Finite Automata synthesis from interactions

Erwan Mahe, Boutheina Bannour, Christophe Gaston, Arnault Lapitre, Pascale Le Gall

Finite Automata synthesis from interactions

Erwan Mahe
Boutheina Bannour
X.Y@cea.fr
Université Paris-Saclay, CEA, List
Palaiseau, France

Christophe Gaston
Arnault Lapitre
X.Y@cea.fr
Université Paris-Saclay, CEA, List
Palaiseau, France

Pascale Le Gall
pascale.legall@centralesupelec.fr
Université Paris-Saclay,
CentraleSupélec
Gif-sur-Yvette, France

ABSTRACT

Interactions are graphical models representing communication flows between actors. Well-known interaction languages include UML Sequence Diagrams or Message Sequence Charts. Even though interactions allow for concise and intuitive specifications, their use remains limited in formal verification, partly because the subsets of formalized languages often lack expressiveness. As many verification methods, such as model-checking or runtime verification, are routinely available for finite automata, we propose a new approach to generate finite automata from an expressive interaction language with operators such as the concurrent region. Our approach leverages an operational semantics to compute derivatives of an interaction and assimilate them to states of a finite automata. In addition, we use term rewriting to merge states on-the-fly so as to obtain small automata without relying on costly a-posteriori minimization techniques.

KEYWORDS

Interaction language, Sequence Diagram, Message Sequence Chart, Non-deterministic Finite Automaton, Operational Semantics, Term Rewriting

ACM Reference Format:
Erwan Mahe, Boutheina Bannour, Christophe Gaston, Arnault Lapitre, and Pascale Le Gall. 2024. Finite Automata synthesis from interactions. In *Formal Methods in Software Engineering (FormaliSE) (FormaliSE '24), April 14–15, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3644033.3644382

1 INTRODUCTION

Context. Interactions are behavioral models describing communication flows between actors. Interaction languages include, among others, Message Sequence Charts (MSC) [15, 38], or UML Sequence Diagrams (UML-SD) [41]. Their main advantage is their easy-to-read graphical representation, which we may call Sequence Diagrams (SD). Let us describe their main elements using the SD drawn on Fig.1. Each actor is associated to a vertical line, called a *lifeline* (here l_1, l_2 and l_3). Behaviors are described as *traces* i.e., successions of atomic communication actions (abbrv. *actions*) which are either the emission or the reception of a message by a lifeline. In the diagrammatic representation, the emission (resp. reception) of a message is represented by an arrow exiting (resp. entering) a lifeline. By extension, message exchanges (i.e., an emission and a corresponding reception) are represented by contiguous horizontal arrows connecting two lifelines. The name of the emitted or received message is represented above the corresponding arrow (e.g., m_1, m_2 and m_3 in our example).

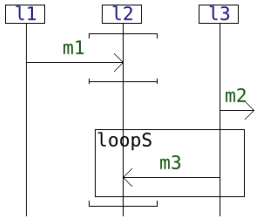


Figure 1: SD representation

Actions being discrete and atomic, message exchanges impose a causal order in which the emission of a message must occur before its reception (on our example, l_1 must emit m_1 before l_2 can receive it). High-level operators such as various kinds of sequencing, parallel composition, choice and repetition can be used to schedule atomic communication actions into more complex structured scenarios. A particularity of interactions is the distinction between strict and weak sequencing. While the former always enforces an order between actions, the latter only does so between actions occurring on the same lifeline. Concurrent regions (abbrv. *co-regions*) further improve expressiveness with the ability to detail which lifelines allow interleavings and which ones do not. In practice, co-regions behave like parallel composition on certain lifelines and like weak sequencing on the others.

Most operators are drawn as annotated boxes (for instance, $loop_S$ denotes a strictly sequential loop in our example). This is not the case for the weak sequencing operator, which corresponds to the top-to-bottom direction of the diagram, and for concurrent regions which are represented using brackets on specific lifelines (e.g., on l_2 in our example). Our example illustrates how weak sequencing constrains the order between actions, depending on where they occur. If one considers the order of the emission of m_2 w.r.t. that of m_3, weak sequencing forces l_3 to emit m_2 before it can emit any instance of m_3 because those two actions occur on the same lifeline l_3. By contrast, if one considers the emission of m_1 and that of m_2, weak sequencing allows them to occur in any order because they occur on two different lifelines (l_1 and l_3). In our example, the loop allows arbitrarily many instances of the reception of m_3 to occur sequentially. The co-region on l_2 then allows the reception of m_1 to occur before, in-between or after any of these instances of the reception of m_3.

Interestingly, for a subset of interactions, associated trace languages are regular. Consequently, it becomes feasible to associate them with Finite Automata (FA). Integrating interaction languages with FA significantly broadens their applicability across various domains, particularly model-checking [2] or monitoring [46]. This

FormaliSE '24, April 14–15, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0589-2/24/04
https://doi.org/10.1145/3644033.3644382

is because FA are instrumental in the verification of language inclusion [1] and the validation of safety properties [6].

Related work. The synthesis of Non-deterministic Finite Automata (NFA) has been addressed in [2, 46]. In [2], basic Message Sequence Charts (bMSC) are described using partial order relations induced by message send-receive pairs and local orders of actions on each lifeline. Several bMSC can then be combined to form graphs of bMSC (bMSC-g), which support more complex behaviors. Nodes of a bMSC-g contain a bMSC, and its edges, linking two bMSC, correspond to concatenating the behaviors specified by each. This concatenation can be interpreted as strict or weak sequencing, corresponding to resp. the synchronous or asynchronous interpretation of edges in [2]. Synthesis of NFA from bMSC-g then amounts to linearizing the partial orders associated to each bMSC node and combining them along graph edges. Yet, although this is always possible under the synchronous interpretation of edges, [2] points out that this is not the case in the asynchronous case (thus the claim that model checking of bMSC-g is undecidable).

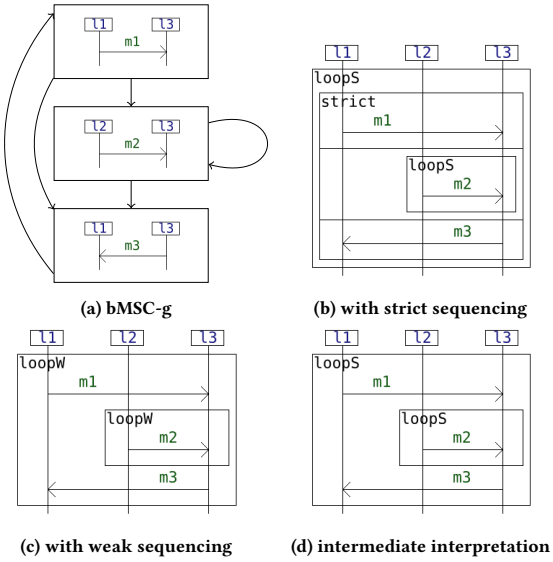


Figure 2: Different interpretations of bMSC-g edges

Fig.2a provides an example bMSC-g built over 3 bMSC b_1, b_2 and b_3 s.t. each bMSC b_i (i in $\{1, 2, 3\}$) involves a message m_i. Under the synchronous interpretation of graph edges, the trace language of that bMSC-g would equate to that of the interaction from Fig.2b. Similarly, under the asynchronous one, it would be that of the interaction from Fig.2c, with the weakly sequential loop operator ($loop_W$) from [36]. While the trace language of the former interaction (Fig.2b) is regular, this is not the case for the latter (Fig.2c). Indeed, it accepts the consecutive emission of n message m_2 followed by their receptions for any integer n, which leads to the non-regularity of language [26]. Because NFA cannot capture non-regular languages, it is impossible to translate this bMSC-g under the asynchronous interpretation of edges.

While the asynchronous interpretation cannot reliably be used for NFA synthesis, the synchronous interpretation is rather conservative and limits considerably the use of weak sequencing (by preventing its use outside basic interactions). Indeed, overapproximating weak sequencing using strict sequencing does not always produce the same trace language. On Fig.2a, let us consider the bMSG-g edge from b_1 to b_2. Here, the emission of m_2 by l_2 from b_2 might be interleaved with the emission of m_1 by l_1 and the reception of m_1 by l_3 from b_1 under the asynchronous interpretation, while it must occur after them under the synchronous interpretation.

In [46], basic SD (only containing message passing and weak sequencing, akin to bMSC) are identified within an overall UML-SD and individually transformed into NFA. Those NFA are then composed following the structure of the SD. For this, operators of UML-SD are mapped to operators on NFA [17]. Alternatives are mapped to NFA union, loops to NFA Kleene star (hence it corresponds to a strictly sequential loop), parallel fragments to NFA parallel composition, and both strict and weak sequencing (outside basic SD) to NFA concatenation. Thus [46] uses the synchronous interpretation of [2].

Both approaches [2, 46] are *compositional* by nature. Indeed, NFA synthesis for complex interactions (represented either by UML-SD or bMSC-g) is achieved by systematically composing smaller NFAs using classic NFA operators (concatenation, union, etc.), the smallest ones corresponding to translations of basic interactions (with only sequencing and atomic actions). Because there is no NFA operator equivalent to weak sequencing, the use of weak sequencing in [2, 46] is thus restricted to basic interactions. Yet, as illustrated on Fig.2d, a finer interpretation of graph edges is possible and would permit the use of weak sequencing outside basic interactions while still preserving the regularity of the trace language.

Contribution. This paper proposes a novel approach for synthesizing concise NFA from complex interactions. This approach does not have the limitations of [2, 46] related to weak sequencing. Instead of being *compositional*, our approach could be described as *incremental* [44]. We indeed leverage a Structural Operational Semantics (SOS) [42] of interactions (marginally adapted from [36]) to construct a NFA whose states correspond to successive *derivatives* (coined by Brzozowski [8]) of the input interaction. This approach is similar to related works on NFA synthesis [5, 45] and testing [44] from regular expressions.

In our approach, there is no difference in the manner with which weak sequencing is handled w.r.t. the other operators of the interaction language and w.r.t. its context within the input SD. Incidentally, this allows us to include concurrent regions (which are hybrid operators between weak sequencing and parallel composition) to our input interaction language, thus making it more flexible and easy to use (w.r.t. bMSG-g in [2] and the subset of UML-SD from [46]). It also facilitates integrating other operators in future works.

Additionally, we use term rewriting to merge NFA states on-the-fly during the synthesis so that each state, instead of corresponding to a single (derivative) interaction rather corresponds to an equivalence class. This enables the direct synthesis of concise NFAs without relying on computationally expensive state reduction techniques, a problem proven to be PSPACE-complete [20].

Paper organisation. Sec.2 covers preliminary notions. In Sec.3, we present an interaction language and an associated SOS adapted from [36]. Sec.4 details our method for NFA synthesis. While Lem.4.3

ensures we synthesize NFAs with a finite set of states, Th.4.8 proves they have the same trace language as the corresponding input interaction. Sec.5 presents experimental results, including comparisons of our approach and those based on composition (e.g., [46]). The paper concludes in Sec.6.

2 PRELIMINARIES

For a set A, we resp. denote by $\mathcal{P}(A)$ and A^* the sets of all subsets of A and all finite sequences over A. The concatenation operation is denoted by "." and the empty sequence by ε. For $w \in A^*$ and for $a \in A$, we resp. denote by $|w|$ and $|w|_a$ the length of w and the number of occurrences of a in w. A word u is said to be a prefix of w if there exists a word v such that $w = u.v$.

Given two sets X and Y, we denote by $X \times Y$ their Cartesian product and by Y^X the set of all functions $\phi : X \to Y$ of domain X and codomain Y.

Definition 2.1. A NFA is a tuple $(A, Q, q_0, F, \rightsquigarrow)$ s.t. A is a finite alphabet, Q is a finite set of states, $q_0 \in Q$ is an initial state, $F \subseteq Q$ is a set of accepting states and $\rightsquigarrow \subseteq Q \times A \times Q$ is a finite set of transitions.

Non-deterministic Finite Automata (NFA). Def.2.1 recalls the definition of NFA. A transition (q, x, q'), with q and q' states in Q and $x \in A$ is denoted as $q \overset{x}{\rightsquigarrow} q'$ and is said to be labelled by x. A path from q to q' is a finite sequence of $k > 0$ consecutive transitions $q_i \overset{x_{i+1}}{\rightsquigarrow} q_{i+1}$ with $i \in [0, k-1]$ s.t. $q_0 = q$ and $q_k = q'$. If we denote by w the word obtained from concatenating their labels i.e., $w = x_1.\cdots.x_k$, we may then write $q \overset{w}{\rightsquigarrow} q'$. In particular, we have $q \overset{\varepsilon}{\rightsquigarrow} q$ for any $q \in Q$. The language recognized by a NFA $\mathcal{A} = (A, Q, q_0, F, \rightsquigarrow)$ is then the set of words $\mathcal{L}(\mathcal{A}) = \{w \in A^* \mid \exists\, q_f \in F \; s.t. \; q_0 \overset{w}{\rightsquigarrow} q_f\}$.

Term rewriting. A set of operation symbols $\mathcal{F}$ is a structured set $\mathcal{F} = \biguplus_{j \geq 0} \mathcal{F}_j$ s.t. for any integer $j \geq 0$, the set of all symbols of arity j is denoted by $\mathcal{F}_j$. Symbols of arity 0 are constants. For any set of variables $\mathcal{X}$, we denote by $\mathcal{T}_{\mathcal{F}}(\mathcal{X})$ the set of terms over $\mathcal{X}$ defined as the smallest set s.t., **(1)** $\mathcal{F}_0 \cup \mathcal{X} \subset \mathcal{T}_{\mathcal{F}}(\mathcal{X})$ and **(2)** for any symbol $f \in \mathcal{F}_j$ of arity $j > 0$ and for any terms $t_1, \cdots, t_j$ from $\mathcal{T}_{\mathcal{F}}(\mathcal{X})$, $f(t_1, \cdots, t_j) \in \mathcal{T}_{\mathcal{F}}(\mathcal{X})$. $\mathcal{T}_{\mathcal{F}} = \mathcal{T}_{\mathcal{F}}(\emptyset)$ denotes the set of ground terms. A function $\phi \in \mathcal{T}_{\mathcal{F}}^{\mathcal{X}}$ is extended as a substitution $\phi \in \mathcal{T}_{\mathcal{F}}^{\mathcal{T}_{\mathcal{F}}(\mathcal{X})}$ s.t. $\forall\, t \in \mathcal{F}_0$, $\phi(t) = t$ and for all terms of the form $f(t_1, \cdots, t_j)$ with $f \in \mathcal{F}_j$, $\phi(f(t_1, \cdots, t_j)) = f(\phi(t_1), \cdots, \phi(t_n))$.

For any $t \in \mathcal{T}_{\mathcal{F}}$, we denote by $pos(t) \in \mathcal{P}(\mathbb{N}^*)$ its set of positions [11] which is s.t., $\forall\, t \in \mathcal{F}$ of the form $f(t_1, \cdots, t_j)$ with $f \in \mathcal{F}_j$ we have $pos(f(t_1, \cdots, t_j)) = \{\varepsilon\} \cup \bigcup_{k \in [1,j]} \{k.p \mid p \in pos(t_k)\}$. For any terms t and s and any position $p \in pos(t)$, $t_{|p}$ denotes the sub-term of t at position p while $t[s]_p$ denotes the term obtained by substituting $t_{|p}$ with s in t.

Definition 2.2. A set R of rewrite rules over $\mathcal{T}_{\mathcal{F}}(\mathcal{X})$ defines a TRS as a relation:

$$\rightarrow_R = \left\{ (t, t') \in \mathcal{T}_{\mathcal{F}}^2 \;\middle|\; \begin{array}{l} \exists\, p \in pos(t),\ \exists\, (x, y) \in R,\ \exists\, \phi \in \mathcal{T}_{\mathcal{F}}^{\mathcal{T}_{\mathcal{F}}(\mathcal{X})} \\ t_{|p} = \phi(x) \text{ and } t' = t[\phi(y)]_p \end{array} \right\}$$

A rewrite rule $x \rightsquigarrow y$ relates two terms x and y from $\mathcal{T}_{\mathcal{F}}(\mathcal{X})$. A set of rewrite rules R characterizes a Term Rewrite System (TRS) (see Def.2.2) $\rightarrow_R$ which relates ground terms. These terms are s.t. we can obtain the right-hand-side by applying a rewrite rule modulo a substitution at a specific position in the left-hand-side i.e., we have $t \rightarrow_R t[\phi(y)]_p$ with $x \rightsquigarrow y$ in R and $t_{|p} = \phi(x)$. For instance the integer expression simplification $3 + (5 + 0) \rightarrow_R 3 + 5$ can be obtained using $R = \{x + 0 \rightsquigarrow x\}$ on $\mathcal{X} = \{x\}$ by applying $\rightsquigarrow$ with $\phi(x) = 5$ at position 2 within $3 + (5 + 0)$.

Given a TRS $\rightarrow_R$, a term $t \in \mathcal{T}_{\mathcal{F}}$ is irreducible iff there are no $t' \neq t$ s.t. $t \rightarrow_R t'$. We denote by $\rightarrow_R^*$ the reflexive and transitive closure of $\rightarrow_R$. A TRS is convergent iff all consecutive applications of $\rightarrow_R$ from a term t necessarily converge to the same irreducible term, i.e., iff $\exists!\ \widetilde{t} \in \mathcal{T}_{\mathcal{F}}$ s.t. $t \rightarrow_R^* \widetilde{t}$ and $\widetilde{t}$ is irreducible. For a convergent TRS $\rightarrow_R$, given a term t, the notation $t \rightarrow_R^! \widetilde{t}$ signifies that $\widetilde{t}$ is the unique irreducible term s.t. $t \rightarrow_R^* \widetilde{t}$.

3 INTERACTION LANGUAGE

3.1 Syntax

In the sequel, a finite set L of lifelines and a finite set M of messages are supposed given. The set of actions is then denoted by $\mathbb{A} = \{l\Delta m \mid l \in L,\ \Delta \in \{!, ?\},\ m \in M\}$. The notations ! and ? resp. correspond to emissions and receptions. For any $a \in \mathbb{A}$ of the form $l\Delta m$ with $\Delta \in \{!, ?\}$, $\theta(a)$ refers to the lifeline l on which it occurs. Sequences of actions called *traces* characterize executions of systems. The set of all possible traces is given by $\mathbb{A}^*$.

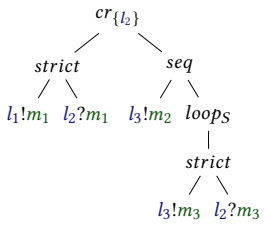


Figure 3: Fig.1 as a term

As done in [36], we encode interactions as ground terms $\mathcal{T}_{\mathcal{F}}$ of a language with constants $\mathcal{F}_0$ being either atomic communication actions (elements $a \in \mathbb{A}$) or the empty interaction (denoted as $\varnothing$) which expresses the empty behavior ε. Symbols of arities 1 and 2 are then used to encode high-level operators. Def.3.1 formalizes our encoding in the fashion of [36]. The language includes strict sequencing *strict*, non-deterministic choice *alt*, strictly sequential repetition $loop_S$ and a set of concurrent region operators cr_ℓ one per subset $\ell \subseteq L$ of lifelines.

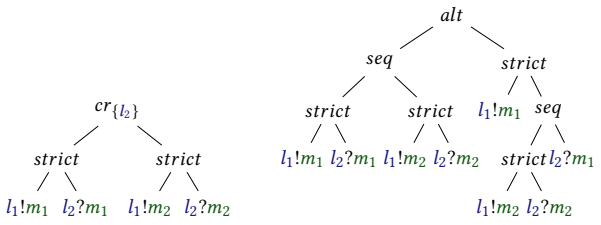
Definition 3.1. The set $\mathbb{I}$ of interactions is the set of ground terms $\mathcal{T}_{\mathcal{F}}$ built over $\mathcal{F}$ s.t.:

$$\mathcal{F}_0 = \mathbb{A} \cup \{\varnothing\} \qquad \mathcal{F}_1 = \{loop_S\}$$
$$\mathcal{F}_2 = \{strict, alt\} \cup \bigcup_{\ell \subseteq L} \{cr_\ell\} \qquad \forall\, k > 2,\ \mathcal{F}_k = \emptyset$$

In our formalism *strict* is used to enforce a strict order between any two behaviors. We use it to encode the asynchronous passing or broadcast of a message. For instance, $strict(l_1!m, l_2?m)$ encodes the passing of m from l_1 to l_2. On Fig.3, which corresponds to an encoding as a term of $\mathbb{I}$ of the SD from Fig.1 we may remark two instances of this pattern, one for each of the arrows carrying m_1 and m_3.

strict is related to $loop_S$, the latter being the Kleene closure of the former. *alt* corresponds to an exclusive choice between two alternative behaviors, each specified by an interaction.

The co-region operators encode both parallel composition and weak sequencing which are usually resp. denoted as *par* and *seq*.



(a) with a co-region **(b) without a co-region**

Figure 4: Illustrating the concision and scaleability of co-regions

Indeed, *par* corresponds to the case where the two scheduled behaviors are concurrent on all lifelines i.e. $par = cr_L$, while we have $seq = cr_\emptyset$. This saves the formalism one additional symbol and the associated set of rules.

In addition, co-regions can be used to express more concisely specific patterns of communications, e.g. for specifying that some messages are emitted in a specific order but may be received in any order. The example of Fig.4 illustrates this on a scenario where two messages m_1 and m_2 are passed from l_1 to l_2, which occurs sequentially on l_1 and concurrently on l_2. We can represent this same behavior either with the interaction on Fig.4a, using a co-region, or the one on Fig.4b, without. Here, avoiding the use of $cr_{\{l_2\}}$ requires considering two alternative branches: one where m_1 is received first and another where it is m_2. The more complex the behavior underneath a *cr* is, the more alternative branches and reshuffling of actions would be required to achieve the same behavior without a co-region.

3.2 Structural operational semantics

We introduce the semantics of interactions in a structural operational style [42]. It borrows usual rules of process algebraic languages [12] and considers specific rules (akin to [36, 37]) related to weak sequencing. We adapt the presentation introduced in [35] (in which the semantics is proven equivalent to a denotational formulation à la [22]) to handle concurrent regions[1].

Definition 3.2 (Evasion). The predicate $\downarrow \subseteq \mathbb{I} \times \mathcal{P}(L)$ is s.t. for any $\ell \subseteq L$, any i_1 and i_2 from $\mathbb{I}$, any $f \in \{strict\} \cup \bigcup_{\ell' \subseteq L} \{cr_{\ell'}\}$:

$$\frac{}{\varnothing \downarrow^\ell} \qquad \frac{\theta(a) \notin \ell}{a \downarrow^\ell} \qquad \frac{i_j \downarrow^\ell}{alt(i_1, i_2) \downarrow^\ell}\ j \in \{1,2\}$$

$$\frac{i_1 \downarrow^\ell \qquad i_2 \downarrow^\ell}{f(i_1, i_2) \downarrow^\ell} \qquad \frac{}{loop_S(i_1) \downarrow^\ell}$$

We say that an interaction $i \in \mathbb{I}$ evades a set of lifelines $\ell \subseteq L$ if it expresses at least one trace that contains no actions occurring on lifelines of ℓ. Def.3.2 introduces the *evasion* predicate accordingly. For example, we always have $loop_S(i) \downarrow^\ell$ for any loop term $loop_S(i)$ and any set of lifelines $\ell \subseteq L$ because the loop expresses the empty behavior ε which corresponds to repeating it 0 times.

The pruning relation from Def.3.3 characterizes, for any interaction i, the existence and uniqueness of another interaction i' that exactly accepts all executions of i that do not involve any lifelines in $\ell \subseteq L$.

Definition 3.3 (Pruning). The pruning relation $\not\leftrightarrow \subset \mathbb{I} \times \mathcal{P}(L) \times \mathbb{I}$ is s.t. for any $\ell \in L$, any $x \in \{\varnothing\} \cup \mathbb{A}$, any i_1 and i_2 from $\mathbb{I}$ and any $f \in \{strict, alt\} \cup \bigcup_{\ell' \subseteq L} \{cr_{\ell'}\}$:

$$\frac{x \downarrow^\ell}{x \not\leftrightarrow^\ell x} \qquad \frac{i_1 \not\leftrightarrow^\ell i_1'}{loop_S(i_1) \not\leftrightarrow^\ell loop_S(i_1')}$$

$$\frac{\neg(i_1 \downarrow^\ell)}{loop_S(i_1) \not\leftrightarrow^\ell \varnothing} \qquad \frac{i_1 \not\leftrightarrow^\ell i_1' \qquad i_2 \not\leftrightarrow^\ell i_2'}{f(i_1, i_2) \not\leftrightarrow^\ell f(i_1', i_2')}$$

$$\frac{i_j \not\leftrightarrow^\ell i_j' \qquad \neg(i_k \downarrow^\ell)}{alt(i_1, i_2) \not\leftrightarrow^\ell i_j'}\ \{j,k\} = \{1,2\}$$

The semantics from [36] relies on the definition of an expression relation which relates interactions to **(1)** the actions which are immediately expressible and **(2)** the interactions which remain to be expressed afterwards. With Def.3.4, we can determine which actions cannot be immediately expressed via the $\not\rightarrow$ relation.

Definition 3.4 (Non-expression). The predicate $\not\rightarrow \subseteq \mathbb{I} \times \mathbb{A}$ is s.t. for any $a \in \mathbb{A}$, any $x \in \{\varnothing\} \cup \mathbb{A}$, any i_1 and i_2 from $\mathbb{I}$ and any $\ell \subseteq L$:

$$\frac{x \neq a}{x \overset{a}{\not\rightarrow}} \qquad \frac{i_1 \overset{a}{\not\rightarrow}}{loop_S(i_1) \overset{a}{\not\rightarrow}} \qquad \frac{i_1 \overset{a}{\not\rightarrow} \qquad (\neg(i_1 \downarrow^L)) \vee (i_2 \overset{a}{\not\rightarrow})}{strict(i_1, i_2) \overset{a}{\not\rightarrow}}$$

$$\frac{i_1 \overset{a}{\not\rightarrow} \qquad i_2 \overset{a}{\not\rightarrow}}{alt(i_1, i_2) \overset{a}{\not\rightarrow}} \qquad \frac{i_1 \overset{a}{\not\rightarrow} \qquad (\neg(i_1 \downarrow^{\{\theta(a)\} \setminus \ell})) \vee (i_2 \overset{a}{\not\rightarrow})}{cr_\ell(i_1, i_2) \overset{a}{\not\rightarrow}}$$

The relation $\rightarrow$ given in Def.3.5 uses the predicates from Def.3.2, Def.3.3 and Def.3.4 to characterize transformations of the form $i \xrightarrow{a} i'$ where $i \in \mathbb{I}$ is an interaction, $a \in \mathbb{A}$ is an action that is immediately executable from i and $i' \in \mathbb{I}$ is a derivative interaction characterizing continuations of behaviors specified by i that start with a.

Definition 3.5 (Expression). The predicate $\rightarrow \subseteq \mathbb{I} \times \mathbb{A} \times \mathbb{I}$ is defined by the rules from Fig.5 with $a \in \mathbb{A}$, $(i_1, i_2, i_1', i_2') \in \mathbb{I}^4$, $f \in \{strict\} \cup \bigcup_{\ell' \subseteq L} \{cr_{\ell'}\}$ and $\{j,k\} = \{1,2\}$.

The relation from Def.3.5 resembles those found in process algebra. The rules "*act*", "*loop*" and "*f-left*" are self-explanatory.

The "*strict-right*" rule enables the execution of actions a on the right of a $strict(i_1, i_2)$ under the condition that i_1 terminates (i.e., any action from i_2 can only be executed after i_1 terminates). This termination is possible iff i_1 can express the empty behavior ε which is characterized by $i_1 \downarrow^L$ because ε is the only behavior which involves no action occurring on any one of the lifelines of L. If a is executed, we then consider that i_1 has terminated (otherwise we might subsequently observe actions a_1 from i_1 which contradicts the order imposed by *strict*) and there only remains to execute i_2' which is s.t. $i_2 \xrightarrow{a} i_2'$.

Similarly, the "*cr-right*" rule dictates the execution of actions on the right of a $cr_\ell(i_1, i_2)$ (with $\ell \subseteq L$). It only does so under the condition that the partial orders imposed by cr_ℓ between the actions of i_1 and those of i_2 are respected. Let us first consider the two edge cases $\ell = L$ and $\ell = \emptyset$.

[1] see [31] for a Coq proof of the equivalence of denotational and operational semantics of interactions that include co-regions

$$\frac{}{a \xrightarrow{a} \varnothing}\ \text{act} \qquad \frac{i_1 \xrightarrow{a} i_1'}{loop_S(i_1) \xrightarrow{a} strict(i_1', loop_S(i_1))}\ \text{loop} \qquad \frac{i_1 \xrightarrow{a} i_1'}{f(i_1, i_2) \xrightarrow{a} f(i_1', i_2)}\ \text{f-left}$$

$$\frac{i_2 \xrightarrow{a} i_2' \qquad i_1 \downarrow^L}{strict(i_1, i_2) \xrightarrow{a} i_2'}\ \text{strict-right} \qquad \frac{i_2 \xrightarrow{a} i_2' \qquad i_1 \not\rightleftarrows^{\{\theta(a)\}\setminus\ell} i_1'}{cr_\ell(i_1, i_2) \xrightarrow{a} cr_\ell(i_1', i_2')}\ \text{cr-right}$$

$$\frac{i_j \xrightarrow{a} i_j' \qquad i_k \not\xrightarrow{a}}{alt(i_1, i_2) \xrightarrow{a} i_j'}\ \text{alt-choice} \qquad \frac{i_1 \xrightarrow{a} i_1' \qquad i_2 \xrightarrow{a} i_2'}{alt(i_1, i_2) \xrightarrow{a} alt(i_1', i_2')}\ \text{alt-delay}$$

Figure 5: Rules for the expression relation

If $\ell = L$, we are in the presence of the interleaving operator $par = cr_L$. In that case, it is always possible to execute a from i_2 because any action a_1 from i_1 can occur either before or after a. This is reflected by the fact that $\{\theta(a)\} \setminus L = \emptyset$ and that $i_1 \not\rightleftarrows^{\emptyset} i_1$ always holds. Hence, we have $i_1' = i_1$ and the predicate $i_1 \not\rightleftarrows^{\emptyset} i_1$ holds, which makes rule "*cr-right*" coincide with the classical right-rule for interleaving [12].

If $\ell = \emptyset$, we are in the presence of the weak sequencing operator $seq = cr_\emptyset$. In that case, let us consider an action a_1 from i_1 occurring on the lifeline $\theta(a)$ on which a occurs. If a_1 must occur whenever i_1 is executed, then it must logically occur before a (as imposed by *seq*). Hence, if a occurs first, this means a_1 must not occur at all. It is possible for i_1 not to express a_1 (and any other action occurring on $\theta(a)$) iff $i_1 \downarrow^{\{\theta(a)\}}$. In that case, in order to compute a derivative to the execution of a from i_2 in $seq(i_1, i_2)$, we need to clean up i_1 from any action occurring on $\theta(a)$. This is the role of the pruning predicate $\not\rightleftarrows$, which intervenes in "*cr-right*" via the condition of the existence of a i_1' s.t. $i_1 \not\xrightarrow{\{\theta(a)\}} i_1'$. In the pruned interaction i_1', we only preserve the behaviors of i_1 that do not contradict the fact that a from i_2 occurs before any action from i_1. For this reason, and because $\ell = \emptyset$, the "*cr-right*" rule coincides with the classical right-rule for weak sequencing [36].

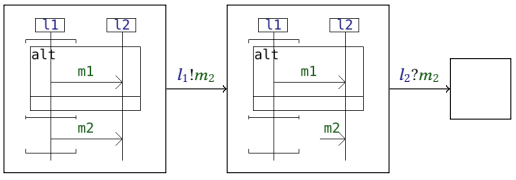
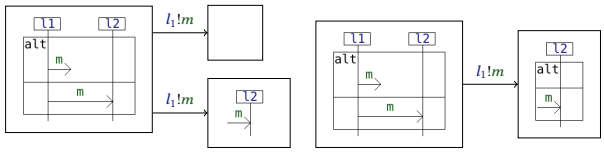


Figure 6: Action expression with co-region and pruning

For all the other cases i.e., whenever $\ell \subsetneq L$ and $\ell \neq \emptyset$, the condition $i_1 \not\xrightarrow{\{\theta(a)\}\setminus\ell} i_1'$ of rule "*cr-right*" makes cr_ℓ behave like interleaving for actions occurring on lifelines of ℓ and like weak sequencing for those occurring on $L \setminus \ell$. Fig.6 illustrates the use of co-region and of pruning on an example. Here, $cr_{\{l_1\}}$ enables the emission of m_1 and m_2 to occur in any order, but requires that m_1 is received before m_2 if it is ever received. When we execute $l_1!m_2$, because it can occur after $l_1!m_1$ due to the concurrent region on l_1, the pruning predicate $\not\rightleftarrows^{l_1}$ does not force the choice of a branch of the alternative. However, when we execute $l_2?m_2$, pruning forces the right branch of the alternative (containing the empty interaction $\varnothing$) to be chosen. Otherwise, we would risk having $l_2?m_1$ occur afterwards, which is forbidden by the weak sequencing.

Let us remark that the $par = cr_L$ operator is the most permissive scheduling operator (among *strict* and all the cr_ℓ with $\ell \subseteq L$ which includes $seq = cr_\emptyset$). Indeed, all the left rules have the same form and thus allow the same derivations while the right rules contain restrictive conditions for *strict* and all cr_ℓ (resp. $i_1 \downarrow^L$ for *strict* and $i_1 \not\xrightarrow{\{\theta(a)\}\setminus\ell} i_1'$ for cr_ℓ with $\ell \subsetneq L$), but not for $par = cr_L$ because $\{\theta(a)\} \setminus L = \emptyset$ and we always have that $i_1 \not\rightleftarrows^{\emptyset} i_1$ holds. As a result, any $f \in \{strict\} \cup \bigcup_{\ell \subseteq L}\{cr_\ell\}$ allows fewer derivations than $par = cr_L$ by construction.



(a) without delayed choice **(b) with delayed choice**

Figure 7: Delayed choice

There are several trace-equivalent (as opposed to bisimilar) manners to define an operational rule for the non-deterministic choice *alt* operator. Fig.7a and Fig.7b present two such manners. In [33, 34] they use the manner described on Fig.7a, in which the choice of alternative branches is made as soon as possible. In this paper, we rather favor the one described on Fig.7b that is called delayed choice in [37] as its use will further reduce the number of states of the generated NFA.

Definition 3.6 (Semantics). $\sigma : \mathbb{I} \to \mathcal{P}(\mathbb{A}^*)$ is the least function (w.r.t. pointwise set-inclusion) satisfying the following rules:

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

The predicates from Def.3.2 and Def.3.5 enable us to define an operational semantics of interactions in Def.3.6. The set $\sigma(i)$ of traces of an interaction i contains all (possibly empty) sequences $a_1 a_2 \dots a_k$ of actions such that there exist some execution steps $i \xrightarrow{a_1} i_1$, $i_1 \xrightarrow{a_2} i_2$, $\dots i_{k-1} \xrightarrow{a_k} i_k$ with $i_k \downarrow^L$. In particular, if the

sequence is empty, then $i \downarrow^L$ and $\varepsilon \in \sigma(i)$. We denote by $\xrightarrow{*}$ the reflexive and transitive closure of the expression relation $\rightarrow$. By extension, for any two interactions i and i' related by $\xrightarrow{*}$, we use the notation $i \xrightarrow{t} i'$ to signify that the successive execution of the actions of a trace $t \in \mathbb{A}^*$ from i leads to i'. On the other hand, if, given $i \in \mathbb{I}$ and $t \in \mathbb{A}^*$ there are no $i' \in \mathbb{I}$ s.t. $i \xrightarrow{t} i'$, then we may write $i \xrightarrow{t} \!\!\!\!\!/$.

4 DERIVING NFA FROM INTERACTIONS

4.1 Derivative interactions

The gist of our method is to execute the input interaction i using the expression relation $\rightarrow$ in order to compute successive derivatives. The set of equivalence classes (up to a rewrite relation) of these derivatives then constitutes the set of states of the generated NFA. For any interaction $i \in \mathbb{I}$, we define its set of reachable derivatives $\mathtt{reach}(i)$ in Def.4.1.

Definition 4.1 (Reachable derivatives). $\mathtt{reach} : \mathbb{I} \rightarrow \mathcal{P}(\mathbb{I})$ is s.t.:

$$\forall\, i \in \mathbb{I},\ \mathtt{reach}(i) = \{i' \mid i \xrightarrow{*} i'\}$$

A first step towards NFA generation is to ascertain that the set $\mathtt{reach}(i)$ is always finite (the set of derivatives of regular expressions has been proven finite in [8]). To do so we rely on a measure $|\ |$ which corresponds to counting (in the syntactic term) the number of atomic actions within an interaction (see Def.4.2).

Definition 4.2 (A measure for interactions). We define $|\ | : \mathbb{I} \rightarrow \mathbb{N}$ s.t., for any a in $\mathbb{A}$, any f in $\mathcal{F}_2$ and any i_1 and i_2 in $\mathbb{I}$:

$$|\varnothing| = 0 \qquad |a| = 1$$
$$|f(i_1, i_2)| = |i_1| + |i_2| \qquad |loop_S(i_1)| = |i_1|$$

We prove in Lem.4.3 that the cardinality of $\mathtt{reach}(i)$ is bounded.

Lemma 4.3. *For any $i \in \mathbb{I}$, we have $|\mathtt{reach}(i)| \leq 2^{|i|}$*

Proof. Let us reason by structural induction over i.

Basic cases ($i \in \{\varnothing\} \cup \mathbb{A}$): $\mathtt{reach}(\varnothing) = \{\varnothing\}$ (because no rule of $\rightarrow$ can be applied to $\varnothing$) hence $|\mathtt{reach}(\varnothing)| = 1$. $\mathtt{reach}(a) = \{a, \varnothing\}$ (by applying rule "*act*", $\varnothing$ is reachable) hence $|\mathtt{reach}(a)| = 2$. In both cases, $|\mathtt{reach}(i)| \leq 2^{|i|}$ (since $|\varnothing| = 0$ and $|a| = 1$).

Inductive cases (i of the form $f(i_1, i_2)$ with $f \in \{strict, alt, cr_\ell\}$ or $loop_S(i_1)$). In the following, we consider two interactions i_1 and i_2 and use the notations: $r_1 = \mathtt{reach}(i_1)$ and $r_2 = \mathtt{reach}(i_2)$.

Considering $par = cr_L$, for $i = par(i_1, i_2)$, we have $\mathtt{reach}(i) = \{par(j_1, j_2) \mid j_1 \in r_1, j_2 \in r_2\}$. Indeed, from terms of this form, we can apply either the rule "*f-left*" or the rule "*cr-right*". The former replaces the sub-term i_1 by a certain i_1' leading to a term of the form $par(i_1', i_2)$. Whichever is the action a that is executed we have $\{\theta(a)\} \setminus L = \emptyset$, and thus $i_1 \overset{\emptyset}{\not\longrightarrow} i_1$. As a result, applying "*cr-right*" yields a term of the form $par(i_1, i_2')$, with i_1 being preserved and i_2' reached from i_2. The same two rules are again the only ones applicable from these terms so that by applying them several times, successive derivations result in all the terms of the form $par(j_1, j_2)$ with j_1 (resp j_2) reachable from i_1 (resp. i_2), i.e., $j_1 \in r_1$ (resp. $j_2 \in r_2$). Then we have $|\mathtt{reach}(i)| = |r_1| * |r_2|$. By induction hypothesis, $|r_1| * |r_2| \leq 2^{|i_1|} * 2^{|i_2|}$, hence $|\mathtt{reach}(i)| \leq 2^{|i|}$ since $|i| = |i_1| + |i_2|$.

For $i = f(i_1, i_2)$ with $f \in \bigcup_{\ell \subseteq L} \{strict, cr_\ell\}$, we have $|\mathtt{reach}(i)| \leq |\mathtt{reach}(par(i_1, i_2))|$. Indeed, these operators enable fewer interleavings than *par*. As, by the previous point, $|\mathtt{reach}(par(i_1, i_2))| \leq 2^{|i|}$, we also have $|\mathtt{reach}(f(i_1, i_2))| \leq 2^{|i|}$.

For $i = alt(i_1, i_2)$, let us use, for $x \in \{1, 2\}$, the notations:

$$r_x^c = \{i_x' \in r_x \mid \forall\, t \in \mathbb{A}^* \text{ s.t. } (i_x \xrightarrow{t} i_x'),\ \text{we have } (i_{3-x} \xrightarrow{t} \!\!\!\!\!/)\}$$
$$r_x^d = \{i_x' \in r_x \mid \exists\, t \in \mathbb{A}^* \text{ s.t. } (i_x \xrightarrow{t} i_x') \wedge (\exists\, i_{3-x}' \text{ s.t. } i_{3-x} \xrightarrow{t} i_{3-x}')\}$$

This implies that $r_x = r_x^c \uplus r_x^d$ (i.e., also $r_x^c \cap r_x^d = \emptyset$) and:

$$\mathtt{reach}(i) = r_1^c \cup r_2^c \cup \{alt(i_1', i_2') \mid (i_1' \in r_1^d) \wedge (i_2' \in r_2^d)\}$$

As for any interaction i, we have $i \xrightarrow{\varepsilon} i$, we always have $i_1 \in r_1^d$ and $i_2 \in r_2^d$ and thus $|r_1^d| > 0$ and $|r_2^d| > 0$. Hence:

$$\begin{array}{lcll}
|\mathtt{reach}(i)| & \leq & |r_1^c| + |r_2^c| + |r_1^d| * |r_2^d| & \\
& \leq & |r_1^c| * |r_2^d| + |r_2^c| * |r_1^d| + |r_1^d| * |r_2^d| & \text{(multiply by } > 0) \\
& \leq & (|r_1^c| + |r_1^d|) * (|r_2^c| + |r_2^d|) & \\
& \leq & |r_1| * |r_2| & (r_x = r_x^c \uplus r_x^d) \\
& \leq & 2^{|i_1|} * 2^{|i_2|} & \text{(by induction)} \\
& \leq & 2^{|i_1|+|i_2|} = 2^{|i|} &
\end{array}$$

For $i = loop_S(i_1)$, we have $\mathtt{reach}(i) \subseteq \{i\} \cup \{strict(i_1', i) \mid i_1' \in r_1 \setminus \{i_1\}\}$. Indeed, only the rule "*loop*" is applicable, so that the first derivation (if it exists) is of the form $strict(i_1', loop_S(i_1))$. From there, **(1)** if we apply "*f-left*", we obtain a term $strict(i_1'', loop_S(i_1))$ with $i_1'' \in \mathtt{reach}(i_1')$. With further applications of the rule "*f-left*", we obtain all the terms of the form $strict(i_1^\dagger, loop_S(i_1))$ with $i_1^\dagger \in r_1$. If, on the other hand **(2)**, we apply "*strict-right*", it means that $i_1' \downarrow^L$ and that we have $loop_S(i_1) \xrightarrow{a} strict(i_1^*, loop_S(i_1))$ with $i_1 \xrightarrow{a} i_1^*$ so that we obtain $strict(i_1', loop_S(i_1)) \xrightarrow{a} strict(i_1^*, loop_S(i_1))$. As i_1^* belongs to r_1, by applying the "*strict-right*" rule, we find yet again an interaction of the form $strict(i_1^\ddagger, i)$ with $i_1^\ddagger \in r_1$. Thus, we have:

$$\begin{array}{lcl}
|\mathtt{reach}(i)| & \leq & 1 + |r_1 \setminus \{i_1\}| = |r_1| \\
& \leq & 2^{|i_1|} = 2^{|i|}
\end{array}$$

□

This enables the construction of a NFA in which each state corresponds to an interaction. We explain in the following how we reduce the number of states on-the-fly by simplifying interactions as they are discovered (using term rewriting). Memorization of already encountered simplified terms allows us to build the set of states of the NFA, while identifying transformations $i \xrightarrow{a} i'$ allows building its set of transitions. A state corresponding to a term i is accepting iff $i \downarrow^L$.

4.2 State reduction and synthesis of the NFA

In most cases, the iterative application of the expression relation $\rightarrow$ creates many useless occurrences of $\varnothing$ in the derived interaction terms. These sub-terms can be removed without the set of traces being modified. For example, because of the rule '*f-left*', the derivations from $strict(i, \varnothing)$ are exactly those from i. Hence we say that $strict(i, \varnothing)$ and i are semantically equivalent. The construction of the NFA associated to an interaction can take advantage of considering such semantically equivalent terms in order to reduce the number of states given that each state corresponds to a term.

In order to set up these simplification mechanisms, we provide in Def.4.4 a set R of rewrite rules aimed at eliminating the useless occurrences of $\emptyset$.

Definition 4.4. Given a variable x, we define the set R of rewrite rules over $\mathcal{T}_{\mathcal{F}}(\{x\})$ as follows:

$$R = \left(\begin{array}{l} \cup_{f \in \cup_{t \subseteq L}\{strict, cr_t\}} \{f(\emptyset, x) \rightsquigarrow x,\ f(x, \emptyset) \rightsquigarrow x\} \\ \cup \left\{ \begin{array}{l} alt(\emptyset, loop_S(x)) \rightsquigarrow loop_S(x), \\ alt(loop_S(x), \emptyset) \rightsquigarrow loop_S(x), \\ alt(\emptyset, \emptyset) \rightsquigarrow \emptyset, \\ loop_S(\emptyset) \rightsquigarrow \emptyset \end{array} \right\} \end{array} \right)$$

These rewrite rules define a TRS $\rightarrow_R$ which is convergent and preserves the semantics σ (as per Lem.4.5).

Lemma 4.5. *The TRS characterized by R is convergent and semantically sound i.e., for any $i \in \mathbb{I}$, $\exists!\ \widetilde{i} \in \mathbb{I}$ s.t. $i \rightarrow_R^! \widetilde{i}$ and we have $\sigma(\widetilde{i}) = \sigma(i)$.*

Proof. $\rightarrow_R$ is both terminating and confluent (see the automated proof in [32] using resp. the TTT2 [23] and CSI [47] tools) and hence convergent. To prove the semantic equivalence it suffices to prove it for the interactions related by all $\rightsquigarrow \in R$, which is trivial. □

Because Lem.4.5 states that the TRS is convergent, for any $i \in \mathbb{I}$, there exists a unique $\widetilde{i}$ such that $i \rightarrow_R^! \widetilde{i}$. Given the absence of ambiguities on the TRS, let us use the notation $\widetilde{i}$ to designate the simplified form of any interaction $i \in \mathbb{I}$.

In Def.4.6, we define NFA synthesis as a function `nfa` which translates interactions from $\mathbb{I}$ to NFA.

Definition 4.6. For any $i_0 \in \mathbb{I}$, $\texttt{nfa}(i_0) = (\mathbb{A}, Q, q_0, F, \rightsquigarrow)$ is the NFA whose elements are defined by:

$$Q = \{\widetilde{i} \mid i \in \texttt{reach}(i_0)\} \qquad q_0 = \widetilde{i_0}$$
$$\rightsquigarrow = \{(\widetilde{i}, a, \widetilde{i'}) \mid (i, i') \in \texttt{reach}(i_0)^2, a \in \mathbb{A},\ i \xrightarrow{a} i'\}$$
$$F = \{\widetilde{i} \mid \widetilde{i} \in Q, \widetilde{i} \downarrow^L\}$$

In a few words, states of $\texttt{nfa}(i_0)$ correspond to unique representatives of equivalence classes on interactions defined by: $i \equiv i'$ iff $\widetilde{i} = \widetilde{i'}$. In particular, the transitions between two simplified interactions j and j' include all the transitions between interactions i and i' verifying $\widetilde{i} = j$ and $\widetilde{i'} = j'$. Those results can be generalized for any convergent and semantically sound TRS for interactions. In [36] the converge of such a TRS (taking advantage of additional properties such as idempotence of loops) is proven modulo AC[2].

The NFA $\texttt{nfa}(i_0)$ in Def.4.6 is well defined. Indeed, the set Q is finite because $|Q| \leq |\texttt{reach}(i)|$ (there is at most one simplified interaction per interactions of $\texttt{reach}(i)$) and $|\texttt{reach}(i)| \leq 2^{|i|}$ according to Lem.4.3). The subset F of accepting states and the transition relation $\rightsquigarrow$ are defined thanks to the predicates $\downarrow^L$ and $\rightarrow$ from Def.3.2 and Def.3.5, which ensures that the trace language of the synthesized NFA is the same as that of the initial interaction (i.e., $\mathcal{L}(\texttt{nfa}(i)) = \sigma(i)$, which is proven in Th.4.8).

Lemma 4.7. *For any $i \in \mathbb{I}$, we have $(i \downarrow^L) \Leftrightarrow (\widetilde{i} \downarrow^L)$.*

[2]modulo Associativity and Commutativity of binary operators c.f. rewriting modulo theories [11]

Proof. Because $\sigma(i) = \sigma(\widetilde{i})$ we have $(\varepsilon \in \sigma(i)) \Leftrightarrow (\varepsilon \in \sigma(\widetilde{i}))$ and thus $(i \downarrow^L) \Leftrightarrow (\widetilde{i} \downarrow^L)$. □

Theorem 4.8. *For any $i \in \mathbb{I}$, we have $\mathcal{L}(\texttt{nfa}(i)) = \sigma(i)$*

Proof. Let us consider a trace $t = a_1.\cdots.a_n \in \mathbb{A}^*$.

If $t \in \sigma(i)$ then (Def.3.6) there exist $i_0, \cdots, i_n$ in $\texttt{reach}(i)$ s.t. $i_0 = i$ and $\forall\ k \in [0, n-1], i_k \xrightarrow{a_{k+1}} i_{k+1}$ and $i_n \downarrow^L$. Thus, as per Def.4.6, we have $q_0 = \widetilde{i_0} \overset{a_1}{\rightsquigarrow} \widetilde{i_1} \overset{a_2}{\rightsquigarrow} \cdots \overset{a_n}{\rightsquigarrow} \widetilde{i_n}$. Because $i_n \downarrow^L$, we have $\widetilde{i_n} \downarrow^L$ (Lem.4.7) which implies $\widetilde{i_n} \in F$ and thus $t = a_1.\cdots.a_n \in \mathcal{L}(\texttt{nfa}(i))$.

If $t \in \mathcal{L}(\texttt{nfa}(i))$, there exists states q_1 through q_n in Q s.t. $q_0 \overset{a_1}{\rightsquigarrow} q_1 \overset{a_2}{\rightsquigarrow} \cdots \overset{a_n}{\rightsquigarrow} q_n$ and $q_n \in F$. Thus, as per Def.4.6, we have n interactions i'_0 to i'_n s.t. $\forall\ k \in [0, n]$, $q_k = \widetilde{i'_k}$ and $\forall\ k \in [0, n-1]$, $i'_k \xrightarrow{a_{k+1}} i'_{k+1}$. Because $q_n = \widetilde{i'_n} \in F$, we have $\widetilde{i'_n} \downarrow^L$ and thus $i'_n \downarrow^L$ (Lem.4.7). Hence, as per Def.3.6, we have $t \in \sigma(i'_0) = \sigma(\widetilde{i'_0})$. But because $\widetilde{i'_0} = q_0 = \widetilde{i}$, this implies $t \in \sigma(\widetilde{i}) = \sigma(i)$. □

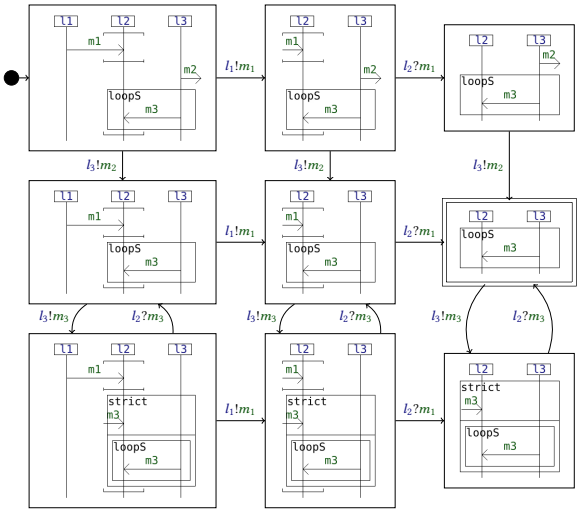


Figure 8: NFA obtained using our incremental approach

Fig.8 represents a NFA obtained using our approach on our initial interaction example (whose SD and term representations are resp. given in Fig.1 and Fig.3). Each of the 9 states is represented by a rectangle that contains the SD representation of the simplified interaction $\widetilde{i}$ associated to it. The initial state is the target of the transition exiting •, and rectangles with a double border correspond to accepting states. Transitions are labeled by the corresponding atomic actions in $\mathbb{A}$. This example illustrates that our incremental approach might facilitate explainability and traceability in e.g., model-checking or testing, as each state can be uniquely traced back to an instant in the unfolding of a global scenario represented by a SD (the representative interaction of the state).

4.3 Remarks on generating small NFA

A FA generated from an interaction describing a real-world system may be extremely large (in number of states) which poses problems related to the time for building the FA or the space for storing it.

State reduction and state minimization aim at obtaining an equivalent FA with resp. fewer states and as few states as possible. It is known that a minimal DFA may have an exponentially larger number of states than that of an equivalent minimal NFA [25]. For instance, given the alphabet $\{a, b\}$, the regular expression $(a|b)^*a(a|b)^n$ can be encoded as a NFA with $n+2$ states while an equivalent minimal DFA has 2^{n+1} states. Because interactions can encode such expressions, choosing NFA as a target formalism is preferable if we want to optimize towards state reduction. Additionally, operators such as weak sequencing, parallelism and repetition introduce non-deterministic and cyclic behaviors into interactions.

On the other hand, the problem of state minimization for NFA is established as PSPACE-complete [20]. Various algorithms exist, such as Kameda-Weiner [21] or [40, 43]. However, their scalability remains an issue, as evidenced by several works [9, 14, 18, 19] seeking approximal solutions (i.e., reduced but not minimal NFA) within a more reasonable time. Still, experimental validations are limited to small NFAs, typically with at most 15 states [14]. Although more efficient algorithms exist for specific NFA subclasses (e.g., DFA[16] or acyclic automata[3, 10]), they cannot be directly applied to minimize FAs obtained from interactions where the minimal FA can be cyclic and non-deterministic.

It is essential to acknowledge the inherent limitations in generating compact NFAs from interactions. For basic interactions, using weak and strict sequencing to compose atomic actions, we may have, in the worst case, up to $O(n^k)$ states, where n is the number of actions, and k the number of lifelines [2]. The computational cost for NFA generation from regular expressions with interleaving over finite words is $\Omega(2^n)$, where n is the number of letters [13, 39]. This result directly applies to interactions as it recalls Lem.4.3: an NFA synthesized from an interaction i could require $2^{|i|}$ states, where $|i|$ is the number of atomic actions it contains.

This discussion underlines the importance of considering state reduction during the generation of the NFA itself and not a posteriori. Our method addresses this concern via its on-the-fly merging of states using term rewriting.

5 EXPERIMENTS

We have implemented our NFA generation method in the tool HIBOU [27]. Manipulation of FA is handled by the AUTOUR [29] toolbox. For the sake a simplicity, we do not consider co-regions cr_ℓ with $\ell \notin \{\emptyset, L\}$ i.e., we only consider the classical *seq* and *par* variants. We performed two sets of experiments on an Intel(R) Core(TM)i5-6360U CPU (2.00GHz) with 8GB RAM with HIBOU version 0.8.5. The main set of experiments we present consists in comparing our incremental method to a compositional approach. The second applies our method to more realistic use cases from the literature. The details of the experiments and the artifacts required to reproduce them are available in [30] and [28].

For the purpose of comparison, we have also implemented a baseline compositional approach which corresponds to adapting [46] to our language. It consists of identifying basic interactions, translating them to NFA (using our method as a fallback) and then composing them recursively by mapping higher-level interaction operators to classical NFA operators. Because of the inherent limitations of compositional [2, 46] methods (there is no NFA operator for weak sequencing, see discussion in Sec.1), we limit ourselves, for the comparison, to interactions in which *seq* operators cannot nest other complex operators (beside other *seq* and *strict*).

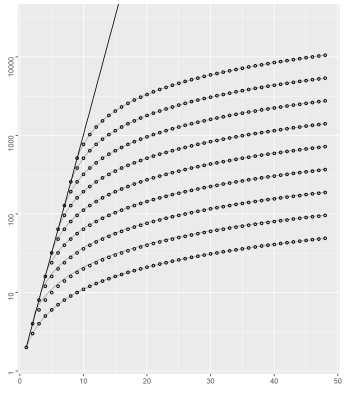


Figure 9: Common grid

To set a scale and thus improve the understanding of the size of generated NFA against that of the corresponding input interactions, we use the common grid represented on Fig.9. The x axis corresponds to $|i|$ (the number of actions in an input interaction, see Def.4.2), and the y axis corresponds to the number of states in a generated NFA. Note that the y axis is in logarithmic scale, allowing for the visualization of a wide range of values, although the distinction between values is less pronounced.

The straight black line on the left corresponds to $x \mapsto 2^x$, giving the upper bound $2^{|i|}$ on the cardinal of the set $\texttt{reach}(i)$ (Lem.4.3). The 432 points on Fig.9 each corresponds to an interaction involving n distinct actions $a_1, \ldots, a_k, \ldots, a_n$ of the form:

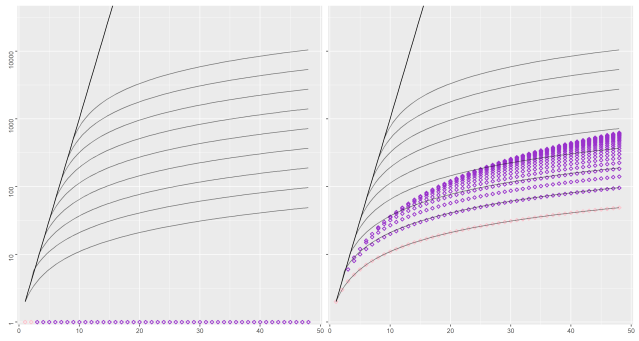
$par(a_1, \cdots, par(a_k, strict(a_{k+1}, \cdots, strict(a_{n-1}, a_n) \cdots)) \cdots)$.

If $k = 0$, the interaction is a strict sequencing of the n actions, which yields a minimal NFA of $n+1$ states. If $k = n$, the interaction is a parallel composition of the n actions, which yields a minimal NFA of size 2^n. The dataset is built by varying n from 1 to 48 and k from 0 to 8 (hence, 432 interactions with some overlap).

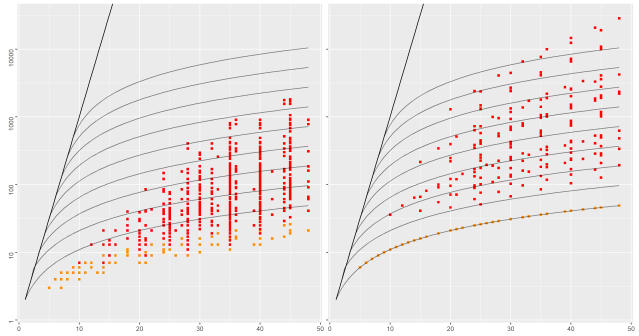
The position of each of the 432 points on Fig.9 corresponds to $|i|$ on the x axis and the number of states of the minimal NFA associated to the trace language of i on the y axis. With these simple interactions, both approaches to generating NFA (incremental and compositional) trivially return the minimal NFA. The data points form a series of curves. We approximate these curves using functions $x \mapsto 2^k * (x - k) + 1$ (e.g., coincide with $x \mapsto x + 1$ for $k = 0$). These curves serve as a common grid to appreciate and compare both approaches w.r.t. each other and w.r.t. the size of minimal NFAs obtained from interactions with the same number of actions (at $|i|$ fixed).

Using our grid, we conduct a comparative analysis of both approaches across 3 datasets illustrated on Fig.10. For each dataset, the size of generated NFA using our method is depicted on the left diagram, while the results involving the compositional approach are shown on the right. As both diagrams have the same referential grid, it is easy to compare the approaches using the y position of the data points (the NFAs of higher points have exponentially more states than lower ones given the logarithmic scale on y). Because the *par* operator has an important effect on the minimal number of states, we have colored interactions without *par* operators using a lighter color (pink instead of purple on Fig.10a, orange instead of red on Fig.10b, and a lighter shade of green on Fig.10c).

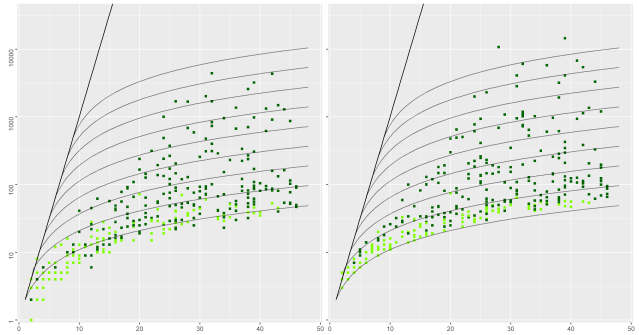
Fig.10a displays a 'loop-alt' dataset, containing interactions of the form: $par(loop_S(alt(a_1, \cdots, a_k)), loop_S(alt(a_{k+1}, \ldots, a_n)))$, with



(a) 'loop-alt' dataset



(b) 'locks' dataset



(c) 'random' dataset

Figure 10: Comparing our approach (left) and composition (right)

variations in n and k up to 48 and 24 respectively. Using our approach, whichever are the numbers n and k, we always have a NFA with a single state and n self transitions. This is not the case for the compositional approach, in which the number of states grows rapidly with the number of actions and the degree of parallelization (up to 625 states for 48 actions).

Fig.10b displays a 'locks' dataset that adapts the regular expression example from Sec.4.3 into digital lock use cases. A digital lock is represented by an interaction of the form:

$strict(loop_S(alt(r_1, \cdots, r_n)), r_1^c, alt(r_1, \cdots, r_n)^{c'})$

where r_1, ..., r_d are d reception actions on the same lifeline for different possible digits in an alphabet. In order to open the lock, a correct sequence of c digits must be entered (r_1^c in the example) followed by c' other digits ($alt(r_1, \cdots, r_d)^{c'}$ in the example). Several such locks may then be composed either strictly sequentially, or in parallel. The dataset is built by varying the number of locks, how these locks are related (via *strict* or *par*) and, for each lock, which are the values of d, c and c'. As discussed in Sec.4.3, minimal DFA associated to such interactions have exponentially more states than their NFA counterpart. Experimental results show that our approach limits this state space explosion, which is less visible for the compositional approach.

Fig.10c displays a 'random' dataset, containing randomly generated interactions. This roughly consists in drawing symbols in $\mathcal{F}$ recursively until a term is built. Special cases are made to ensure finite depth and that we have *seq* only within basic interactions (so as to conform to the limitations of the compositional approach). We generate 350 interactions with less than 50 actions and less than 6 *par* symbols. While the first two datasets ('loop-alt' and 'locks') contain interactions with specific structures that could advantage our approach, the 'random' dataset does not. Still, experimental results shows that our method consistently outputs smaller NFAs than the compositional approach.

All materials required to replicate this first set of experiments are available in [30].

In our second set of experiments, we applied our method to use cases found in the literature. Without the need for comparison w.r.t. the compositional approach, we can lift the restriction on having no *seq* outside basic sub-interactions. More precisely, the compositional approach, applied to any interaction $i \in \mathbb{I}$, does not necessarily produce a nfa $\mathcal{A}$ s.t. $\mathcal{L}(\mathcal{A}) = \sigma(i)$. For $\mathcal{L}(\mathcal{A}) = \sigma(i)$ to hold, it is necessary that, for all the sub-terms $seq(i_1, i_2)$ of i outside basic interactions, we have $\sigma(seq(i_1, i_2)) = \sigma(strict(i_1, i_2))$ i.e. it is equivalent to interpret the corresponding edge on the bMSG-g synchronously or asynchronously. Generally, while lower-level specifications (which need to be implementable) respect this property, this is rarely the case for higher-level requirements.

	i		$\texttt{nfa}(i)$		
	$\|L\|$	$\|a\|$	$\|Q\|$	$\|\rightsquigarrow\|$	time
ABP [37]	4	60	68	94	8, 1 *ms*
HR [24]	6	24	101	214	7, 5 *ms*
Sensor [7]	12	26	168	368	14, 8 *ms*
Platoon 3 [4]	3	17	90	189	3, 1 *ms*
Platoon 4 [4]	4	31	752	1 874	57, 5 *ms*
Platoon 5 [4]	5	48	6 440	18 855	2 338 *ms*

Figure 11: Use cases

In [28], we provide all the necessary materials to reproduce these experiments and summarize metrics obtained from using our method to 4 use cases, including a low-level specification of the Alternating Bit Protocol [37] (ABP) and three high level requirements for: a DApp system for managing Human-Resources [24] (HR), a system for querying sensor data [7] (Sensor) and a connected platoon of autonomous rovers [4] (Platoon). For the Platoon use case, we consider several interaction models, each corresponding to a number of rovers. Fig.11 presents some of these metrics, $|L|$ and $|a|$ resp. corresponding to the number of lifelines and actions in the term structure of the interaction i while $|Q|$ and $|\rightsquigarrow|$ to, resp., the

numbers of states and transitions in the synthesized NFA $\texttt{nfa}(i)$. The "time" column gives the time required to compute the NFA.

6 CONCLUSION

We have proposed an approach for synthesizing Non-deterministic Finite Automata (NFA) from interactions. In contrast to previous approaches [2, 46], that are based on composition using classical NFA operators, our approach relies on computing the set of interactions that can be derived from the initial interaction i using a structural operational semantics. As this set is finite, we use it as the set of states of the NFA we synthesize from i. State reduction can be performed on-the-fly using term rewriting, thus avoiding the use of costly NFA reduction techniques. Further research on semantically sound rewrite systems for interactions may further improve this state reduction by allowing more states to be merged. Moreover, unlike [2, 46], our method handles interactions in which weak sequencing may nest other complex operators such as alternatives, parallel composition or loops.

Additionally, an instrumented comparison of our method and a compositional approach adapted from [46] underlined the ability of our method to consistently synthesize compact NFAs, their size (in number of states) being lower (sometimes significantly) than that of NFAs obtained via composition. Further experiments on use cases from the literature endorse the flexibility and applicability of our method.

The results of this paper open new perspectives regarding model-checking and online runtime verification from interactions. Another avenue of research concerns finer characterizations of interactions by adapting properties of graphs of MSC related to the implementability of systems.

Acknowledgements. The research leading to these results has received funding from the European Union's Horizon Europe programme under grant agreement No 101069748 – SELFY project. This work has been partially supported by the French National Research Agency under the France 2030 label (NF-HiSec ANR-22-PEFT-0009). The views reflected herein do not necessarily reflect the opinion of the French government.

REFERENCES

[1] Parosh Aziz Abdulla, Yu-Fang Chen, Lukáš Holík, Richard Mayr, and Tomáš Vojnar. 2010. When Simulation Meets Antichains. In *Tools and Algorithms for the Construction and Analysis of Systems*, Javier Esparza and Rupak Majumdar (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 158–174.

[2] Rajeev Alur and Mihalis Yannakakis. 1999. Model Checking of Message Sequence Charts. In *CONCUR '99: Concurrency Theory, 10th International Conference, Eindhoven, The Netherlands, August 24-27, 1999, Proceedings (Lecture Notes in Computer Science, Vol. 1664)*, Jos C. M. Baeten and Sjouke Mauw (Eds.). Springer, 114–129. https://doi.org/10.1007/3-540-48320-9_10

[3] Jerome Amilhastre, Philippe Janssen, and Marie-Catherine Vilarem. 2001. FA Minimisation Heuristics for a Class of Finite Languages. In *Automata Implementation*, Oliver Boldt and Helmut Jurgensen (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–12.

[4] Maroneze Andre, Massonnet Philippe, Dupont Sebastien, Nigam Vivek, Plate Henrik, Sykosch Arnold, Cakmak Eren, Thanasis Sfetsos, Jimenez Victor, Amparan Estibaliz, Martinez Cristina, Lopez Angel, Garcia-Alfaro Joaquin, Segovia Mariana, Rubio-Hernan Jose, Blanc Gregory, Debar Herve, Carbone Roberto, Ranise Silvio, Verderame Luca, Spaziani-Brunella Marco, Yautsiukhin Artsiom, Morgagni Andrea, Klein Jacques, Bissyande Tegawende, and Samhi Jordan. 2020. Assessment specifications and roadmap. In *Technical report*, EC (Ed.). SPARTA project.

[5] Valentin Antimirov. 1995. Partial derivatives of regular expressions and finite automata constructions. In *STACS 95*, Ernst W. Mayr and Claude Puech (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 455–466.

[6] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of model checking*. MIT Press.

[7] Mohamed Bakillah, Steve H. L. Liang, Alexander Zipf, and Mir Abolfazl Mostafavi. 2013. A dynamic and context-aware semantic mediation service for discovering and fusion of heterogeneous sensor data. *J. Spatial Inf. Sci.* 6, 1 (2013), 155–185. https://doi.org/10.5311/JOSIS.2013.6.104

[8] Janusz A. Brzozowski. 1964. Derivatives of Regular Expressions. *J. ACM* 11, 4 (1964), 481–494. https://doi.org/10.1145/321239.321249

[9] Jean-Marc Champarnaud and Fabien Coulon. 2003. NFA Reduction Algorithms by Means of Regular Inequalities. In *Developments in Language Theory*, Zoltan Esik and Zoltan Fulop (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 194–205.

[10] Jan Daciuk, Bruce W. Watson, Stoyan Mihov, and Richard E. Watson. 2000. Incremental Construction of Minimal Acyclic Finite-State Automata. *Comput. Linguist.* 26, 1 (mar 2000), 3–16. https://doi.org/10.1162/089120100561601

[11] Nachum Dershowitz and Jean-Pierre Jouannaud. 1991. *Rewrite Systems*. MIT Press, Cambridge, MA, USA, 243–320.

[12] Wan Fokkink. 2009. Process Algebra: An Algebraic Theory of Concurrency. In *Algebraic Informatics*, Symeon Bozapalidis and George Rahonis (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 47–77.

[13] Wouter Gelade. 2010. Succinctness of regular expressions with interleaving, intersection and counting. *Theor. Comput. Sci.* 411, 31-33 (2010), 2987–2998. https://doi.org/10.1016/j.tcs.2010.04.036

[14] Jaco Geldenhuys, Brink van der Merwe, and Lynette van Zijl. 2010. Reducing Nondeterministic Finite Automata with SAT Solvers. In *Finite-State Methods and Natural Language Processing*, Anssi Yli-Jyra, Andras Kornai, Jacques Sakarovitch, and Bruce Watson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 81–92.

[15] Jesper G. Henriksen, Madhavan Mukund, K. Narayan Kumar, Milind A. Sohoni, and P. S. Thiagarajan. 2005. A theory of regular MSC languages. *Inf. Comput.* 202, 1 (2005), 1–38. https://doi.org/10.1016/j.ic.2004.08.004

[16] John E. Hopcroft. 1971. *An n Log n Algorithm for Minimizing States in a Finite Automaton.* Technical Report. Stanford, CA, USA.

[17] John E. Hopcroft and Jeffrey D. Ullman. 1979. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley.

[18] Lucian Ilie, Gonzalo Navarro, and Sheng Yu. 2004. *On NFA Reductions.* Springer Berlin Heidelberg, Berlin, Heidelberg, 112–124. https://doi.org/10.1007/978-3-540-27812-2_11

[19] Lucian Iliea and Sheng Yu. 2002. Algorithms for Computing Small NFAs. In *Mathematical Foundations of Computer Science 2002*, Krzysztof Diks and Wojciech Rytter (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 328–340.

[20] Tao Jiang and B. Ravikumar. 1991. Minimal NFA problems are hard. In *Automata, Languages and Programming*, Javier Leach Albert, Burkhard Monien, and Mario Rodriguez Artalejo (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 629–640.

[21] T. Kameda and P. Weiner. 1970. On the State Minimization of Nondeterministic Finite Automata. *IEEE Trans. Comput.* C-19, 7 (1970), 617–627. https://doi.org/10.1109/T-C.1970.222994

[22] Alexander Knapp and Till Mossakowski. 2017. UML Interactions Meet State Machines - An Institutional Approach. In *7th Conf. on Algebra and Coalgebra in Computer Science (CALCO) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 72)*.

[23] Martin Korp, Christian Sternagel, Harald Zankl, and Aart Middeldorp. 2009. Tyrolean Termination Tool 2. In *Rewriting Techniques and Applications*, Ralf Treinen (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 295–304.

[24] Giorgia Lallai, Andrea Pinna, Michele Marchesi, and Roberto Tonelli. 2020. Software Engineering for DApp Smart Contracts Managing Workers Contracts. In *Proceedings of the 3rd Distributed Ledger Technology Workshop Co-located with ITASEC 2020 (CEUR Workshop Proceedings, Vol. 2580)*. CEUR-WS.org.

[25] Ernst Leiss. 1981. Succinct representation of regular languages by boolean automata. *Theoretical Computer Science* 13, 3 (1981), 323–330. https://doi.org/10.1016/S0304-3975(81)80005-9

[26] P. Madhusudan and B. Meenakshi. 2001. Beyond Message Sequence Graphs. In *FST TCS 2001: Foundations of Software Technology and Theoretical Computer Science, 21st Conference, Bangalore, India, December 13-15, 2001, Proceedings (Lecture Notes in Computer Science, Vol. 2245)*, Ramesh Hariharan, Madhavan Mukund, and V. Vinay (Eds.). Springer, 256–267. https://doi.org/10.1007/3-540-45294-X_22

[27] Erwan Mahe. 2022. HIBOU tool. https://github.com/erwanM974/hibou_label.

[28] Erwan Mahe. 2023. applying an incremental approach to NFA synthesis on interaction use cases from the literature. https://github.com/erwanM974/hibou_nfagen_concrete_usecases.

[29] Erwan Mahe. 2023. AUTOUR toolbox. https://github.com/erwanM974/autour_core.

[30] Erwan Mahe. 2023. comparing incremental and compositional approaches to NFA synthesis from interactions. https://github.com/erwanM974/hibou_nfa_synthesis_benchmark.

[31] Erwan Mahe. 2023. coq proof for the equivalence of interaction semantics with co-regions. https://github.com/erwanM974/coq_interaction_semantics_equivalence_with_coregions.
[32] Erwan Mahe. 2023. proof of convergence for interaction simplification. https://github.com/erwanM974/hibou_trs_simplify_empty.
[33] Erwan Mahe, Boutheina Bannour, Christophe Gaston, Arnault Lapitre, and Pascale Le Gall. 2021. A Small-Step Approach to Multi-Trace Checking against Interactions. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing* (Virtual Event, Republic of Korea) *(SAC '21)*. Association for Computing Machinery, New York, NY, USA, 1815–1822. https://doi.org/10.1145/3412841.3442054
[34] Erwan Mahe, Christophe Gaston, and Pascale Le Gall. 2020. Revisiting Semantics of Interactions for Trace Validity Analysis. In *Fundamental Approaches to Software Engineering*, Heike Wehrheim and Jordi Cabot (Eds.). Springer International Publishing, Cham, 482–501.
[35] Erwan Mahe, Christophe Gaston, and Pascale Le Gall. 2022. Equivalence of Denotational and Operational Semantics for Interaction Languages. In *Theoretical Aspects of Software Engineering - 16th International Symposium, TASE 2022, Cluj-Napoca, Romania, July 8-10, 2022, Proceedings (Lecture Notes in Computer Science, Vol. 13299)*, Yamine Aït Ameur and Florin Craciun (Eds.). Springer, 113–130. https://doi.org/10.1007/978-3-031-10363-6_8
[36] Erwan Mahe, Christophe Gaston, and Pascale Le Gall. 2024. Denotational and operational semantics for interaction languages: Application to trace analysis. *Science of Computer Programming* 232 (2024), 103034. https://doi.org/10.1016/j.scico.2023.103034
[37] Sjouke Mauw and Michel Adriaan Reniers. 1997. High-level message sequence charts. In *SDL '97 Time for Testing, SDL, MSC and Trends - 8th International SDL Forum, Proceedings*. Elsevier, 291–306.
[38] Sjouke Mauw and Michel Adriaan Reniers. 1999. Operational Semantics for MSC'96. *Computer Networks* 31, 17 (1999), 1785–1799. https://doi.org/10.1016/S1389-1286(99)00060-2
[39] A.J. Mayer and L.J. Stockmeyer. 1994. The Complexity of Word Problems - This Time with Interleaving. *Information and Computation* 115, 2 (1994), 293–311. https://doi.org/10.1006/inco.1994.1098
[40] B. F. Melnikov. 1999. A new algorithm of the state-minimization for the non-deterministic finite automata. *Korean Journal of Computational and Applied Mathematics* (may 1999), 277–290. https://doi.org/10.1007/BF03014374
[41] Zoltán Micskei and Hélène Waeselynck. 2011. The many meanings of UML 2 Sequence Diagrams: a survey. *Software & Systems Modeling* 10, 4 (2011), 489–514.
[42] Gordon Plotkin. 2004. A Structural Approach to Operational Semantics. *The Journal of Logic and Algebraic Programming* 60-61 (07 2004), 17–139. https://doi.org/10.1016/j.jlap.2004.05.001
[43] Libor Polak. 2004. Minimalizations of NFA Using the Universal Automaton. In *Proceedings of the 9th International Conference on Implementation and Application of Automata* (Kingston, Canada) *(CIAA'04)*. Springer-Verlag, Berlin, Heidelberg, 325–326. https://doi.org/10.1007/978-3-540-30500-2_37
[44] Grigore Roşu and Mahesh Viswanathan. 2003. Testing Extended Regular Language Membership Incrementally by Rewriting. In *Rewriting Techniques and Applications*, Robert Nieuwenhuis (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 499–514.
[45] Koushik Sen and Grigore Roşu. 2003. Generating Optimal Monitors for Extended Regular Expressions. *Electronic Notes in Theoretical Computer Science* 89, 2 (2003), 226–245. https://doi.org/10.1016/S1571-0661(04)81051-X RV '2003, Run-time Verification (Satellite Workshop of CAV '03).
[46] Jocelyn Simmonds, Yuan Gan, Marsha Chechik, Shiva Nejati, Bill O'Farrell, Elena Litani, and Julie Waterhouse. 2009. Runtime Monitoring of Web Service Conversations. *IEEE Trans. Serv. Comput.* 2, 3 (2009), 223–244. https://doi.org/10.1109/TSC.2009.16
[47] Harald Zankl, Bertram Felgenhauer, and Aart Middeldorp. 2011. CSI – A Confluence Tool. In *Automated Deduction – CADE-23*, Nikolaj Bjørner and Viorica Sofronie-Stokkermans (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 499–505.

Received 15 12 2023; accepted 13 01 2024