Muller Transition Automata for Describing Infinite Execution Sequences and their Determinisation

Gregor v. Bochmann [1] and Martin Fränzle [2]

[1] University of Ottawa, Canada
[2] Carl von Ossietzky Universität Oldenburg, Germany
bochmann@uottawa.ca, martin.fraenzle@informatik.uni-oldenburg.de

Abstract. System requirements are often modeled by state machines. The finite-length execution sequences define the safeness properties, while infinite-length execution sequences can be used for defining fairness and liveness properties. Various extensions to state machines have been proposed for describing infinite-length sequences, such as Büchi or Muller automata. We suggest that in many cases, Muller transition automata (MTA, a variation of the traditional Muller automata) with a single acceptance set are a natural model for the desired system properties. The main contribution of this paper is an algorithm that finds an equivalent deterministic MTA for a given non-deterministic MTA. Such an algorithm is important for component-based system design, such as for control systems. To our knowledge, this algorithm is the first determinisation algorithm that works directly on Muller automata. It has a high complexity but it is relatively simple.

Keywords: Muller transition automata, Determinisation, Modeling liveness and fairness properties.

1 Introduction

System requirements are often modeled by state machines. The finite-length execution sequences define the safeness properties, while infinite-length execution sequences can be used for defining fairness and liveness properties. Various extensions to state machines have been proposed for describing infinite-length sequences, such as Büchi, Muller, Rabin or Streett automata, in the following also called ω-automata [2]. To be part of the language accepted by such an automaton, the sequence of interactions must not only correspond to a run $r(\sigma)$, that is, an execution path of the automaton, but also satisfy a certain acceptance condition of the automaton. Usually, the acceptance condition depends on $Inf(r(\sigma))$, the set of states of the automaton that are encountered infinitely often during the execution. In the case of a Muller automaton (MA), the acceptance condition is defined by a set Acc containing one or more sets of states M, and a sequence σ is accepted if there exists a run $r(\sigma)$ such that $Inf(r(\sigma))$ is equal to one of the acceptance sets $M \epsilon Acc$. All these different types of automata can be used to define the same set of (infinite-length) sequences, called the ω-regular languages. Various algorithms are known for the conversion of one type of automaton to an equivalent automaton of a different type and for obtaining a deterministic automaton equivalent to a given non-deterministic one.

The purpose of this paper is to promote a particular way of modeling liveness and fairness properties during system design using the formalism of ω-regular languages. We propose for this purpose Muller transition automata (MTA) which, in many cases, only require a single acceptance set M ε Acc for describing the desired behavior. Muller transition automata (a variation of Muller automata, see, e.g. [1], Section 8), have acceptance sets M that contain transitions rather than states to be visited infinitely often. MTA and MA define the same set of languages, namely the ω-regular languages. This fact has been formulated in [1] as Proposition 8.2, but their proof sketch factually only covers the conversion from transition to state Muller automata, which is of polynomial complexity. The conversion in the other direction is in general of exponential complexity as the following simple example shows: Consider the MA with a single state s with looping transitions for n different interactions and Acc = {{s}}. Since all infinite interaction sequences are accepted, a corresponding MTA needs an acceptance set M for each subset of the interactions, a blow-up of order exp(n).

Figure 1 contains an example. The states that must be encountered infinitely often, or the transitions that must be executed infinitely often, are shown in bold. The *a*-transition from state 1 can only be executed a finite number of times. We note that there is no equivalent MA with only three states defining the same language.

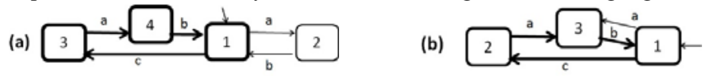

Fig. 1. (a) MA or MTA accepting the language (a b | c a b)* $(c a b)^{\omega}$. (b) Equivalent MTA.

The paper contains in the next section the definition of Muller Transition Automata (MTA). Section 2.2 contains a discussion of system properties, such as loop termination and fairness, and how these concept can be modeled with MTAs. The implementation of these properties is also discussed, using the concept of “fairly scheduled transitions”.

Section 3 is dedicated to the definition of an algorithm for finding, for a given non-deterministic MTA (which is possibly obtained through hiding of certain interactions from a deterministic one), of an equivalent deterministic MTA. To the best of our knowledge, a direct determinisation procedure for Muller automata (MA or MTA) has not been given before. We note that Schewe [6] actually stated his complexity results for determinisation of Büchi automata in terms of transition. It seems that as of yet, the difference between state and transition automata has mostly been considered a minor detail without profound consequences, such that the different representations can freely be selected as one sees fit. The tacit assumption apparently is that the two representations are so similar that results would easily carry over between them. We show that this actually is not the case: (a) As mentioned above, there is an exponential gap in conciseness between MA and MTA, and (b) as the latter are less concise, some constructions, like determinisation, may be easier on MTA. We demonstrate this fact in Section 3, where an algorithm is given for the case that the nondeterministic MTA has a single acceptance set M ε Acc. In contrast to most existing determinisation algorithms for ω-automata, this new algorithm is relatively straightforward. In Section 3.4 it is shown how the deterministic union operation can be used to generalize this construction to arbitrary nondeterministic MTA. The algorithm is demonstrated on a non-trivial example.

Most closely related to our work is the work of Colcombet and Zdanowksi [7], who have investigated the issue of employing transition rather than state automata in the translation of non-deterministic Büchi automata to deterministic Rabin automata. Their focus is mainly on the size of the state set, which unveils only minor differences between determinisation of state or transition automata. With some of the results being derived from Schewe's determinisation procedure [6], it does however indicate that the size of the acceptance set is of paramount importance, as also observed by Boker [5]. Schewe [6] in fact employed transition automata and, while providing tight bounds on the necessary state-set size, observed a necessity for exponentially sized acceptance sets – in his case in the form of Rabin pairs.

2 Describing infinite execution sequences by Muller transition automata

2.1 Definition of Muller transition automata

A (possibly nondeterministic) Muller transition automaton is an automaton $\mathcal{M}$ = <S, A, T, Acc, si> where S is a finite set of states; A is an alphabet of transition labels, also called interactions; T is a set of transitions, a subset of S x A x S; Acc is a set of acceptance sets (each a subset of T); and si is the initial state. A transition (s1, a, s2) ε (S x A x S) is said to be a transition from state s1 to state s2 with the interaction a, in the following also written <s1, a, s2>. For a given acceptance set M ε Acc, we call the transitions in M the **live** transitions of M.

A deterministic Muller transition automaton is a Muller transition automaton for which, in each state s ε S, there is at most one transition from s with a given interaction a ε A. In this case, we also write $a^{(s)}$ for a transition t = <s, a, s2>.

Given an infinite sequence of interactions σ = (a1, a2, …) ϵ A^{ω}, a run of σ on $\mathcal{M}$ is a sequence of transitions (t1, t2, …) such that t1 = <si, a1, s1> and for all j >1 : tj = <sj-1, aj, sj>.

A sequence σ ϵ A^{ω} is accepted by $\mathcal{M}$ if there exists a run r(σ) on $\mathcal{M}$ such that the set of transitions that occur infinitely often in r(σ), written inf(r(σ)), is equal to one of the sets M ϵ Acc. For a deterministic Muller transition automaton, there is at most one run for each infinite sequence σ ϵ A^{ω} . We write Lang($\mathcal{M}$) for the set of interaction sequences that are accepted by $\mathcal{M}$, which is called the language accepted by $\mathcal{M}$.

We say that the state machine (without the acceptance condition) defines the **finitary behavior**, which defines the allowed finite prefixes of the accepted infinite interaction sequences. The **infinitary behavior** of a MTA is defined by the acceptance sets M ε Acc. Each acceptance set M defines a submachine of the given MTA which consists of all transitions in M. For each accepted infinite sequence, there is a point from where onwards, the sequence of interactions follows the infinitary behavior.

Note: One can make the assumption that in a well-formed MTA all acceptance sets M ϵ Acc are **admissible** in the following sense: A subset M of T is admissible if the transitions form a strongly connected subgraph of the automaton's transition graph.

Note on language operations: In the proof of Theorem 6.4 in [3] it was shown how one can obtain a timed MA that represents the union, intersection or complement of languages defined by given timed MA. It is easy to adapt these operations to the context of MTA (see also [1]). These operations are important for component-based system design (see e.g. [4]). The operation of union will be used in Section 3.4.

2.2 Fairly scheduled transitions (some practical considerations)

When there is more than one outgoing transition from a given state of a state machine, a choice between these transitions must be performed by an implementation of that state machine. Often, this choice should be "fair". For example, we may interpret the state machine in Figure 2(a) as follows: There are two processes *Pb* and *Pc*, and one resource. During transition *a*, both processes request the resource, and in transition *b* (or *c*) process *Pb* (or *Pc*) receives the resource. The definition of "strong fairness" is satisfied for a process if it eventually receives the resource after having requested it an unlimited number of times. We apply this concept to transitions and use the following definition:

Definition: A transition of a state machine is **fairly scheduled** if it is eventually executed after being enabled an unlimited (arbitrary) number of times.

Lemma: If a fairly scheduled transition of a state machine is enabled infinitely often, then the transition is executed infinitely often. (Note: The proof in temporal logic is trivial).

Lemma: Given a system specification in the form of a MTA containing a single acceptance set M, the infinitary behavior of the system can be implemented by providing fairly scheduled implementations for all live transitions that start from a state which has a choice between several outgoing transitions. (Different methods may be employed for such implementations.)

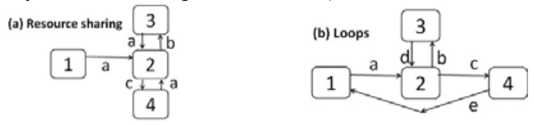

Fig. 2. (a) Fair resource sharing among states 3 and 4. (b) Program with embedded loops.

We conjecture that the behavior of a reasonable system would have only a single acceptance set. Another example is shown in Figure 2(b) which represents the simplified control flow of a program that performs a task (transition ***a*** followed by the **while** loop (*b, d*)* and transition ***c***) and finally goes back to the initial state through transition ***e***. This machine has the following admissible acceptance sets:

1. {b, d} : the case that eventually, the inner loop does not terminate.

2. {a, c, e} : the case that finally, the inner loop is never executed.
3. {a, b, d, c, e} : the case that the inner loop is infinitely often entered and always terminates.

The case 3 above occurs if both transitions b and c are fairly scheduled. If transition b is fairly scheduled, but not transition c, we may have cases 1 or 3. If inversely, transition c is fairly scheduled, but not transition b, we may have cases 2 or 3.

Considering that Figure 2(c) represents a system that performs a task involving a loop and then goes back to the initial state to repeat this process forever, it appears that one normally would expect a behavior corresponding to case 3 above. The behavior of case 2 makes only sense if the loop is required during some initialization process, but never used after some time.

3 Determinisation

3.1 Introduction

We first consider the determinisation of an MTA that has a single acceptance set. The case of multiple acceptance sets is discussed in Section 3.4. After this introduction to the problem with several small examples, the determinisation algorithm is described in Section 3.2, and a more complex example is discussed in Section 3.3. The determinisation algorithm has the following five steps:

Step 1: Standard determinisation

The standard determinisation algorithm results in a deterministic machine where each state (which we call **macro-state**) corresponds to a set of states in the original machine. A simple example of a nondeterministic machine is shown in Figure 3(a). The equivalent deterministic one is shown in Figure 3(b). It has three macro-states, where the second corresponds to two states of the original machine: states 2 and 3. We write Σ{s1, … sn} for a macro-state that corresponds to the states s1, … sn of the original machine. Using this notation, we can say that the deterministic machine has the states Σ{1}, Σ{2,3} and Σ{3}.

Step 2: Further information on the transitions

The example of Figure 3 demonstrates the issue of transition identification. Assuming that all transitions of Figure 3(a) should be live, one may think that the transition loop a – b – c in Figure 3(b) would be a possible acceptance set for the deterministic machine. However, both c-transitions of the machine must be included in the acceptance set since one is executed after the a-transition leading to state 2 while the other is executed after the a-transition leading into state 3, and both transitions must be executed infinitely often.

In order to clearly identify the transitions in the deterministic machine and the correspondence with the nondeterministic one, we use diagrams as shown in Figure 3(c). Each macro-state includes explicitly the corresponding states of the nondeterministic machine, called **detailed states**, and the arrows of the transitions, called **detailed transitions**, go from their detailed start state to their detailed ending state. We also annotate the detailed transitions with their starting and ending (detailed) states, as shown in the figure. We note that in Figure 3(c) the two a-transitions are clearly identified, and the c-transition occurs twice in the deterministic machine.

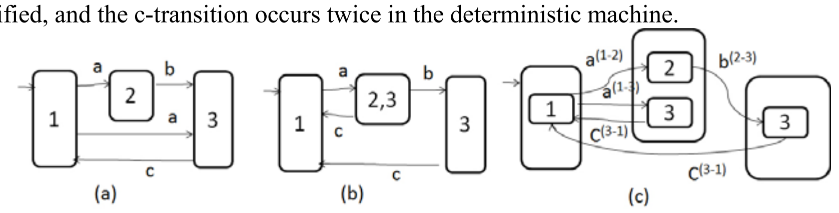

Fig. 3. (a) A nondeterministic state machine. (b) Equivalent deterministic state machine obtained by standard determination algorithm, containing three macro-states. (c) State machine (b) with more detail: detailed states included in macro-states and detailed transitions.

Step 3: Considering the live transitions

In this step, we mark the detailed transition in the detailed deterministic state machine obtained in Step 2 and find the subset of transitions that are executed during the infinitary behavior. For this purpose we consider the sub-graph of all detailed transitions which correspond to a live transition of the original machine and find the largest fully connected subgraph which represents the transitions of the deterministic machine which are executed during the infinitary behavior.

Another simple example is shown in Figure 4 (the two a-transitions of the nondeterministic machine are hidden, the live transitions are drawn as thick arrows). The transitions during the infinitary behavior are the b – c loop within the macro-state Σ{2,3}, which means alternative execution of b and c (see Figure 4(c)).

Step 4: Considering the active detailed states of macro-states

The example of Figure 4 presents the issue of active (and non-active) detailed states within macro-states during the infinitary behavior. The standard determinisation of Figure 4(b) allows the execution of the b- and c-transitions in arbitrary order in the infinitary behavior (which is not allowed by Figure 4(a)). The more detailed view of Figure 4(c) indicates alternative execution (like the nondeterministic machine of Figure 4(a)). Figure 4(c) shows that after the execution of a b-transition, the machine is in the detailed state 3 (while the detailed state 2 of the macro-state Σ{2,3} is inactive). This is not true during the finitary behavior because the non-live b-transition $b^{(3-2)}$ may be executed instead (from the environment, these two b-transitions are not distinguishable).

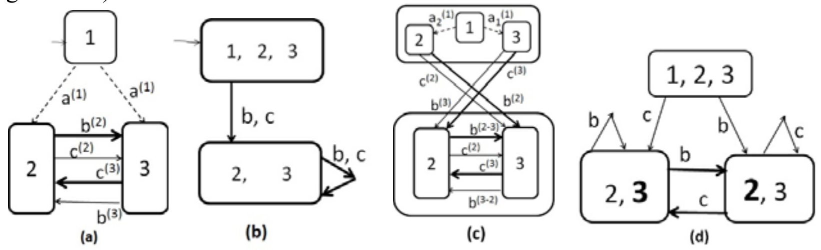

Fig. 4. (a) A nondeterministic MTA. (b) Standard determination. (c) More detailed view of (b). Deterministic MTA corresponding to (c) with duplicated macro-state (2, 3).

In order to deal with this problem, we replace such a macro-state with several (safety-equivalent) copies, one for each subset of detailed states that may be simultaneously active. **Safety-equivalent** means that they are equivalent for the finitary behavior. For the example of Figure 4, we obtain two safety-equivalent states, one reached after the (live) b-transition, and one after the c-transition, as shown in Figure 4(d) – the active detailed states are indicated in **bold**. We note that the c-arrow from Σ{2,**3**} to Σ{**2**,3} includes the c-transitions $c^{(3-2)}$ and $c^{(2-3)}$, and the c-loop transition from Σ{**2**,3} includes the same transitions - in this loop, $c^{(3-2)}$ is not shown in bold because it cannot be executed during the infinitary behavior (as the detailed state 3 is not active in Σ{**2**,3}.

The result of this step is a deterministic MTA with a maximal acceptance set which accepts only sequences that are also accepted by the original nondeterministic MTA, but not all of them. The remaining sequences will be covered by the following step.

Step 5: Acceptable subsets of the maximal acceptance set

There are often smaller subsets that are also valid acceptance sets in the sense that they ensure the liveness of all live transitions in the original machine. But they restrict the order in which these transitions can be executed during the infinitary behavior. An example is shown in Figure 5. The original MTA shown in Figure 5(a) becomes nondeterministic when the c-interactions are hidden. We assume that all transitions are live. Figure 5(b) shows the equivalent deterministic MTS where all transitions from the macro-states Σ{1,2} and Σ{1,2,3} are in the maximal acceptance set. In this case, the following four acceptance sets assure the liveness of all transitions of the original nondeterministic MTA:

- M1 : the a-b-loop between the macro-states Σ{1,2} and Σ{1,2,3} – in the infinitary behavior, the interactions a and b are executed alternatively.
- M2 : the a-b-loop plus the a-self- loop – in the infinitary behavior, consecutive a-interactions may occur.
- M3 : the a-b-loop plus the b-self- loop – in the infinitary behavior, consecutive b-interactions may occur.
- M4 : the maximal acceptance set shown in Figure 5(b) - there is no restriction on the order in which the a- and b-transitions occur during the infinitary behavior.

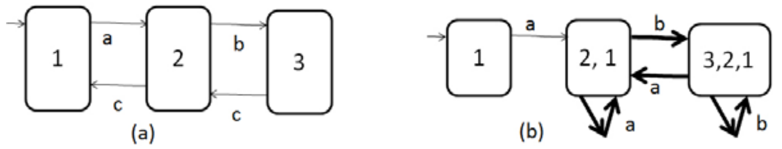

Fig. 5. (a) MTA which becomes nondeterministic when interaction c is hidden. (b) Equivalent MTA with maximal acceptance set.

We note that the sequences accepted with the acceptance set M1 (or M2 or M3) are not accepted with the maximal acceptance set, because the latter requires that all transitions of the maximal acceptance set in the deterministic MTA be executed infinitely often, while these sequences do not execute the a-self-loop and b-self-loop infinitely often.

To deal with this problem, Step 5 identifies all acceptance sets that include all transitions in the acceptance set of the original MTA. For instance, the a-self-loop in Figure 7(b) alone would not be a valid acceptance set because it would not cover the transition $b^{(2\text{-}3)}$.

3.2 Determinisation Algorithm for MTAs with a single acceptance set

Given a nondeterministic MTA (NMTA) $\mathcal{M}$ = <S, A, T, Acc, si> where Acc contains a single set L of live transitions. A is the set of transition labels (interactions), including the empty word ϵ which is an invisible interaction that may be obtained by hiding. We want to find a deterministic MTA $\mathcal{M}'$ = <S', A' , T', Acc', si'> that accepts the same language as $\mathcal{M}$, where A' is equal to A minus the empty word. Such a $\mathcal{M}'$ is found by the following algorithm which was informally introduced in the previous section. A non-trivial example is discussed in Section 3.3.

Step 1: Standard determination

According to the standard determinisation algorithm, a safety-equivalent state machine $\mathcal{M}1$ = <S1, A', T1, si1> is obtained by the following algorithm:

- Each state Σ ε S1 represents a different subset of S, the states of the original MTA. We call these states of $\mathcal{M}1$ **macro-states**, and write Σ_V ε S1 for the state of $\mathcal{M}1$ that represents the subset V of states of $\mathcal{M}$. The states in V are called the **detailed states** of V.
- The initial state si1 is Σ_{Init} where Init is the subset of S that includes si, the initial state of $\mathcal{M}$, and all states of $\mathcal{M}$ that can be reached from si by only ϵ-transitions.
- For any macro-state Σ_{V1} ε S1 and any interaction b ε A' , if there is a b-transition from some state s1 ε V1 then there is a b-transition t ε T1 from Σ_{V1} to Σ_{V2} ε S1 where V2 is the subset of S that includes all states of $\mathcal{M}$ that can be reached from some state of V1 by a b-transition ε T, and all states that can be reached from those states by only ϵ-transitions – otherwise there is no b-transition from Σ_{V1}. We call the b-transition t a **macro-transition** and the b-transitions ε T the (detailed) **b-transitions contributing to t**.

Step 2: Further information: The detailed transition graph

In this step, we construct the **detailed transition graph**. It has as nodes the detailed states of the macro-states of $\mathcal{M}1$ and as edges the detailed transitions. The graph is constructed as follows:

- Notation: We write $\Sigma_V(s)$ for the detailed state s ε V belonging to the macro-state Σ_V ε S1.
- The nodes of the transition graph are all the detailed states of all macro-states of $\mathcal{M}1$*,* that is all $\Sigma_V(s)$ where s ε V and Σ_V ε S1.

- The edges of the transition graph are detailed transitions. More precisely, for any interaction b ε A, there is a b-transition from $\Sigma_{V1}(s1)$ to $\Sigma_{V2}(s2)$ if and only if there is a b-transition in $\mathcal{M}$ from s1 to s2. This includes the ϵ-transitions.

Step 3: Construction of the live subgraph of the detailed transition graph

- We construct the **subgraph of live transitions** which is the subgraph of the detailed transition graph which contains only those edges that correspond to live transitions of $\mathcal{M}$, that is, to transitions that are in L.
- We construct the **live subgraph** which is the maximal fully connected subgraph of the subgraph of live transitions.
- All nodes and detailed transitions that are part of the live subgraph are **marked as active**. (Note: This means that they are part of the infinitary behavior).
- The macro-transitions that have an active contributing (detailed) transition are also marked **active.**

Step 4: Duplicating certain macro-states

As the example of Figure 4 shows, during infinitary behavior, there is less non-determinism because not all transitions can be executed. In the macro-state Σ{2. 3} of Figure 4(c) either the detailed state 2 or 3 is active during the infinitary behavior depending on what the last transition was. During the infinitary behavior, when the MTA enters a macro-state, it will be in one of the active detailed states. We duplicate a macro-state if different active detailed states can be distinguished based on the transition by which the macro-state is entered. The algorithm is a **while**-loop where a given macro-state is duplicated in each round. (**Notation:** We write $\Sigma_{V1}{}^{A1}$ for a macro-state Σ_{V1} for which the active detailed states are exactly those in A1 (A1 is a subset of V1).)

Algorithm: While there is a macro-state $\Sigma_{V2}{}^{A2}$ which has an incoming macro-transition with interaction b ε A' from a macro-state $\Sigma_{V1}{}^{A1}$ such that the detailed ending states of the contributing active b-transitions form a subset A3 that is smaller than A2, do the following:

- Replace the macro-state $\Sigma_{V2}{}^{A2}$ by N macro-states $\Sigma_{V2}{}^{Ai}$ (i = 1, 2, … N) such that the detailed ending states of the contributing active (detailed) transitions for each incoming active macro-transition form exactly one of the subsets Ai.
- Each outgoing detailed transition from a detailed state of the original macro-state $\Sigma_{V2}{}^{A2}$ will be duplicated for the corresponding detailed states of all new macro-states $\Sigma_{V2}{}^{Ai}$. If the original transition was marked active and the new copied transition starts from a detailed state that is active, the transition will be marked active, otherwise as inactive. (Note: Therefore, all the macro-states $\Sigma_{V2}{}^{Ai}$ will be safety-equivalent to the original macro-state).
- Each outgoing macro-transition from the original macro-state $\Sigma_{V2}{}^{A2}$ will be duplicated for all new macro-states $\Sigma_{V2}{}^{Ai}$. If and only if a duplicated macro-transition has an active contributing detailed transition, then the macro-transition will be marked active.
- Each incoming active macro-transition of the original macro-state leads to the macro-state $\Sigma_{V2}{}^{Ai}$ which has the corresponding subset of active detailed states Ai, and

all contributing detailed transitions lead to their corresponding detailed state. (No change in activity marking).

- The non-active incoming macro-transitions of the original macro-state (and their contributing detailed transitions) will be assigned to any one of the newly created macro-states (since they are safety-equivalent).

The result of this step is a deterministic MTA ***M'* =** <S', A', T', Acc', si'> where S' is the set of macro-states of the detailed transition graph (after Step 4), T' are the macro-transitions, Acc' contains a single set L' which contains all active macro-transitions, and si' = si1 as determined in Step 1. L' is the maximal acceptance set, as discussed in Section 3.1. This MTA ***M'*** accepts only sequences that are also accepted by the original ***M***, but in general not all.

Step 5: Acceptable subsets of the maximal acceptance set

This step identifies additional acceptance sets to be included in Acc' in order to make ***M'*** equivalent to ***M***.

Acc' contain the maximal acceptance set L' and all subsets L'' of L' that satisfy the following conditions:

- L'' forms a fully connected graph of active macro-transitions. Note: This is the standard admissibility criterion.
- All transitions in L are represented by L'' (see example discussed in Section 3.1).
- If a macro-transition in L'' leads to a macro-state Σ with more than one active detailed state, then L'' contains for each active state s ε Σ a macro-transition with a supporting active detailed transition that leaves the state s. (Note: In the example discussed in Section 3.3, the state $\Sigma_{\{\mathbf{1},\,2,\,\mathbf{4}\}}$ is such a state, both outgoing transitions c and d must be included in each acceptance set.)

Complexity: We note that the complexity of this algorithm is doubly exponential because of the exponential blow-up of the number of states during Step 1 and Step 4.

3.3 Discussion of an example

A non-trivial example is discussed in the following. The given nondeterministic MTA is shown in Figure 6(a). The transitions with interaction label *a* are hidden, drawn as dotted arrows. This MTA is nondeterministic (a) due to the hiding of the a-transitions, and (b) due to several c-transitions from state 4. These are the steps:

- The standard determinisation algorithm of Step 1 leads to the state machine of Figure 6(c).
- The result of Step 2 is shown in Figure 6(d).
- The result of Step 3 is also shown in Figure 6(d). The active detailed transitions and detailed states of the live subgraph are shown in bold, while the live detailed transitions that are not part of the live subgraph are shown as bold dashed arrows.
- Concerning Step 4, we note that the macro-state Σ{**1**, 2} (see Figure 6(d)) does not need duplication since it has only one active detailed state. Also Σ{1, **2, 3**} does not need duplication since during the infinitary behavior, it can only be reached

through the active d-transitions that lead to detailed state 3 from where the detailed state 2 is reached by a hidden transition. However, Σ{1, **2, 4**} can be replaced by the three macro-states Σ{1, **2**, 4} , Σ{1, 2, **4**} , and Σ{**1**, 2, **4**} (see Figure 7). When the latter state is reached during the infinitary behavior, it is not known whether the detailed state 1 or 4 is reached by the b-macro-transition. Also the Σ{0, 1, **2, 4**} macro-state must be replaced by the two states Σ{0, 1, **2**, 4} and Σ{0, 1, 2, **4**} . The non-active c-macro-transitions from Σ{0, 1, 2, **4**} (self-loop in Figure 6(d)) and from Σ{1, 2, **4**} could lead to Σ{0, 1, **2**, 4} or Σ{0, 1, 2, **4**} . In Figure 6(b), which shows the final result of this step (including only the macro-transitions), the choice was made that these transitions lead to Σ{0, 1, **2**, 4} .

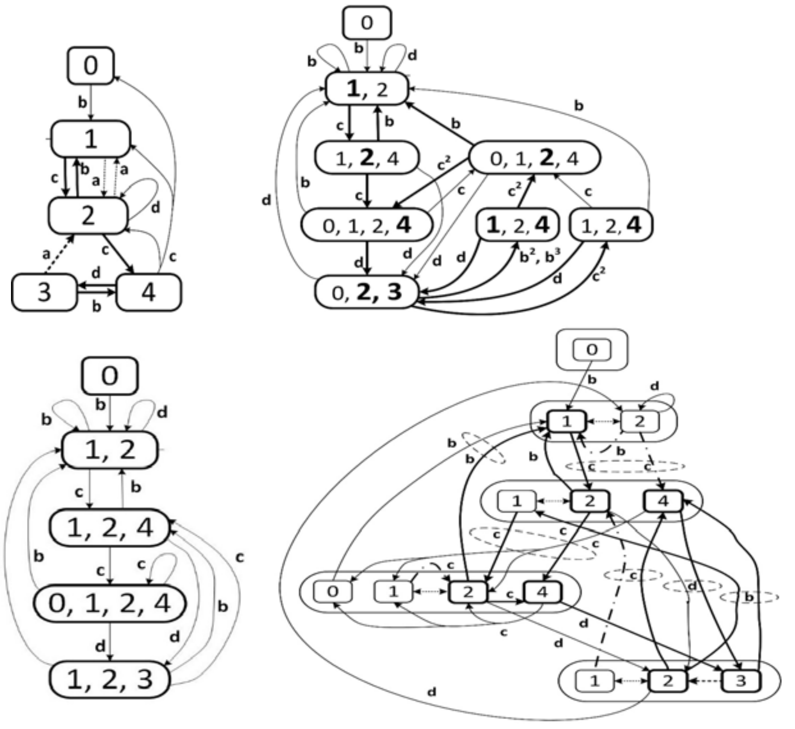

Fig. 6. (a),top-left: Given nondeterministic MTA; (b) top-right: equivalent deterministic MTA; (c) low-left: Standard determinisation of (a); (d) low-right: Detailed view of (c) [notation: the dashed ellipses represent macro-transitions]

- Figure 6(b) is used in Step 5. The macro-transitions shown in bold form the maximal acceptance set L''. The smallest subset of L'' that satisfies the conditions of Step 5 is the transition loop (d, b, c, c) starting in macro-state Σ{0, 1, 2, **4**}. This subset must also include the d-transition from Σ{**1**, 2, **4**} in order to satisfy the last condition of Step 5. We conclude that all fully connected subsets of L'' that include this smallest subset are the acceptance sets of ***M'***.

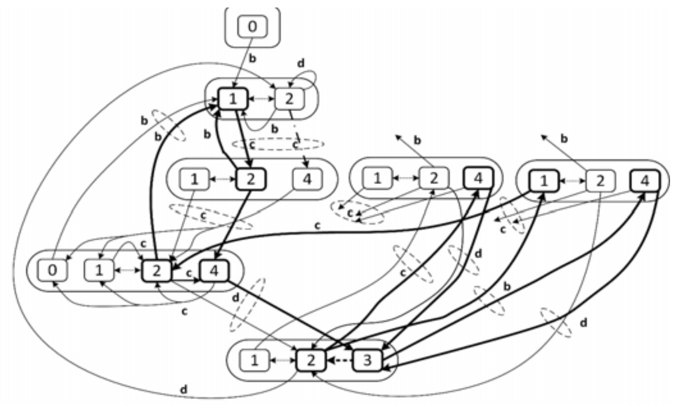

Fig. 7. Detailed view of Fig. 6(d) after duplicating the macro-state $\Sigma_{\{\mathbf{1}, \mathbf{2}, \mathbf{4}\}}$.

3.4 Determinisation of an MTA with multiple acceptance sets

A sequence σ ϵ A^{ω} is accepted by a MTA ***M*** = <S, A, T, Acc, si> if it satisfies the acceptance condition of one of the acceptance sets L ϵ Acc. If ***M*** is nondeterministic, we can obtain an equivalent deterministic MTA ***M'*** by applying the algorithm above to ***M*** for each L ϵ Acc separately, and then use the union operation to construct ***M'*** which is the union of all the deterministic MTAs obtained for the different acceptance sets L. Similar to the approach in [3], the union of two deterministic MTA can be defined in terms of a synchronous automaton product yielding a deterministic MTA. We note that Steps 1 and 2 need only be performed once, since they are independent of the acceptance set L. A discussion of this approach for the example above is not included because of space constraints.

References

1. Perrin, D., Pin, J.-É.: Infinite Words: Automata, Semigroups, Logic and Games, Elsevier, (2004).
2. Farwer, B.: "ω-Automata", in Grädel, Erich; Thomas, Wolfgang; Wilke, Thomas, Automata, Logics, and Infinite Games, LNCS, Springer, pp. 3–21 (2002).
3. Alur, R. and Dill, D.L.: A theory of timed automata, Theor. CS, 126 (1994) 183 – 235.
4. Bochmann, G. v.: Using logic to solve the submodule construction problem, Journal on Discrete Event Dynamic Systems, Vol. 23 (1), Springer, March 2013, pp. 27-59.
5. Boker, U.: On the (In)Succinctness of Muller Automata, in 26th EACSL Annual Conference on Computer Science Logic (CSL 2017), Editors: Goranko and Dam, pp. 12:1–12, InformaticsSchloss Dagstuhl, Germany.
6. Schewe, S.: Tighter Bounds for the Determinisation of Büchi Automata, in de Alfaro (Ed.): FOSSACS 2009, LNCS 5504, pp. 167–181, 2009.
7. Colcombet, T. and Zdanowski, K. : A Tight Lower Bound for Determinisation of Transition Labeled Büchi Automata, in ICALP 2009: Automata, Lang. and Progr., pp. 151-162.