Y$ and therefore, the semantics of Y is included, by construction, in that of i' defined by $loop_W(i_1) \simeq^{\divideontimes}_{\theta(a)} i'$.

Now that we have provided an intuition of the reasoning behind the structure of the rule, let us discuss it from the perspective of the actions that are successively executed. In the rule for $loop_W$, we prepend (via seq) the pruned version i' of the loop $loop_W(i_1)$ (i.e., defined by $loop_W(i_1) \simeq^{\divideontimes}_{\theta(a)} i'$) so that the transition is

$$loop_W(i_1) \xrightarrow{a} seq(i', seq(i'_1, loop_W(i_1)))$$

rather than $loop_H(i_1) \xrightarrow{a} seq(i'_1, loop_H(i_1))$, which distinguishes it from the rule for $loop_H$. By doing so, we allow the first action executed after a (so the second action in total) to be taken from $seq(i', seq(i'_1, loop_W(i)))$ instead of being limited to $seq(i'_1, loop_W(i))$. This allows the second action (after a) to be taken from earlier iterations of the loop (earlier w.r.t. that which has yielded i'_1), thus revealing traces that would not otherwise be reachable paths using the rule for $loop_H$. The example from Figure 10 illustrates this. Let us consider the path starting from the interaction denoted with Ⓒ. The first action $l_1!m_1$ is taken from a particular iteration of the loop (not necessarily the first). What remains to be executed in that iteration of the loop corresponds to the reception $l_2?m_1$. We can see that the pruned version of the loop i' drawn as $loop_W(l_2!m_2)$ is prepended (above) that remaining $l_2?m_1$. This allows the second action executed in that path to be $l_2!m_2$, which is not possible using the rule for $loop_H$ (as demonstrated with the path starting from the interaction denoted with Ⓑ). In summary, prepending the pruned version of the loop enables us to delay the determination of the iteration of the loop to which the first executed action belongs. In our example, if the trace is $l_1!m_1.l_2?m_1$ then necessarily $l_1!m_1$ belongs to the first iteration. By contrast if the trace is $l_1!m_1.l_2!m_2.l_2?m_1$, this implies that $l_1!m_1$ is taken from the second iteration and if it is $l_1!m_1.l_2!m_2.l_2!m_2.l_2?m_1$, this entails that $l_1!m_1$ is taken from the third iteration and so on. The use of the pruned version i' of the loop as the artifact that we prepend is justified by weak sequencing. In our example, from the perspective of lifeline l_1, the first action that is executed is $l_1!m_1$, so we cannot have another action occurring on l_1 and corresponding to an earlier iteration of the loop. As a result, we may only allow iterations of the loop that do not involve any action occurring on l_1, i.e., the lifeline $\theta(a)$ on which the action a is executed. Hence, because via $loop_W(i_1) \simeq^{\divideontimes}_{\theta(a)} i'$, the pruning relation characterizes an interaction i' in which only these possible iterations remain, the artifact that we must prepend indeed corresponds to i'.

Denotational perspective on $loop_W$. Recalling results from Section 3, we can reason as follows. 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 $i \simeq^{\divideontimes}_{\theta(a)} i'$ and $i_1 \xrightarrow{a} i'_1$. Given that i is a loop, it is always prunable (Lemma 6), so there exists i' s.t. $i \simeq^{\divideontimes}_{\theta(a)} i'$. 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 $\varepsilon \in \sigma_d(i')$ (Lemma 7), we have $t \in \{\varepsilon\};_{\divideontimes} \{t'_1\};_{\divideontimes} \sigma_d(i)$ 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 7 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 (and not only the first, which is enforced by the rule for $loop_H$).



(a) Before execution

(b) After execution

(c) Executing $l_2!m_4$

(d) Resulting term

Figure 9: Illustration of execution

Figure 9 extends the example from Figure 8 and illustrates the use of the execution relation. Here, action $l_2!m_4$ is to be executed. At the root of the interaction is a *seq* operator. Here, it is possible to execute $l_2!m_4$ because, as we have seen with the example from Figure 7 and Figure 8, the left sub-term evades lifeline l_2. $l_2!m_4$ is executable within the right sub-term because it is its left-most leaf. Then, in order to obtain the resulting term, the left sub-term is pruned w.r.t. l_2 and action $l_2!m_4$ is executed within the right sub-term. At the root of the right sub-term is a $loop_H$ operator. Applying the corresponding rule yields a repetition using *seq*. What remains to be executed in the loop instance is $l_1?m_4$, and the initial loop is scheduled afterwards using *seq*.

Figure 10, which is the interaction counterpart of the trace example from Figure 3 further illustrates the rule for the $loop_W$ operator and makes explicit its difference with $loop_H$. Let us consider repetitions of the interaction $i = alt(strict(l_1!m_1, l_2?m_1), l_2!m_2)$. The interaction designated with Ⓐ at the top left of Figure 10 corresponds to $i_A^0 = seq(i,i)$ (i is repeated twice using weak sequencing). The path $i_A^0 \xrightarrow{l_1!m_1} i_A^1 \xrightarrow{l_2!m_2} i_A^2 \xrightarrow{l_2?m_1} i_A^3$ starting from that interaction and going to the right on Figure 10 corresponds to a behavior that can be expressed by i_A^0 (i.e., a trace accepted by i_A^0). In $i_A^0 = seq(i,i)$, there are two distinct occurrences of action $l_1!m_1$ (in the top *alt* and the bottom *alt*). In the trace highlighted on Figure 10, the first executed action corresponds to

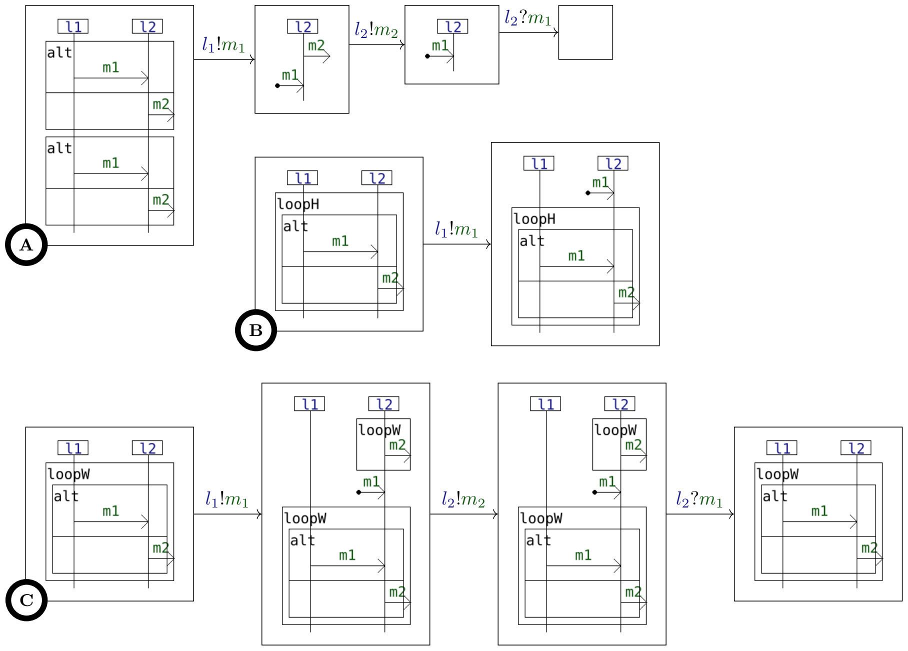


Figure 10: Illustrating the operational semantics and counter-example of Figure 3

the second occurrence of $l_1!m_1$ (at the bottom). This action is immediately 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 $seq(i,i)$. Let us now consider the interaction designated with Ⓑ which is $i^0_B = loop_H(i)$. Here, 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 from being executed. As $loop_H$ is associated with the weak HF-closure $\overset{\hookleftarrow}{;_{\times}}{}^*$ and not to the K-closure $;_{\times}{}^*$, it is therefore expected that t could not be accepted by $loop_H(i)$ in this example. However, considering the interaction $i^0_C = loop_W(i)$ designated with Ⓒ, the semantics of that $loop_W(i)$ contains t. Indeed, prepending 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.

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

Definition 12 (Operational semantics). *$\sigma_o : \mathbb{I}_\Omega \rightarrow \mathcal{P}(\mathbb{T}_\Omega)$ is s.t.:*

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

With i and i' interactions in $\mathbb{I}_\Omega$, t trace in $\mathbb{T}_\Omega$ and a action in $\mathbb{A}_\Omega$.

5.4. Correctness of operational semantics with respect to denotational semantics

In the following we prove the equivalence of σ_o and σ_d. The proof is a transcript of a formal Coq proof available in [23]. 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 $\rightarrow$ in relation to σ_d:

Lemma 8. *For any a in $\mathbb{A}_\Omega$, t in $\mathbb{T}_\Omega$ and i, i' in $\mathbb{I}_\Omega$,*

$$\Big(\; (i \xrightarrow{a} i') \wedge (t \in \sigma_d(i')) \; \Big) \Rightarrow (a.t \in \sigma_d(i))$$

Proof. By induction on the rules from Definition 11 that make the hypothesis $i \xrightarrow{a} i'$ possible. See Appendix A. □

Lemma 8 and Lemma 4 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 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. Let us consider $i \in \mathbb{I}_\Omega$ and $t \in \sigma_o(i)$ and let us reason by induction on the trace t.

- If $t = \varepsilon$, the definition of σ_o entails that $i \downarrow$. Then as per Lemma 4, we necessarily have $\varepsilon \in \sigma_d(i)$.
- If $t = a.t'$ then, by definition of σ_o, $a.t' \in \sigma_o(i)$ iff $\exists\ i' \in \mathbb{I}_\Omega$ s.t. $i \xrightarrow{a} i'$ and $t' \in \sigma_o(i')$. Let us consider such an interaction i'. By the induction hypothesis on t', we have $(t' \in \sigma_o(i')) \Rightarrow (t' \in \sigma_d(i'))$. As a result we have $i \xrightarrow{a} i'$ and $t' \in \sigma_d(i')$. By Lemma 8, we conclude that $a.t' \in \sigma_d(i)$.

□

Let us now prove the reciprocate, which is that for any interaction i, we have $\sigma_d(i) \subseteq \sigma_o(i)$. We provide, with Lemma 9, a second characterization of $\rightarrow$ with regard to σ_d.

Lemma 9. *For any $a \in \mathbb{A}_\Omega$, $t \in \mathbb{T}_\Omega$ and $i \in \mathbb{I}_\Omega$,*

$$(a.t \in \sigma_d(i)) \Rightarrow \Big(\exists\ i' \in \mathbb{I}_\Omega, \quad (i \xrightarrow{a} i') \wedge (t \in \sigma_d(i')) \; \Big)$$

Proof. By induction on the term structure of interactions. See Appendix A. □

Thanks to Lemma 9 and Lemma 4 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. Let us consider $i \in \mathbb{I}_\Omega$ and $t \in \sigma_d(i)$ and let us reason by induction on the trace t.

- If $t = \varepsilon$, the fact that $t = \varepsilon \in \sigma_d(i)$ implies, as per Lemma 4, that $i \downarrow$. Then, the definition of σ_o implies that $\varepsilon \in \sigma_o(i)$.
- If $t = a.t'$ then, the fact that $a.t' \in \sigma_d(i)$ implies, as per Lemma 9, that there exists an interaction i' such that $i \xrightarrow{a} i'$ and $t' \in \sigma_d(i')$. Let us therefore consider such an interaction i'. By the induction hypothesis on trace t', we have $(t' \in \sigma_d(i')) \Rightarrow (t' \in \sigma_o(i'))$. As a result we have $i \xrightarrow{a} i'$ and $t' \in \sigma_o(i')$. By definition of the operational semantics, this implies that $a.t' \in \sigma_o(i)$. Hence the property holds.

□

We have therefore proven that the sets of traces associated with interactions respectively by denotational semantics and operational semantics are equal. Both semantics σ_d and σ_o are equivalent.

Incidentally, for any two interactions i_1 and i_2, if $i_1 \approx_E i_2$ (with the axiom system E from Definition 6) then $\sigma_o(i_1) = \sigma_o(i_2)$. All syntactic transformations which are sound w.r.t. σ_d are also sound w.r.t. σ_o.

6. Implementing action-traceable interaction execution

6.1. Motivation

We illustrate on Figure 11 how the operational formulation of the trace semantics of interactions (via the execution relation $\rightarrow$) might be used to execute interaction models and explore their semantics.

Let us consider the initial interaction i:

$$\begin{aligned} i = seq(alt(\quad & seq(strict(l_1!m_1, l_2?m_1), strict(l_2!m_2, l_3?m_2)), \\ & loop_W(strict(l_1!m_2, l_3?m_2)) \ ), \\ strict(l_1!m_1, l_3?m_1) \ ) \end{aligned}$$

According to the operational semantics described in Section 5, the unique derivation tree for executing action $l_1!m_1$ which is at position 1111 in i is the following:

$$\cfrac{\cfrac{\cfrac{\cfrac{i_{|1111} = l_1!m_1 \xrightarrow{l_1!m_1} \varnothing}{i_{|111} = strict(l_1!m_1, l_2?m_1) \xrightarrow{l_1!m_1} strict(\varnothing, l_2?m_1)}}{i_{|11} = seq(\cdots) \xrightarrow{l_1!m_2} seq(strict(\varnothing, l_2?m_1), strict(l_2!m_2, l_3?m_2))}}{i_{|1} = alt(\cdots) \xrightarrow{l_1!m_2} seq(strict(\varnothing, l_2?m_1), strict(l_2!m_2, l_3?m_2))}}{i \xrightarrow{l_1!m_2} seq(seq(strict(\varnothing, l_2?m_1), strict(l_2!m_2, l_3?m_2)), strict(l_1!m_1, l_3?m_1))}$$

From the bottom to the top, we successively apply the rule for the *seq* operator for its first argument (relative position 1), then the rule for the *alt* operator (relative position 1), then *seq* (relative position 1), *strict* (relative position 1) and finally the rule for executing the action $l_1!m_1$ (relative position

ε). By concatenation of the successive positions, we get the position 1111 of the executed action $l_1!m_1$ within the initial interaction i.

The derivation tree of any derivation $i \xrightarrow{a} i'$ is uniquely defined by the position p in i of the action a that is executed. We can therefore designate unambiguously any and all such derivations using the position of the executed action. Using positions is notably pertinent as there can be several instances of the same action. Indeed, for a given action a there may be several i'_j such that $i \xrightarrow{a} i'_j$. On Figure 11 are represented (using diagrammatic representations of interactions) all the three possible derivations of our example interaction. The one on the left corresponds to the derivation which we have presented (executing action $l_1!m_1$ at position 1111). Let us remark the presence of two distinct instances of action $l_1!m_1$ at different positions (1111 and 21) which can be immediately executed and yield different "follow-up" interactions.

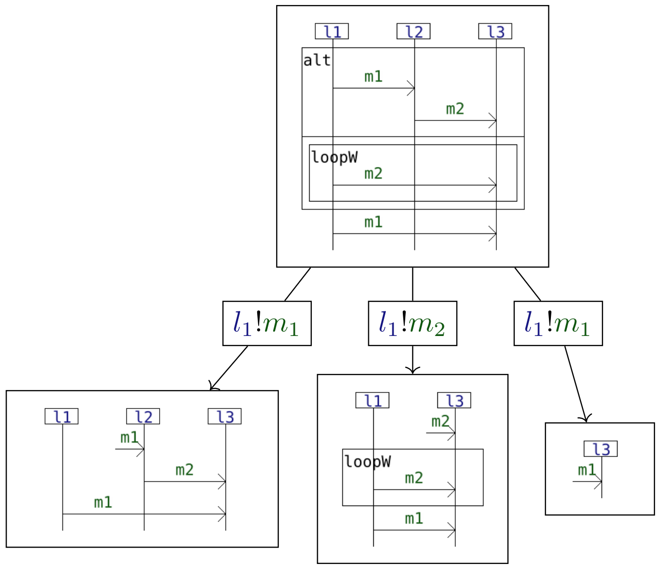


Figure 11: Deriving term transformations from the $\rightarrow$ execution relation

Via the use of the positions p of immediately executable actions, it is possible to infer the derivations $i \xrightarrow{i_{|p}} i'$ without having to construct the corresponding derivation trees. As a result, the operational-style semantics may be reformulated in a functional-style, based on the definition of two functions:

- one which determines the positions of immediately executable actions
- and one which, given an interaction and such a position, computes the corresponding follow-up interaction

This reformulation separates these two concerns which are intertwined in the operational formulation of the predicate $i \xrightarrow{a} i'$. The advantages of this reformulation includes **(1)** an easy implementation, **(2)** that we do not need to compute all derivations and follow-up interactions to know which actions are immediately executable and **(3)** that by using it, we can trace the unique positions of the actions which are executed thus resolving the inherent non-determinism of interactions.

To mirror the denotational and operational semantics, we call this reformulation an *execution semantics*. In this section, we ground this semantics denoted as σ_e with respect to the two previous formal semantics σ_d and σ_o and we propose implementation optimizations via the use of the semantically sound transformations of Definition 6.

6.2. Pruning

The termination $\downarrow$ and evasion $\downarrow^{\divideontimes}$ predicates can be trivially expressed in functional style. In Definition 13, we propose a functional-style formulation of the pruning relation. Lemma 10 states that the pruning relation is equivalent to its functional-style formulation.

Definition 13 (Pruning in functional style)**.** *The function* $\boldsymbol{prn} : \mathbb{I}_\Omega \times L \rightarrow \mathbb{I}_\Omega$ *is defined for any couple* (i, l) *in* $\mathbb{I}_\Omega \times L$ *verifying* $i \downarrow^{\divideontimes} l$ *as follows*[4]*:*

$$
\bullet\ \textbf{\textit{provided}}\ i \downarrow^{\divideontimes} l\ \textbf{\textit{then}}\ \boldsymbol{prn}(i,l) = \ \textbf{\textit{match}}\ i\ \textbf{\textit{with}}
$$

$$
\begin{array}{lll}
\mid \varnothing & \rightarrow & \varnothing \\
\mid a \in \mathbb{A}_\Omega & \rightarrow & a \\
\mid alt(i_1, i_2) & \rightarrow & \left\{ \begin{array}{ll} \boldsymbol{prn}(i_2, l) & \textbf{\textit{if}}\ i_2 \downarrow^{\divideontimes} l\ \wedge\ \neg i_1 \downarrow^{\divideontimes} l \\ \boldsymbol{prn}(i_1, l) & \textbf{\textit{if}}\ i_1 \downarrow^{\divideontimes} l\ \wedge\ \neg i_2 \downarrow^{\divideontimes} l \\ alt(\boldsymbol{prn}(i_1, l), \boldsymbol{prn}(i_2, l)) & \textbf{\textit{if}}\ i_1 \downarrow^{\divideontimes} l\ \wedge\ i_2 \downarrow^{\divideontimes} l \end{array} \right. \\
\mid f(i_1, i_2) & \rightarrow & f(\boldsymbol{prn}(i_1, l), \boldsymbol{prn}(i_2, l))\ \ \textbf{\textit{for}}\ f \in \{strict, seq, par\} \\
\mid loop_k(i_1) & \rightarrow & \left\{ \begin{array}{ll} loop_k(\boldsymbol{prn}(i_1, l)) & \textbf{\textit{if}}\ i_1 \downarrow^{\divideontimes} l \\ \varnothing & \textbf{\textit{else}} \end{array} \right. \quad \textbf{\textit{for}}\ k \in \{S, H, W, P\}
\end{array}
$$

Lemma 10. *For any* $i \in \mathbb{I}_\Omega$ *and any* $l \in L$*:*

$$
(\exists\ i' \in \mathbb{I}_\Omega,\ s.t.\ i \simeq^{\divideontimes}_l i') \Rightarrow (\boldsymbol{prn}(i, l) = i') \qquad \textit{and} \qquad (i \downarrow^{\divideontimes} l) \Rightarrow (i \simeq^{\divideontimes}_l \boldsymbol{prn}(i, l))
$$

Proof. By induction on the term structure of interactions. See Appendix A. □

6.3. Frontier

As previously discussed in Section 6.1, within an interaction i, there can be several occurrences of the same action a. Hence we can have two distinct interactions i'_A and i'_B such that $i \xrightarrow{a} i'_A$ and $i \xrightarrow{a} i'_B$. As illustrated in the example in Section 6.1, we unambiguously identify actions via their position within the interaction.

Actions can only be found at the leaf nodes of the syntactic tree of an interaction. Among these actions, only some are immediately executable. Given an interaction $i \in \mathbb{I}_\Omega$, Definition 14 introduces the set $\mathtt{frt}(i) \subseteq pos(i)$ of positions of these immediately executable actions, which we call the frontier of execution.

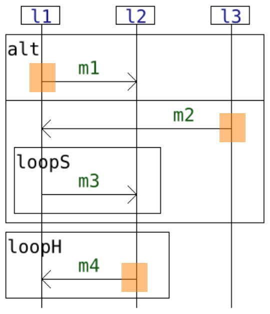
[4]The presentation is freely inspired by the syntax of the OCaml functional language.

Definition 14 (Frontier of execution). *The function* $\texttt{frt} : \mathbb{I}_\Omega \rightarrow \mathcal{P}(\{1,2\}^*)$ *is defined for each* i *in* $\mathbb{I}_\Omega$ *by:*

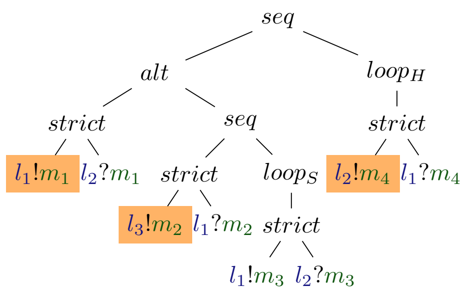
- $\texttt{frt}(i) =$ ***match*** i ***with***

$$\begin{array}{lll}
\mid \varnothing & \rightarrow & \emptyset \\
\mid a \in \mathbb{A}_\Omega & \rightarrow & \{\varepsilon\} \\
\mid \mathit{strict}(i_1, i_2) & \rightarrow & \begin{cases} 1.\texttt{frt}(i_1) \cup 2.\texttt{frt}(i_2) & \textbf{\textit{if}}\ i_1 \downarrow \\ 1.\texttt{frt}(i_1) & \textbf{\textit{else}} \end{cases} \\
\mid \mathit{seq}(i_1, i_2) & \rightarrow & 1.\texttt{frt}(i_1) \cup \{p \mid p \in 2.\texttt{frt}(i_2)\ ,\ i_1 \downarrow^{\divideontimes} \theta(i_{|p})\} \\
\mid f(i_1, i_2) & \rightarrow & 1.\texttt{frt}(i_1) \cup 2.\texttt{frt}(i_2)\ \textbf{\textit{for}}\ f \in \{\mathit{alt}, \mathit{par}\} \\
\mid \mathit{loop}_k(i_1) & \rightarrow & 1.\texttt{frt}(i_1)\ \textbf{\textit{for}}\ k \in \{S, H, W, P\}
\end{array}$$

$\texttt{frt}(i)$ is inferred statically from the term structure of i. The empty interaction has an empty frontier: $\texttt{frt}(\varnothing) = \emptyset$. For any atomic action $a \in \mathbb{A}_\Omega$, $\texttt{frt}(a) = \{\varepsilon\}$, because the position ε designates the root node of the interaction. For any interaction i of the form $\mathit{loop}_k(i_1)$, with $k \in \{S, H, W, P\}$, we have $\texttt{frt}(i) = 1.\texttt{frt}(i_1)$ because the loop can be instantiated through the execution of any one of the immediately executable actions from its sub-interaction, which specifies the behaviors that can be repeated.



(a) On diagram



(b) On term

Figure 12: Frontier of execution

Given $i_1, i_2 \in \mathbb{I}_\Omega$ and $f \in \{\mathit{strict}, \mathit{seq}, \mathit{par}, \mathit{alt}\}$, if an interaction i is of the form $f(i_1, i_2)$, then $\texttt{frt}(i)$ is inferred from $\texttt{frt}(i_1)$ and $\texttt{frt}(i_2)$.

- In all cases, actions that are immediately executable within the left sub-interaction i_1 are also immediately executable in i. Indeed, ordering constraints are only introduced on the right sub-term i_2. Thus $1.\texttt{frt}(i_1)$ is included in $\texttt{frt}(i)$.
- As for the actions that are immediately executable within the right sub-interaction i_2, whether or not they are also immediately executable in i depends on the nature of the constructor f and that of the left sibling interaction i_1.
  - If $f = \mathit{alt}$ or $f = \mathit{par}$, $2.\texttt{frt}(i_2)$ is also included in $\texttt{frt}(i)$ because no constraint may prevent the execution of actions from i_2.

- If $f = strict$, any action from i_2 can only be executed in i if no action from i_1 is executed in the same execution (otherwise it would violate the strict sequencing). Therefore $2.\texttt{frt}(i_2)$ is included in $\texttt{frt}(i)$ iff i_1 accepts the empty trace i.e., iff $i_1 \downarrow$.
- If $f = seq$, elements p from $2.\texttt{frt}(i_2)$ are included in $\texttt{frt}(i)$ iff i_1 accepts an execution that does not involve the lifeline on which the action $i_{|p}$ occurs i.e., iff $i_1 \downarrow^{*} \theta(i_{|p})$.

Figure 12 illustrates the use of frt on our running example. Here we have three immediately executable actions: $l_1!m_1$, $l_3!m_2$ and $l_2!m_4$ which are respectively at positions 111, 1211 and 221.

6.4. Execution

In Definition 15, we define a function exe which returns the unique follow-up interaction resulting from the execution of a specific frontier action specified by its position.

Definition 15 (Interaction Execution)**.** *The function* $\texttt{exe} : \mathbb{I}_\Omega \times \{1,2\}^* \rightarrow \mathbb{I}_\Omega$ *is defined for any* $i \in \mathbb{I}_\Omega$ *and* $p \in \texttt{frt}(i)$ *by:*

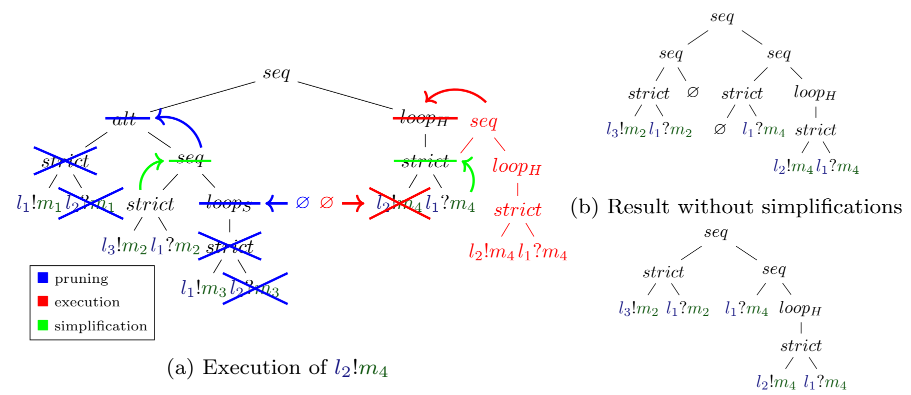
- $\texttt{exe}(i,p) =$ ***match*** (i,p) ***with***

$$
\begin{array}{lll}
\mid (act, \varepsilon) & \rightarrow & \varnothing \\
\mid (alt(i_1,i_2), 1.p_1) & \rightarrow & \texttt{exe}(i_1,p_1) \\
\mid (alt(i_1,i_2), 2.p_2) & \rightarrow & \texttt{exe}(i_2,p_2) \\
\mid (strict(i_1,i_2), 1.p_1) & \rightarrow & strict(\texttt{exe}(i_1,p_1), i_2) \\
\mid (strict(i_1,i_2), 2.p_2) & \rightarrow & \texttt{exe}(i_2,p_2) \\
\mid (seq(i_1,i_2), 1.p_1) & \rightarrow & seq(\texttt{exe}(i_1,p_1), i_2) \\
\mid (seq(i_1,i_2), 2.p_2) & \rightarrow & seq(\boldsymbol{prn}(i_1, \theta(i_{2|p_2})), \texttt{exe}(i_2,p_2)) \\
\mid (par(i_1,i_2), 1.p_1) & \rightarrow & par(\texttt{exe}(i_1,p_1), i_2) \\
\mid (par(i_1,i_2), 2.p_2) & \rightarrow & par(i_1, \texttt{exe}(i_2,p_2)) \\
\mid (loop_S(i_1), 1.p_1) & \rightarrow & strict(\texttt{exe}(i_1,p_1), i) \\
\mid (loop_H(i_1), 1.p_1) & \rightarrow & seq(\texttt{exe}(i_1,p_1), i) \\
\mid (loop_W(i_1), 1.p_1) & \rightarrow & seq(\boldsymbol{prn}(i, \theta(i_{1|p_1})), seq(\texttt{exe}(i_1,p_1), i)) \\
\mid (loop_P(i_1), 1.p_1) & \rightarrow & par(\texttt{exe}(i_1,p_1), i)
\end{array}
$$

6.5. Simplifications

The barebones definition of the pruning prn and execution exe functions makes it so that the empty interaction $\varnothing$ might appear at different positions in the resulting terms. The presence of these $\varnothing$ sub-terms may sometimes be redundant and warrant simplifications which are justified by the semantically sound syntactic transformations from Definition 6. We illustrate this on Figure 13 which further develops on the example from Figure 9.

These more trivial simplifications (such as simplifying occurrences of $\varnothing$) can be directly included in the functional style definitions from Definitions 13 and 15 to simplify interaction terms on-the-fly as they are being built. We denote by $\texttt{prn}_\approx$ and $\texttt{exe}_\approx$ these modified versions of prn and exe which are such that for



(a) Execution of $l_2!m_4$

(b) Result without simplifications

(c) Result with simplifications

Figure 13: Execution with simplifications (example from Figure 9)

any $i \in \mathbb{I}$, any $l \in L$ s.t. $i \downarrow^{\divideontimes} l$ and any $p \in \texttt{frt}(i)$ we have $\texttt{prn}_{\approx}(i,l) \approx_E \texttt{prn}(i,l)$ and $\texttt{exe}_{\approx}(i,p) \approx_E \texttt{exe}(i,p)$.

For instance, for the execution function we may define $\texttt{exe}_{\approx}$ as follows:

- $\texttt{exe}_{\approx}(i,p) =$ **match** (i,p) **with**

$$
\begin{array}{lll}
\mid (act, \varepsilon) & \rightarrow & \varnothing \\
\mid (alt(i_1,i_2), j.p_j) \ \textbf{for } j \in \{1,2\} & \rightarrow & \texttt{exe}_{\approx}(i_j,p_j) \\
\mid (f(i_1,i_2), 1.p_1) \ \textbf{for } f \in \{strict, seq, par\} & \rightarrow & \left\{ \begin{array}{ll} f(\texttt{exe}_{\approx}(i_1,p_1), i_2) & \textbf{if } \texttt{exe}_{\approx}(i_1,p_1) \neq \varnothing \\ i_2 & \textbf{else} \end{array} \right. \\
\mid (strict(i_1,i_2), 2.p_2) & \rightarrow & \texttt{exe}(i_2,p_2) \\
\mid (par(i_1,i_2), 2.p_2) & \rightarrow & \left\{ \begin{array}{ll} par(i_1, \texttt{exe}(i_2,p_2)) & \textbf{if } \texttt{exe}(i_2,p_2) \neq \varnothing \\ i_1 & \textbf{else} \end{array} \right. \\
\ldots & &
\end{array}
$$

6.6. *The execution semantics and its correctness*

The functions `frt` and `exe` ground the execution semantics σ_e given in Definition 16 below.

Definition 16 (Execution semantics)**.** *For any implementation* $\texttt{exe}_{\approx}$ *of the execution function, we define* $\sigma_e : \mathbb{I}_\Omega \rightarrow \mathcal{P}(\mathbb{T}_\Omega)$ *as:*

$$
\sigma_e(i) = empty(i) \cup \bigcup_{p \in \texttt{frt}(i)} i_{|p}.\sigma_e(\texttt{exe}_{\approx}(i,p))
$$

with:

$$
empty(i) = \left\{ \begin{array}{ll} \{\epsilon\} & \textit{if } i \downarrow \\ \emptyset & \textit{else} \end{array} \right.
$$

In the following we prove the equivalence of σ_e and σ_o. A formalisation of the proofs using Coq is available in [23].

Lemma 11 states that every derivation $i \xrightarrow{a} i'$ starting from i that can be inferred with the execution relation can also be inferred using the frontier $\texttt{frt}$ and the execution function $\texttt{exe}$.

Lemma 11. *For any interactions i and i' in $\mathbb{I}_\Omega$, for any action $a \in \mathbb{A}_\Omega$, we have:*

$$\left(\; i \xrightarrow{a} i' \; \right) \Rightarrow \left(\; \exists \, p \in \texttt{frt}(i), \;\; s.t. \; (i_{|p} = a) \;\; \wedge \;\; (i' = \texttt{exe}(i,p)) \; \right)$$

Proof. By induction on the term structure of interactions. See Appendix A. □

The inclusion of σ_o in σ_e, given in Theorem 3 is a simple consequence of Lemma 11.

Theorem 3. *For any interaction $i \in \mathbb{I}_\Omega$:*

$$\sigma_o(i) \subseteq \sigma_e(i)$$

Proof. Let us consider $i \in \mathbb{I}_\Omega$ and $t \in \sigma_o(i)$ and let us reason by induction on the (size of the) trace t.

- If $t = \varepsilon$, the fact that $t = \varepsilon \in \sigma_o(i)$ implies that $i \downarrow$. Then, this implies that $empty(i) = \{\varepsilon\}$ and therefore that $\varepsilon \in \sigma_e(i)$.
- If $t = a.t'$ then, the fact that $a.t' \in \sigma_o(i)$ implies the existence of an action a and an interaction i' such that $i \xrightarrow{a} i'$ and $t' \in \sigma_o(i')$.
  - On the one hand, we can apply the induction hypothesis on t', which implies that $t' \in \sigma_e(i')$
  - On the other hand, we can apply Lemma 11 to reveal the existence of a position $p \in \texttt{frt}(i)$ such that $i_{|p} = a$ and $i' = \texttt{exe}(i,p) \approx_E \texttt{exe}_\approx(i,p)$
  - As a result we have $p \in \texttt{frt}(i)$ and $t' \in \sigma_e(\texttt{exe}_\approx(i,p))$. Therefore, by definition of σ_e we have $t = a.t' = i_{|p}.t' \in \sigma_e(i)$

□

Lemma 12 states that every follow-up $\texttt{exe}(i,p)$ determined using frontier positions from $\texttt{frt}(i)$ can also be inferred using the execution relation $\rightarrow$.

Lemma 12. *For any $i \in \mathbb{I}_\Omega$ and $p \in \texttt{frt}(i)$ we have $i \xrightarrow{i_{|p}} \texttt{exe}(i,p)$*

Proof. By induction on the term structure of interactions. See Appendix A. □

Finally, the inclusion of σ_e in σ_o, given in Theorem 4 follows from Lemma 12.

Theorem 4. *For any interaction $i \in \mathbb{I}_\Omega$:*

$$\sigma_e(i) \subseteq \sigma_o(i)$$

Proof. Let us consider $i \in \mathbb{I}_\Omega$ and $t \in \sigma_e(i)$ and let us reason by induction on the trace t.

- If $t = \varepsilon$, the fact that $t = \varepsilon \in \sigma_e(i)$ implies that $empty(i) = \{\varepsilon\}$ and therefore that $i \downarrow$. Then, the definition of σ_o entails that $\varepsilon \in \sigma_o(i)$.
- If $t = a.t'$ then, the fact that $a.t' \in \sigma_e(i)$ entails the existence of a frontier position $p \in \mathtt{frt}(i)$ such that $i_{|p} = a$ and $t \in \sigma_e(\mathtt{exe}_\approx(i,p))$.
  - On the one hand, given that trace t' is strictly smaller than t, we can apply the induction hypothesis which implies that $t' \in \sigma_o(\mathtt{exe}_\approx(i,p)) = \sigma_o(\mathtt{exe}(i,p))$
  - On the other hand, we can apply Lemma 12 to get that $i \xrightarrow{a} \mathtt{exe}(i,p)$
  - These two facts combined imply that $t = a.t' \in \sigma_o(i)$

$\square$

We have therefore proven both inclusions and can conclude that the execution semantics σ_e is indeed equivalent to the operational-style semantics σ_o and hence that the three semantics presented in this paper are equivalent.

The execution semantics σ_e is used in several other works as a basis to implement algorithms manipulating interactions (e.g., for generating NFA [26] or for analyzing behaviors against specifications written as interactions [29, 28, 27]). It is also implemented in the HIBOU tool [25] for the execution of interactions, the visualization of their semantics and the implementation of the aforementioned algorithms. In this paper, by formalizing σ_e and proving its equivalence to more widely accepted denotational (σ_d) and operational (σ_o) semantics of interactions we therefore formally ground all these approaches.

7. Related works

Semantics of Interaction Languages. Extensive literature concerns the semantics of interaction languages. This is exemplified for UML-SD by the review [33] which underscores the variety of manners with which this language can be understood and interpreted. It is notable that UML-SDs are described semi-formally in the norm [34]. This allows for a rich language with operators such as *assert* or *negate* [10] which are not covered in our IL. However a full formalisation proves difficult, as explained in [33, 10]. The extent to which UML-SDs are formalised may vary [33]: some works formalize loops [21], others do not [16], and some only allow finitely many iterations [40]. In all cases where there are loops, only one loop operator is proposed and corresponds to either $loop_S$ [14, 1, 39, 19], $loop_W$ [40, 15, 37] (none of them being implemented/tooled) or it may be $loop_W$ limited to only contain basic interactions [17]. The semantics found in the literature are mostly given either **(1)** via a translation towards another formalism or **(2)** in denotational-style or operational-style semantics.

Semantics of interactions via translation. Approaches based on translations are the most abundant in the literature. They are also more oriented towards applications and tools, given that the target formalisms generally benefit from a more mature ecosystem and tool support. In [8], a subset of UML-SD is translated towards multivalued nets (a variety of Petri Nets). The proposed translation uses composition to link parts of the Net which correspond to part of the sequence diagram. In [9] a translation from UML-SD with synchronous exchanges to timed automata is proposed. The automata, which are encoded within the UPPAAL tool, are used to verify an implementation of a communication protocol for audiovisual applications. In [4], annotated UML-SD are translated as sets of communicating Timed Input Output Symbolic Transition Systems (TIOSTS). Each TIOSTS keeps track of the region (combined fragment) of the diagram in which it currently is so as to resolve non-determinism (enforcing that different TIOSTS may not take different branches of alternatives). [1] proposes a translation from graphs of basic MSC to Non-Deterministic Finite Automata (NFA). However, this translation is only possible if the edges of the graph are interpreted as strict sequencing (and hence it is not possible to represent a $loop_W$). Indeed, [1] states that the problem, under the asynchronous interpretation, is undecidable. Similarly, [39] translates UML-SD into NFA. However, they use composition, mapping UML-SD operators to NFA operators (e.g., *alt* to union, *loop* to Kleene star). Here, only the strictly sequential loop is handled and weak sequencing can only be used within basic SD (i.e., we cannot weakly sequence more complex SD). [17] proposes a dedicated automaton language to translate UML-SD. In these automata, counters V record how many times lifelines are executed in loops. This allows them to translate weakly sequential loops but in a restricted manner (i.e., these loops can only contain a basic SD).

[14] proposes translating UML-SD into Communicating Sequential Processes (CSP). The local order of events on each lifeline is encoded as a CSP process $\boldsymbol{l}$. Each emission-reception pair is translated to a dedicated CSP process $\boldsymbol{m}$ which enforces that a reception must occur after the corresponding emission. Then, weak sequencing is translated as a generalized parallel composition of the $\boldsymbol{l}$ and $\boldsymbol{m}$ CSP processes, synchronizing on the emission and reception events. However, in [14], the only loop operator that is proposed is the strictly sequential loop (they sequence different instances of the loop with the CSP sequencing operator which requires termination of the first process before the second can be executed). This may be due to the fact that their synchronization mechanism relies on identifying emission-reception pairs via a (`id`,`from`,`to`) tuple which is unique in the diagram. If several instances of a $loop_W$ are active at the same time, without mechanisms to rename identifiers `id`, there would be ambiguity on the instance of the pair on which to synchronize. This signifies that for a single (`id`,`from`,`to`) in a $\boldsymbol{m}$, there may be several matching events in a $\boldsymbol{l}$.

Unlike some other works (e.g., [16, 32, 14]), we do not have a dedicated constructor 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. For broadcasts, we may use patterns of the form $strict(a!m, par(b?m, c?m))$. This has the advantage of isolating emission-reception pairs or broadcast patterns within the term structure of interactions itself and each such pattern can be unambiguously referred to via its position in the term. This allows several distinct instances of the passing of the same message to be executed in parallel without losing track of which instances are ready to receive the given messages. In contrast to that, in approaches that rely on a form of synchronization based on static labels (such as message names), this ambiguity cannot be resolved. In [14] they avoid the issue by using a strictly sequential loop instead of a weakly sequential loop.

Denotational and operational style semantics of Interactions. Denotational semantics found in the literature are often based on partial order sets [40, 15] or algebraic operators on sets of traces [16]. In [29] we have introduced a denotational semantics in the former style which we have then reformulated in the latter style in [30]. This semantics can be seen as an extension of the one from [16] with repetition operators in the form of variants of the algebraic Kleene closure.

Operational semantics of interaction languages are quite rare [33]. One of the most complete and well-known, which is based on the Algebra of Communicating Processes, is found in [37] for Message Sequence Charts. The authors from [19], which provide an alternative based on a theory of agents and environments, mention that [37] is quite complex and thus difficult to implement. Let us remark that in [19] they use a strict sequential loop instead of a weakly sequential one. A simpler presentation than that of [37] is found in [32]. This approach, which inspired ours for the operational semantics, uses a termination predicate and an execution relation. Similarities between [32] and our work include the use of pruning which, in [32], relates to a "permission relation". In [32], loops are not handled and there is no *strict* constructor: direct causal relations between actions occurring on different lifelines (such as that between an emission and the corresponding reception of a message) are handled by maps, updated during executions. Moreover, rules involve negative conditions such as $i \overset{a}{\not\to}$ expressing that it is not possible to find an interaction i' verifying $i \xrightarrow{a} i'$. This approach 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 [37], a loop construction for weak sequencing composition is considered in addition to the constructions discussed in [32]. Rules in [37] 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.

In contrast to [37], we have not included operators and rules related to delayed choice or delayed parallel composition. Indeed, thanks to its equivalence to our denotational semantics, our operational semantics can be combined with syntactic transformations corresponding to distributive properties such as that $alt(seq(a_1, a_2), seq(a_1, a_3)) \approx seq(a_1, alt(a_2, a_3))$. Hence delayed choice (here that of the alternative branch upon observation of a_1) can be handled without a dedicated operator and rule. Such transformations preserve trace semantics but not the bissimulation.

We have introduced our operational semantics in [30] in the style of Plotkin's structural operational semantics [36]. It is defined in the fashion of process algebras [2] and adopts some of the ideas introduced in [32, 37] (which, as a distinct feature, uses maps between sent and received messages) but is closer to usual structural operational semantics. The present paper extends [30] with the definition of a third "execution" semantics which is closer to the implementation in our tool (HIBOU [25]) and the initial formulation in [29]. This semantics separates two concerns intertwined in the operational semantics which correspond to the determination of the "frontier" of immediately executable actions and the computation of "follow-up" interactions which specify continuations of behaviors of an initial interaction after the execution of a specific action.

The present paper also extends [30] via the definition of classes of syntactically distinct but semantically equivalent interactions in which we can find a unique representative. In combination with the "execution" semantics implemented in HIBOU [25], this enabled us to generate NFA with few states (close to the minimal NFA) from interactions [26].

Trace analysis from interactions. Interactions can be used to recognize behaviors that they specify via incremental membership tests (in a similar manner as Finite Automata or Regular Expressions [38] can be used to recognize words of a language). Behaviors of distributed systems can be expressed as global sequences of actions called *traces* [20] (in the case where there is a global clock) or tuples of local traces called *multi-traces* [5] (one per sub-system when there is a local clock per sub-system).

This problem of recognizing a trace specified by a behavioral model is called trace membership or trace analysis. In [29], in order to implement a trace analysis algorithm for interactions, we introduced σ_e. The analysis then tries to re-enact a behavior characterized by a trace t in an interaction i. If $t = \varepsilon$, this consists in verifying whether or not $i \downarrow$. If $t = a.t'$, then it suffices to use the frontier $\texttt{frt}$ to check whether or not a is immediately executable (i.e., if $\exists\ p \in \texttt{frt}(i)\ s.t.\ i_{|p} = a$). If this is not the case, then the trace is not accepted. If this is the case, then the analysis boils down to whether or not any of the follow-ups $i' = \texttt{exe}(i, p)$ for the $p \in \texttt{frt}(i)\ s.t.\ i_{|p} = a$ accepts t. In [28], we used our execution semantics as a reference to prove the correctness of an algorithm dedicated to analysing the conformity of DS executions expressed as multi-traces against interactions. [27] extends [28] to handle "partially observed" multi-traces defined as multi-traces in which some local traces are not observed or their observation stops too early.

In this paper, by proving the equivalence of σ_e to more widely accepted denotational and operational semantics of interactions, we formally ground the approaches developed in [29], [28] and [27] for resp. trace, multi-trace, and partially observed multi-trace analysis. In other words, we bridge the gap between the formal semantics from [30] and the applications developed in [29, 28, 27]. In addition, our reformulation of the execution semantics includes syntactic term simplifications which are proven to be semantically sound. These term simplifications benefit implementations of trace and multi-trace analysis algorithms by keeping the successive interaction terms compact.

The use of frameworks of equivalent semantics. Unifying Theories of Programming (UTP) [11] **(1)** motivates the definition of several equivalent semantics for programming language and **(2)** proposes techniques to derive one from the other and prove their equivalence. It discusses denotational, algebraic and operational semantics as well as their respective advantages and drawbacks. For instance, operational semantics are especially important in the diagnosis of unexpected behavior of a program but since their use requires a fully written program, they are of no use for direct reasoning about specification and design. These discussions can also be made for modeling languages and more particularly sequence diagrams.

In the same spirit, [3] focuses on notions of control flow in various paradigms of programming languages (object-oriented, functional, etc.). It argues that although operational semantics are the more widely used and intuitive manners to define control flow semantics, denotational semantics are a precious assistance for developing logics to reason on programs. A proof of equivalence is of course required to use them both jointly.

[12] proves the equivalence of denotational and operational semantics for a subset of the Hardware Description Language Verilog. Their proof of equivalence justify the use of program transformations and system partitioning for hardware/software co-design.

In the same manner as [12] does for Verilog, we follow UTP [11] principles by deriving a unifying semantics for MSC. The relevance of this approach is justified by applications such as trace and multi-trace analysis [29, 28, 27] which combine aspects of both representations of the semantics for an efficient implementation (see [25]).

8. Conclusion

With this paper we provided a formal basis for an expressive Interaction Language which can be used in a variety of applications thanks to the three semantics which we have defined and proven equivalent (a Coq proof is available in [23]). Particular care has been given to the handling of the weak sequencing operator which is a hallmark of sequence diagrams, as well as loop operators. The denotational-style semantics, which makes use of composition and algebraic operators is well-suited for the definition of semantically-sound transformations which can be used, among others, to simplify interactions and compute canonical forms. The operational-style semantics, through the definition of an execution relation, allows constructing accepted traces via the concatenation of successively executed actions. A functional-style implementation of the operational semantics, which we call the execution semantics can then integrate term transformations justified by the denotational semantics and be used in trace analysis for recognizing accepted behavior in an efficient manner. These three trace semantics are proven equivalent and constitute a formal basis for further results and applications.

References

[1] Alur, R., Yannakakis, M., 1999. Model Checking of Message Sequence Charts, in: Baeten, J.C.M., Mauw, S. (Eds.), CONCUR '99: Concurrency Theory, 10th International Conference, Eindhoven, The Netherlands, August 24-27, 1999, Proceedings, Springer. pp. 114–129. doi:10.1007/3-540-48320-9_10.

[2] Baeten, J., 2000. Process Algebra with Explicit Termination. Computing science reports, Technische Universiteit Eindhoven.

[3] de Bakker, J., de Vink, E., 1996. Control Flow Semantics. MIT Press, Cambridge, MA, USA.

[4] Bannour, B., Gaston, C., Servat, D., 2011. Eliciting Unitary Constraints from Timed Sequence Diagram with Symbolic Techniques: Application to Testing, in: Thu, T.D., Leung, K.R.P.H. (Eds.), 18th Asia Pacific Software Engineering Conference, APSEC 2011, Ho Chi Minh, Vietnam, December 5-8, 2011, IEEE Computer Society. pp. 219–226. doi:10.1109/APSEC.2011.40.

[5] Benharrat, N., Gaston, C., Hierons, R.M., Lapitre, A., Le Gall, P., 2017. Constraint-Based Oracles for Timed Distributed Systems, in: Yevtushenko, N., Cavalli, A.R., Yenigün, H. (Eds.), Testing Software and Systems, Springer International Publishing, Cham. pp. 276–292.

[6] Damm, W., Harel, D., 1998. LSCs: Breathing Life into Message Sequence Charts. Technical Report. ISR.

[7] Dershowitz, N., Jouannaud, J.P., 1991. Rewrite Systems. MIT Press, Cambridge, MA, USA. p. 243–320.

[8] Eichner, C., Fleischhack, H., Meyer, R., Schrimpf, U., Stehno, C., 2005. Compositional Semantics for UML 2.0 Sequence Diagrams Using Petri Nets, in: Prinz, A., Reed, R., Reed, J. (Eds.), SDL 2005: Model Driven, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 133–148.

[9] Firley, T., Huhn, M., Diethers, K., Gehrke, T., Goltz, U., 1999. Timed Sequence Diagrams and Tool-Based Analysis - A Case Study, in: France, R., Rumpe, B. (Eds.), UML'99 — The Unified Modeling Language, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 645–660.

[10] Harel, D., Maoz, S., 2008. Assert and negate revisited: Modal semantics for UML sequence diagrams. Software and Systems Modeling 7, 237–252.

[11] Hoare, C.A.R., Jifeng, H., 1998. Unifying Theories of Programming. Prentice Hall International Series in Computer Science.

[12] Huibiao, Z., Bowen, J.P., Jifeng, H., 2001. From Operational Semantics to Denotational Semantics for Verilog, in: Margaria, T., Melham, T. (Eds.), Correct Hardware Design and Verification Methods, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 449–464.

[13] ITU, 2011. Message Sequence Chart (MSC). itu.int/rec/T-REC-Z.120.

[14] Jacobs, J., Simpson, A., 2015. On a Process Algebraic Representation of Sequence Diagrams, in: Canal, C., Idani, A. (Eds.), Software Engineering and Formal Methods, Springer International Publishing, Cham. pp. 71–85.

[15] Katoen, J.P., Lambert, L., 1998. Pomsets for Message Sequence Charts, pp. 197–207. Formale Beschreibungstechniken fuer verteilte Systeme, 8. GI/ITG-Fachgespraech ; Conference date: 04-06-1998 Through 05-06-1998.

[16] Knapp, A., Mossakowski, T., 2017. UML Interactions Meet State Machines - An Institutional Approach, in: 7th Conf. on Algebra and Coalgebra in Computer Science (CALCO).

[17] Knapp, A., Wuttke, J., 2007. Model Checking of UML 2.0 Interactions, in: Kühne, T. (Ed.), Models in Software Engineering, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 42–51.

[18] Korp, M., Sternagel, C., Zankl, H., Middeldorp, A., 2009. Tyrolean Termination Tool 2, in: Treinen, R. (Ed.), Rewriting Techniques and Applications, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 295–304.

[19] Letichevsky, A., Kapitonova, J., Kotlyarov, V., Volkov, V., Letichevsky, A., Weigert, T., 2005. Semantics of Message Sequence Charts, pp. 117–132. doi:10.1007/11506843_8.

[20] Longuet, D., 2012. Global and Local Testing from Message Sequence Charts, in: Proceedings of the 27th Annual ACM Symposium on Applied Computing, Association for Computing Machinery, New York, NY, USA. p. 1332–1338. doi:10.1145/2245276.2231987.

[21] Lu, L., Kim, D.K., 2014. Required Behavior of Sequence Diagrams: Semantics and Conformance. ACM Trans. Softw. Eng. Methodol. 23. doi:10.1145/2523108.

[22] Mahe, E., a. Coq proof for soundness of equivalence relation for semantically equivalent interactions. erwanm974.github.io/coq_hibou_label_equivalent_terms/. Accessed: 2021-10-14.

[23] Mahe, E., b. Coq proof for the equivalence of the semantics. erwanm974.github.io/coq_hibou_label_semantics_equivalence/. Accessed: 2021-10-14.

[24] Mahe, E., c. Proof of convergence of the TRS using TTT2 and CSI. github.com/erwanM974/hibou_trs_basic. Accessed: 2023-06-19.

[25] Mahe, E., 2022. HIBOU tool. github.com/erwanM974/hibou_label.

[26] Mahe, E., Bannour, B., Gaston, C., Lapitre, A., Gall, P.L., 2023a. A Term-based Approach for Generating Finite Automata from Interaction Diagrams. arXiv:2306.02983.

[27] Mahe, E., Bannour, B., Gaston, C., Lapitre, A., Gall, P.L., 2023b. Interaction-Based Offline Runtime Verification of Distributed Systems, in: Hojjat, H., Ábrahám, E. (Eds.), Fundamentals of Software Engineering, Springer Nature Switzerland, Cham. pp. 88–103.

[28] Mahe, E., Bannour, B., Gaston, C., Lapitre, A., Le Gall, P., 2021. A Small-Step Approach to Multi-Trace Checking against Interactions, in: Proceedings of the 36th Annual ACM Symposium on Applied Computing, Association for Computing Machinery, New York, NY, USA. p. 1815–1822. doi:10.1145/3412841.3442054.

[29] Mahe, E., Gaston, C., Le Gall, P., 2020. Revisiting Semantics of Interactions for Trace Validity Analysis, in: Wehrheim, H., Cabot, J. (Eds.), Fundamental Approaches to Software Engineering, Springer International Publishing, Cham. pp. 482–501.

[30] Mahe, E., Gaston, C., Le Gall, P., 2022. Equivalence of Denotational and Operational Semantics for Interaction Languages, in: Ameur, Y.A., Craciun, F. (Eds.), Theoretical Aspects of Software Engineering - 16th International Symposium, TASE 2022, Cluj-Napoca, Romania, July 8-10, 2022, Proceedings, Springer. pp. 113–130. doi:10.1007/978-3-031-10363-6_8.

[31] Mauw, S., Reniers, M.A., 1997. High-level message sequence charts, in: SDL '97 Time for Testing, SDL, MSC and Trends - 8th International SDL Forum, Proceedings, Elsevier. pp. 291–306.

[32] Mauw, S., Reniers, M.A., 1999. Operational Semantics for MSC'96. Computer Networks 31, 1785–1799. doi:10.1016/S1389-1286(99)00060-2.

[33] Micskei, Z., Waeselynck, H., 2011. The many meanings of UML 2 Sequence Diagrams: a survey. Software & Systems Modeling 10, 489–514.

[34] OMG, 2017. Unified Modeling Language. omg.org/spec/UML/.

[35] Parrow, J., 2001. An Introduction to the π-Calculus, in: Bergstra, J.A., Ponse, A., Smolka, S.A. (Eds.), Handbook of Process Algebra. North-Holland / Elsevier, pp. 479–543.

[36] Plotkin, G., 2004. A Structural Approach to Operational Semantics. The Journal of Logic and Algebraic Programming 60-61, 17–139. doi:10.1016/j.jlap.2004.05.001.

[37] Reniers, M., 1999. Message Sequence Chart : Syntax and Semantics. Ph.D. thesis. Mathematics and Computer Science. doi:10.6100/IR524323.

[38] Roşu, G., Viswanathan, M., 2003. Testing Extended Regular Language Membership Incrementally by Rewriting, in: Nieuwenhuis, R. (Ed.), Rewriting Techniques and Applications, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 499–514.

[39] Simmonds, J., Gan, Y., Chechik, M., Nejati, S., O'Farrell, B., Litani, E., Waterhouse, J., 2009. Runtime Monitoring of Web Service Conversations. IEEE Trans. Serv. Comput. 2, 223–244. doi:10.1109/TSC.2009.16.

[40] Störrle, H., 2003. Semantics of interactions in UML 2.0, in: IEEE Symposium on Human Centric Computing Languages and Environments, 2003. Proceedings. 2003, pp. 129–136. doi:10.1109/HCC.2003.1260216.

[41] Zankl, H., Felgenhauer, B., Middeldorp, A., 2011. CSI – A Confluence Tool, in: Bjørner, N., Sofronie-Stokkermans, V. (Eds.), Automated Deduction – CADE-23, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 499–505.

Appendix A. Detailed proofs

Lemma (Lemma 4). *For any $i \in \mathbb{I}_\Omega$, $(i \downarrow) \Leftrightarrow (\varepsilon \in \sigma_d(i))$*

Proof. Let us reason by induction on the term structure of i.

- If i is of the form $\varnothing$, then we have both $\varnothing \downarrow$ and $\varepsilon \in \sigma_d(\varnothing)$.
- If i belongs to $\mathbb{A}_\Omega$, we have neither $i \downarrow$ nor $\varepsilon \in \sigma_d(i)$.
- If i is of one of the following forms $strict(i_1, i_2)$, $par(i_1, i_2)$ or $seq(i_1, i_2)$ with i_1 and i_2 two sub-interactions that satisfy the induction hypotheses, the reasoning is quite similar for the three operators *strict*, *par* and *seq*. Consider the case of the *strict* operator:
  $\Leftarrow$ Let us suppose that $\varepsilon \in \sigma_d(i)$. By definition of σ_d for the *strict* constructor, there exist $t_1 \in \sigma_d(i_1)$ and $t_2 \in \sigma_d(i_2)$ such that $\varepsilon \in (t_1; t_2)$. This trivially implies that $t_1 = \varepsilon$ and $t_2 = \varepsilon$. By the induction hypotheses, we obtain $i_1 \downarrow$ and $i_2 \downarrow$. This, in turn, implies that $strict(i_1, i_2) \downarrow$.
  $\Rightarrow$ If $strict(i_1, i_2) \downarrow$, then we have both $i_1 \downarrow$ and $i_2 \downarrow$. By the induction hypotheses, we get $\varepsilon \in \sigma_d(i_1)$ and $\varepsilon \in \sigma_d(i_2)$. Therefore $\varepsilon \in \sigma_d(i)$.
- If i is of the form $alt(i_1, i_2)$:
  $\Leftarrow$ The fact that $\varepsilon \in \sigma_d(i)$ entails that either $\varepsilon \in \sigma_d(i_1)$ or $\varepsilon \in \sigma_d(i_2)$ or both. Let us suppose $\varepsilon \in \sigma_d(i_1)$. As per the induction hypothesis, we have $i_1 \downarrow$, which implies that $alt(i_1, i_2) \downarrow$.
  $\Rightarrow$ If $alt(i_1, i_2) \downarrow$, then either $i_1 \downarrow$ or $i_2 \downarrow$ (or both). Let us suppose $i_1 \downarrow$. The induction hypothesis implies that $\varepsilon \in \sigma_d(i_1)$ and hence $\varepsilon \in \sigma_d(i)$.
- If i is of the form $loop_k(i_1)$, with $k \in \{S, H, W, P\}$ we have $i \downarrow$ and $\varepsilon \in \sigma_d(i)$.

□

Lemma (Lemma 5)**.** *For any $l \in L$ and $i \in \mathbb{I}_\Omega$,$(i \downarrow^{\divideontimes} l) \Leftrightarrow (\exists\ t \in \sigma_d(i), \neg(t \divideontimes l))$*

Proof. Given $l \in L$, let us reason by induction on the term structure of i.

- If i is of the form $\varnothing$, then we have $\varnothing \downarrow^{\divideontimes} l$ and $\varepsilon \in \sigma_d(\varnothing)$ satisfies $\neg(\varepsilon \divideontimes l)$.
- If i is an action in $\mathbb{A}_\Omega$ of the form $l'!m$ or $l'?m$ (i.e $\theta(a) = l'$), we have, on the one hand, $a \downarrow^{\divideontimes} l$ iff $l' \neq l$ and on the other hand, a is the only trace of $\sigma_d(i)$ verifying $\neg(a \divideontimes l)$ iff $\theta(a) \neq l$. This gives the result for $i = a$.
- If i is of one of the forms $strict(i_1, i_2)$, $par(i_1, i_2)$ or $seq(i_1, i_2)$ with i_1 and i_2 two sub-interactions that satisfy the induction hypotheses, the reasoning is similar for the three operators. Let us consider the case of *strict*:
$\Rightarrow$ If $i \downarrow^{\divideontimes} l$ then both $i_1 \downarrow^{\divideontimes} l$ and $i_2 \downarrow^{\divideontimes} l$. We can therefore apply the induction hypotheses, which lets us consider two traces $t_1 \in \sigma_d(i_1)$ and $t_2 \in \sigma_d(i_2)$ such that $\neg(t_1 \divideontimes l)$ and $\neg(t_2 \divideontimes l)$. By definition of σ_d, we have $(t_1; t2) \subset \sigma_d(i)$. By distributivity of $\divideontimes$ over ;, this implies that $\exists\ t \in (t_1; t2)$ s.t. $\neg(t \divideontimes l)$ and this trace is in $\sigma_d(i)$.
$\Leftarrow$ Reciprocally, if there exists t in $\sigma_d(strict(i_1, i_2))$ s.t. $\neg(t \divideontimes l)$, then, by definition of σ_d, this entails that there exist two traces $t_1 \in \sigma_d(i_1)$ and $t_2 \in \sigma_d(i_2)$ s.t. $t \in (t_1; t_2)$. Then, we have $\neg(t_1 \divideontimes l)$ and $\neg(t_2 \divideontimes l)$. We can therefore apply the induction hypothesis which implies that $i_1 \downarrow^{\divideontimes} l$ and $i_2 \downarrow^{\divideontimes} l$, which, per the definition of $\downarrow^{\divideontimes}$, implies that $i \downarrow^{\divideontimes} l$.

  For interactions of the form $par(i_1, i_2)$ and $seq(i_1, i_2)$, the reasoning is the same except that we reason on respectively the operators $||$ and $;_{\divideontimes}$ over both of which the conflict $\divideontimes$ is also distributive.
- If $i = alt(i_1, i_2)$:
$\Rightarrow$ If $i \downarrow^{\divideontimes} l$ then either or both $i_1 \downarrow^{\divideontimes} l$ and $i_2 \downarrow^{\divideontimes} l$. Let us suppose we have $i_1 \downarrow^{\divideontimes} l$ (the other case is similar). By the induction hypothesis, we have $\exists\ t_1 \in \sigma_d(i_1)$ s.t. $\neg(t_1 \divideontimes l)$. We then simply observe that $\sigma_d(i_1) \subset \sigma_d(i)$.
$\Leftarrow$ Reciprocally, if there exists t in $\sigma_d(alt(i_1, i_2))$ s.t. $\neg(t \divideontimes l)$, this trace t is in either $\sigma_d(i_1)$ or $\sigma_d(i_2)$. Let us suppose the first case. By the induction hypothesis, this implies that $i_1 \downarrow^{\divideontimes} l$ (the other case is handled similarly). Then, by the definition of $\downarrow^{\divideontimes}$, this implies that $i \downarrow^{\divideontimes} l$.
- If i is of the form $loop_k(i_1)$ with $k \in \{S, H, W, P\}$, by definition, we have $i \downarrow^{\divideontimes} l$ and the empty trace $\varepsilon \in \sigma_d(i)$ verifies $\neg(\varepsilon \divideontimes l)$.

□

Lemma (Lemma 6)**.** *For any $i \in \mathbb{I}_\Omega$ and any $l \in L$, $(i \downarrow^{\divideontimes} l) \Leftrightarrow (\exists!\ i' \in \mathbb{I}_\Omega$ s.t. $i \simeq^{\divideontimes}_l i')$*

Proof. Given $l \in L$, let us reason by induction on the term structure of i.

- If i is of form $\varnothing$, then we have $\varnothing \downarrow^{\divideontimes} l$ and $i' = \varnothing$ is the unique i' s.t. $i \simeq^{\divideontimes}_l i'$

- If i is of form $a \in \mathbb{A}_\Omega$, we have, on one hand, $a \downarrow^{\divideontimes} l$ iff $\theta(a) \neq l$ and on the other hand, provided that $\theta(a) \neq l$, a is the unique term s.t. $a \simeq_l^{\divideontimes} a$. The two conditions are therefore equivalent.

- Let us now suppose that i is of the form $f(i_1, i_2)$, with $f \in \{strict, seq, par\}$, and i_1 and i_2 two sub-interactions that satisfy the induction hypotheses:
  $\Rightarrow$ If $i \downarrow^{\divideontimes} l$ then both $i_1 \downarrow^{\divideontimes} l$ and $i_2 \downarrow^{\divideontimes} l$. We can therefore apply the induction hypotheses, which allow us to consider two interactions i'_1 and i'_2 such that $i_1 \simeq_l^{\divideontimes} i'_1$ and $i_2 \simeq_l^{\divideontimes} i'_2$. Then by Definition 10 we have $f(i_1, i_2) \simeq_l^{\divideontimes} f(i'_1, i'_2)$. Given that this is the only rule which can apply to infer an i' s.t. $f(i_1, i_2) \simeq_l^{\divideontimes} i'$ and given that i'_1 and i'_2 are unique then $i' = f(i'_1, i'_2)$ is also unique.
  $\Leftarrow$ Reciprocally, if there exists an unique interaction i' in $\mathbb{I}_\Omega$ such that $f(i_1, i_2) \simeq_l^{\divideontimes} i'$), then, as per Definition 10 there must exist unique i'_1 and i'_2 such that $i_1 \simeq_l^{\divideontimes} i'_1$ and $i_2 \simeq_l^{\divideontimes} i'_2$. We can then apply the induction hypothesis to obtain that $i_1 \downarrow^{\divideontimes} l$ and $i_2 \downarrow^{\divideontimes} l$ which imply $i \downarrow^{\divideontimes} l$.

- If i is of the form $alt(i_1, i_2)$:
  $\Rightarrow$ If $i \downarrow^{\divideontimes} l$ then (a) either $i_1 \downarrow^{\divideontimes} l$ and $i_2 \downarrow^{\divideontimes} l$, and by induction, we have unique i'_1 and i'_2 such that $i_1 \simeq_l^{\divideontimes} i'_1$ and $i_2 \simeq_l^{\divideontimes} i'_2$ and hence a unique $i' = alt(i'_1, i_2)$ s.t. $i \simeq_l^{\divideontimes} i'$ because the other two rules of the pruning relation on the alternative cannot apply (since we have $\neg(i_2 \not\downarrow^{\divideontimes} l)$ and $\neg(i_1 \not\downarrow^{\divideontimes} l)$); (b) or only one of i_1 and i_2 verifies the evasion predicate. Let us suppose $i_1 \downarrow^{\divideontimes} l$ and $i_2 \not\downarrow^{\divideontimes} l$ (the opposite case is handled similarly), then, we have by induction $i_1 \simeq_l^{\divideontimes} i'_1$, and by Definition 10 $i \simeq_l^{\divideontimes} i'_1$ is unique because the other two rules cannot apply.

  $\Leftarrow$ Reciprocally, let us suppose that there exists an unique i' in $\mathbb{I}_\Omega$ such that $i \simeq_l^{\divideontimes} i'$. We reason by case according to the rules defining $\simeq^{\divideontimes}$: (a) if it is $i \simeq_l^{\divideontimes} alt(i'_1, i'_2)$ then, we have the hypotheses of unique i'_1 and i'_2 s.t. $i_1 \simeq_l^{\divideontimes} i'_1$ and $i_2 \simeq_l^{\divideontimes} i'_2$ which imply, by induction, that $i_1 \downarrow^{\divideontimes} l$ and $i_2 \downarrow^{\divideontimes} l$ and hence $i \downarrow^{\divideontimes} l$ (b) if it is $i \simeq_l^{\divideontimes} i'_1$ then we have a unique i'_1 s.t. $i_1 \simeq_l^{\divideontimes} i'_1$ and the hypothesis that $i_2 \not\downarrow^{\divideontimes} l$. By induction, we have $i_1 \downarrow^{\divideontimes} l$, and because $i = alt(i_1, i_2)$ is an alternative, we have $i \downarrow^{\divideontimes} l$ (c) the third case ($i \simeq_l^{\divideontimes} i'_2$ with $i_1 \not\downarrow^{\divideontimes} l$) is handled similarly.

- If i is of the form $loop_k(i_1)$, with $k \in \{S, H, W, P\}$. We have $i \downarrow^{\divideontimes} l$ and:
  - if $i_1 \downarrow^{\divideontimes} l$ then by induction there is a unique i'_1 s.t. $i_1 \simeq_l^{\divideontimes} i'_1$ and hence a unique $i' = loop_k(i'_1)$ s.t. $i \simeq_l^{\divideontimes} i'$ because the other rule cannot apply given $\neg(i_1 \not\downarrow^{\divideontimes} l)$
  - if $i_1 \not\downarrow^{\divideontimes} l$ then we have $i \simeq_l^{\divideontimes} \varnothing$ and this $i' = \varnothing$ is unique because the other rule cannot apply (otherwise we would have $i_1 \downarrow^{\divideontimes} l$)

□

Lemma (Lemma 7)**.** *For any $l \in L$ and any i and i' from $\mathbb{I}_\Omega$ verifying $i \simeq_l^{\divideontimes} i'$:*

$$\sigma_d(i') = \{t \in \sigma_d(i) \mid \neg(t \divideontimes l)\}$$

Proof. Given $l \in L$, let us reason by induction on the term structure of i.

- For $i = \varnothing$, we have $\varnothing \simeq_l^{\divideontimes} \varnothing$ and $\sigma_d(\varnothing) = \{\varepsilon\}$. ε having no conflict with l, the property holds.

- For $i = a \in \mathbb{A}_\Omega$ with $i \simeq_l^{\divideontimes} i'$, then we have $\theta(i) \neq l$ and $i' = a$. Hence $a \simeq_l^{\divideontimes} a$ and $\sigma_d(a) = \{a\}$ has a single trace with no conflict with l.

- For i of the form $\mathit{strict}(i_1, i_2)$, with i_1 and i_2 two sub-interactions that satisfy induction hypotheses, i.e. such that, given $i_j \simeq_l^{\divideontimes} i'_j$ we have $\sigma_d(i'_j) = \{t_j \in \sigma_d(i_j) \mid \neg(t_j \divideontimes l)\}$ with $j \in \{1,2\}$, by definition of pruning, we have $i \simeq_l^{\divideontimes} \mathit{strict}(i'_1, i'_2)$ and let us denote it by $i' = \mathit{strict}(i'_1, i'_2)$ for short.
$\subset$ If $t \in \sigma_d(i')$ then there exist $t_1 \in \sigma_d(i'_1)$ and $t_2 \in \sigma_d(i'_2)$ s.t. $t \in (t_1; t_2)$. By the induction hypothesis, we have $t_1 \in \sigma_d(i_1)$ and $\neg(t_1 \divideontimes l)$ and $t_2 \in \sigma_d(i_2)$ and $\neg(t_2 \divideontimes l)$. Therefore $t \in \sigma_d(i_1); \sigma_d(i_2) = \sigma_d(i)$, and $\neg(t \divideontimes l)$.
$\supset$ If $t \in \sigma_d(i)$ is s.t. $\neg(t \divideontimes l)$ then this implies the existence of $t_1 \in \sigma_d(i_1)$ and $t_2 \in \sigma_d(i_2)$ s.t. $t \in (t_1; t_2)$. The fact that $\neg(t \divideontimes l)$ implies that both $\neg(t_1 \divideontimes l)$ and $\neg(t_2 \divideontimes l)$. According to the induction hypothesis, $t_1 \in \sigma_d(i'_1)$ and $t_2 \in \sigma_d(i'_2)$. Therefore $(t_1; t_2) \subset \sigma_d(i')$ and hence $t \in \sigma_d(i')$.

- For interactions of the form $\mathit{par}(i_1, i_2)$ and $\mathit{seq}(i_1, i_2)$, the reasoning is the same as for the previous case except that we reason on (respectively) the operators $||$ and $;_{\divideontimes}$ over both of which the conflict $\divideontimes$ is also distributive.

- For i of the form $\mathit{loop}_k(i_1)$ with $k \in \{S, H, W, P\}$, let us note $\diamond$ the corresponding operator on sets of traces, i.e. $\diamond =;$ if $k = S$, $\diamond =;_{\divideontimes}^{\Lsh}$ if $k = H$, $\diamond =;_{\divideontimes}$ if $k = W$ and $\diamond = ||$ if $k = P$. We then have:
  - if $i_1 \not\downarrow^{\divideontimes} l$, then $\mathit{loop}_k(i_1) \simeq_l^{\divideontimes} \varnothing$. As per the reciprocate of Lemma 5, if i_1 does not avoid l, all the traces from $\sigma_d(i_1)$ have conflicts with l. Then, all the traces obtained from merging traces from i_1 have conflicts with l. Therefore ε is the only trace from $\sigma_d(\mathit{loop}_k(i_1))$ which has no conflict with l. Since $\sigma_d(\varnothing) = \{\varepsilon\}$, the property holds.
  - if $i_1 \downarrow^{\divideontimes} l$, then there exists a unique i'_1 such that $i_1 \simeq_l^{\divideontimes} i'_1$ with i'_1 verifying $\sigma_d(i'_1) = \{t_1 \in \sigma_d(i_1) \mid \neg(t_1 \divideontimes l)\}$. Then we have:

$$\begin{array}{rcl} \sigma_d(\mathit{loop}_k(i'_1)) & = & \sigma_d(i'_1)^{\diamond *} \\ & = & \{t \in \sigma_d(i_1) \mid \neg(t \divideontimes l)\}^{\diamond *} \\ & = & \{t \in \sigma_d(i_1)^{\diamond *} \mid \neg(t \divideontimes l)\} \\ & = & \{t \in \sigma_d(\mathit{loop}_k(i_1))) \mid \neg(t \divideontimes l)\} \end{array}$$

    Indeed, the conflict $\divideontimes$ distributes over $\diamond$, and hence any trace obtained from merging traces from i_1 has no conflict with l iff it is obtained from merging traces from i_1 that all have no conflict with l. Therefore, traces that have no conflict with l are precisely those that are obtained from merging traces with no conflicts with l. These traces come from $\mathit{loop}_k(i'_1)$ as per the induction hypothesis. Therefore the property holds.

$\square$

Lemma (Lemma 8)**.** *For any a in $\mathbb{A}_\Omega$, t in $\mathbb{T}_\Omega$ and i, i' in $\mathbb{I}_\Omega$,*

$$\Big(\ (i \xrightarrow{a} i') \wedge (t \in \sigma_d(i'))\ \Big) \Rightarrow (a.t \in \sigma_d(i))$$

Proof. Given i and i' in $\mathbb{I}_\Omega$, a in $\mathbb{A}_\Omega$ and $t \in \mathbb{T}_\Omega$, let us suppose that $i \xrightarrow{a} i'$ and that $t \in \sigma_d(i')$. In order to prove $a.t \in \sigma_d(i)$ let us reason by induction on the rules from Definition 11 that makes the hypothesis $i \xrightarrow{a} i'$ possible.

- When executing $i \in \mathbb{A}_\Omega$, we have $i' = \varnothing$. Then $\sigma_d(i) = \{i\}$ and $\sigma_d(\varnothing) = \{\varepsilon\}$. The property $i.\varepsilon = i \in \sigma_d(i)$ holds.

- When executing an action on the left of an alternative $i = alt(i_1, i_2)$, we have $i' = i'_1$ such that $i_1 \xrightarrow{a} i'_1$. As a result, $i_1 \xrightarrow{a} i'_1$ and $t \in \sigma_d(i'_1)$. By induction hypothesis on i_1, $a.t \in \sigma_d(i_1)$. Given that $\sigma_d(i_1) \subset \sigma_d(i)$, the property holds. The case for executing an action on the right of $i = alt(i_1, i_2)$ is treated similarly.

- When executing an action on the left of $i = par(i_1, i_2)$, we have $i' = par(i'_1, i_2)$ with $i_1 \xrightarrow{a} i'_1$ and $t \in \sigma_d(par(i'_1, i_2))$. By definition of σ_d, there exists $(t'_1, t_2) \in \sigma_d(i'_1) \times \sigma_d(i_2)$ s.t. $t \in (t'_1 || t_2)$. Therefore we have $i_1 \xrightarrow{a} i'_1$ and $t'_1 \in \sigma_d(i'_1)$. By induction hypothesis on i_1, we have $a.t'_1 \in \sigma_d(i_1)$. Given that $\sigma_d(par(i_1, i_2))$ is the union of all the $(t_\alpha || t_\beta)$ with t_α and t_β traces from i_1 and i_2, $(a.t'_1 || t_2) \subset \sigma_d(i)$. In particular, $t \in (t'_1 || t_2)$, so, by definition of the $||$ operator, we have that $a.t \in (a.t'_1 || t_2)$. Executing an action on the right of $i = par(i_1, i_2)$ is treated similarly.

- Executing an action on the left of a *strict* is treated like executing an action on the left of a *par*. When executing an action on the right of $i = strict(i_1, i_2)$, we have $i' = i'_2$ with $i_2 \xrightarrow{a} i'_2$ and $i_1 \downarrow$. Given that $t \in \sigma_d(i'_2)$ and $i_2 \xrightarrow{a} i'_2$, we apply the induction hypothesis on i_2 to obtain that $a.t \in \sigma_d(i_2)$. Given that $\sigma_d(strict(i_1, i_2))$ includes $\sigma_d(i_2)$ when $i_1 \downarrow$, the property holds.

- Executing an action on the left of a *seq* is treated like executing an action on the left of a *par*. When executing an action on the right of $i = seq(i_1, i_2)$, we have $i' = seq(i'_1, i'_2)$ with $i_1 \simeq^{\divideontimes}_{\theta(a)} i'_1$ and $i_2 \xrightarrow{a} i'_2$. Then, $i_1 \downarrow^{\divideontimes} \theta(a)$ is implied by $i_1 \simeq^{\divideontimes}_{\theta(a)} i'_1$. Given that $t \in \sigma_d(i')$, there exists $t_1 \in \sigma_d(i'_1)$ and $t_2 \in \sigma_d(i'_2)$ s.t. $t \in (t_1 ;_{\divideontimes} t_2)$. By the induction hypothesis on i_2, we have $a.t_2 \in \sigma_d(i_2)$. Since $t_1 \in \sigma_d(i'_1)$, Lemma 7 gives that $\neg(t_1 \divideontimes \theta(a))$. As a result, by definition of the *seq* operator, $(t_1 ;_{\divideontimes} a.t_2)$ includes $a.t$. Finally, given that $\sigma_d(i)$ includes all $(t_\alpha ;_{\divideontimes} t_\beta)$ s.t. $t_\alpha \in \sigma_d(i_1)$ and $t_\beta \in \sigma_d(i_2)$, we have, in particular $(t_1 ;_{\divideontimes} a.t_2) \subset \sigma_d(i)$.

- When executing an action a in $i = loop_S(i_1)$, we have $i' = strict(i'_1, loop_S(i_1))$ with $i_1 \xrightarrow{a} i'_1$. We have that $t \in \sigma_d(i')$. Therefore there exists $t_1 \in$

$\sigma_d(i'_1)$ and $t_2 \in \sigma_d(i)$ s.t. $t \in (t_1;t_2)$. We then remark that $i_1 \xrightarrow{a} i'_1$ and $t_1 \in \sigma_d(i'_1)$. Hence we can apply the induction hypothesis on sub-interaction i_1, which implies that $a.t_1 \in \sigma_d(i_1)$. As a result, given that $t_2 \in \sigma_d(loop_S(i_1)) = \sigma_d(i_1)^{;*}$, and $a.t_1 \in \sigma_d(i_1)$, we have, $(a.t_1;t_2) \subset \sigma_d(i_1)^{;*}$ i.e. $(a.t_1;t_2) \subset \sigma_d(i)$. Then, given that $t \in (t_1;t_2)$, we have immediately that $a.t \in (a.t_1;t_2)$ because it is always possible to add actions from the left. Therefore $a.t \in \sigma_d(i)$, so the property holds.

- The cases for $loop_H$ and $loop_P$ are treated like the one for $loop_S$.

- When executing a from $i = loop_W(i_1)$, we have $i' = seq(i'_0, seq(i'_1, loop_W(i_1)))$ with $i_1 \xrightarrow{a} i'_1$ and $i \simeq^{\divideontimes}_{\theta(a)} i'_0$. Given that $t \in \sigma_d(i')$, there exists $t_0 \in \sigma_d(i'_0)$, $t_1 \in \sigma_d(i'_1)$ and $t_2 \in \sigma_d(i)$ s.t. $t \in (t_0;_{\divideontimes} t_1;_{\divideontimes} t_2)$. Given that $i \simeq^{\divideontimes}_{\theta(a)} i'_0$ we have that $\sigma_d(i'_0) \subset \sigma_d(i)$ and, given that $\sigma_d(i) = \sigma_d(i_1)^{;_{\divideontimes}*}$, we have that $\sigma_d(i);_{\divideontimes} \sigma_d(i) \subset \sigma_d(i)$ and therefore $\sigma_d(i'_0);_{\divideontimes} \sigma_d(i) \subset \sigma_d(i)$. Given that $t_0 \in \sigma_d(i'_0)$, per Lemma 7, $\neg(t_0 \divideontimes \theta(a))$ i.e. no action contained in t_0 occurs on $\theta(a)$. Therefore, if we consider any t_β and any $t_\alpha \in (t_0;_{\divideontimes} t_\beta)$, we have that $a.t_\alpha \in (t_0;_{\divideontimes} a.t_\beta)$ because action a can be taken from the right-hand-side of $(t_0;_{\divideontimes} a.t_\beta)$. With $i_1 \xrightarrow{a} i'_1$ and $t_1 \in \sigma_d(i'_1)$, we can apply the induction hypothesis on sub-interaction i_1, which implies that $a.t_1 \in \sigma_d(i_1)$. Given that $t_2 \in \sigma_d(loop_W(i_1)) = \sigma_d(i_1)^{;_{\divideontimes}*}$, and $a.t_1 \in \sigma_d(i_1)$, we have, $(a.t_1;_{\divideontimes} t_2) \subset \sigma_d(i_1)^{;_{\divideontimes}*}$ i.e. $(a.t_1;_{\divideontimes} t_2) \subset \sigma_d(i)$. Finally, given $a.t \in (t_0;_{\divideontimes} a.t_1;_{\divideontimes} t_2) \subset (\sigma_d(i'_0);_{\divideontimes} \sigma_d(i)) \subset \sigma_d(i)$, the property holds.

□

Lemma (Lemma 9)**.** *For any $a \in \mathbb{A}_\Omega$, $t \in \mathbb{T}_\Omega$ and $i \in \mathbb{I}_\Omega$,*

$$(a.t \in \sigma_d(i)) \Rightarrow \big(\exists\ i' \in \mathbb{I}_\Omega, \quad (i \xrightarrow{a} i') \wedge (t \in \sigma_d(i'))\ \big)$$

Proof. Let us consider $i \in \mathbb{I}_\Omega$, $a \in \mathbb{A}_\Omega$ and $t \in \mathbb{T}_\Omega$. Let us suppose that $a.t \in \sigma_d(i)$ and let us reason by induction on the term structure of i.

- We cannot have $i = \varnothing$ because it contradicts $a.t \in \sigma_d(i)$

- For $i \in \mathbb{A}_\Omega$, $a.t \in \sigma_d(i)$ implies that $i = a$ and $t = \varepsilon$. We then have the existence of $i' = \varnothing$ which indeed satisfies $a \xrightarrow{a} \varnothing$ and $\varepsilon \in \sigma_d(\varnothing)$

- For $i = alt(i_1, i_2)$, then $a.t \in \sigma_d(i)$ implies either $a.t \in \sigma_d(i_1)$ or $a.t \in \sigma_d(i_2)$. Let us suppose it is the first case (the second is identical). Then, by the induction hypothesis on i_1, there exists i'_1 such that $i_1 \xrightarrow{a} i'_1$ and $t \in \sigma_d(i'_1)$. By definition of the relation $\rightarrow$, this implies that $alt(i_1, i_2) \xrightarrow{a} i'_1$. As a result, we have identified $i' = i'_1$ which satisfies the property.

- For $i = par(i_1, i_2)$, then $a.t \in \sigma_d(i)$ implies the existence of traces t_1 and t_2 such that $t_1 \in \sigma_d(i_1)$ and $t_2 \in \sigma_d(i_2)$ and $a.t \in (t_1||t_2)$. This then implies either **(1)** that t_1 is of the form $a.t'_1$ and $t \in (t'_1||t_2)$ or **(2)** that t_2 is of the form $a.t'_2$ and $t \in (t_1||t'_2)$. As both case can be treated identically, let us suppose it is the first case. Given that we have $a.t'_1 \in \sigma_d(i_1)$, by induction hypothesis on i_1, there exists i'_1 such that $i_1 \xrightarrow{a} i'_1$ and $t'_1 \in \sigma_d(i'_1)$. By definition of $\rightarrow$, $par(i_1, i_2) \xrightarrow{a} par(i'_1, i_2)$. By definition of σ_d, given that $t'_1 \in \sigma_d(i'_1)$ and $t_2 \in \sigma_d(i_2)$, we have $(t'_1 || t_2) \subset \sigma_d(par(i'_1, i_2))$. Then, given that $t \in (t'_1 || t_2)$, this implies that $t \in \sigma_d(par(i'_1, i_2))$. We have identified $i' = par(i'_1, i_2)$ which satisfies the property.

- For $i = strict(i_1, i_2)$ then $a.t \in \sigma_d(i)$ implies the existence of traces t_1 and t_2 such that $t_1 \in \sigma_d(i_1)$ and $t_2 \in \sigma_d(i_2)$ and $a.t \in (t_1; t_2)$. This then implies: either **(1)** that t_1 is of the form $a.t'_1$ and $t \in (t'_1; t_2)$. This case is similar to that for par and $||$. or **(2)** that $t_1 = \varepsilon$ and $t_2 = a.t$. On the one hand, the fact that $t_1 = \varepsilon \in \sigma_d(i_1)$ implies, as per Lemma 4, that $i_1 \downarrow$. On the other hand, with $t_2 = a.t \in \sigma_d(i_2)$, the induction hypothesis on i_2 establishes the existence of i'_2 s.t. $i_2 \xrightarrow{a} i'_2$ and $t \in \sigma_d(i'_2)$. Because $i_1 \downarrow$, this implies that $strict(i_1, i_2) \xrightarrow{a} i'_2$. As a result, we have identified $i' = i'_2$ which satisfies the property.

- For $i = seq(i_1, i_2)$, $a.t \in \sigma_d(i)$ implies the existence of traces t_1 and t_2 such that $t_1 \in \sigma_d(i_1)$ and $t_2 \in \sigma_d(i_2)$ and $a.t \in (t_1 ;_{\divideontimes} t_2)$. This then implies either **(1)** that $t_1 = a.t'_1$ and $t \in (t'_1 ;_{\divideontimes} t_2)$, similar case to that for par and $||$, or **(2)** that $\neg(t_1 \divideontimes \theta(a))$ and that t_2 is of the form $a.t'_2$ with $t \in (t_1 ;_{\divideontimes} t'_2)$. On the one hand, $t_1 \in \sigma_d(i_1)$ together $\neg(t_1 \divideontimes \theta(a))$, ensures, as per Lemma 5, that $i_1 \downarrow^{\divideontimes} \theta(a)$ and therefore, as per Lemma 6, that there exists a unique i'_1 such that $i_1 \simeq^{\divideontimes}_{\theta(a)} i'_1$. On the other hand, with $t_2 = a.t'_2 \in \sigma_d(i_2)$, the induction hypothesis on i_2 establishes the existence of i'_2 s.t. $i_2 \xrightarrow{a} i'_2$ and $t'_2 \in \sigma_d(i'_2)$. By definition of $\rightarrow$, this implies that $seq(i_1, i_2) \xrightarrow{a} seq(i'_1, i'_2)$. Given that $t_1 \in \sigma_d(i_1)$ is such that $\neg(t_1 \divideontimes \theta(a))$, as per Lemma 7, this implies that $t_1 \in \sigma_d(i'_1)$. By definition of σ_d, given that $t_1 \in \sigma_d(i'_1)$ and $t'_2 \in \sigma_d(i'_2)$, we have $(t_1 ;_{\divideontimes} t'_2) \subset \sigma_d(seq(i'_1, i'_2))$. Then, given that $t \in (t_1 ;_{\divideontimes} t'_2)$, $t \in \sigma_d(seq(i'_1, i'_2))$. We therefore have identified $i' = seq(i'_1, i'_2)$ which satisfies the property.

- For $i = loop_S(i_1)$ the fact that $a.t \in \sigma_d(i)$ implies that $a.t \in \sigma_d(i_1)^{;*}$ and hence, as per Lemma 1, there exists a trace t' such that $a.t' \in \sigma_d(i_1)$ and $t \in \{t'\}; \sigma_d(i_1)^{;*}$. Then, with $a.t' \in \sigma_d(i_1)$, the induction hypothesis on i_1 gives us an interaction i'_1 such that $i_1 \xrightarrow{a} i'_1$ and $t' \in \sigma_d(i'_1)$. By definition of $\rightarrow$, $loop_S(i_1) \xrightarrow{a} strict(i'_1, loop_S(i_1))$. By definition of σ_d, given that $t' \in \sigma_d(i'_1)$ and that $t \in \{t'\}; \sigma_d(i_1)^{;*}$, we have $t \in \sigma_d(strict(i'_1, loop_S(i_1)))$ and $i' = strict(i'_1, loop_S(i_1))$ satisfies the property.

- The case for $loop_P$ is treated like the case for $loop_S$.

- For $i = loop_H(i_1)$, $a.t \in \sigma_d(i)$ entails $a.t \in \sigma_d(i_1)^{;^{\Lsh}_{\divideontimes} *}$. By definition, there exists $j > 0$ such that $a.t \in \sigma_d(i_1)^{;^{\Lsh}_{\divideontimes} j} = \sigma_d(i_1) ;^{\Lsh}_{\divideontimes} \sigma_d(i_1)^{;^{\Lsh}_{\divideontimes} (j-1)} \subset \sigma_d(i_1) ;^{\Lsh}_{\divideontimes} \sigma_d(i_1)^{;^{\Lsh}_{\divideontimes} *}$. Because the restricted operator $;^{\Lsh}_{\divideontimes}$ only allows to take the first action of recomposed traces from the left-hand-side, we can identify a trace t' s.t. $a.t' \in \sigma_d(i_1)$ and $t \in \{t'\};_{\divideontimes} \sigma_d(i_1)^{\overset{\Lsh}{;_{\divideontimes}}*}$. Hence, the induction hypothesis on i_1 establishes the existence of i'_1 such that $i_1 \xrightarrow{a} i'_1$ and $t' \in \sigma_d(i'_1)$. By definition of $\rightarrow$, this implies that $loop_H(i_1) \xrightarrow{a} seq(i'_1, loop_H(i_1))$. By definition of σ_d, given that $t' \in \sigma_d(i'_1)$ and that $t \in \{t'\};_{\divideontimes} \sigma_d(i_1)^{\overset{\Lsh}{;_{\divideontimes}}*}$, we have that $t \in \sigma_d(seq(i'_1, loop_H(i_1)))$. We have identified $i' = seq(i'_1, loop_H(i_1))$ which satisfies the property.

- For $i = loop_W(i_1)$, $a.t \in \sigma_d(i)$ entails $a.t \in \sigma_d(i_1)^{;_{\divideontimes}*}$. There exists $n > 0$ such that $a.t \in \sigma_d(i_1)^{;_{\divideontimes}n}$. As a result, we can identify traces t_1 through t_n such that for any $j \in [1, n]$, $t_j \in \sigma_d(i_1)$ and $a.t \in \{t_1\};_{\divideontimes} \cdots ;_{\divideontimes} \{t_n\}$. By definition of the seq operator, action a is taken from a certain t_j with $j \in [1, n]$ and $t_j = a.t'_j$ and we have, for any $k < j$, $\neg(t_k \divideontimes \theta(a))$ and $t \in \{t_1\};_{\divideontimes} \cdots ;_{\divideontimes} \{t_{j-1}\};_{\divideontimes} \{t'_j\};_{\divideontimes} \{t_{j+1}\};_{\divideontimes} \cdots ;_{\divideontimes} \{t_n\}$. Considering i'_0 s.t. $loop_W(i_1) \simeq^{\divideontimes}_{\theta(a)} i'_0$ (which existence is guaranteed by Lemma 6), because, for any $k < j$, we have $\neg(t_k \divideontimes \theta(a))$ then, as per Lemma 7, for all $k < j$, we have $t_k \in \sigma_d(i'_0)$. Then, either **(1)** all the t_k are ε, then $\{t_1\};_{\divideontimes} \cdots ;_{\divideontimes} \{t_{j-1}\} = \{\varepsilon\} \subset \sigma_d(i'_0)$ or **(2)** at least one t_k is not the empty trace then i'_0 is a non empty $loop_W$, and, given that for all $k < j$, we have $t_k \in \sigma_d(i'_0)$ then $\{t_1\};_{\divideontimes} \cdots ;_{\divideontimes} \{t_{j-1}\} \subset \sigma_d(i'_0)$ (because a $loop_W$ is closed under repetition by $;_{\divideontimes}$). Hence $\{t_1\};_{\divideontimes} \cdots ;_{\divideontimes} \{t_{j-1}\} \subset \sigma_d(i'_0)$ and therefore $t \in \sigma_d(i'_0);_{\divideontimes} \{t'_j\};_{\divideontimes} \{t_{j+1}\};_{\divideontimes} \cdots ;_{\divideontimes} \{t_n\}$. Given that, for $k > j$ we have that $t_k \in \sigma_d(i)$ and because $i = loop_W(i_1)$, then $\{t_{j+1}\};_{\divideontimes} \cdots ;_{\divideontimes} \{t_n\} \subset \sigma_d(i)$ and therefore $t \in \sigma_d(i'_0);_{\divideontimes} \{t'_j\};_{\divideontimes} \sigma_d(i)$. Given that $t_j = a.t'_j \in \sigma_d(i_1)$, by the induction hypothesis on i_1, there exists i'_1 such that $i_1 \xrightarrow{a} i'_1$ and $t'_j \in \sigma_d(i'_1)$. By definition of $\rightarrow$, $loop_W(i_1) \xrightarrow{a} seq(i'_0, seq(i'_1, loop_W(i_1)))$. Finally, given that we have shown that $t \in \sigma_d(i'_0);_{\divideontimes} \{t'_j\};_{\divideontimes} \sigma_d(i)$ and because $t'_j \in \sigma_d(i'_1)$, by definition of σ_d we have $t \in \sigma_d(seq(i'_0, seq(i'_1, loop_W(i_1))))$. Therefore we have identified $i' = seq(i'_0, seq(i'_1, loop_W(i_1)))$ which satisfies the property.

$\square$

Lemma (Lemma 10)**.** *For any $i \in \mathbb{I}_\Omega$ and any $l \in L$:*

$$(\exists\ i' \in \mathbb{I}_\Omega,\ s.t.\ i \simeq^{\divideontimes}_l i') \Rightarrow (\mathtt{prn}(i, l) = i') \qquad and \qquad (i \downarrow^{\divideontimes} l) \Rightarrow (i \simeq^{\divideontimes}_l \mathtt{prn}(i, l))$$

Proof. Let us consider a certain $l \in L$ and reason by induction on i:

If $i = \varnothing$ then the only possible i' s.t. $\varnothing \simeq^{\divideontimes}_l i'$ is $\varnothing$ and by definition $\varnothing \downarrow^{\divideontimes} l$ and $\mathtt{prn}(\varnothing, l) = \varnothing$.

If $i = a \in \mathbb{A}_\Omega$ then, for pruning to be possible (in both definitions) we must have $\theta(a) \neq l$ and we then have $a \simeq^{\divideontimes}_l a$, $a \downarrow^{\divideontimes} l$ and $\mathtt{prn}(a, l) = a$.

If $i = alt(i_1, i_2)$ then (using Lemma 6):

- if $i_1 \downarrow^{\divideontimes} l$ and $\neg i_2 \downarrow^{\divideontimes} l$ then there exists i'_1 s.t. $i_1 \simeq^{\divideontimes}_l i'_1$ and $i \simeq^{\divideontimes}_l i'_1$ and we have $i \downarrow^{\divideontimes} l$ and $\mathtt{prn}(i, l) = \mathtt{prn}(i_1, l)$. By the induction hypothesis we have $i'_1 = \mathtt{prn}(i_1, l)$ and hence $i \simeq^{\divideontimes}_l \mathtt{prn}(i, l)$.

- the case if $\neg i_1 \downarrow^{\divideontimes} l$ and $i_2 \downarrow^{\divideontimes} l$ can be treated similarly.
- if both $i_1 \downarrow^{\divideontimes} l$ and $i_2 \downarrow^{\divideontimes} l$ then $\forall\, j \in \{1,2\}$, there exists i'_j s.t. $i_j \simeq_l^{\divideontimes} i'_j$ and $i \simeq_l^{\divideontimes} alt(i'_1, i'_2)$ and we have $i \downarrow^{\divideontimes} l$ and $\mathtt{prn}(i,l) = alt(\mathtt{prn}(i_1,l), \mathtt{prn}(i_2,l))$. By the induction hypothesis we have $\forall\, j \in \{1,2\}$, $i'_j = \mathtt{prn}(i_j,l)$ and hence $i \simeq_l^{\divideontimes} \mathtt{prn}(i,l)$.
- the case $\neg i_1 \downarrow^{\divideontimes} l$ and $\neg i_2 \downarrow^{\divideontimes} l$ implies that $\neg i \downarrow^{\divideontimes} l$ and that there are no i' s.t. $i \simeq_l^{\divideontimes} i'$.

If $i = f(i_1, i_2)$ with $f \in \{strict, seq, par\}$ then (using Lemma 6):

- if both $i_1 \downarrow^{\divideontimes} l$ and $i_2 \downarrow^{\divideontimes} l$ then $\forall\, j \in \{1,2\}$, there exists i'_j s.t. $i_j \simeq_l^{\divideontimes} i'_j$ and $i \simeq_l^{\divideontimes} f(i'_1, i'_2)$ and we have $i \downarrow^{\divideontimes} l$ and $\mathtt{prn}(i,l) = f(\mathtt{prn}(i_1,l), \mathtt{prn}(i_2,l))$. By the induction hypothesis we have $\forall\, j \in \{1,2\}$, $i'_j = \mathtt{prn}(i_j,l)$ and hence $i \simeq_l^{\divideontimes} \mathtt{prn}(i,l)$.
- if this is not the case then $\neg i \downarrow^{\divideontimes} l$ and there are no i' s.t. $i \simeq_l^{\divideontimes} i'$.

If $i = loop_k(i_1)$ with $k \in \{S, H, W, P\}$ then (using Lemma 6):

- if $i_1 \downarrow^{\divideontimes} l$ then there exists i'_1 s.t. $i_1 \simeq_l^{\divideontimes} i'_1$ and $i \simeq_l^{\divideontimes} loop_k(i'_1)$ and we have $i \downarrow^{\divideontimes} l$ and $\mathtt{prn}(i,l) = loop_k(\mathtt{prn}(i_1,l))$. By the induction hypothesis we have $i'_1 = \mathtt{prn}(i_1,l)$ and hence $i \simeq_l^{\divideontimes} \mathtt{prn}(i,l)$.
- else, even though $\neg i_1 \downarrow^{\divideontimes} l$ we still have $i \downarrow^{\divideontimes} l$ because i has a loop at its root. Also, we have both $i \simeq_l^{\divideontimes} \varnothing$ and $\mathtt{prn}(i,l) = \varnothing$ and thus $i \simeq_l^{\divideontimes} \mathtt{prn}(i,l)$.

□

Lemma (Lemma 11). *For any interactions i and i' in $\mathbb{I}_\Omega$, for any action $a \in \mathbb{A}_\Omega$, we have:*

$$\Big(\ i \xrightarrow{a} i'\ \Big) \Rightarrow \Big(\ \exists\, p \in \mathtt{frt}(i),\ s.t.\ (i_{|p} = a)\ \wedge\ (i' = \mathtt{exe}(i,p))\ \Big)$$

Proof. Let us suppose that $i \xrightarrow{a} i'$ and let us reason by induction on the structure of i.

- If $i = \varnothing$ then it is not possible to have an i' such that $i \xrightarrow{a} i'$
- If $i = a \in \mathbb{A}_\Omega$ then: **(1)** the only possible i' such that $i \xrightarrow{a} i'$ is $\varnothing$ i.e. we have $a \xrightarrow{a} \varnothing$. And **(2)** we have $\varepsilon \in \mathtt{frt}(a)$ and $a_{|\varepsilon} = a$ and $\mathtt{exe}(a, \varepsilon) = \varnothing$. Therefore the property holds.
- If $i = strict(i_1, i_2)$ then $i \xrightarrow{a} i'$ implies:
  - either that there exists i'_1 such that $i_1 \xrightarrow{a} i'_1$ and $i' = strict(i'_1, i_2)$. We can then apply the induction hypothesis on sub-interaction i_1 to obtain the existence of a position $p_1 \in \mathtt{frt}(i_1)$ such that $i_{1|p_1} = a$ and $i'_1 = \mathtt{exe}(i_1, p_1)$. Then, by construction $1.p_1 \in \mathtt{frt}(i)$ and $i_{|1.p_1} = i_{1|p_1} = a$ and $\mathtt{exe}(i, 1.p_1) = strict(\mathtt{exe}(i_1, p_1), i_2) = strict(i'_1, i_2) = i'$.

- or that we have $i_1 \downarrow$ and that there exists i'_2 such that $i_2 \xrightarrow{a} i'_2$ and $i' = i'_2$. We can then apply the induction hypothesis on sub-interaction i_2 to obtain the existence of a position $p_2 \in \mathtt{frt}(i_2)$ such that $i_{2|p_2} = a$ and $i'_2 = \mathtt{exe}(i_2, p_2)$. Then, given that $i_1 \downarrow$, we have by construction that $2.p_2 \in \mathtt{frt}(i)$ and $i_{|2.p_2} = i_{2|p_2} = a$ and $\mathtt{exe}(i, 2.p_2) = \mathtt{exe}(i_2, p_2) = i'_2 = i'$.

- If $i = seq(i_1, i_2)$ then $i \xrightarrow{a} i'$ implies:
  - either that there exists i'_1 such that $i_1 \xrightarrow{a} i'_1$ and $i' = seq(i'_1, i_2)$. This case is similar to that of *strict*.
  - or that we have $i_1 \downarrow^{\divideontimes} \theta(a)$ and that there exists i'_2 such that $i_2 \xrightarrow{a} i'_2$ and $i' = seq(\mathtt{prn}(i_1, \theta(a)), i'_2)$. We can then apply the induction hypothesis on sub-interaction i_2 to obtain the existence of a position $p_2 \in \mathtt{frt}(i_2)$ such that $i_{2|p_2} = a$ and $i'_2 = \mathtt{exe}(i_2, p_2)$ and $i_{|2.p_2} = i_{2|p_2} = a$. Then, given that $i_1 \downarrow^{\divideontimes} \theta(i_{|2.p_2})$, we have by construction that $2.p_2 \in \mathtt{frt}(i)$ and $i_{|2.p_2} = i_{2|p_2} = a$ and $\mathtt{exe}(i, 2.p_2) = seq(\mathtt{prn}(i_1, \theta(i_{2|p_2})), \mathtt{exe}(i_2, p_2)) = seq(\mathtt{prn}(i_1, \theta(a)), i'_2) = i'$.
- The case $i = par(i_1, i_2)$ can be treated as the left side of *strict*.
- The case $i = alt(i_1, i_2)$ is a simpler variant of *par*.
- If $i = loop_k(i_1)$ with $k \in \{S, H, W, P\}$, then $i \xrightarrow{a} i'$ implies that there exists i'_1 such that $i_1 \xrightarrow{a} i'_1$ and i' is either $strict(i'_1, i)$ or $seq(i'_1, i)$ or $seq(\mathtt{prn}(i, \theta(a)), seq(i'_1, i))$ (as per Lemma 10) or $par(i'_1, i)$. In any case, we can apply the induction hypothesis on sub-interaction i_1 to obtain the existence of a position $p_1 \in \mathtt{frt}(i_1)$ such that $i_{1|p_1} = a$ and $i'_1 = \mathtt{exe}(i_1, p_1)$. Then, by construction $1.p_1 \in \mathtt{frt}(i)$ and $i_{|1.p_1} = i_{1|p_1} = a$ and $\mathtt{exe}(i, 1.p_1) = i'$.

□

Lemma (Lemma 12)**.** *For any* $i \in \mathbb{I}_\Omega$ *and* $p \in \mathtt{frt}(i)$ *we have* $i \xrightarrow{i_{|p}} \mathtt{exe}(i, p)$

Proof. Let us suppose that $p \in \mathtt{frt}(i)$ and then reason by induction on the structure of i:

- If $i = \varnothing$ then we have $\mathtt{frt}(\varnothing) = \emptyset$ so it is not possible to have a $p \in \mathtt{frt}(i)$.
- If $i = a \in \mathbb{A}_\Omega$ then we have $\mathtt{frt}(a) = \{\varepsilon\}$ and p must be ε. Then because $a_{|\varepsilon} = a$, $a \xrightarrow{a} \varnothing$ and $\mathtt{exe}(a, \varepsilon) = \varnothing$, the property holds.
- If $i = strict(i_1, i_2)$ then $p \in \mathtt{frt}(i)$ implies:
  - either that p is of the form $1.p_1$ and $p_1 \in \mathtt{frt}(i_1)$. Also we have $i_{1|p_1} = i_{|p}$. We can then apply the induction hypothesis on sub-interaction i_1 to get that we have $i_1 \xrightarrow{i_{1|p_1}} \mathtt{exe}(i_1, p_1)$. This implies,

as per the definition of the execution relation, that $strict(i_1, i_2) \xrightarrow{i_{1|p_1}} strict(\texttt{exe}(i_1, p_1), i_2)$. Then, given that $\texttt{exe}(i, p) = strict(\texttt{exe}(i_1, p_1), i_2)$ this equates to $i \xrightarrow{i_{|p}} \texttt{exe}(i, p)$.

  - or that we have $i_1 \downarrow$, that p is of the form $2.p_2$ and that $p_2 \in \texttt{frt}(i_2)$. Also we have $i_{2|p_2} = i_{|p}$. We can then apply the induction hypothesis on sub-interaction i_2 to get that we have $i_2 \xrightarrow{i_{2|p_2}} \texttt{exe}(i_2, p_2)$. Given that we have $i_1 \downarrow$, this implies, as per the definition of the execution relation, that $strict(i_1, i_2) \xrightarrow{i_{2|p_2}} \texttt{exe}(i_2, p_2)$. Then, given that $\texttt{exe}(i, p) = \texttt{exe}(i_2, p_2)$ this equates to $i \xrightarrow{i_{|p}} \texttt{exe}(i, p)$.

- If $i = seq(i_1, i_2)$ then $p \in \texttt{frt}(i)$ implies:
  - either that p is of the form $1.p_1$ and $p_1 \in \texttt{frt}(i_1)$. This can be treated as above.
  - or that we have $i_1 \downarrow^{\divideontimes} \theta(i_{|p})$, that p is of the form $2.p_2$ and that $p_2 \in \texttt{frt}(i_2)$. Also we have $i_{2|p_2} = i_{|p}$. We can then apply the induction hypothesis on sub-interaction i_2 to get that we have $i_2 \xrightarrow{i_{2|p_2}} \texttt{exe}(i_2, p_2)$. Given that $i_1 \downarrow \theta(i_{|p})$, this implies, as per the definition of the relation $\rightarrow$, that $seq(i_1, i_2) \xrightarrow{i_{|p}} seq(\texttt{prn}(i_1, \theta(i_{|p})), \texttt{exe}(i_2, p_2))$. Then, given that $\texttt{exe}(i, p) = seq(\texttt{prn}(i_1, \theta(i_{|p})), \texttt{exe}(i_2, p_2))$ this equates to $i \xrightarrow{i_{|p}} \texttt{exe}(i, p)$.

- The case $i = par(i_1, i_2)$ can be treated as the left side of *strict*.

- The case $i = alt(i_1, i_2)$ can be treated similarly to *par*.

- If $i = loop_k(i_1)$ with $k \in \{S, H, W, P\}$ then $p \in \texttt{frt}(i)$ implies that p is of the form $1.p_1$ and $p_1 \in \texttt{frt}(i_1)$. Also we have $i_{1|p_1} = i_{|p}$. We can apply the induction hypothesis on sub-interaction i_1 to get that we have $i_1 \xrightarrow{i_{|p}} \texttt{exe}(i_1, p_1)$. Let us then denote by i'_1 the interaction $\texttt{exe}(i_1, p_1)$. This then implies, as per the definition of the execution relation, that $loop_k(i_1) \xrightarrow{i_{|p}} i'$ with i' being either $strict(i'_1, i)$ (if $k = S$), $seq(i'_1, i)$ (if $k = H$), $seq(\texttt{prn}(i, \theta(i_{|p})), seq(i'_1, i))$ (if $k = W$, the existence of $\texttt{prn}(i, \theta(i_{|p}))$ being proved by Lemma 10) or $par(i'_1, i)$ (if $k = P$). Then, in any case, given that $\texttt{exe}(i, 1.p_1)$ is exactly defined as i', this equates to $i \xrightarrow{i_{|p}} \texttt{exe}(i, p)$.

$\square$