State-Merging DFA Induction Algorithms with Mandatory Merge Constraints

Bernard Lambeau[1], Christophe Damas[1], and Pierre Dupont[1,2]

[1] Department of Computing Science and Engineering (INGI)*
Université catholique de Louvain
Place Sainte Barbe, 2
B-1348 Louvain-la-Neuve - Belgium
{bernard.lambeau, christophe.damas,
pierre.dupont}@uclouvain.be
[2] UCL Machine Learning Group
http://www.ucl.ac.be/mlg/

Abstract. Standard state-merging DFA induction algorithms, such as RPNI or Blue-Fringe, aim at inferring a regular language from positive and negative strings. In particular, the negative information prevents merging incompatible states: merging those states would lead to produce an inconsistent DFA. Whenever available, domain knowledge can also be used to extend the set of incompatible states. We introduce here mandatory merge constraints, which form the logical counterpart to the usual incompatibility constraints. We show how state-merging algorithms can benefit from these new constraints. Experiments following the Abbadingo contest protocol illustrate the interest of using mandatory merge constraints. As a side effect, this paper also points out an interesting property of state-merging techniques: they can be extended to take any pair of DFAs as inputs rather than simple strings.

1 Introduction

Deterministic Finite Automaton (DFA) induction is a popular technique to infer a regular language from positive and negative strings defined over a finite alphabet Σ. Several state-merging algorithms have been proposed to tackle this task, including RPNI [1], EDSM and Blue-Fringe (also known as redBlue) [2]. These algorithms start from a tree-shaped automaton, the so-called prefix tree acceptor (PTA), that accepts the positive sample S_+ only, and successively merge states to generalize the induced language. The order in which pairs of states are considered for merging is the key difference between the respective algorithms. In all cases, the generalization is controlled by the negative sample S_- to prevent merging incompatible states. Merging those states would lead to build an inconsistent machine, that is a DFA which accepts at least one negative string.

The availability of negative information is theoretically motivated since positive and negative samples are required to identify in the limit any super-finite class of languages, including the regular language class [3]. The convergence proof of RPNI, for instance,

* This work is partially supported by the Regional Government of Wallonia (Gisele project, RW Conv. 616425).

A. Clark, F. Coste, and L. Miclet (Eds.): ICGI 2008, LNAI 5278, pp. 139–153, 2008.

is related to the definition of a characteristic sample. When RPNI receives as input such a sample, it is guaranteed to output the target machine, that is, the canonical automaton of the target language. Such a characteristic sample includes $\mathcal{O}(n^2)$ negative strings, where n denotes the number of states of the target machine.

When few negative strings are available, one can rely on another kind of knowledge, typically provided by the application domain. For example, when some additional information is available about incompatibilities between states of the initial automaton, state merging algorithms can easily be extended to avoid merging states that are known to be incompatible [4,5]. This technique, sometimes referred to as *state coloring*, may be seen as incompatibility constraints on the induction algorithm.

Our previous work applies grammatical inference to the Requirements Engineering (RE) domain [6]. In such an applicative context, domain-specific information, represented as incompatibility constraints, can be used in conjunction with negative strings to avoid poor generalizations while reducing the induction search space. In the same work, we introduced the QSM algorithm, an active learning extension to RPNI or Blue-Fringe with membership queries [7]. These queries are generated during the induction process and provide additional positive and negative strings, overcoming the limitations of an initially sparse learning sample.

The present work introduces a new kind of constraints, called *mandatory merge constraints*. They form the logical counterpart to the incompatibility constraints. This paper investigates this kind of constraints and how to extend state-merging algorithms to handle them. The extended algorithm, called MSM, has been evaluated using an experimental protocol inspired by the Abbadingo competition [2]. We show experimentally that MSM converges faster than existing algorithms. This result is actually quite straightforward since MSM uses a richer information as input. However, two additional observations are arguably the actual contributions of this work.

Firstly, DFA induction algorithms using state-merging typically include a recursive merging process to reduce the non-determinism of temporary solutions. Such a *merging for determinization* process is often implemented assuming a *tree invariant property*. This property states that, when considering two states to be merged, at least one of them is the root of a tree. Such a property, which holds for RPNI and Blue-Fringe, but interestingly not for EDSM, is a sufficient condition for the determinization process to be finite. We argue here that, even though it is convenient, the tree invariant property is not required as the determinization process stops by itself. The important consequence of this observation is that the initial automaton to be considered no longer needs to be a tree representing a finite sample. Hence MSM naturally gives rise to the *Automaton State Merging* (ASM) algorithm, that is a state-merging induction algorithm which takes as input a positive regular language represented by a DFA A_+ and a negative sample S_-, and outputs a regular language L with $L(A_+) \subseteq L \subseteq \Sigma^* \setminus S_-$. Secondly, we discuss a natural extension to ASM which takes as input a positive DFA A_+ and a "negative" DFA A_- and returns a regular language L with $L(A_+) \subseteq L \subseteq \Sigma^* \setminus L(A_-)$. The initialization of this extended algorithm would immediately detect an inconsistency whenever $L(A_+) \cap L(A_-) \neq \emptyset$.

The rest of this paper is structured as follows. Section 2 briefly reviews the Requirement Engineering application context which originally motivated the present work.

Section 3 reviews the DFA state-merging induction algorithms and the use of incompatibility constraints. Section 4 introduces mandatory merge constraints and describes the MSM algorithm. Section 5 presents the experimental protocol and the results obtained using MSM on synthetic data and on a RE case study. The ASM algorithm to deal with positive and negative automata as inputs is discussed in section 6. Our conclusions and future works are presented in section 7.

2 The Synthesis of Software Behavior Models Seen as a DFA Induction Problem

It has been claimed that the hardest part in building a software system is deciding precisely what the system should do. This is the general objective of Requirements Engineering (RE). Automating parts of this process can be addressed by learning behavior models from scenarios of interactions between the software-to-be and its environment. Indeed, scenarios can be represented as strings over an alphabet of possible events and they can be generalized to form a language of acceptable behaviors. Whenever behaviors are modeled by finite-state machines, the problem becomes equivalent to automaton induction from positive and negative strings.

Scenarios are typical examples of system usage provided by an end-user involved in the requirements elicitation process. A simple message sequence chart (MSC) formalism is used for representing scenarios. A MSC is composed of vertical lines representing time-lines associated with agent instances, and horizontal arrows representing interactions among such agents. Figure 1 depicts some scenarios for a simple train system. The system is composed of four agents: a train controller, train motor and doors, and a passenger. For example, the scenario "Start/Stop" expresses that the train controller can start the train and then stop the train from the initial state.

Related works on RE are described in [7]. The technique proposed in the present paper allows one to inductively synthesize behavior models from positive and negative scenarios while taking into account their flowcharting in a high-level Message Sequence Chart (hMSC). A hMSC is a directed graph where each node references a scenario. Edges indicate the acceptable flowcharting of these scenarios, allowing the end-user to reuse scenarios within a specification and to introduce sequences, loops, and alternatives. Figure 1 presents an example of scenario flowcharting for the train example. An induction approach for behavior synthesis is relevant because a PTA can easily be de-

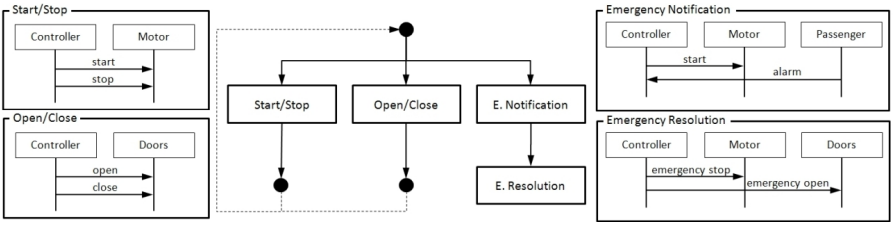

Fig. 1. Some scenarios for a simple train system and their flowcharting in a hMSC (center)

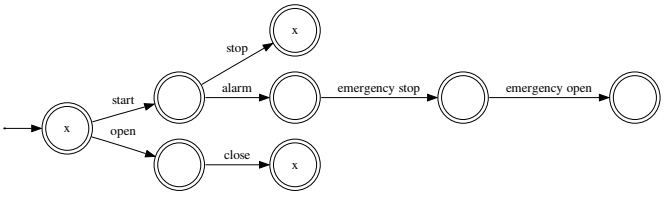

Fig. 2. A labeled Prefix Tree Acceptor generated for the train system

rived from the positive and negative scenarios: the transitions drawn as solid lines form a spanning tree of the hMSC resulting in the initial PTA of Figure 2. Moreover, the hMSC provides additional information that must be used during the induction process: dashed lines allow one to identify equivalent system states. A label is associated to those equivalent states in the PTA (x in the example). The MSM algorithm introduced in section 4 ensures that they will be merged by the induction process.

3 State-Merging DFA Induction with Incompatibility Constraints

This section briefly reviews the DFA induction problem using state-merging algorithms and incompatibility constraints. The concept of quotient automaton, strongly related with state-merging, is introduced in section 3.1. Section 3.2 presents a family of state-merging algorithms, RPNI being one representative example. Section 3.3 describes one efficient way to handle incompatibility constraints in such algorithms.

3.1 State-Merging and Quotient Automaton

State-merging DFA induction algorithms, such as RPNI, start from an initial automaton called a prefix tree acceptor $PTA(S_+)$. It is the largest trimmed DFA accepting exactly S_+ (see the automaton on the left of Fig. 3). The generalization operation obtained by merging states is defined through the concept of *quotient automaton*, relative to a partition π of the state set of the original atomaton. States belonging to the same subset, or block, of π are merged in the quotient automaton (see the automaton on the right of Fig. 3). Any accepting path in the PTA is also an accepting path in PTA/π. As a quotient automaton corresponds to a particular partition, the set of possible generalizations which can be obtained by merging states of an automaton A can be searched through a lattice of partitions $Lat(A)$ [8].

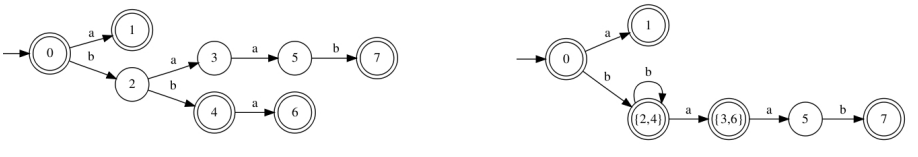

Fig. 3. (left) $PTA(S+)$ with $S_+ = \{\lambda, a, bb, bba, baab\}$ and λ denoting the empty string; **(right)** a quotient automaton $A = PTA/\pi$ where $\pi = \{\{0\}, \{1\}, \{2,4\}, \{3,6\}, \{5\}, \{7\}\}$. Accepting states are represented as doubly circled nodes.

3.2 State-Merging DFA Induction Algorithms

RPNI is a very well known DFA induction algorithm [1]. It can be seen as a particular case of the state-merging algorithm described in Algorithm 1. It takes a positive and a negative sample as input. The first step constructs the PTA[1]. An initial partition is initialized and successively updated by the main loop of the algorithm. At each step of this loop, two blocks of the partition are selected as candidate for merging. These partition blocks precisely define the states of the quotient automaton PTA/π. In other words, each step can be interpreted as merging two states of a quotient automaton, which forms an intermediate solution of the algorithm. When compatible with the negative sample, this intermediate solution is kept for the next step, otherwise it is simply discarded. The algorithm continues by selecting other blocks to merge, until no more state pairs can be considered.

Algorithm 1. A state-merging DFA induction algorithm

Algorithm STATE-MERGING DFA INDUCTION ALGORITHM
Input: A positive and negative sample (S_+, S_-)
Output: A DFA A consistent with (S_+, S_-)

// Compute a PTA, let N denote the number of its states
$PTA \leftarrow$ `Initialize`$(S_+,\ S_-)$; $\pi \leftarrow \{\{0\}, \{1\}, ..., \{N-1\}\}$

// Main state-merging loop
while $(B_i, B_j) \leftarrow$ `ChoosePair`(π) **do**
  $\pi_{new} \leftarrow$ `Merge`(π, B_i, B_j)
  if `Compatible`$(PTA/\pi_{new}, S_-)$ **then**
    $\pi \leftarrow \pi_{new}$

return PTA/π

// This function merges two blocks and removes non-determinism recursively
`Merge`(π, B_i, B_j) **begin**
  $\pi \leftarrow \pi \backslash \{B_i, B_j\} \cup \{B_i \cup B_j\}$
  while $(B_k, B_l) \leftarrow$ `FindNonDeterminism`(π, B_i, B_j) **do**
    $\pi \leftarrow$ `Merge`(π, B_k, B_l)
  return π
end

The pseudo code of the `Merge` function is shown below the main loop. The first line simply updates the partition π by effectively merging the two block arguments. Merging two blocks B_i and B_j may lead to a non-deterministic quotient automaton; the partition is then recursively updated in order to reduce the non-determinism.

The `ChoosePair` function determines which pairs of blocks to consider for merging. In the particular case of the RPNI algorithm, it relies on the standard lexicographical order on strings. The Blue-Fringe algorithm uses an heuristic approach according to the order in which state pairs are chosen, while identifying as soon as possible the states which are incompatible with all their predecessors [2]. For both algorithms, the block B_i in the main loop and, by extension, the block B_k in the `Merge` function, are always the roots of a tree. This is the *tree invariant property* already mentioned in section 1. This property also helps implementing particularly simple and fast algorithms [2].

[1] The reason why the PTA is built using S_+ as well as S_- results from a variant motivated in section 3.3.

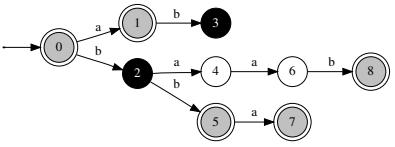

Fig. 4. Augmented PTA from the sample $(S_+, S_-) = (\{\lambda, a, bb, bba, baab\}, \{ab, b\})$. The positive sample can be represented by the PTA illustrated on the left of Fig.3. This PTA can be augmented by the negative sample, by coloring states reached by a negative string as black. Accepting states (of positive strings) are filled in grey in this figure.

3.3 Handling Incompatibility Constraints

Negative strings play a crucial role in automaton induction algorithms, as they are used to avoid merging incompatible PTA states. When few negative strings are provided in the initial sample S_-, alternative sources of knowledge can be used to play a similar role. In particular, knowledge about incompatibilities between states of the PTA can easily be incorporated in the induction process as coloring constraints. As the negative sample S_- can itself be encoded as such constraints, we used this particular example to illustrate the technique in Fig. 4.

To ensure that the solution returned by the induction algorithm correctly rejects S_-, it suffices to avoid merging black and grey states of the PTA. In other words black and grey states form incompatible pairs. This black/grey coloring can be captured by a partial function $f_{col}(Q) \rightarrow \{grey, black\}$, where Q is the set of PTA states. A quotient automaton PTA/π respects the coloring constraint if, $\forall q_1, q_2 \in Q$ such that $f_{col}(q_1) \neq f_{col}(q_2)$ and both are defined, if $q_1 \in B_i$ and $q_2 \in B_j$ then $B_i \neq B_j$. In other words, each block B may contain any number of uncolored states (4 and 6 in our example) but grey and black states must be kept separate from each other. Checking the constraints can be efficiently performed in the `Merge` function. The `Compatible` function in the main loop can thus be replaced by a compatibility test between merged blocks in the `Merge` function. This algorithm is able to detect inadequate solutions during the determinization procedure itself, speeding up the induction process. This improvement is illustrated in section 4 where the MSM algorithm is introduced. More details about this state coloring technique can be found in [5] and examples of RE domain-specific information in [7]. Domain knowledge represented as incompatibility constraints has also been used for subsequential transducer learning [9].

4 State-Merging with Mandatory Merge Constraints

Section 2 explains why *mandatory merge constraints* arise in our RE application context. We present here, the MSM (Mandatory State Merging) algorithm, a straightforward adaptation of Algorithm 1 to deal with such constraints.

Mandatory merge constraints are the logical counterpart to the incompatibility constraints. It is interesting to elaborate briefly on the logical differences between them. Consider for example the PTA of Fig. 4. Grey and black states are known to be incompatible: *merging them would lead to a solution that accepts at least one negative string*.

In addition, we assume that some domain-specific knowledge allows the user to state that q_2 and q_6 in this PTA do in fact represent the same state in the target machine. In our application domain, this kind of knowledge takes the following form: "From the initial state, the software accepts exactly the same future behaviors after the occurrence of event b (state 2) and the sequence of events baa (state 6)". By virtue of the mapping between states of a canonical DFA and residual languages (the "future behaviors" in our example), these states must be merged by the induction process: *not merging them will lead to a solution that does not respect this positive domain knowledge*.

Capturing mandatory merge constraints uses a similar mechanism to *state coloring*, this one being called *state labeling*. Formally, we introduce a partial labeling function $f_{lab}(Q) \rightarrow \{r, s, t, ...\}$, where Q is the set of PTA states and r, s, t are labels. States with the same label must be merged by the induction process. In our example, the fact that states q_2 and q_6 must be merged is captured by $f_{lab}(q_2) = f_{lab}(q_6)$. A quotient automaton PTA/π respects the labeling constraints if, $\forall q_1, q_2 \in Q$ such that $f_{lab}(q_1) = f_{lab}(q_2)$ and both are defined, if $q_1 \in B_i$ and $q_2 \in B_j$ then $B_i = B_j$.

Given a particular partial coloring function f_{col} and a particular partial labeling function f_{lab}, their influence on the induction process is the following. States with different colors *may not* be merged, while states with the same color *may* be merged. States with different labels *may* be merged, while states with the same label *must* be merged. It is worth stressing that the labeling information does not replace the negative knowledge, which is included in the coloring constraints. In the extreme case where all states are correctly labeled but no incompatibility constraints (hence no negative examples) are present, all states can, and actually will, be merged. In short, *labeling* and *coloring* information help together to achieve a good generalization accuracy.

The apparent symmetry between the two types of constraints is however not reflected in the merging process. Coloring (or incompatibility) constraints must be enforced at each step of the algorithm while the labeling (or mandatory) constraints need not be. Indeed, if any intermediate solution violates the coloring constraint, it will also be violated by any quotient automaton of this intermediate solution. In other words, when such a constraint is violated no more path exists in the search space to an adequate solution. This is the reason why intermediate solutions are checked at each step of the algorithm and discarded if required.

In contrast, an intermediate solution not respecting a labeling constraint does not imply that a quotient automaton of this solution would not. A trivial example of this fact is the PTA itself: in our example, states q_2 and q_6 must be merged while not being originally in the same block, thus initially violating the labeling constraint. Under the hypothesis that labeling and coloring constraints are consistent with each other, some quotient automaton of the PTA may however be consistent with all constraints. Dynamically checking whether an intermediate solution PTA/π not satisfying a labeling (or mandatory merge) constraint, can indeed lead to a consistent automaton looks to be a difficult task. The MSM algorithm described below adopts a straightforward approach to satisfy labeling constraints without the need for such a dynamic check.

MSM (see Algorithm 2) is a state-merging induction algorithm ensuring that both kinds of constraints are satisfied. As motivated in section 3.3, incompatibility constraints between states provided by the state coloring (including those provided by the

negative sample) are checked in the `Merge` function instead of the main loop; an exception mechanism can be used to handle incompatibility notifications. Secondly, all blocks that must be merged according to the labeling information are merged immediately, that is, before entering the main loop. The `FindSameBlocks` function identifies pairs of blocks in π having the same label. These blocks are merged using the `Merge` function. This ensures that potential non-determinism is actually reduced. If *coloring* and *labeling* constraints are inconsistent, the $avoid$ exception is not caught by the algorithm and typically raised to the user. If successful, the first while loop produces a quotient automaton PTA/π respecting both coloring and labeling constraints. The partition π is then further updated by the usual state-merging loop. While this loop takes care of respecting coloring constraints, the labeling constraints could simply not be violated anymore. Indeed, after the initial phase, states that must be merged are necessarily in the same blocks of π and those states will remain merged forever.

Algorithm 2. MSM, a DFA induction algorithm that satisfies *incompatibility* (coloring) and *mandatory merge* (labeling) constraints

```
Algorithm MSM
Input: A non-empty initial positive and negative sample (S+, S−)
Input: Labeling and coloring constraints
Output: A DFA A consistent with (S+, S−) and all constraints

// Compute a PTA, let N denote the number of its states
PTA ← Initialize(S+, S−); π ← {{0}, {1}, ..., {N − 1}}

// Merge all states according to labeling constraints
while (Bi, Bj) ← FindSameBlocks(π) do
    π ← Merge(π, Bk, Bl)

// Main state-merging loop
while (Bi, Bj) ← ChoosePair(π) do
    try
        π ← Merge(π, Bi, Bj)
    catch avoid
        // next state pair to consider
return PTA/π

// This function merges two blocks and removes non-determinism recursively
// while checking coloring constraints
Merge(π, Bi, Bj) begin
    if Incompatible(Bi, Bj) then
        raise avoid
    π ← π\{Bi, Bj} ∪ {Bi ∪ Bj}
    while (Bk, Bl) ← FindNonDeterminism(π, Bi, Bj) do
        π ← Merge(π, Bk, Bl)
    return π
end
```

The implementation of MSM looks *a priori* straightforward. Starting from the PTA, a DFA A is built by merging all states that must be merged. Next, the main state-merging loop is executed from A. It is however worth stressing that the *tree invariant property* does not hold in MSM. This observation may require to significantly review the actual implementation of this algorithm. Interestingly, the main merging loop and the `Merge`

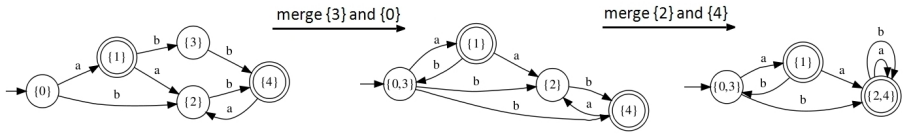

Fig. 5. Recursive determinization process. States $\{3\}$ and $\{0\}$ of an arbitrary DFA are merged, which causes a non-determinism on letter b from state $\{0,3\}$. The destination states $\{2\}$ and $\{4\}$ are subsequently merged to reduce the non-determinism.

function can be implemented without the tree invariant property because the recursive determinization process stops naturally on the first DFA encountered. This observation allows one to start from an arbitrary DFA and, as soon as non-determinism occurs, to reduce it. Figure 5 gives an example of such a recursive operation.

5 Evaluation

MSM has been evaluated on synthetic data as well as on the RE train case study briefly introduced in section 2. Sections 5.1 and 5.2 discuss the respective results of these experiments.

5.1 Experiments on Synthetic Data

To evaluate MSM on synthetic data, we used an experimentation protocol inspired from the Abbadingo contest [2]. In our current implementation, MSM is equivalent to RPNI when no mandatory merged constraints are present. In other words, the merging order is exactly the one of RPNI if no labeling constraints are considered. In this context, the objective of this evaluation is mostly to quantify the proportion of domain-specific information required to get better generalization results. The experimentation protocol that we used to achieve this objective is outlined below.

Experiments are made on randomly generated target DFAs with 32 and 64 states and an alphabet of 2 letters. Accepting states are chosen randomly by flipping a fair coin. These automata are trimmed to remove unreachable states and minimized to obtain canonical target machines. The number of states of a DFA generated using this procedure is approximately 3/5 of the requested size, which has been increased accordingly. As in Abbadingo, if the depth of the resulting automaton is not equal to $p = 2 * log_2(n) - 2$, it is simply discarded.

A learning and testing set for a target DFA with n states consists of n^2 randomly generated strings. These strings are generated using a uniform distribution over the collection of all binary strings of length $[0, p + 5]$. This set is randomly divided into two samples of the same size: a learning sample on which MSM is run and a testing sample used to measure the adequacy of the resulting solution. Strings of the learning sampled are labeled as positive or negative according to the target DFA. MSM is run on increasing proportions of the learning sample.

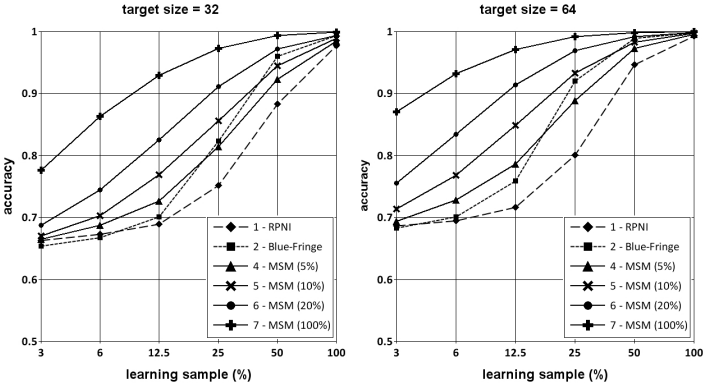

Fig. 6. Classification accuracy for RPNI, Blue-Fringe and MSM

In order to simulate domain-specific information leading to mandatory merge constraints, unique labels are associated to randomly chosen states of the target DFA. Increasing proportions of the number of states labeled in this way have been used: 5%, 10%, 20% and 100%. States of the PTA used by MSM are labeled by jointly visiting it with the target DFA and reporting encountered labels. This labeling constrains the induction process as explained in section 4.

Figure 6 reports the proportion of independent test samples correctly classified while increasing the learning sample. Curves in this plots correspond to executions of RPNI, Blue-Fringe and MSM with different labeling proportions. Each point in these plots is the average value computed over 200 independent runs. MSM overcomes RPNI on all executions, which was actually expected, but illustrates experimentally that forcing to merge initially some (correctly labeled!) states does not prevent from converging. 5% of labeling information is comparable, from the point of view of the generalization accuracy, to the use of the Blue-Fringe heuristic for selecting state pairs to be merged. Beyond this proportion, the accuracy continues to increase. Interestingly, it is already visible when the sample is sparse. This fact is particularly helpful in our RE context, where learning sample is initially provided by an end-user. Moreover, it is worth noting that, as pointed out in section 4, the identification of the target does not reduce to a trivial problem even with 100% of labeling information when only few negative examples are available.

Although not yet implemented, we are confident that using mandatory merge constraints would also improve the generalization accuracy of the Blue-Fringe and QSM algorithms (the interested reader may refer to [7] for an experimental comparison between RPNI, Blue-Fringe and QSM without labeling constraints).

5.2 Experiments on a RE Case Study

An accuracy gain is also expected when using MSM for behavior model synthesis. In order to quantify this gain, the algorithm has been evaluated on an extended version of the train system introduced in section 2. In this respect, the evaluation protocol is slightly different from the one presented in the previous section: the target model of

the train system has been built manually as a DFA of 18 states with an alphabet of size 10 (see Figure 7). A typical collection of scenarios has also been built for the system and represented as an augmented PTA: 9 positive and 5 negative strings for a total of 55 states.

Mandatory merge constraints are defined by labeling pairs of equivalent PTA states. These pairs are chosen following a "loop identification" heuristic, representative of the way such a specification is incrementally built by an end-user using an hMSC. For instance, opening then closing the doors from the initial state (without any intermediate event) naturally returns in the initial state (see the loop between states 0 and 2 in Figure 7 and the way this loop is easily represented in the hMSC of Figure 1). However, some loops are less natural to identify from the scenarios: the sequence of events ($leaving$, $high$, $approaching$, low, $atstation$) form a loop starting from state 3 for example. Equivalent state pairs of the PTA identified in this way have been classified in four categories, following the expected difficulty for an end-user of discovering them in the scenarios. MSM has been evaluated on increasing proportion of mandatory merge constraints, following this classification.

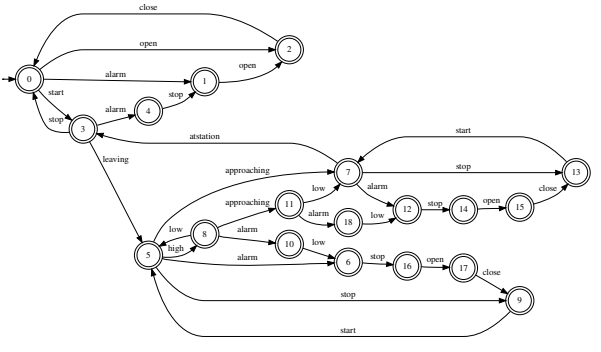

Fig. 7. Target model of the train system

Table 1. Classification accuracy obtained with different setups on the train case study. The number of labeling constraints $|lab|$ refers to the number of PTA states pairs declared to be equivalent.

Algorithm	$\|lab\|$	Accuracy
RPNI	-	0.55
BlueFringe	-	0.83
MSM	0	0.55
	3	0.71
	6	0.73
	10	0.88
	15	0.90

Table 1 compares the accuracy of the DFA induced using RPNI and BlueFringe as well as MSM with an increasing number of mandatory merge constraints. The reported accuracy is the average classification rate computed over 10 independent test samples, each one containing 80 (positive or negative) strings. The results confirm what has been observed on synthetic data. Increasing the number of mandatory merge constraints (that is, enriching the hMSC with additional transitions) leads to a better accuracy, outperforming BlueFringe when such information is rich enough. The results also show that no algorithm has been able to identify perfectly the target DFA on such sparse samples. It is worth noting that, for the need of this evaluation, only few sources of negative information have been used while such sources do exist in this application domain [7]. Further improvements could also be obtained by using mandatory merge constraints with the BlueFringe search order.

6 DFA Induction from Positive and Negative DFAs (Deterministic Finite Automata) as Inputs

Relaxing the tree invariant property has additional benefits which we discuss in this section. The main state-merging loop of MSM (see Algorithm 2) actually generalizes a language represented by an arbitrary DFA, under the control of all negative knowledge represented as incompatibility constraints. It is possible to factor out the state-merging loop from MSM as a new algorithm ASM (for Automaton State Merging), which generalizes a positive DFA A_+ taken as input, under the control of a negative sample S_-, ignoring here other incompatibility constraints for the simplicity of the discussion. The pseudo-code of ASM is given in Algorithm 3.

Algorithm 3. ASM, a DFA induction algorithm that generalizes a positive DFA A_+ under the control of a negative sample S_-

Algorithm ASM
Input: A positive DFA A_+ and a negative sample S_-
Output: A DFA A consistent with (A_+, S_-)

// Augment the automaton A_+ with states
// marked/added from S_-
$M \leftarrow$ `Augment`$(A_+,\ S_-)$

// Compute the natural order on M
$\pi \leftarrow$ `NatOrder`(M)

// Main state-merging loop
$\pi \leftarrow$ `Generalize`(π)

return M/π

The first step of this algorithm augments A_+ with the negative sample S_- in a way similar to the augmentation of a $PTA(S_+)$. Each negative string can be decomposed as uv, where u is the longest prefix already present in A_+ and reaches a state q, and v is the suffix of this negative string. When v is empty, q is marked as a negatively accepting (= black) state if not yet marked as positively accepting; if q is already a positively accepting (= grey) state, an inconsistency error between A_+ and S_- is reported to the user. When v is not empty, a new branch rooted at q is added to the automaton, ending in a new negatively accepting state.

The function `NatOrder` computes the natural order of the states of M using a breadth first search of the states and numbering each of them when encountered. The search ends when each state has been reached. We assume here for simplicity that this function returns the trivial partition π with each state in its own block, the blocks being naturally ordered. The `Generalize` function corresponds to the main state-merging loop and returns the updated partition.

ASM is the actual algorithm we use for the synthesis of behavior models in our RE application domain. Unfolding the hMSC as a labeled PTA is in fact not required and one can directly generalize, using ASM, the automaton that would be produced from the hMSC by the deductive technique of [10].

The ASM algorithm itself may be further extended. Indeed, the extension which consists in replacing finite positive string S_+ by a regular language represented as a DFA A_+, can also be applied to the negative sample. This lead to an induction algorithm ASM*, which takes as input both a positive DFA A_+ and a negative DFA A_-. In ASM*, the `Augment` function produces the composition of A_+ and A_- as a colored automaton M. More precisely, the state space of M is the product of the states of A_+ and A_- (extended whenever necessary to be complete DFAs). A state of M is

positively accepting if the corresponding state is accepting in A_+. Similarly, it is negatively accepting if the corresponding state in A_- is accepting. If both conditions hold at the same time for at least one state of M, an inconsistency error between A_+ and A_- is reported to the user. Figure 8 illustrates this composition mechanism on a simple example. The natural order is computed on M. The main state-merging loop is then executed while checking for the coloring constraints as usual in the recursive `Merge` function.

There could be no need to compute explicitly the product automaton M since the associated coloring constraints, which follow from the common prefixes between A_+ and A_-, can be computed and updated dynamically, as nicely shown in [5]. We believe however that it is useful to consider such a product automaton for defining the state ordering relation computed in `NatOrder`. Doing so, the prefixes of the negative strings in $L(A_-)$, including possibly some prefixes that do not belong to $L(A_-)$ themselves, can be used to define the search order. In this sense, strict prefixes of positive and negative strings are considered in the same way. This is indeed the natural extension to the augmented $PTA(S_+, S_-)$ used to define the search order in the Blue-Fringe algorithm. We also note that [5] generalizes those ideas to non-deterministic automata. However the authors do not stress the possibility to start the whole induction process from both a positive and a negative automaton, as a consequence of relaxing the tree invariant property.

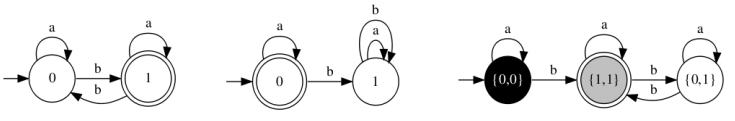

Fig. 8. Composition of A_+ (left) and A_- (middle) to get a product automaton M (right) with coloring constraints

Finally, it is worth noting that RPNI2, the incremental version of the RPNI algorithm introduced in [11], unfolds the current DFA when a new negative example is received and wrongly accepted by the current machine. The unfolded DFA is subsequently merged following a different path in the search space. As such, RPNI2 is already able to restart the generalization process from a DFA not restricted to be a tree. However the tree invariant property is satisfied in RPNI2 since the unfolding is guaranteed to go back to a temporary solution that RPNI would have considered if run on the updated samples.

7 Conclusion and Future Work

Coloring constraints are classically used in automaton induction techniques to control the state merging process while generalizing the positive sample. Such coloring constraints define which states are incompatible, that is, cannot be merged without giving rise to an inconsistent machine. We introduce here mandatory merge constraints, implemented using a partial labeling function, as the logical counterpart to the incompatibility

constraints. We propose the MSM algorithm, a natural extension to state-merging algorithms such as RPNI or Blue-Fringe, that can deal both with coloring and labeling constraints. We present experimental comparisons between MSM, RPNI and Blue-Fringe following a protocol inspired by the Abbadingo competition.

The MSM extension looks straightforward from an algorithmic point of view but it actually relaxes the tree invariant property. This property states that, among two states considered for merging, at least one is always the root of a tree-shaped (sub-)automaton. The tree invariant property is often assumed in DFA induction algorithms when recursively merging pairs of states to reduce non-determinism. However such a merging for determinization process naturally stops by itself and the tree invariant property is thus not required. As a consequence, the MSM algorithm gives rise to the ASM algorithm that takes a DFA A_+ and a negative sample S_- as inputs. ASM is the actual induction algorithm we use in our Requirements Engineering application domain which motivated, in the first place, the definition of mandatory merge constraints. We also describe the ASM* algorithm, which further extends ASM, and takes as input both a positive and a negative DFA.

Our future work includes several points. Our current implementation of MSM, and hence of ASM, relies on the RPNI search order. MSM and ASM can also be adapted to the Blue-Fringe strategy typically with the EDSM scoring function to adapt the search order. Doing so would only require to compute the scoring function between state pairs as a side product of the recursive merging operation applied here on general graphs. In this regard, the implementation would be similar to the original EDSM algorithm (*i.e.* without the Blue-Fringe strategy) but with coloring and labeling constraints. A further step is to extend the QSM algorithm [7] to add the active learning feature.

ASM* raises interesting theoretical questions since inferring from a positive and a negative DFA no longer fits exactly in the identification in the limit framework. The definition of a characteristic sample would need to be adapted as well as the experimental protocol.

References

1. Oncina, J., García, P.: Identifying regular languages in polynomial time. In: Bunke, H. (ed.) Advances in Structural and Syntactic Pattern Recognition. Series in Machine Perception and Artificial Intelligence, vol. 5, pp. 99–108. World Scientific, Singapore (1992)
2. Lang, K., Pearlmutter, B., Price, R.: Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In: Honavar, V.G., Slutzki, G. (eds.) ICGI 1998. LNCS (LNAI), vol. 1433, pp. 1–12. Springer, Heidelberg (1998)
3. Gold, E.: Language identification in the limit. Information and Control 10(5), 447–474 (1967)
4. Coste, F., Nicolas, J.: How considering incompatible state mergings may reduce the DFA induction search tree. In: Honavar, V.G., Slutzki, G. (eds.) ICGI 1998. LNCS (LNAI), vol. 1433, pp. 199–210. Springer, Heidelberg (1998)
5. Coste, F., Fredouille, D., Kermorvant, C., de la Higuera, C.: Introducing domain and typing bias in automata inference. In: Paliouras, G., Sakakibara, Y. (eds.) ICGI 2004. LNCS (LNAI), vol. 3264, pp. 115–126. Springer, Heidelberg (2004)

6. Damas, C., Lambeau, B., Dupont, P., van Lamsweerde, A.: Generating annotated behavior models from end-user scenarios. IEEE Transactions on Software Engineering 31(12), 1056–1073 (2005)
7. Dupont, P., Lambeau, B., Damas, C., van Lamsweerde, A.: The QSM algorithm and its application to software behavior model induction. Applied Artificial Intelligence 22, 77–115 (2008)
8. Dupont, P., Miclet, L., Vidal, E.: What is the search space of the regular inference? In: Carrasco, R.C., Oncina, J. (eds.) ICGI 1994. LNCS, vol. 862, pp. 25–37. Springer, Heidelberg (1994)
9. Oncina, J., Varó, M.A.: Using domain information during the learning of a subsequential transducer. In: Miclet, L., de la Higuera, C. (eds.) ICGI 1996. LNCS, vol. 1147, pp. 301–312. Springer, Heidelberg (1996)
10. Uchitel, S., Kramer, J., Magee, J.: Synthesis of behavorial models from scenarios. IEEE Transactions on Software Engineering 29(2), 99–115 (2003)
11. Dupont, P.: Incremental regular inference. In: Miclet, L., de la Higuera, C. (eds.) ICGI 1996. LNCS, vol. 1147, pp. 222–237. Springer, Heidelberg (1996)