Recursive Parametric Automata and ϵ-Removal*

Lin Liu[1] and Jonathan Billington[2]

[1] School of Computer and Information Science
[2] Computer Systems Engineering Centre
University of South Australia
{lin.liu | jonathan.billington}@unisa.edu.au

Abstract. This work is motivated by and arose from the parametric verification of communication protocols over unbounded channels, where the channel capacity is the parameter. Verification required the use of finite state automata (FSA) reduction, including ϵ-removal, for a specific infinite family of FSA. This paper generalises this work by introducing Recursive Parametric FSA (RP-FSA), an infinite family of FSA that can be represented recursively in a single parameter. Further, the paper states and proves a necessary and sufficient condition regarding the transformation of a RP-FSA to its language equivalent ϵ-removed family of FSA that is also a RP-FSA in the same parameter. This condition also guarantees a further structural property regarding the RP-FSA and its ϵ-removed family.

Keywords: Parametric Automata, Protocol Verification, Automata Reduction, Language Equivalence

1 Introduction

The Capability Exchange Signalling (CES) protocol [9] is a multimedia control protocol that allows a communication party to inform its peer of its multimedia (e.g. audio and/or video) transmission and reception capabilities. To verify the CES protocol against its service specification, we need to obtain the CES service language: the set of allowable sequences of CES service primitives (i.e. user observable events). Our approach [4] is to extract service languages from state spaces of Coloured Petri Net (CPN) [10] models of service specifications by using automata reduction [2]. The CES service CPN has transitions that model CES service primitives and a transition that models message loss, an internal event that is not to be included in the CES service language (but is needed to capture sequences of primitives). We derive a Finite State Automaton (FSA) from the state space [4] by designating initial and final states and mapping the CPN transition modelling message loss to an ϵ-transition. Then we use FSA reduction to remove ϵ-transitions and non-determinism as steps towards proving language equivalence or inclusion (with respect to the protocol).

Our CPN model [11] of the CES service is parameterised by a positive integer (channel capacity), so it has an infinite family of state spaces. To verify the CES

* This work was partially supported by ARC Discovery Grant, DP0559927.

protocol against its service for any value of the parameter, we firstly obtain symbolic representations for the state spaces and the associated FSAs. In [11] we exploit regularities in the state spaces to obtain a recursive representation. We then derive an infinite family of FSAs from the state spaces. In [12, 14] we proved that the language equivalent ϵ-removed (LE-ER) family of automata can also be represented recursively. Furthermore, after removing non-determinism, we obtain a language equivalent recursively represented family of automata that represents the CES service language for arbitrary capacity [13]. These results lead us to the following generalisation: if a parametric FSA can be represented recursively, under what conditions can its LE-ER (or determinised) family also be represented recursively? In this paper, we determine a sufficient condition for a recursively represented parametric FSA to retain its recursive representation under ϵ-removal. We also determine a necessary condition to satisfy another structural property regarding these families of automata.

We firstly define a (first order) Recursive Parametric FSA (RP-FSA) in terms of a system parameter $l \in \mathcal{N}^+$ (the positive integers). Intuitively, FSA_l comprises a *base component*, FSA_{l-1}, plus another component, ADD_l. We then consider the LE-ER family derived from a RP-FSA, which we denote FSA_l^{ER}. We identify and prove the necessary and sufficient condition for a) FSA_l^{ER} to be a RP-FSA in l and b) the base component of FSA_l^{ER} to be identical to FSA_{l-1}^{ER}. The result contributes to the development of automata theory which we believe will be applicable to the verification of a class of parametric systems, as already demonstrated for the CES service [12–14].

There has been work on Recursive State Machines [1] and Unrestricted Hierarchical State Machines [3] where nodes correspond to ordinary states or to recursive invocations of other state machines. In contrast, we develop recursive representations of an *infinite family* of FSAs in an integer parameter and examine its related LE-ER family. We are not aware of any other work in this area.

The rest of the paper is organised as follows. Section 2 defines a RP-FSA. A theorem regarding transforming a RP-FSA to its LE-ER family is given in Section 3 with its proof in Section 4. Section 5 summarises the paper and suggests future work.

2 Definition of a Recursive Parametric FSA

Our work on the CES service has motivated us to define an infinite family of FSAs over a parameter $l \in \mathcal{N}^+$. Members of the family are related in that the FSA for l includes the FSA for $(l-1)$, and the alphabet and initial state are the same for all members of the family.

Definition 1 *A Recursive Parametric FSA is an infinite family of FSAs in a system parameter $l \in \mathcal{N}^+$, where its l^{th} member, $FSA_l = (V_l, \Sigma, A_l, v_0, F_l)$, is given by*

- *V_l is a finite set of states or nodes that depends on l,*
- *Σ is a finite set, known as the alphabet,*

– $A_l \subseteq V_l \times (\Sigma \cup \{\epsilon\}) \times V_l$, *is the transition relation, where* ϵ *is the empty string,*
– $v_0 \in V_l$ *is the initial state, and every state in* V_l *is accessible from* v_0*,*
– F_l *is the set of final states.*

Given $FSA_1 = (V_1, \Sigma, A_1, v_0, F_1)$*, the family,* FSA_l *(for* $l \geq 2$*) is obtained recursively as follows.*

$$V_l = V_{l-1} \cup V_l^{add} \tag{1}$$

$$A_l = A_{l-1} \cup A_l^{add} \tag{2}$$

$$F_{l-1} \subseteq F_l \tag{3}$$

where

$$V_{l-1} \cap V_l^{add} = \emptyset \tag{4}$$

$$A_l^{add} \subseteq (V_{l-1} \times (\Sigma \cup \{\epsilon\}) \times V_l^{add}) \cup (V_l^{add} \times (\Sigma \cup \{\epsilon\}) \times \ V_l) \tag{5}$$

An example of a RP-FSA is the parametric FSA, FSA_{CES_l}, derived from the state space of our parameterised CES service CPN [12, 14]. Fig. 1 shows FSA_{CES_3}. The subgraph that ignores the shaded nodes (13 to 20) and their associated arcs is FSA_{CES_1}. It has 12 nodes and 33 arcs, including 4 dashed arcs (ϵ-transitions). The initial state and the only final state of FSA_{CES_1} are both node 1. The alphabet, $\Sigma = \{Treq, Tind, Tres, Tcnf, Rreq, Rind, RindU, RindP\}$, is the set of (abbreviated) names of the CES service primitives. From Fig. 1, by ignoring the darkly shaded nodes (17 to 20) and their associated arcs, we obtain a subgraph that is FSA_{CES_2}. We see that FSA_{CES_2} can be constructed from FSA_{CES_1} by adding the 4 grey nodes (13 to 16) and the 16 grey arcs. By denoting $v_2^{add_1} = 13,\ v_2^{add_2} = 14,\ v_2^{add_3} = 15$ and $v_2^{add_4} = 16$, and $v_1^{add_1} = 2,\ v_1^{add_2} = 10,\ v_1^{add_3} = 4$ and $v_1^{add_4} = 12$, the 16 additional arcs are given in Table 1 for $l = 2$. When looking at the whole graph in Fig. 1, FSA_{CES_3} can be constructed from FSA_{CES_2} by adding 4 nodes (17 to 20) and 16 arcs (Table 1 for $l = 3$), when denoting $v_3^{add_1} = 17,\ v_3^{add_2} = 18,\ v_3^{add_3} = 19$ and $v_3^{add_4} = 20$.

It has been shown in [12, 14] that, for $l \geq 2$, FSA_{CES_l} can be recursively constructed in the following way:

$$V_l = V_{l-1} \cup V_l^{add}$$
$$A_l = A_{l-1} \cup A_l^{add}$$
$$F_l = F_{l-1} = \{v_0\} = \{1\}$$

where

$$V_l^{add} = \{v_l^{add_i} \mid i \in \{1, \ldots, 4\}\},\ V_{l-1} \cap V_l^{add} = \emptyset$$

and A_l^{add} is given in Table 1, where $v_1^{add_1} = 2,\ v_1^{add_2} = 10,\ v_1^{add_3} = 4,\ v_1^{add_4} = 12$ (Fig. 1). From Definition 1, the family FSA_{CES_l} ($l \in \mathcal{N}^+$) is a RP-FSA.

3 RP-FSA and ϵ-Removal

In this section, we propose the necessary and sufficient condition for the LE-ER automata family derived from FSA_l ($l \in \mathcal{N}^+$) to be a RP-FSA in l, where its base component is the LE-ER automaton of FSA_{l-1}. We firstly provide some definitions related to FSA_l, where $l \geq 2$.

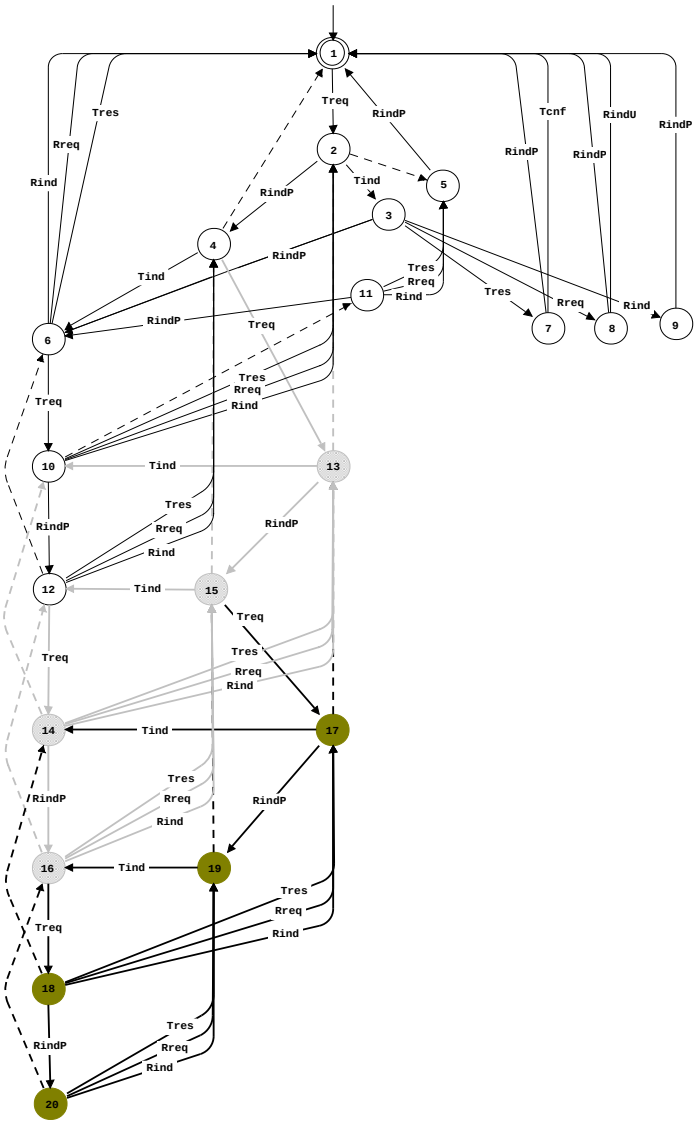

Fig. 1. FSA_{CES_3}

Table 1. A_l^{add} $(l \geq 2)$

Name	Arc	Name	Arc	Name	Arc
$a_l^{add_1}$	$(v_{l-1}^{add_3}, Treq, v_l^{add_1})$	$a_l^{add_2}$	$(v_{l-1}^{add_4}, Treq, v_l^{add_2})$	$a_l^{add_3}$	$(v_l^{add_1}, RindP, v_l^{add_3})$
$a_l^{add_4}$	$(v_l^{add_1}, Tind, v_{l-1}^{add_2})$	$a_l^{add_5}$	$(v_l^{add_1}, \epsilon, v_{l-1}^{add_1})$	$a_l^{add_6}$	$(v_l^{add_2}, RindP, v_l^{add_4})$
$a_l^{add_7}$	$(v_l^{add_2}, \epsilon, v_{l-1}^{add_2})$	$a_l^{add_8}$	$(v_l^{add_2}, Tres, v_l^{add_1})$	$a_l^{add_9}$	$(v_l^{add_2}, Rreq, v_l^{add_1})$
$a_l^{add_{10}}$	$(v_l^{add_2}, Rind, v_l^{add_1})$	$a_l^{add_{11}}$	$(v_l^{add_3}, Tind, v_{l-1}^{add_4})$	$a_l^{add_{12}}$	$(v_l^{add_3}, \epsilon, v_{l-1}^{add_3})$
$a_l^{add_{13}}$	$(v_l^{add_4}, Tres, v_l^{add_3})$	$a_l^{add_{14}}$	$(v_l^{add_4}, Rreq, v_l^{add_3})$	$a_l^{add_{15}}$	$(v_l^{add_4}, Rind, v_l^{add_3})$
$a_l^{add_{16}}$	$(v_l^{add_4}, \epsilon, v_{l-1}^{add_4})$				

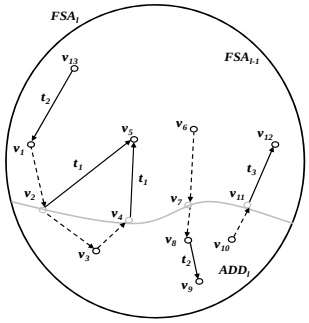

Fig. 2. An illustration of the definitions related to a RP-FSA

Definition 2 *A node $v \in V_{l-1}$ is an* **entry node** *to FSA_{l-1}, iff $\exists\ (v', t, v) \in A_l$ where $v' \in V_l^{add}$. The set of entry nodes of FSA_{l-1} is denoted by V_{l-1}^{EN}.*

Definition 3 *A node $v \in V_{l-1}$ is an* **exit node** *from FSA_{l-1}, iff $\exists\ (v, t, v') \in A_l$ where $v' \in V_l^{add}$. The set of exit nodes of FSA_{l-1} is denoted by V_{l-1}^{EX}.*

Definition 4 *The* **base component** *of FSA_l is FSA_{l-1}, and the* **added component** *of FSA_l is a labelled directed graph, $ADD_l = (V_l^{add} \cup V_{l-1}^{EN} \cup V_{l-1}^{EX}, A_l^{add})$.*

Definition 5 *A node of FSA_l is a* **candidate node** *(denoted v^C), iff $\exists\ (v', t, v^C) \in A_l$ and $t \in \Sigma$. We denote the set of candidate nodes of FSA_l, V_l^C, which comprises $V_{l-1}^C = \{v^C \mid (v, t, v^C) \in A_{l-1}, t \in \Sigma\}$ and $V_l^{addC} = V_l^C \setminus V_{l-1}^C = \{v^C \mid (v, t, v^C) \in A_l^{add}$ and $(v', t', v^C) \notin A_{l-1}, t, t' \in \Sigma\}$.*

Definition 6 *A finite sequence of transitions of FSA_l that starts at node vs and ends at node ve is a* **candidate sequence** *(denoted $s_{(vs,ve,t)}^C$), iff for $n \geq 1$*

$$s_{(vs,ve,t)}^C = vs \xrightarrow{\epsilon} v_1 \xrightarrow{\epsilon} \ldots \xrightarrow{\epsilon} v_n \xrightarrow{t} ve \tag{6}$$

where $t \in \Sigma$. The set of candidate sequences of FSA_l is denoted by S_l^C.

Definition 7 *A finite sequence of transitions of FSA_l that starts at node vs and ends at node ve is a* **return sequence** *(denoted $s_{(vs,ve,t)}^R$), iff for $n \geq 1$, $s_{(vs,ve,t)}^R \in S_l^C$, $vs, ve \in V_{l-1}$ and $\exists\ v_i \in \{v_1, \ldots, v_n\}$ such that $v_i \in V_l^{add}$. The set of return sequences of FSA_l is denoted by S_l^R.*

Definition 8 *An* **empty cycle** *of a FSA is a sequence of ϵ-transitions that starts and ends in the same state.*

Fig. 2 illustrates the definitions, where dashed arcs show ϵ-transitions. The base component and added components of FSA_l are the parts above and under the curly grey line respectively. v_4 and v_{11} are entry nodes, v_2 and v_7 are exit nodes, and they are drawn at the boundary of the base and added components. v_1, v_5, v_9 and v_{12} are candidate nodes. $s_1 = v_1 \xrightarrow{\epsilon} v_2 \xrightarrow{\epsilon} v_3 \xrightarrow{\epsilon} v_4 \xrightarrow{t_1} v_5$, $s_2 = v_1 \xrightarrow{\epsilon} v_2 \xrightarrow{t_1} v_5$, $s_3 = v_6 \xrightarrow{\epsilon} v_7 \xrightarrow{\epsilon} v_8 \xrightarrow{t_2} v_9$, $s_4 = v_{10} \xrightarrow{\epsilon} v_{11} \xrightarrow{t_3} v_{12}$, are candidate sequences, and s_1 is a return sequence while the other three are not.

Another example is FSA_{CES_l}. In Table 1, $v_{l-1}^{add_3}$ and $v_{l-1}^{add_4}$ are exit nodes because, from each of them, there is a transition to a node in V_l^{add}, i.e. $a_l^{add_1}$ and $a_l^{add_2}$. $v_{l-1}^{add_1}$, $v_{l-1}^{add_2}$, $v_{l-1}^{add_3}$ and $v_{l-1}^{add_4}$ are entry nodes because there are transitions that start at nodes of V_l^{add} and end at each of them (i.e. transitions $a_l^{add_4}$, $a_l^{add_5}$, $a_l^{add_7}$, $a_l^{add_{11}}$, $a_l^{add_{12}}$, and $a_l^{add_{16}}$). FSA_{CES_l} contains candidate sequences, but it does not have return sequences because a return sequence requires at least one ϵ-transition that starts at a node in V_{l-1} and ends at a node in V_l^{add}. From Table 1 and Fig. 1 (FSA_{CES_1} only), FSA_{CES_l} does not have such ϵ-transitions.

Now we formalise the necessary and sufficient condition in the theorem below.

Theorem 1 *For $l \in \mathcal{N}^+$, let $FSA_l = (V_l, \Sigma, A_l, v_0, F_l)$ be a RP-FSA without empty cycles, and $FSA_l^{ER} = (V_l^{ER}, \Sigma, A_l^{ER}, v_0, F_l^{ER})$ its family of LE-ER automata. FSA_l^{ER} is a RP-FSA in l, where, for $l \geq 2$, its base component is FSA_{l-1}^{ER} iff for $vs, ve \in V_{l-1}^C$, $s_{(vs,ve,t)}^R \in S_l^R \Rightarrow (s_{(vs,ve,t)}^C \in S_{l-1}^C$ or $(vs, t, ve) \in A_{l-1})$.*

The theorem states that when the presence of a return sequence ($s_{(vs,ve,t)}^R$) between two candidate nodes in FSA_l, implies the presence of either a corresponding candidate sequence ($s_{(vs,ve,t)}^C$) between the same candidate nodes in the base component of FSA_l (i.e. FSA_{l-1}) or a direct transition between them ($(vs, t, ve) \in A_{l-1}$), then the LE-ER automaton of FSA_l is also a RP-FSA in l, with its base component being the LE-ER automaton of FSA_{l-1}. The converse also holds.

Referring to Fig. 2 and assuming that s_1 is the only return sequence, this RP-FSA satisfies the condition as a candidate sequence s_2 that belongs to FSA_{l-1} is between v_1 and v_5 and the last transitions of s_2 and s_1 are both t_1. If s_2 did not exist (assuming no other candidate sequences from v_1 to v_5 with t_1 as their last transition and that (v_1, t_1, v_5) does not exist), the FSA would not satisfy the condition. As mentioned earlier, FSA_{CES_l} does not have return sequences, so the condition is satisfied, and its LE-ER FSA can be represented recursively based on the LE-ER FSA of $FSA_{CES_{l-1}}$, a result that we proved in [12, 14].

4 Proving the Necessary and Sufficient Condition

4.1 Preliminaries

We can remove ϵ-transitions by constructing ϵ-closures [15]. The ϵ-closure of a state or a set of states is the set of all states that are accessible by only ϵ-transitions from that state or set of states. Our goal is to determine the condition under which the LE-ER automaton of FSA_l is also a RP-FSA, where the base component of the LE-ER automaton is the LE-ER automaton of the base component of FSA_l. To derive the condition we remove ϵ-transitions in the base component (FSA_{l-1}) and the added component (ADD_l) separately, so that it is easier to identify if the LE-ER automaton of FSA_{l-1} is included in the LE-ER automaton of FSA_l. When an ϵ-closure includes ϵ-transitions of FSA_{l-1} and ADD_l, the above approach removes all the ϵ-transitions in an ϵ-closure at one time, and is thus not appropriate for our procedure.

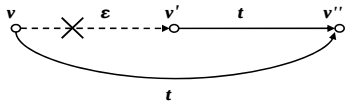

Fig. 3. Removing an ϵ-transition using Algorithm 1

Barrett et al [2] present an incremental approach. To transform a FSA to its LE-ER automaton, we firstly remove empty cycles (Definition 8), then remove all the remaining ϵ-transitions one by one. The algorithm for removing ϵ-transitions from a FSA without empty cycles is formalised as follows, based on [2].

Algorithm 1 *For $FSA = (V, \Sigma, A, v_0, F)$ without empty cycles, its LE-ER automaton, $FSA^{ER} = (V^{ER}, \Sigma, A^{ER}, v_0, F^{ER})$ is created as follows:*

1. *Initially, let $V^{ER} = V$, $A^{ER} = A$, $F^{ER} = F$.*
2. **While** $A^{ER}_{\epsilon} = \{(v, \epsilon, v') \mid (v, \epsilon, v') \in A^{ER}\} \neq \emptyset$, **do***:*
   (a) *choose any $(v, \epsilon, v') \in A^{ER}_{\epsilon}$*
   (b) *update A^{ER} to $(A^{ER} \setminus \{(v, \epsilon, v')\}) \cup \{(v, t, v'') \mid (v', t, v'') \in A^{ER},\ t \in (\Sigma \cup \{\epsilon\})\}$*
   (c) *if $v' \in F^{ER}$, update F^{ER} to $F^{ER} \cup \{v\}$.*
3. *Update V^{ER} to $V^{ER} \setminus \{v \mid \nexists s = v_0 \xrightarrow{t_0} \ldots v_n \xrightarrow{t_n} v, n \geq 0, t_i \in \Sigma, 0 \leq i \leq n\}$*
4. *Update A^{ER} to $A^{ER} \setminus \{(v, t, v') \mid v \notin V^{ER}$ or $v' \notin V^{ER}\}$*

As shown in Fig. 3, when using the algorithm to remove (v, ϵ, v'), if $(v', t, v'') \in A^{ER}$, (v, t, v'') is included in A^{ER} (step 2(b)). Furthermore, if v' is a final state of FSA, v is included in F^{ER} (step 2(c)). After all the ϵ-transitions are removed by following step 2, some nodes of V^{ER} may become inaccessible from the initial state v_0, so in steps 3 and 4, we exclude inaccessible states and their associated transitions from V^{ER} and A^{ER} respectively.

We now state three lemmas to be used in the proof of the theorem. Lemma 1 gives the result of removing a sequence of ϵ-transitions. Lemma 2 states the necessary and sufficient condition for adding a non-ϵ-transition when removing ϵ-transitions from FSA_l. Lemma 3 presents the necessary and sufficient condition for a node of FSA_l to remain accessible after all ϵ-transitions are removed.

Lemma 1 *Let $FSA = (V, \Sigma, A, v_0, F)$ be an FSA that includes ϵ-transitions. Consider a transition sequence $s = vs \xrightarrow{\epsilon} v_1 \xrightarrow{\epsilon} \ldots \xrightarrow{\epsilon} v_n \xrightarrow{t} ve$ where $n \geq 1$ and $t \in \Sigma$. After the n ϵ-transitions are removed from this sequence using steps 1 and 2 of Algorithm 1, transitions $\{(v, t, ve) \mid v \in \{vs, v_1, \ldots, v_n\}\}$ are in A^{ER}.*

Proof Referring to Fig. 4, we firstly remove the second last transition in s, (v_{n-1}, ϵ, v_n), then the preceding ϵ-transition, and keep moving backwards until the first ϵ-transition, (vs, ϵ, v_1) is removed.

On removing (v_{n-1}, ϵ, v_n), arc (v_{n-1}, t, ve) is added according to Algorithm 1. When the third last transition $(v_{n-2}, \epsilon, v_{n-1})$ (if $n \geq 3$) is removed, because (v_{n-1}, t, ve) has just been added, (v_{n-2}, t, ve) has to be added. In general, when removing any ϵ-transition (v_i, ϵ, v_{i+1}) $(1 \leq i < n-1)$ in this way, because (v_{i+1}, t, ve) exists, (v_i, t, ve) has to be added. This process is continued until (vs, ϵ, v_1) is removed while (vs, t, ve) is added because (v_1, t, ve) has been added previously or was the last in the sequence if $n = 1$.

So after the n ϵ-transitions are removed, the set of arcs $\{(v, t, ve) \mid v \in \{vs, v_1, \ldots, v_n\}\}$, $n \geq 1$ are added to A^{ER}. Hence, Lemma 1 is proved. □

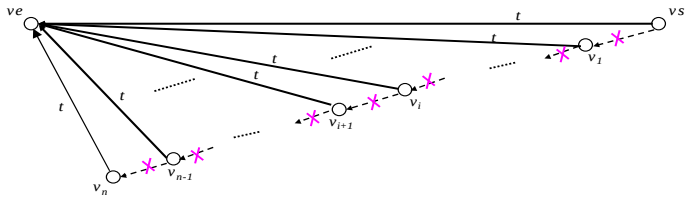

Fig. 4. Removing ϵ-transitions from the transition sequence in Lemma 1

Lemma 2 *For $t \in \Sigma$ and $(vs, t, ve) \notin A_l$, when applying Algorithm 1 to FSA_l, after steps 1 and 2 are completed, a non-ϵ-transition (vs, t, ve) is in A_l^{ER} iff $s^C_{(vs,ve,t)} \in S_l^C$.*

Proof The sufficient condition follows immediately from Lemma 1 as (vs, t, ve) is one of the arcs added when removing ϵ-transitions from $s^C_{(vs,ve,t)}$.

The necessary condition states that if transition (vs, t, ve) is added then there must be a candidate sequence $s^C_{(vs,ve,t)}$ (and $(vs, t, ve) \notin A_l$). To show this holds, we prove its contrapositive, i.e. if there does not exist $s^C_{(vs,ve,t)} \in S_l^C$, (vs, t, ve) can not be added when removing ϵ-transitions from FSA_l.

From Algorithm 1 if there does not exist a transition sequence from vs to ve at all, no new arc (vs, t, ve) can be added. So we only need to show that, (vs, t, ve) still can not be added when none of sequences from vs to ve are candidate sequences, i.e. any sequence from vs to ve is of the form $s' = vs \xrightarrow{t_1} v_1 \xrightarrow{t_2} \ldots \xrightarrow{t_{n-1}} v_n \xrightarrow{t} ve$, where $n \geq 1$, $t \in \Sigma$, and $\exists t' \in \{t_1, \ldots, t_{n-1}\}$ such that $t' \neq \epsilon$, a sequence that can only be made up of a chain of non-ϵ-transitions, or of both ϵ-transitions and non-ϵ-transitions.

In the first case, s' contains only non-ϵ-transitions, no ϵ-transitions to be removed from s', hence no arcs can be added. For the second case, assume that (vs, t', v_1) is a non-ϵ-transition. According to Algorithm 1, when removing any of the ϵ-transitions between v_1 and v_n, it is not possible to add an arc that starts from vs because vs is not the source node of an ϵ-transition. Now assume that (v_m, t', v_{m+1}) $(1 \leq m \leq n-1)$ is the first non-ϵ-transition we encounter in the chain (i.e. all preceding transitions are ϵ-transitions), then from Lemma 1, arcs $\{(v, t', v_{m+1}) \mid v \in \{vs, v_1, \ldots, v_{m-1}\}\}$ will be added when all the ϵ-transitions from vs to v_m are removed. However, these added arcs are not ϵ-transitions because $t' \neq \epsilon$. When removing any ϵ-transitions between v_{m+1} and v_n, again according to Algorithm 1, it is not possible to add an arc starting from vs. Therefore, for the transition sequence s' described above, arc (vs, t, ve) cannot be added.

Since we have used s' to represent any of the possible transition sequences existing from vs to ve, we have proved that if all of the transition sequences from vs to ve are not candidate sequences, then (vs, t, ve) can not be added. So the necessary condition is proved as well.

Therefore Lemma 2 holds. □

Lemma 3 *When applying Algorithm 1 to FSA_l, after steps 1 and 2 are completed, a state v ($v \neq v_0$) remains accessible iff $v \in V_l^C$ (a candidate node).*

Proof Because all the states of FSA_l are accessible, there must exist at least one transition sequence from v_0 to a state v. It can be seen that a transition sequence from v_0 to a predecessor of v, v', may be of one of the 3 types: Type 1: a sequence that comprises ϵ-transitions only; Type 2: a sequence that comprises non-ϵ-transitions only; Type 3: a sequence that comprises both ϵ and non-ϵ-transitions.

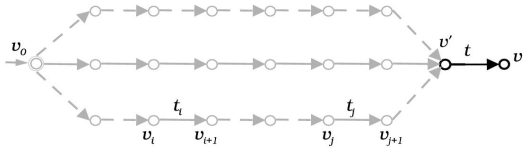

Fig. 5. Example transition sequences from v_0 to a predecessor (v') of node v

Fig. 5 shows an example for each of the 3 types of transition sequences from v_0 to v'. In this figure, an ϵ-transition is drawn as a dashed arc and a non-ϵ-transition is shown as a solid arc. The sequence on the top that comprises dashed arcs only is a type 1 sequence. The sequence in the middle that has solid arcs only is a type 2 sequence, and the sequence at the bottom that has two solid arcs and some dashed arcs is of type 3.

In the following we prove the sufficient condition first, i.e. if there exists (v', t, v) and $t \in \Sigma$, v is still accessible after ϵ-removal.

With a type 1 sequence, from Lemma 1, on the removal of ϵ-transitions the transition (v_0, t, v) is added, making v directly accessible.

For a type 2 sequence no ϵ-transitions are removed and the sequence remains ensuring v is accessible.

For a type 3 sequence, consider the example sequence shown at the bottom of Fig. 5. It has two non-ϵ-transitions (v_i, t_i, v_{i+1}) and (v_j, t_j, v_{j+1}), and $t_i \neq \epsilon$, $t_j \neq \epsilon$. When removing all the ϵ-transitions before v_i in the sequence, according to Lemma 1, from each node in the sequence before v_i, an arc labelled with t_i and pointing to v_{i+1} is added. So we have (v_0, t_i, v_{i+1}) in A_l^{ER}. When ϵ-transitions between v_{i+1} and v_j are removed, similarly from each node in the sequence from v_{i+1} to v_{j-1} an arc labelled with t_j and pointing to v_{j+1} is added. So we have (v_{i+1}, t_j, v_{j+1}) in A_l^{ER}. Finally when the ϵ-transitions between v_{j+1} and v' are removed, from each node of this part of the sequence (including v_{j+1} but excluding v'), a non-ϵ-transition that points to v and is labelled with t is added, including (v_{j+1}, t, v). Therefore, from v_0, via three non-ϵ-transitions (v_0, t_i, v_{i+1}), (v_{i+1}, t_j, v_{j+1}) and (v_{j+1}, t, v), we can reach v. So v must be accessible after all of the ϵ-transitions of FSA_l are removed.

The necessary condition states if v remains accessible after ϵ-removal, then there must exist $(v', t, v) \in A_l$ and $t \in \Sigma$. To show that this statement is correct, we prove its contrapositive, that is, if there does not exist $(v', t, v) \in A_l$ where $t \in \Sigma$, v is inaccessible after ϵ-removal.

As v is accessible before removing ϵ-removal, there must be a set of arcs $A_l^{2v} \subseteq A_l$ such that $A_l^{2v} = \{a \mid a = (v', t, v) \in A_l\}$. Because $\nexists(v', t, v) \in A_l$ and $t \in \Sigma$, all the arcs in A_l^{2v} are ϵ-transitions. Furthermore, as no empty cycles are allowed in FSA_l, so we have $A_l^{2v} = \{a \mid a = (v', \epsilon, v) \in A_l, v' \neq v\}$. Then if applying Algorithm 1 to FSA_l, transitions $\{(v', t', v'') \mid \exists(v, t', v'') \in A_l, v'' \neq v'\}$ are added, and none of the added arcs pointing to v. Meanwhile, all the arcs of A_l^{2v} are removed. So no arcs will point to v after ϵ-removal, i.e. v has become inaccessible. The necessary condition has been proved.

Therefore Lemma 3 holds. □

From Lemma 2, when removing ϵ-transitions, transitions are added only between starting and ending nodes of candidate sequences (Definition 6). So such sequences are candidates for adding new transitions, that is why we call them candidate sequences. From Lemma 3 the destination node of a non-ϵ-transition is a candidate to be kept in the LE-ER automaton of FSA_l as it remains accessible after ϵ-removal, so we call such a node a candidate node (Definition 5).

4.2 Proving the Sufficient Condition

Lemma 4 *For $l \in \mathcal{N}^+$, let $FSA_l = (V_l, \Sigma, A_l, v_{0_l}, F_l)$ be a RP-FSA without empty cycles, and $FSA_l^{ER} = (V_l^{ER}, \Sigma, A_l^{ER}, v_0, F_l^{ER})$ its LE-ER automata family. If for $vs, ve \in V_{l-1}^C$, $s_{(vs,ve,t)}^R \in S_l^R \Rightarrow (s_{(vs,ve,t)}^C \in S_{l-1}^C$ or $(vs, t, ve) \in A_{l-1})$, then for $l \geq 2$,*

$$V_l^{ER} = V_{l-1}^{ER} \cup (V_l^{ER})^{add} \tag{7}$$

$$A_l^{ER} = A_{l-1}^{ER} \cup (A_l^{ER})^{add} \tag{8}$$

$$F_{l-1}^{ER} \subseteq F_l^{ER} \tag{9}$$

where

$$V_{l-1}^{ER} \cap (V_l^{ER})^{add} = \emptyset \tag{10}$$

$$(A_l^{ER})^{add} \subseteq (V_{l-1}^{ER} \times \Sigma \times (V_l^{ER})^{add}) \cup ((V_l^{ER})^{add} \times \Sigma \times \ V_l^{ER}) \tag{11}$$

Proof The proof is structured into 3 lemmas concerning the states of FSA_l^{ER} (Lemma 5), its arcs (Lemma 6) and its final states (Lemma 7).

Lemma 5 *Let FSA_l and FSA_l^{ER} be the automata referred to in Lemma 4, then for $l \geq 2$, the set of states of FSA_l^{ER} is given by Equations (7) and (10).*

Proof From step 3 of Algorithm 1, V_l^{ER} comprises states of FSA_l that remain accessible after ϵ-removal. Based on Lemma 3, a state of FSA_l is accessible after ϵ-removal iff it is a candidate node, i.e. $V_l^{ER} = V_l^C$, for $l \in \mathcal{N}^+$. Thus we have $V_{l-1}^{ER} = V_{l-1}^C$. From Definition 5, $V_l^C = V_{l-1}^C \cup V_l^{addC}$ and $V_{l-1}^C \cap V_l^{addC} = \emptyset$. Let $(V_l^{ER})^{add} = V_l^{addC}$, then $V_l^{ER} = V_{l-1}^{ER} \cup (V_l^{ER})^{add}$ and $V_{l-1}^{ER} \cap (V_l^{ER})^{add} = \emptyset$. So Equations (7) and (10) always hold, thus Lemma 5 is proved. □

Lemma 6 *Let FSA_l and FSA_l^{ER} be the automata referred to in Lemma 4. If for $vs, ve \in V_{l-1}^C$, $s_{(vs,ve,t)}^R \in S_l^R \Rightarrow (s_{(vs,ve,t)}^C \in S_{l-1}^C$ or $(vs, t, ve) \in A_{l-1})$, then for $l \geq 2$, the set of arcs of FSA_l^{ER} is given by (8) and (11).*

Proof As A_l^{ER} is obtained from A_l by removing and adding transitions, we have

$$A_l^{ER} = (A_l \cup A_l^A) \setminus A_l^D \tag{12}$$

where A_l^A represents the set of non-ϵ-transitions added and A_l^D the set of transitions that are removed. From Algorithm 1, the set of transitions removed from A_l comprises ϵ-transitions of FSA_l and the transitions whose source and/or destination nodes become inaccessible. From Lemma 3, the set of inaccessible states is $(V_l \setminus V_l^C)$. Therefore we have,

$$\begin{aligned} A_l^D = &\{(v, \epsilon, v') \in A_l\} \\ &\cup \{(v, t, v') \in A_l \mid t \in \Sigma,\ v \in (V_l \setminus V_l^C) \textit{ or } v' \in (V_l \setminus V_l^C)\} \end{aligned} \tag{13}$$

Note that if the source and/or destination nodes of a transition added in step 2 become inaccessible, this transition is removed in step 4. However, we do not include these transitions in A_l^D as we use A_l^A to only represent the non-ϵ-transitions that are added between candidate nodes (which remain accessible after ϵ-removal).

Because $A_l = A_{l-1} \cup A_l^{add}$ (Equation (2)), A_l^D can be represented as $A_l^D = A_{l-1}^D \cup (A_l^{add})^D$, where

$$\begin{aligned} A_{l-1}^D = &\{(v, \epsilon, v') \in A_{l-1}\} \\ &\cup \{(v, t, v') \in A_{l-1} \mid t \in \Sigma,\ v \in (V_{l-1} \setminus V_{l-1}^C) \textit{ or } v' \in (V_{l-1} \setminus V_{l-1}^C)\} \\ (A_l^{add})^D = &\{(v, \epsilon, v') \in A_l^{add}\} \\ &\cup \{(v, t, v') \in A_l^{add} \mid t \in \Sigma,\ v \in (V_l \setminus V_l^C) \textit{ or } v' \in (V_l \setminus V_l^C)\} \end{aligned} \tag{14}$$

From Definition 5, V_{l-1}^C comprises all the states in V_{l-1} that remain accessible via states in V_{l-1}, i.e. $(V_{l-1} \setminus V_{l-1}^C)$ is the set of states of V_{l-1} that are inaccessible when only considering V_{l-1}. So A_{l-1}^D comprises all the transitions removed from A_{l-1} when transforming FSA_{l-1} to FSA_{l-1}^{ER} by itself. So we can revise Equation (12) to:

$$A_l^{ER} = (A_{l-1} \setminus A_{l-1}^D) \cup (A_l^{add} \setminus (A_l^{add})^D) \cup A_l^A \tag{15}$$

We now look at the details of A_l^A. From Lemma 2, a non-ϵ-transition is added between vs and ve iff there is $s_{(vs,ve,t)}^C \in S_l^C$. So we have:

$$A_l^A = \{(vs, t, ve) \mid vs, ve \in V_l^C, t \in \Sigma \textit{ and } s_{(vs,ve,t)}^C \in S_l^C\} \tag{16}$$

As $V_l^C = V_{l-1}^C \cup V_l^{addC}$, there are four cases for the location of the starting and ending nodes of $s_{(vs,ve,t)}^C$. They are **case 1**: $vs, ve \in V_{l-1}^C$; **case 2**: $vs \in V_{l-1}^C$, $ve \in V_l^{addC}$; **case 3**: $vs \in V_l^{addC}$, $ve \in V_{l-1}^C$; and **case 4**: $vs, ve \in V_l^{addC}$.

We use A_l^{A1} to represent all the transitions in A_l^A that are added based on case 1 candidate sequences, and A_l^{A234} for all the transitions that are added based on cases 2, 3 and 4. So $A_l^A = A_l^{A1} \cup A_l^{A234}$, where

$$A_l^{A1} = \{(vs, t, ve) \in A_l^A \mid vs, ve \in V_{l-1}^C\} \tag{17}$$

$$A_l^{A234} = \{(vs, t, ve) \in A_l^A \mid vs \in V_l^{addC} \textit{ or } ve \in V_l^{addC}\} \tag{18}$$

We have proved $V_l^{ER} = V_{l-1}^{ER} \cup (V_l^{ER})^{add}$, with $V_{l-1}^{ER} = V_{l-1}^C$, $(V_l^{ER})^{add} = V_l^{addC}$. So

$$A_l^{A234} \subseteq (V_{l-1}^{ER} \times \Sigma \times (V_l^{ER})^{add}) \cup ((V_l^{ER})^{add} \times \Sigma \times\ V_l^{ER}) \tag{19}$$

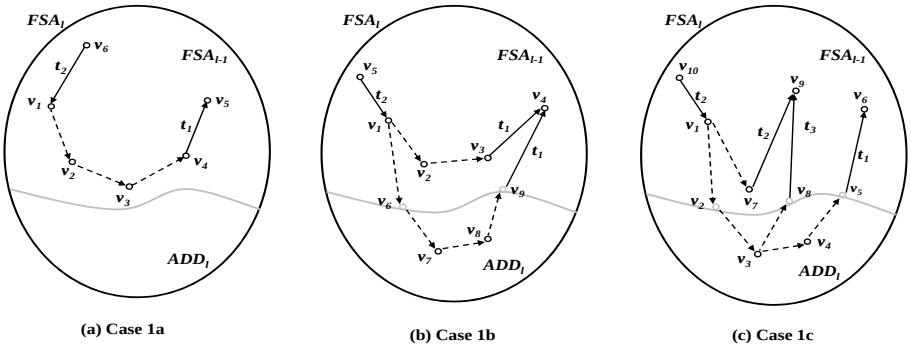

Fig. 6. An illustration of Case 1 sequences

That is, a transition in A_l^{A234} is in $(A_l^{ER})^{add}$ (see (11)).

The candidate sequence, based on which a transition in A_l^{A1} is added, can belong to FSA_{l-1}, denoted **case 1a** (see the example sequence from v_1 to v_5 in Fig. 6(a)), or be a return sequence. When it is a return sequence, $s^R_{(vs,ve,t)}$, there are two cases. In the first case (**case 1b**) there is also either a direct transition $(vs, t, ve) \in A_{l-1}$ or a candidate sequence, $s^C_{(vs,ve,t)}$ that belongs to FSA_{l-1}. Fig. 6(b) provides an example where $s_1 = v_1 \xrightarrow{\epsilon} v_6 \xrightarrow{\epsilon} v_7 \xrightarrow{\epsilon} v_8 \xrightarrow{\epsilon} v_9 \xrightarrow{t_1} v_4$ is a return sequence, and $s_2 = v_1 \xrightarrow{\epsilon} v_2 \xrightarrow{\epsilon} v_3 \xrightarrow{t_1} v_4$ belongs to FSA_{l-1}. s_1 and s_2 have the same starting and ending nodes and their last transitions are both labelled by t_1. In the second case (**case 1c**), there is neither a direct transition $(vs, t, ve) \in A_{l-1}$ nor a candidate sequence, $s^C_{(vs,ve,t)}$ that belongs to FSA_{l-1} ($s^C_{(vs,ve,t)} \notin S^C_{l-1}$). Fig. 6(c) illustrates case 1c, where $s_1 = v_1 \xrightarrow{\epsilon} v_2 \xrightarrow{\epsilon} v_3 \xrightarrow{\epsilon} v_4 \xrightarrow{\epsilon} v_5 \xrightarrow{t_1} v_6$ and $s_2 = v_1 \xrightarrow{\epsilon} v_2 \xrightarrow{\epsilon} v_3 \xrightarrow{\epsilon} v_8 \xrightarrow{t_3} v_9$ are return sequences. From v_1 to v_6 there is no direct transition nor a candidate sequence of FSA_{l-1}. From v_1 to v_9, there is no direct transition and only one candidate sequence in FSA_{l-1} but its last transition is labelled by t_2, rather than t_3, so it is not a corresponding candidate sequence.

In case 1a, (vs, t, ve) is added and must be in A^{ER}_{l-1}, because when using Algorithm 1 to transform FSA_{l-1} to FSA^{ER}_{l-1} (by itself), (vs, t, ve) is included in A^{ER}_{l-1}, from Lemma 1. For case 1b, if $(vs, t, ve) \in A_{l-1}$, then Algorithm 1 will not remove it (as $t \in \Sigma$ and from Lemma 3) so it must be in A^{ER}_{l-1}. Alternatively, if $s^C_{(vs,ve,t)} \in S^C_{l-1}$, then by Lemma 2, $(vs, ve, t) \in A^{ER}_{l-1}$. Hence the return sequences from vs and ve only add transitions that are already in A^{ER}_{l-1} and therefore have no effect. So far the transitions added in cases 1a and 1b are all in A^{ER}_{l-1} and the transitions added in cases 2 to 4 are in $(A_l^{ER})^{add}$.

For case 1c, since $(vs, t, ve) \notin A_{l-1}$ it will only be included in A^{ER}_{l-1} if $s^C_{(vs,ve,t)} \in S^C_{l-1}$ from Lemma 2. Since there is no such candidate sequence, (vs, t, ve) is not in A^{ER}_{l-1}. For example, referring to Fig. 6(c), (v_1, t_3, v_9) is added when using Algorithm 1 on FSA_l, but it is not in A^{ER}_{l-1} because, when removing ϵ-transitions from FSA_{l-1}, $s^C_{(v_1,v_9,t_3)}$ is not in S^C_{l-1} (Lemma 2). Similarly,

(v_1, t_1, v_6) is added when transforming FSA_l based on Lemma 2, but it is also not in A^{ER}_{l-1} as $s^C_{(v_1,v_6,t_1)}$ is not in S^C_{l-1}. The added transition (vs, t, ve) is not in $(A^{ER}_l)^{add}$ either because a transition of $(A^{ER}_l)^{add}$ must have its source and/or destination node in $(V^{ER}_l)^{add}$ (see (11)). Thus if we exclude case 1c, so that for $vs, ve \in V^C_{l-1}$, $s^R_{(vs,ve,t)} \in S^R_l \Rightarrow (s^C_{(vs,ve,t)} \in S^C_{l-1}$ *or* $(vs, t, ve) \in A_{l-1})$, then the base component of the LE-ER FSA_l will be FSA^{ER}_{l-1} because its set of transitions will be A^{ER}_{l-1} (rather than a superset). This is the condition stated in Lemma 4 (and Theorem 1). Thus we have shown that under this condition, all the added arcs during ϵ-removal for FSA_l are in A^{ER}_{l-1} or $(A^{ER}_l)^{add}$, and we can revise the representation of A^{ER}_l from Equation (15) to:

$$\begin{aligned} A^{ER}_l &= (A_{l-1} \setminus A^D_{l-1}) \cup (A^{add}_l \setminus (A^{add}_l)^D) \cup A^{A1ab}_l \cup A^{A234}_l \\ &= ((A_{l-1} \setminus A^D_{l-1}) \cup A^{A1ab}_l) \cup ((A^{add}_l \setminus (A^{add}_l)^D) \cup A^{A234}_l \end{aligned} \tag{20}$$

where

$$\begin{aligned} A^{A1ab}_l = \{(vs, t, ve) \mid vs, ve \in V^C_{l-1}, t \in \Sigma \textit{ and} \\ s^R_{(vs,ve,t)} \in S^R_l \Rightarrow (s^C_{(vs,ve,t)} \in S^C_{l-1} \textit{ or } (vs, t, ve) \in A_{l-1})\} \end{aligned} \tag{21}$$

comprises all the transitions added when transforming FSA_{l-1} to FSA^{ER}_{l-1}. A^D_{l-1} comprises all transitions removed in the same context, so under this condition, $(A_{l-1} \setminus A^D_{l-1}) \cup A^{A1ab}_l$ consists of all the transitions of FSA^{ER}_{l-1}, i.e.

$$A^{ER}_{l-1} = (A_{l-1} \setminus A^D_{l-1}) \cup A^{A1ab}_l \tag{22}$$

Now consider $(A^{add}_l \setminus (A^{add}_l)^D) \cup A^{A234}_l$, the second half of Equation (20). Transitions in A^{add}_l have source and/or destination nodes in V^{add}_l. $(A^{add}_l)^D$ comprises all the ϵ-transitions of A^{add}_l and all the transitions that are removed from A^{add}_l because their source and/or destination nodes are not in $(V^{ER}_l)^{add}$. So all the transitions in $(A^{add}_l \setminus (A^{add}_l)^D))$ have source and/or destination nodes in $(V^{ER}_l)^{add}$. A^{A234}_l satisfies (19), thus

$$((A^{add}_l \setminus (A^{add}_l)^D) \cup A^{A234}_l) \subseteq ((V^{ER}_{l-1} \times \Sigma \times (V^{ER}_l)^{add}) \cup ((V^{ER}_l)^{add} \times \Sigma \times \ V^{ER}_l))$$

If we let $(A^{ER}_l)^{add} = (A^{add}_l \setminus (A^{add}_l)^D)) \cup A^{A234}_l$, then (11) is satisfied. From this and Equations (20) and (22), $A^{ER}_l = A^{ER}_{l-1} \cup (A^{ER}_l)^{add}$, so Equation (8) holds. Therefore Lemma 6 is proved. □

Lemma 7 *Let FSA_l and FSA^{ER}_l be the automata referred to in Lemma 4, then for $l \geq 2$, the set of final states of FSA^{ER}_l is given by (9).*

Proof From Algorithm 1, the final states of FSA^{ER}_l are obtained by adding new final states and keeping accessible states in F_l. That is $F^{ER}_l = F^A_l \cup F^K_l$, where $F^A_l = \{v \in V^C_l \mid (v, \epsilon, v') \in A_l \textit{ where } v' \in F_l\}$, and $F^K_l = F_l \cap V^C_l$. As $A_l = A_{l-1} \cup A^{add}_l$ and $V^C_l = V^C_{l-1} \cup V^{addC}_l$, we have

$$\begin{aligned} F^A_l &= \{v \in V^C_{l-1} \mid (v, \epsilon, v') \in A_{l-1} \textit{ where } v' \in F_{l-1}\} \\ &\quad \cup \{v \in V^{addC}_l \mid (v, \epsilon, v') \in A^{add}_l \textit{ where } v' \in F_l\} \end{aligned} \tag{23}$$

$$F^K_l = (F_{l-1} \cap V^C_{l-1}) \cup (F_l \cap V^{addC}_l) \tag{24}$$

Using the same argument for FSA_{l-1}, we get

$$F_{l-1}^{ER} = \{v \in V_{l-1}^C \mid (v, \epsilon, v') \in A_{l-1} \textit{ where } v' \in F_{l-1}\} \cup (F_{l-1} \cap V_{l-1}^C)$$

so that $F_{l-1}^{ER} \subseteq F_l^{ER}$, i.e. (9) holds, and Lemma 7 is proved. □

Thus (7) to (11) hold under the condition of the lemma and hence Lemma 4 is proved. □

4.3 Proving the Necessary Condition

Lemma 8 *For $l \in \mathcal{N}^+$, if FSA_l^{ER} is a RP-FSA as specified in (7) to (11), then FSA_l satisfies for $vs, ve \in V_{l-1}^C$, $s_{(vs,ve,t)}^R \in S_l^R \Rightarrow (s_{(vs,ve,t)}^C \in S_{l-1}^C$ or $(vs, t, ve) \in A_{l-1})$.*

Proof To prove the lemma, we prove its contrapositive, i.e. for $vs, ve \in V_{l-1}^C$ when $s_{(vs,ve,t)}^R \in S_l^R \not\Rightarrow (s_{(vs,ve,t)}^C \in S_{l-1}^C$ or $(vs, t, ve) \in A_{l-1})$ then FSA_l^{ER} is *not* a RP-FSA as specified in (7) to (11).

From the proof of Lemma 6, if case 1c is not excluded, i.e. for $vs, ve \in V_{l-1}^C$ when $s_{(vs,ve,t)}^R \in S_l^R \not\Rightarrow (s_{(vs,ve,t)}^C \in S_{l-1}^C$ or $(vs, t, ve) \in A_{l-1})$, then (vs, t, ve) is added where $vs, ve \in V_{l-1}^{ER}$. However, this transition is not in A_{l-1}^{ER} or $(A_l^{ER})^{add}$ as explained in the proof of Lemma 6. This means $A_l^{ER} \neq A_{l-1}^{ER} \cup (A_l^{ER})^{add}$, so Equation (8) does not hold. Hence FSA_l^{ER} is *not* a RP-FSA as specified in (7) to (11), and the contrapositive is true. Hence Lemma 8 is proved. □

Therefore, based on Lemma 4 and Lemma 8, Theorem 1 holds.

5 Conclusion and Future Work

In this paper we have defined an infinite family of FSA related by an integer parameter, called Recursive Parametric FSA (RP-FSA). We considered the removal of ϵ-transitions from this family and identified (and proved) the necessary and sufficient condition for which this transformation results in another family which is RP-FSA in the same parameter, where the transformed family's base component is the ϵ-removed base component of the original RP-FSA. This is of theoretical interest and may provide the basis for an algebra of RP-FSA where ϵ-removal and graph addition are operators. However, this work was motivated by the verification of a multimedia protocol. We have developed a more general theory that we believe can be applied to other practical systems. In [5], a structural regularity has been discovered in the data transfer service of the Internet's Transmission Control Protocol (TCP) operating over unbounded channels. This can lead to a recursive (or closed form) expression for the state space in terms of the channel capacity. We have also observed similar regular behaviour in the state space of a simulator model [8]. The symbolic FSAs derived from the state spaces of these systems are RP-FSA. When the RP-FSA contains ϵ-transitions we can use the condition identified in this paper to check if the corresponding LE-ER RP-FSA can be obtained. If this is the case and this ϵ-removed RP-FSA

is (or can be transformed to) a deterministic RP-FSA, then we have a recursive representation of a specification against which the system can be verified.

Future work will include developing the theory on the condition under which an RP-FSA is closed under FSA determinisation, i.e. the determinised family of FSA can also be represented in the same recursive style, and applying it and the result presented in this paper to the verification of industrial systems. We will also consider extending the theory to two integer parameters for protocols with two parameters, as illustrated in the verification of the Stop and Wait Protocol class [6, 7].

References

1. Alur, R., Etessami, K., Yannakakis, M.: Analysis of Recursive State Machines. In: Lecture Notes in Computer Science (LNCS), Vol. 2102, pp. 207–220. Springer (2001)
2. Barrett, W. A., Bates, R. M., Gustafson, D. A., Couch, J. D.: Compiler Construction: Theory and Practice. 2nd Edition. Science Research Associates (1986)
3. Benedikt, M., Godefroid, P., Reps, T.: Model Checking of Unrestricted Hierarchical State Machines. In: LNCS, Vol. 2076, pp. 652–666. Springer (2001)
4. Billington, J., Gallasch, G. E., Han, B.: A Coloured Petri Net Approach to Protocol Verification. In: Lectures on Concurrency and Petri Nets: Advances in Petri Nets. LNCS, Vol. 3098, pp. 210–290. Springer (2004)
5. Billington, J., Han, B.: Formalising TCPs Data Transfer Service Language: A Symbolic Automaton and its Properties. Fundamenta Informaticae. Vol. 80, No. 1-3, pp. 49–74 (2007)
6. Gallasch, G. E., Billington, J.: A Parametric State Space for the Analysis of the Infinite Class of Stop-and-Wait Protocols. In: LNCS, Vol. 3925, pp. 201–218. Springer (2006)
7. Gallasch, G. E., Billington, J.: Parametric Language Analysis of the Class of Stop-and-Wait Protocols. In: LNCS, Vol. 5062, pp. 191–210. Springer (2008)
8. Gordon, S., Billington, J.: Analysing a Missile Simulator with Coloured Petri Nets. Int. J. on Software Tools for Technology Transfer. Vol. 2, No. 2, pp. 144-159. Springer (1998)
9. ITU: Recommendation H.245, Control protocol for multimedia communication (2005)
10. Jensen, K.: Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. 2nd Edition, Vol. 1. Springer (1997)
11. Liu, L., Billington, J.: Tackling the Infinite State Space of a Multimedia Control Protocol Service Specification. In: LNCS, Vol. 2360, pp. 273–293. Springer (2002)
12. Liu, L., Billington, J.: A Proof of the Recursive Formula for the Infinite Service Language of the CES Protocol. Technical report, CSEC-13, Computer Systems Engineering Centre, University of South Australia (2004)
13. Liu, L., Billington, J.: Obtaining the Service Language for H.245's Multimedia Capability Exchange Signalling Protocol: the Final Step. In: 10th Int. Multi-Media Modelling Conference, pp. 323–328. IEEE (2004)
14. Liu, L., Billington, J.: Reducing Parametric Automata: A Multimedia Protocol Service Case Study. In: LNCS, Vol. 3299, pp. 483–486. Springer (2004)
15. Louden, K. C.: Compiler Construction: Principles and Practice. PWS Publishing Company (1997)