DIAGNOSIS OF ACTIVE SYSTEMS BY LAZY TECHNIQUES

Gianfranco Lamperti and Marina Zanella
University of Brescia, Dipartimento di Ingegneria dell'Informazione, Via Branze 38, 25123 Brescia, Italy

Keywords: Diagnosis, Discrete-event systems, Active systems, Communicating automata, Uncertainty, Lazyness.

Abstract: In society, laziness is generally considered as a negative feature, if not a capital fault. Not so in computer science, where lazy techniques are widespread, either to improve efficiency or to allow for computation of unbounded objects, such as infinite lists in modern functional languages. We bring the idea of lazy computation to the context of model-based diagnosis of active systems. Up to a decade ago, all approaches to diagnosis of discrete-event systems required the generation of the global system model, a technique that is impractical when the system is large and distributed. To overcome this limitation, a lazy approach was then devised in the context of diagnosis of active systems, which works with no need for the global system model. However, a similar drawback arose a few years later, when uncertain temporal observations were proposed. In order to reconstruct the system behavior based on an uncertain observation, an *index space* is generated as the determinization of a nondeterministic automaton derived from the graph of the uncertain observation, the *prefix space*. The point is that the prefix space and the index space suffer from the same computational difficulties as the system model. To confine the explosion of memory space when dealing with diagnosis of active systems with uncertain observations, a laziness-based, circular-pruning technique is presented. Experimental results offer evidence for the considerable effectiveness of the approach, both in space and time reduction.

1 INTRODUCTION

Active systems (Lamperti and Zanella, 2003) are a class of asynchronous discrete-event systems that can be used to model, at a high abstraction level, real physical systems in order to carry out diagnosis and monitoring. In the last decade such tasks have been investigated and a number of working algorithms have been proposed (Baroni et al., 1999; Lamperti and Zanella, 2004; Lamperti and Zanella, 2006). Before the notion of active system were defined, synchronous discrete-event systems had already been considered in the literature as a promising modeling abstraction for monitoring and diagnosis purposes (Sampath et al., 1995; Sampath et al., 1996). What was actually more innovative since the very initial introduction of active systems (Baroni et al., 1998) was not, as is seemingly obvious, the different class of modeled systems (asynchronous vs. synchronous), but the ability to come to a diagnosis without previously generating the global behavioral model of the system, an ability which, although dealt with for asynchronous discrete-event systems, applies also to synchronous ones.

This ability may be considered as an instance of a *lazy* computation, as opposed to a *busy* computation. In computer science laziness is a positive feature, aimed at saving computational resources in both space and time, which is adopted in several contexts, such as Boolean expression evaluation and functional languages, including Haskell (Thompson, 1999). Trivially, laziness obeys a general principle which states that a processing step shall be performed only if and when necessary. In the context of diagnosis of discrete-event systems, the global behavioral model of the system, which encompasses all the possible evolutions of the system, compliant with whichever observation, is not strictly necessary in order to reconstruct the dynamic evolutions based on a given specific observation (as is the case in order to solve a single diagnosis/monitoring problem). Therefore, in a lazy perspective, the global behavioral model is not built and only the evolutions compliant with the given observation are reconstructed. If the same diagnostic problem occurs several times, the same on-line computation is performed in each session. In a busy perspective, instead, the global behavioral model is built off-line once and for all and then it is exploited for all diagnostic sessions, bringing a considerable gain in on-line computational complexity. The reason for a lazy computation is to be pre-

Lamperti G. and Zanella M. (2010).
DIAGNOSIS OF ACTIVE SYSTEMS BY LAZY TECHNIQUES.
In *Proceedings of the 12th International Conference on Enterprise Information Systems - Artificial Intelligence and Decision Support Systems*, pages 171-180
DOI: 10.5220/0002855101710180

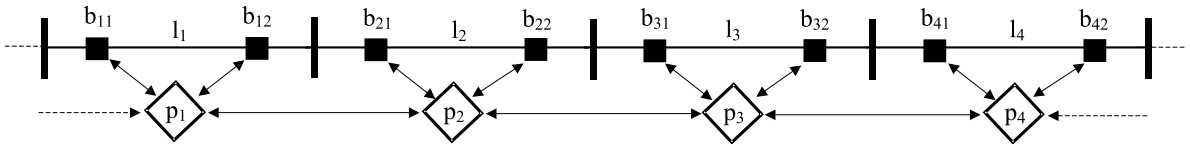

Figure 1: Fragment of power transmission network.

ferred is that the size of the global behavioral model is explosive for real-scale systems, thus a busy approach is practically infeasible. That is why all state-of-the-art approaches to diagnosis of discrete-event systems (Pencolé and Cordier, 2005) do not rely on the generation of the global behavioral system model.

Some years after the introduction of active systems, the concept of a *temporal uncertain observation* was defined (Lamperti and Zanella, 2002), and diagnosis problems featuring such a kind of observations were taken into account. An uncertain temporal observation, being under-constrained, represents several sequences of observable events. For diagnosis purposes, all the evolutions of the active system, consistent with all such sequences, have to be reconstructed on-line. Such a reconstruction is driven by a deterministic acyclic automaton, called the *index space*, which is obtained as the determinization of an acyclic automaton, called the *prefix space*, which, in turn, is drawn from a directed acyclic graph, which is the more natural front-end representation of an uncertain observation.

The approaches proposed so far to diagnose an active system given a temporal uncertain observation suppose that the whole index space is built, necessarily on-line since the observation may vary over sessions. However, the size of the prefix space and the index space is huge even for small observation graphs, therefore such approaches are practically infeasible. Moreover, such spaces may include paths that are physically impossible, that is, sequences of observable events that cannot be generated by the considered active system. The approaches proposed so far can then be considered as busy ones from the point of view of observation handling, while this paper proposes a lazy approach for diagnosis of active systems with uncertain temporal observations, that is, an approach that builds only the useful portions of the prefix space and the index-space (and avoids building physically impossible states). Envisaging such an approach is not straightforward since one can realize that a path of the index space is physically impossible, and, therefore, it has to be pruned, only based on the reconstruction of the evolutions of the system. Thus a circularity arises: on the one hand, the index space drives the evolution reconstruction and, on the other, the performed reconstruction serves as a basis for discarding states in the index space and in the prefix space as well. How to cope with this circularity is the purpose of this paper.

2 APPLICATION DOMAIN

Supervision of power networks is the application domain for which diagnosis of active systems was first conceived. A power network is composed of transmission lines. Each transmission line is protected by two breakers that are commanded by a protection. The protection is designed to detect the occurrence of a short circuit on the line based on the continuous measurement of its impedance: when the impedance goes beyond a given threshold, the two breakers are commanded to open, thereby causing the extinction of the short circuit. In a simplified view, the network is represented by a series of lines, each one associated with a protection, as displayed in Fig. 1, where lines $l_1 \cdots l_4$ are protected by protections $p_1 \cdots p_4$, respectively. For instance, p_2 controls l_2 by operating breakers b_{21} and b_{22}. In normal (correct) behavior, both breakers are expected to open when tripped by the protection. However, the protection system may exhibit an abnormal (faulty) behavior, for example, one breaker or both may not open when required. In such a case, each faulty breaker informs the protection about its own misbehavior. Then, the protection sends a request of recovery actions to the neighboring protections, which will operate their own breakers appropriately. For example, if p_2 operates b_{21} and b_{22} and the latter is faulty, then p_2 will send a signal to p_3, which is supposed to command b_{32} to open. A recovery action may be faulty on its turn. For example, b_{32} may not open when tripped by p_2, thereby causing a further propagation of the recovery to protection p_4. The protection system is designed to propagate the recovery request until the tripped breaker opens correctly. When the protection system is reacting, a subset of the occurring events are visible to the operator in a control room who is in charge of monitoring the behavior of the network and, possibly, to issue explicit commands so as to minimize the extent of the isolated sub-network. Generally speaking, the localization of the short circuit and the identification of the faulty breakers may be impractical in real contexts,

especially when the extent of the isolation spans several lines and the operator is required to take recovery actions within stringent time constraints. On the one hand, there is the problem of observability: the observable events generated during the reaction of the protection system are generally uncertain in nature. On the other, it is impractical for the operator to reason on whatever observation so as to make consistent hypotheses on the behavior of the system and, eventually, to establish the shorted line and the faulty breakers.

3 DIAGNOSIS TASK

An active system is a network of components that are connected to one another through links. Each component is modeled by a communicating automaton that reacts to events either coming from the external world or from neighboring components. Events exchanged between components are queued into links before being consumed. The way a system reacts to an event coming from the external world is constrained by the communicating automata of the involved components and the way such components are connected to one another. The whole set of evolutions of a system Σ, starting at the initial state σ_0, is confined to a finite automaton, the *behavior space* of Σ, $Bsp(\Sigma,\sigma_0)$. However, a strong assumption for diagnosis of active systems is the unavailability of the behavior space since, in real, large-scale applications, the generation of the behavior space is impractical. As such, $Bsp(\Sigma,\sigma_0)$ is intended for formal reasons only. A (possibly empty) path within $Bsp(\Sigma,\sigma_0)$ rooted in σ_0 is a *history* of Σ. When the system reacts, it performs a sequence of transitions within the behavior space, called the *actual history* of the system. Some of these transitions are observable as *visible labels*. Also, each transition can be either *normal* or *faulty*. If faulty, the transition is associated with a *faulty label*. Given a history h, the (possibly empty) set of faulty labels encompassed by h is the *diagnosis* entailed by h. Likewise, the sequence of visible labels encompassed by h is the *trace* of h.

Example 1. Shown in Fig. 2 is an abstraction of the behavior space $Bsp(\Sigma,\sigma_0)$. We assume that each arc corresponds to a component transition, which moves the system from one state to another. In the figure, only the visible labels of observable transitions, namely a, b, and c, are displayed. A possible history is $[\sigma_0,\sigma_2,\sigma_4,\sigma_2,\sigma_4,\sigma_6,\sigma_8]$, with trace $[a,c,b]$.

Ideally, the reaction of a system should be observed as the trace of the actual history. However,

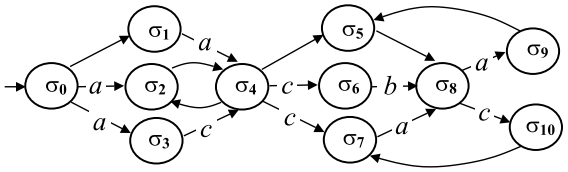

Figure 2: Behavior space $Bsp(\Sigma,\sigma_0)$.

what is actually observed is a temporal observation O. This is a directed acyclic graph, where nodes are marked by sets of *candidate visible labels*, while arcs denote partial temporal ordering among nodes. For each node, only one label is the *actual label* (the one in the actual history), with the others being the *spurious labels*. The set of labels in a node ω of O is denoted as $\|\omega\|$. Since temporal ordering is only partial, several *candidate traces* are possible for O, with each candidate being determined by choosing a label for each node while respecting the ordering constraints imposed by arcs. The set of candidate traces is written $\|O\|$.

Example 2. Depicted in Fig. 3 is a temporal observation O involving nodes $\omega_1,\ldots,\omega_4$. Node ω_2 is marked by labels b and ε, where the latter is the *null* label, which is in fact invisible. Thus, as far as ω_2 is concerned, either b or nothing has been generated by the system. Since ω_3 and ω_4 are connected by an arc, c necessarily precedes this occurrence of b in any trace. Note that trace $[a,c,b]$ belongs to $\|O\|$.

Figure 3: Temporal observation O for system Σ.

A *diagnostic problem* $\wp(\Sigma)$ requires determining the set of *candidate diagnoses* implied by the histories of Σ whose traces are in $\|O\|$. Intuitively, the (possibly infinite) set of histories in $Bsp(\Sigma,\sigma_0)$ is filtered based on the constraints imposed by each trace relevant to O. Since among such traces is the (unknown) actual trace, among the candidate diagnoses will be the diagnosis implied by the actual history, namely the (unknown) *actual diagnosis*. To solve $\wp(\Sigma)$, the diagnostic engine performs three major steps:

1. *Indexing*. An *index space* $Isp(O)$ is generated from O. This is a deterministic automaton whose regular language is $\|O\|$.
2. *Reconstruction*. Based on $Isp(O)$, the set of histories whose trace is in $\|O\|$ is determined in terms of a *behavior*, written $Bhv(\wp(\Sigma))$. This is an automaton such that each state is a pair $(\sigma,\Im)$, where σ is a state in $Bsp(\Sigma,\sigma_0)$ and $\Im$ a state in $Isp(O)$. A transition $(\sigma,\Im)\xrightarrow{T}(\sigma',\Im')$ in $Bhv(\wp(\Sigma))$ is

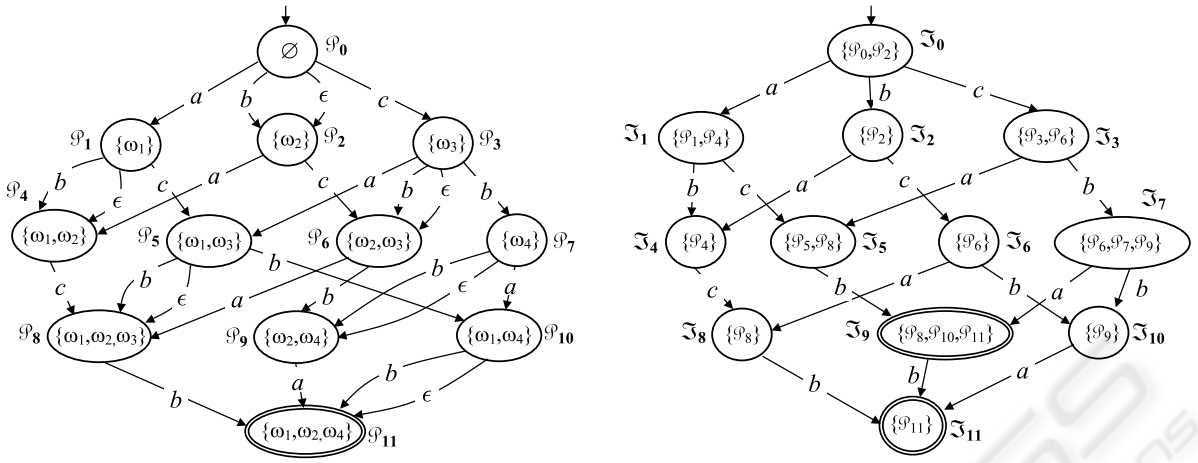

Figure 4: Prefix space $Psp(O)$ (left) and index space $Isp(O)$ (right).

such that $\sigma \xrightarrow{T} \sigma'$ is a transition in $Bsp(\Sigma, \sigma_0)$. Besides, if T is visible with label ℓ, then $\Im \xrightarrow{\ell} \Im'$ is a transition in $Isp(O)$, otherwise $\Im' = \Im$.

3. *Decoration*. Each state in $Bhv(\wp(\Sigma)$ is decorated by the set of diagnoses implied by all histories ending at such a state.

Eventually, the solution of $\wp(\Sigma)$ is determined by distilling the diagnoses, in the decorated behavior, whose state is associated with a final state of $Isp(O)$.

Example 3. Outlined on the right-hand side of Fig. 4 is the index space of observation O (Fig. 3), namely $Isp(O)$. This is generated as the determinization of a nondeterministic automaton called the *prefix space* of O, written $Psp(O)$, outlined on the left-hand side. Each state $\mathcal{P}$ of $Psp(O)$ is a set of nodes of O, called a *prefix* of O. A prefix $\mathcal{P}$ implicitly identifies $\mathcal{P}^*$, where $\mathcal{P}^*$ is the union of $\mathcal{P}$ and the set of ancestors (in O) of all nodes in $\mathcal{P}$. For instance, $\mathcal{P}_9$ identifies $\{\omega_2, \omega_4\} \cup \{\omega_3\}$. $Psp(O)$ is generated starting from the empty set $\mathcal{P}_0$ and, for each state $\mathcal{P}$, selecting one node $\omega \in O$ not included in $\mathcal{P}^*$, whose ancestors are included in $\mathcal{P}^*$. Then, for each label $\ell \in \|\omega\|$, a transition $\mathcal{P} \xrightarrow{\ell} \mathcal{P}'$ is inserted, where $\mathcal{P}'$ is the prefix identifying the set $\mathcal{P}^* \cup \{\omega\}$. Only one final state exists, (in our example, $\mathcal{P}_{11}$), identifying all nodes of O.

Example 4. Shown in Fig. 5 is the reconstructed behavior (plain part of the graph) relevant to a diagnostic problem $\wp(\Sigma)$, where the behavior space of Σ is in Fig. 2, the observation O in Fig. 3, and the index space of O in Fig. 4. The gray part of the graph is generated by the reconstruction algorithm, but is eventually discarded as spurious (since it is not encompassed by any path from initial state $(\sigma_0, \Im_0)$ to final state $(\sigma_8, \Im_9)$. Notice that the regular language of $Bhv(\wp(\Sigma))$ is the

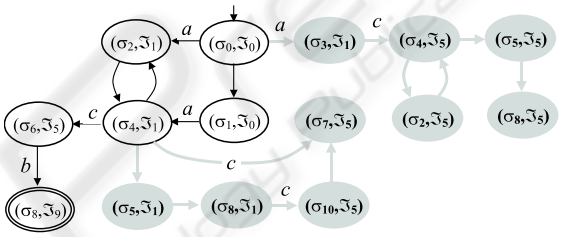

Figure 5: Reconstruction of behavior $Bhv(\wp(\Sigma))$.

singleton $\{[a,c,b]\}$, despite the infinite number of histories (owing to cycles).

The previous example shows that the language of $Bhv(\wp(\Sigma))$ is a subset of the language of $Isp(O)$. Precisely, the language of the reconstructed behavior is the intersection of the language of the index space and the language of the behavior space.

4 LAZY DIAGNOSTIC ENGINE

The systematic approach to problem solving introduced above may become inappropriate owing to the explosion of the prefix space and, consequently, of the index space. This problem arose when experimenting with algorithms for subsumption-checking of temporal observations (Lamperti and Zanella, 2008). The cause for the huge number of nodes can be understood by analyzing how the index space is generated. Given an observation O, the prefix space of O is built by considering all possible ways in which nodes of O can be selected, based on the precedence constraints imposed by the arcs of O. At each choice, we create new transitions in the prefix space, marked by the labels within the selected node of O, and con-

nect each of them to another state of $Psp(O)$. Such a state is marked by a set of nodes (a prefix) of O that identifies the whole set of nodes already chosen in O. Intuitively, the less temporally constrained O, the larger the set of possible sequences of choices. The exact number of states in $Psp(O)$ equals the cardinality of the whole set of prefixes of O. In particular, assuming n nodes in O, if O is linear (nodes totally ordered) then the number of states in $Psp(O)$ equals $n+1$. If O is totally disconnected (nodes temporally unconstrained) then the number of states in $Psp(O)$ equals 2^n. So, in the worst case, the number of states of $Psp(O)$ grows exponentially with the number of nodes in O. In practice, even for disconnected observations of moderate size, say 40 nodes, the prefix space contains 2^{40} states, corresponding to more than 10^{12} states! With such numbers, if the generation of the prefix space is impractical, the transformation of it into the equivalent deterministic automaton, the index space, is simply out of question. So, what to do? Generally speaking, not all the candidate traces included in $Isp(O)$ are consistent with the behavior space of the system, just as not all the histories included in the behavior space are consistent with $Isp(O)$. In fact, in the reconstruction phase, we filter out the histories in $Bsp(\Sigma,\sigma_0)$ based on the constraints imposed by $Isp(O)$, thereby yielding $Bhv(\wp(\Sigma))$. Now, the point is, we might try to perform some sort of pruning of the index space $Isp(O)$ based on the constraints imposed by the behavior space $Bsp(\Sigma,\sigma_0)$. However, this would work only assuming the availability of the latter, which is not the case. A better idea is to filter out the index space based on the reconstructed behavior $Bhv(\wp(\Sigma))$. This allows us to avoid the generation of $Bsp(\Sigma,\sigma_0)$. By contrast, the problem is now that $Bhv(\wp(\Sigma))$ is itself generated based on $Isp(O)$, giving rise to a circularity: we need $Isp(O)$ to generate $Bhv(\wp(\Sigma))$ and we need $Bhv(\wp(\Sigma))$ to generate $Isp(O)$. Interestingly, we can cope with this circularity by building the index space and the reconstructed behavior adopting a lazy approach, where the constructions of the two automata are intertwined. So, the reciprocal constraints can be checked at each step.

A second shortcoming of the systematic approach to problem solving concerns the structure of the reconstructed behavior.

- Let $\beta=(\sigma,\Im)$ be either the initial state or a state reached by a visible transition in $Bhv(\wp(\Sigma))$. Let $Silent(\beta)$ be the subgraph of $Bhv(\wp(\Sigma))$ rooted in β and reached by silent transitions only. Then, all states in $Silent(\beta)$ will share the same index $\Im$.
- Let $\beta_1=(\sigma,\Im_1)$ and $\beta_2=(\sigma,\Im_2)$ be two states in $Bhv(\wp(\Sigma))$ sharing the same system state σ. Then, the projections of $Silent(\beta_1)$ and $Silent(\beta_2)$ on $Bsp(\Sigma,\sigma_0)$ are identical. In other words, if we remove the indexes $\Im_1$ and $\Im_2$ from $Silent(\beta_1)$ and $Silent(\beta_2)$, respectively, we come up with the same fragment of the behavior space.

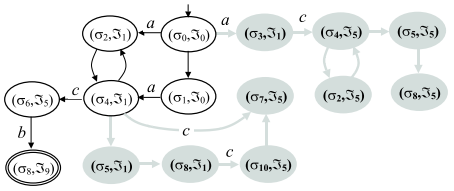

Figure 6: Condensed behavior space $\mathbf{Bsp}(\Sigma,\sigma_0)$.

These peculiarities of $Bhv(\wp(\Sigma))$ suggest that there is some redundancy in its reconstruction. On the one hand, states of $Bhv(\wp(\Sigma))$ marked by the same index $\Im$ can be grouped to form a fragment of $Bsp(\Sigma,\sigma_0)$ involving silent transitions only. This way, index $\Im$ can be associated with the whole fragment rather than with each state within the fragment. On the other, and more importantly, since each fragment functionally depends on its root β (either the initial state of $Bhv(\wp(\Sigma))$ or a state reached by a visible transition), a previous generation of the fragment can be reused with no need for model-based reasoning when β is generated as the next state in $Bhv(\wp(\Sigma))$. This way, we avoid re-generating the duplicated fragment of behavior. This factorization can be defined for the behavior space too, giving rise to the notion of *condensed behavior space*, $\mathbf{Bsp}(\Sigma,\sigma_0)$. Each state $\mathcal{C}\in\mathbf{Bsp}(\Sigma,\sigma_0)$ is a *condensation*, namely $\mathcal{C}=Cond(\sigma)$, where σ is the *root* of $\mathcal{C}$. The *exit states* of $\mathcal{C}$ are those exited by (at least) one visible transition directed towards another (possibly the same) condensation.

Example 5. Shown in Fig. 6 is the condensed behavior space $\mathbf{Bsp}(\Sigma,\sigma_0)$ relevant to $Bsp(\Sigma,\sigma_0)$ in Fig. 2.

Based on $\mathbf{Bsp}(\Sigma,\sigma_0)$ we can define the notion of a *condensed behavior* $\mathbf{Bhv}(\wp(\Sigma))$ as the automaton whose nodes are associations $(\mathcal{C},\Im)$ between a condensation $\mathcal{C}$ in $\mathbf{Bsp}(\Sigma,\sigma_0)$ and a state $\Im$ in $Isp(O)$. In the initial state $(\mathcal{C}_0,\Im_0)$, $\mathcal{C}_0$ is the initial state of $\mathbf{Bsp}(\Sigma,\sigma_0)$ and $\Im_0$ is the initial state of $Isp(O)$. A transition $(\mathcal{C},\Im)\xrightarrow{T}(\mathcal{C}',\Im')$ is such that $\mathcal{C}\xrightarrow{T}\mathcal{C}'$ is a transition in $\mathbf{Bsp}(\Sigma,\sigma_0)$, ℓ is the visible label of T, and $\Im\xrightarrow{\ell}\Im'$ is a transition in $Isp(O)$.

4.1 *LISCA* Algorithm

In order to perform circular pruning, the *lazy diagnostic engine* is required to determinize $Psp(O)$ into $Isp(O)$ incrementally, by exploiting the layered structure of the former. In fact, if n is the number of nodes of O, $Psp(O)$ is made of $n+1$ layers. An algorithm,

called *LISCA*, has been developed as an extended specialization of the *Incremental Subset Construction* algorithm for determinization of finite automata (Lamperti et al., 2008). *LISCA* allows the index space to be updated at the generation of each new layer of the prefix space. The pseudo-code of *LISCA* is outlined below (lines 1–65). *LISCA* takes as input the current portion of the prefix space, **P**, the corresponding portion of the index space, **I**, and the set of transitions **T** extending **P** to the next layer. As a side effect, *LISCA* updates both **P** and **I** based on **T**. Besides, it outputs the sequence $\mathcal{U}$ of the *update actions* performed on **I**, to be exploited subsequently by the diagnostic engine for the layered reconstruction of the condensed behavior. The algorithm makes use of the auxiliary procedure *Extend* (lines 10–31). The latter takes as input a state $\mathfrak{I}$ of **I** and a set $\mathbb{P}$ of states in **P**. Three side effects may hold: the content of $\mathfrak{I}$ is extended by $\mathbb{P}$, the extended $\mathfrak{I}$ is merged with another state $\mathfrak{I}'$, and the *update sequence* $\mathcal{U}$ is extended by the topological action performed on **I**. Note that, based on line 19, the processing of *Extend* is performed only if $\mathbb{P}$ is not a subset of $\|\mathfrak{I}\|$. Since the extension of $\|\mathfrak{I}\|$ by $\mathbb{P}$ may cause a collision with an existing state $\mathfrak{I}'$, a merging of $\mathfrak{I}$ and $\mathfrak{I}'$ is made in lines 22–25. This consists in redirecting towards/from $\mathfrak{I}$ all transitions entering/exiting $\mathfrak{I}'$, in removing $\mathfrak{I}'$, and in renaming to $\mathfrak{I}$ the buds relevant to $\mathfrak{I}'$ (the notion of a bud is introduced shortly). Eventually, the actual update action is recorded into $\mathcal{U}$, namely either $Ext(\mathfrak{I})$ (extension without merging) or $Mrg(\mathfrak{I},\mathfrak{I}')$ (extension with merging). The body of *LISCA* is coded in lines 32–65. After the extension of **P** by the new transitions in **T**, the *bud set* $\mathcal{B}$ is instantiated (line 35). Each *bud* in $\mathcal{B}$ is a triple $(\mathfrak{I},\ell',\mathbb{P}')$, where $\mathfrak{I}$ is a state in **I**, ℓ' is a label marking a transition in **T**, and $\mathbb{P}'$ is the ε-closure[1] of the set of states $\mathcal{P}'$ entered by transitions in **T** from a state $\mathcal{P} \in \|\mathfrak{I}\|$, which are marked by label ℓ'. Intuitively, a bud indicates that $\mathfrak{I}$ is bound to some update, either by the extension of $\|\mathfrak{I}\|$ (when $\ell' = \varepsilon$) or by a transition exiting $\mathfrak{I}$ (when $\ell' \neq \varepsilon$). $Inc(\mathfrak{I})$ denotes the set of inconsistent labels of $\mathfrak{I}$: these labels are determined during the layered reconstruction of the condensed behavior. After the initialization of the update sequence $\mathcal{U}$ at line 36, a loop is iterated within lines 37–63. At each iteration, a bud $(\mathfrak{I},\ell,\mathbb{P})$ is considered. Three main scenarios are possible:

- Lines 39–40: $\ell = \varepsilon$. $\mathfrak{I}$ is extended by $\mathbb{P}$.
- Lines 41–48: $\ell \neq \varepsilon$ and there is no transition exiting $\mathfrak{I}$ and marked by ℓ. Two cases are possible:
  - Lines 42–43: a state $\mathfrak{I}'$ already exists, such that $\|\mathfrak{I}'\| = \mathbb{P}$. A new transition $\mathfrak{I} \xrightarrow{\ell} \mathfrak{I}'$ is created.
  - Lines 44–46: $\nexists$ a state $\mathfrak{I}'$ such that $\|\mathfrak{I}'\| = \mathbb{P}$. Both the (empty) state $\mathfrak{I}'$ and transition $\mathfrak{I} \xrightarrow{\ell} \mathfrak{I}'$ are created. Then, $\mathfrak{I}'$ is extended by $\mathbb{P}$.

  Eventually, $New(\mathfrak{I} \xrightarrow{\ell} \mathfrak{I}')$ is appended to $\mathcal{U}$.
- Lines 49–61: $\ell \neq \varepsilon$ and there exists a transition exiting $\mathfrak{I}$ that is marked by ℓ. Generally speaking, owing to a possible previous merging by *Extend*, several transitions marked by the same label ℓ may exit $\mathfrak{I}$.[2] Thus, each transition $\mathfrak{I} \xrightarrow{\ell} \mathfrak{I}'$ is considered in lines 50–61. After verifying that $\mathbb{P}$ is not contained in $\|\mathfrak{I}'\|$, two cases are possible:
  - Lines 52–53: there does not exist another transition entering $\mathfrak{I}'$, which is extended by $\mathbb{P}$.
  - Lines 55–58: there exists another transition entering $\mathfrak{I}'$. A new state $\mathfrak{I}''$ is created as a copy of $\mathfrak{I}'$. Also, for each transition $\mathfrak{I}' \xrightarrow{x} \bar{\mathfrak{I}}$ a new transition $\mathfrak{I}'' \xrightarrow{x} \bar{\mathfrak{I}}$ is created. Then, $\mathfrak{I} \xrightarrow{\ell} \mathfrak{I}'$ is redirected towards $\mathfrak{I}''$. Eventually, after appending the update action $Dup(\mathfrak{I} \xrightarrow{\ell} \mathfrak{I}',\mathfrak{I}'')$ to $\mathcal{U}$, the content of $\mathfrak{I}''$ is extended by $\mathbb{P}$.

The loop terminates when the bud set $\mathcal{B}$ becomes empty (line 63, all buds processed). This causes the output of $\mathcal{U}$ and the termination of *LISCA*.

```
1  Algorithm LISCA(P, I, T) → U
2   input
3    P: a portion of a (pruned) prefix space up to level k,
4    I: the (pruned) index space equivalent to P (up to level k),
5    T: a set of transitions extending P to level k+1;
6   side effects
7    Update of P and I;
8   output
9    U: the sequence of relevant updates in I;

10 auxiliary procedure Extend(ℑ, ℙ)
11  input
12   ℑ: a state in I,
13   ℙ: a subset of states in P;
14  side effects
15   Extension of ‖ℑ‖ by ℙ,
16   Possible merging of ℑ with another state ℑ′ in I,
17   Extension of the update sequence U;
18  begin {Extend}
19  if ℙ ⊈ ‖ℑ‖ then
20   Insert ℙ into ‖ℑ‖;
21   if I includes a state ℑ′ such that ‖ℑ′‖ = ‖ℑ‖ then
22    Redirect to ℑ all transitions entering ℑ′;
23    Redirect from ℑ all transitions exiting ℑ′ and remove
```

[1] The ε-closure of a set of states $\mathbb{S}$ in a nondeterministic automaton is the union of $\mathbb{S}$ and set of states reachable from each state in $\mathbb{S}$ by paths of transitions marked by label ε.

[2] During the processing of the bud set, **I** may become nondeterministic. However, such nondeterminism always disappears in the end.

duplicated transitions;
Remove $\mathfrak{I}'$ from **I**;
Rename to $\mathfrak{I}$ the buds in $\mathcal{B}$ relevant to $\mathfrak{I}'$;
Append $Mrg(\mathfrak{I},\mathfrak{I}')$ to $\mathcal{U}$
else
Append $Ext(\mathfrak{I})$ to $\mathcal{U}$
end-if
end-if
end {Extend};
begin {LISCA}
Update **P** by the additional transitions in **T**;
Let $\mathcal{L}$ be the set of labels marking transitions in **T**;
$\mathcal{B} := \{(\mathfrak{I},\ell',\mathbb{P}') \mid \mathfrak{I} \in \mathbf{I},\ \ell' \in (\mathcal{L} - Inc(\mathfrak{I})),$
$\mathbb{P}'_{\ell'} = \{\mathcal{P}' \mid \mathcal{P} \xrightarrow{\ell'} \mathcal{P}' \in \mathbf{T}, \mathcal{P} \in \|\mathfrak{I}\|\},$
$\mathbb{P}'_{\ell'} \neq \emptyset,\ \mathbb{P}' = \varepsilon\text{-}closure(\mathbb{P}'_{\ell'})\}$;
$\mathcal{U} := [\,]$;
loop
Remove a bud $(\mathfrak{I},\ell,\mathbb{P})$ from $\mathcal{B}$;
if $\ell = \varepsilon$ **then**
$Extend(\mathfrak{I},\mathbb{P})$
elsif $\nexists$ a transition exiting $\mathfrak{I}$ and marked by ℓ **then**
if I includes a state $\mathfrak{I}'$ such that $\|\mathfrak{I}'\| = \mathbb{P}$ **then**
Insert a new transition $\mathfrak{I} \xrightarrow{\ell} \mathfrak{I}'$ into **I**
else
Create in **I** a new state $\mathfrak{I}'$ and a new transition $\mathfrak{I} \xrightarrow{\ell} \mathfrak{I}'$;
$Extend(\mathfrak{I}',\mathbb{P})$
end-if;
Append $New(\mathfrak{I} \xrightarrow{\ell} \mathfrak{I}')$ to $\mathcal{U}$
else
for each transition $\mathfrak{I} \xrightarrow{\ell} \mathfrak{I}'$ **do**
if $\mathbb{P}\|\mathfrak{I}'\|$ **then**
if $\nexists$ another transition entering $\mathfrak{I}'$ **then**
$Extend(\mathfrak{I}',\mathbb{P})$
else
Create a copy $\mathfrak{I}''$ of $\mathfrak{I}'$, with all exiting transitions;
Redirect $\mathfrak{I} \xrightarrow{\ell} \mathfrak{I}'$ towards $\mathfrak{I}''$;
Append $Dup(\mathfrak{I} \xrightarrow{\ell} \mathfrak{I}',\mathfrak{I}'')$ to $\mathcal{U}$;
$Extend(\mathfrak{I}'',\mathbb{P})$
end-if
end-if
end-for
end-if
while $\mathcal{B} \neq \emptyset$;
return $\mathcal{U}$;
end {LISCA}.

4.2 Circular Pruning

Circular pruning amounts to intertwining the generation of the index space and the reconstruction of the condensed behavior so as to prune them at each *layering step*, with the latter consisting of the following sequence of actions:

1. *Generation of the next layer of the prefix space (states at the same next level along with relevant transitions)*;
2. *Update of the corresponding index space by means of the* LISCA *algorithm*;
3. *Extension of the condensed behavior based on the updates of the index space*;
4. *Pruning of the condensed behavior based on its new topology*;
5. *Pruning of the index space based on the updated condensed behavior*;
6. *Backward propagation of the pruning of the index space to the prefix space*.

Once generated the next layer of $Psp(O)$, *LISCA* extends $Isp(O)$ and returns the sequence of relevant updates $\mathcal{U}$. Once updated $Isp(O)$, the condensed behavior can be extended based on the extensions recorded in $\mathcal{U}$. The update actions are considered in the order they have been stored in $\mathcal{U}$ and processed as follows.

- $Ext(\mathfrak{I})$: Each state $(\mathcal{C},\mathfrak{I}) \in \mathbf{Bhv}(\wp(\Sigma))$ is qualified as belonging to the new frontier of the condensed behavior. This information is exploited for pruning the latter.
- $Mrg(\mathfrak{I},\mathfrak{I}')$: For each pair $(\mathcal{C},\mathfrak{I})$, $(\mathcal{C},\mathfrak{I}')$ of states in $\mathbf{Bhv}(\wp(\Sigma))$, all transitions entering $(\mathcal{C},\mathfrak{I}')$ are redirected towards $(\mathcal{C},\mathfrak{I})$, and, then, $(\mathcal{C},\mathfrak{I}')$ is removed along with all its exiting transitions.
- $New(\mathfrak{I} \xrightarrow{\ell} \mathfrak{I}')$: Each state $(\mathcal{C},\mathfrak{I})$ in $\mathbf{Bhv}(\wp(\Sigma))$ is considered. Let $\mathbb{T}_\ell$ be the set of transitions $\sigma \xrightarrow{T} \sigma'$ leaving an exit state of $\mathcal{C}$ such that T is a visible transition associated with label ℓ. Then, for each $\sigma \xrightarrow{T} \sigma' \in \mathbb{T}_\ell$, $\mathbf{Bhv}(\wp(\Sigma))$ is extended by $(\mathcal{C},\mathfrak{I}) \xrightarrow{T} (\mathcal{C}',\mathfrak{I}')$, where $\mathcal{C}'$ is the condensation rooted in σ'. If there exists a $\mathbb{T}_\ell$ which is not empty, then $\mathfrak{I} \xrightarrow{\ell} \mathfrak{I}'$ is marked as consistent in $\mathcal{U}$.
- $Dup(\mathfrak{I} \xrightarrow{\ell} \mathfrak{I}',\mathfrak{I}'')$: The subsequent redirection of $\mathfrak{I} \xrightarrow{\ell} \mathfrak{I}'$ towards $\mathfrak{I}''$ is mimicked in $\mathbf{Bhv}(\wp(\Sigma))$ as follows. Each transition $(\mathcal{C},\mathfrak{I}) \xrightarrow{T} (\mathcal{C}',\mathfrak{I}')$ in $\mathbf{Bhv}(\wp(\Sigma))$ is replaced by the new transition $(\mathcal{C},\mathfrak{I}) \xrightarrow{T} (\mathcal{C}',\mathfrak{I}'')$, where $(\mathcal{C}',\mathfrak{I}'')$ is a newly created state (notice that $\mathcal{C}'$ is already involved in the condensed behavior, though associated with a different index, namely $\mathfrak{I}'$). Besides, just as for the index space, for each transition $(\mathcal{C}',\mathfrak{I}') \xrightarrow{T^*} (\mathcal{C}^*,\mathfrak{I}^*)$, a new transition $(\mathcal{C}',\mathfrak{I}'') \xrightarrow{T^*} (\mathcal{C}^*,\mathfrak{I}^*)$ is created. These operations do not alter the regular language of the condensed behavior, thereby, there is no need for checking the consistency of the new transitions.

Once extended, the condensed behavior can be pruned as follows. Let $\mathbb{B}$ and $\mathbb{B}'$ be the sets of frontier nodes of $\mathbf{Bhv}(\wp(\Sigma))$ before and after the extension of the

latter, respectively. Formally, a node $(\mathcal{C},\mathfrak{I})$ is within $\mathbb{B}'$ when either $Ext(\mathfrak{I})$, $Mrg(\mathfrak{I},\mathfrak{I}')$, $New(\mathfrak{I}_1 \xrightarrow{\ell} \mathfrak{I})$, or $Dup(\mathfrak{I}_1 \xrightarrow{\ell} \mathfrak{I}_2,\mathfrak{I})$ is in $\mathcal{U}$.

Let $\mathbb{B}^* = \mathbb{B} - \mathbb{B}'$. For each $(\mathcal{C},\mathfrak{I}) \in \mathbb{B}^*$, if $\mathfrak{I}$ is not final in $Isp(O)$ and there does not exist a transition exiting $(\mathcal{C},\mathfrak{I})$, then $(\mathcal{C},\mathfrak{I})$ is removed with its entering transitions, while the parents of the removed node are inserted into $\mathbb{B}^*$ (for upward cascade pruning).

Once the condensed behavior has been extended based on $\mathcal{U}$, the index space can be pruned based on the unmarked (inconsistent) transitions. To this end, each transition $\mathfrak{I} \xrightarrow{\ell} \mathfrak{I}'$ not marked as consistent in $\mathcal{U}$ is removed from the index space. Furthermore, if $\mathfrak{I}'$ becomes isolated (no entering transition) then $\mathfrak{I}'$ too is removed from the index space. The removal of a node from the index space is sound because we can prove that such a node will no longer be reached by any transition in future extensions of the index space. We can also prove that downward cascade pruning cannot hold in $Isp(O)$.

The pruning of the index space is propagated to the prefix space. To this purpose, the *frontier* $\mathbb{I}_i$ of $Isp(O)$ is considered, this being the set of all (not pruned) nodes of $Isp(O)$ that have either been generated or extended by *LISCA* in the current iteration i. Such nodes are reached by the only sequences of observable labels that are consistent with the behavior reconstructed so far, where such sequences are the only ones that will possibly be extended in the further iteration. Let $\mathbb{P}_i$ be the set of states belonging to the i-th layer of the prefix space, which is the layer that has been generated at the current iteration. Each node $\mathcal{P} \in \mathbb{P}_i$ such that $\mathcal{P} \notin \bigcup_{\mathfrak{I}\in\mathbb{I}_i} \|\mathfrak{I}\|$ has to be removed from the prefix space (along with dangling transitions).

Example 6. Consider the diagnostic problem $\wp(\Sigma)$ defined in Example 4. Fig. 7 shows how to solve the same problem by means of the lazy diagnostic engine, namely *LDE*. First, the initial states of $Psp(O)$ (at layer 0), $Isp(O)$, and $\mathbf{Bhv}(\wp(\Sigma))$ are generated. Then, *LDE* loops 4 times, where 4 is the number of nodes in O (which equals the number of successive layers in $Psp(O)$), as detailed below.

1. The first layer of $Psp(O)$ is generated, involving states $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$, with transitions from $\mathcal{P}_0$. Then, $Isp(O)$ is extended by *LISCA*. Updates in $\mathcal{U}$ are $Ext(\mathfrak{I}_0)$, $New(\mathfrak{I}_0 \xrightarrow{a} \mathfrak{I}_1)$, $New(\mathfrak{I}_0 \xrightarrow{b} \mathfrak{I}_2)$, and $New(\mathfrak{I}_0 \xrightarrow{c} \mathfrak{I}_3)$. Then, $\mathbf{Bhv}(\wp(\Sigma))$ is extended by three nodes, with the only consistent transition in $Isp(O)$ being $\mathfrak{I}_0 \xrightarrow{a} \mathfrak{I}_1$. Thus, the other two transitions are pruned from $Isp(O)$ (along with states $\mathfrak{I}_2$ and $\mathfrak{I}_3$). This pruning is propagated to $Psp(O)$, where $\mathcal{P}_3$ is removed, along with its entering (dangling) transition.
2. The second layer of $Psp(O)$ is generated, involving states $\mathcal{P}_4$, $\mathcal{P}_5$, and $\mathcal{P}_6$, and relevant entering transitions. The extension of $Isp(O)$ involves $Ext(\mathfrak{I}_1)$, $New(\mathfrak{I}_1 \xrightarrow{b} \mathfrak{I}_4)$, and $New(\mathfrak{I}_1 \xrightarrow{c} \mathfrak{I}_5)$. Note how $\mathcal{P}_2 \xrightarrow{c} \mathcal{P}_6$ does not cause any update in $Isp(O)$. In fact, we would expect $New(\mathfrak{I}_0 \xrightarrow{c} \mathfrak{I}_6)$. The point is that, once a transition exiting a state $\mathfrak{I}$ and marked by ℓ is removed, $\mathfrak{I}$ is decorated with the inconsistent label ℓ, so that all subsequent attempt to extend $\mathfrak{I}$ with a transition marked by ℓ will be prevented. The extension of $\mathbf{Bhv}(\wp(\Sigma))$ creates four new states, all entered by transitions marked by c: only $\mathfrak{I}_1 \xrightarrow{c} \mathfrak{I}_5$ is marked as consistent. Thus, $\mathfrak{I}_1 \xrightarrow{b} \mathfrak{I}_4$ is removed from $Isp(O)$. Consequently, $\mathcal{P}_6$ and its entering transition are removed from $Psp(O)$.
3. The third layer of $Psp(O)$ is generated, with $\mathcal{P}_8$ and $\mathcal{P}_{10}$ being the newly-created states. This causes the extension of $Isp(O)$ by $Ext(\mathfrak{I}_5)$ and $New(\mathfrak{I}_5 \xrightarrow{b} \mathfrak{I}_9)$. This updates translate into $\mathbf{Bhv}(\wp(\Sigma))$ as the new transition $(\mathcal{C}_4,\mathfrak{I}_5) \xrightarrow{b} (\mathcal{C}_8,\mathfrak{I}_9)$, with the creation of $(\mathcal{C}_8,\mathfrak{I}_9)$. No inconsistent transition is detected. Instead, unlike the previous steps, at this point the pruning of $\mathbf{Bhv}(\wp(\Sigma))$ applies. We have $\mathbb{B}^* = \mathbb{B} - \mathbb{B}' = \{(\mathcal{C}_2,\mathfrak{I}_5),(\mathcal{C}_7,\mathfrak{I}_5),(\mathcal{C}_5,\mathfrak{I}_5)\}$. Since no transition exits either of these three states and $\mathfrak{I}_5$ is not final in $Isp(O)$, these states are removed from $\mathbf{Bhv}(\wp(\Sigma))$, along with entering transitions. This provokes the removal of state $(\mathcal{C}_1,\mathfrak{I}_1)$ and its entering transition, too.
4. The last layer of $Psp(O))$ is generated, involving the final state $\mathcal{P}_{11}$ and three entering transitions. These causes the extension of $Isp(O)$ by $Ext(\mathfrak{I}_9)$ and $New(\mathfrak{I}_9 \xrightarrow{b} \mathfrak{I}_{11})$. However, the latter is inconsistent in $\mathbf{Bhv}(\wp(\Sigma))$, therefore it is removed from $Isp(O)$.

Compared with Fig. 4, the number of states and transitions in both $Psp(O)$ and $Isp(O)$ is reduced in Fig. 7, as expected. Of course, the advantage in using the lazy approach depends on the extent of the set resulting from the difference between the language of $Isp(O)$ and the language of $Bsp(\Sigma,\sigma_0)$, in other words, on the number of spurious traces in $\|O\|$: the larger the set of spurious traces, the better the performances of *LDE* compared with the busy approach.

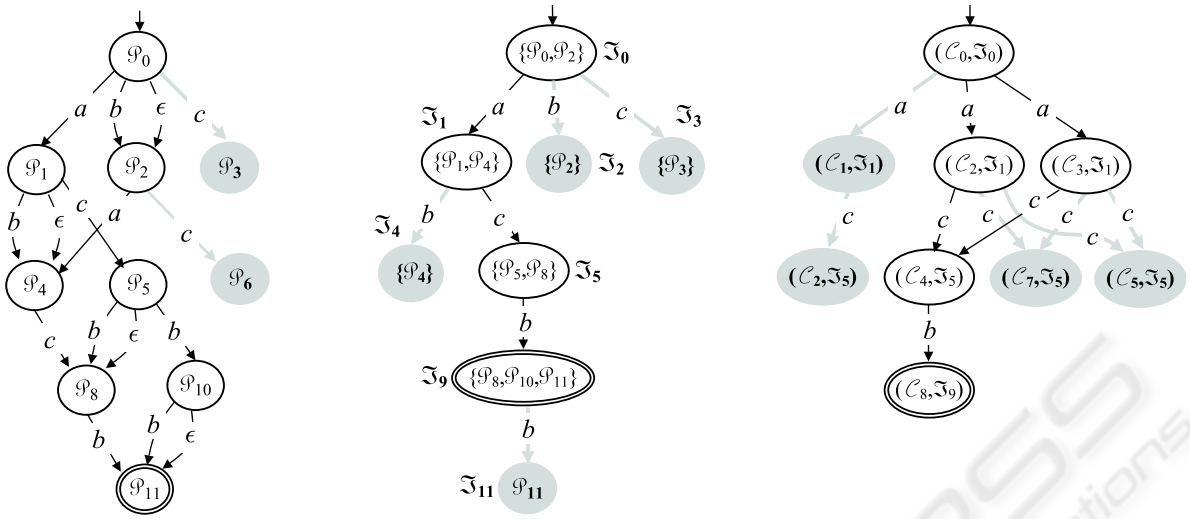

Figure 7: Lazy generation of $Psp(O)$ (left), $Isp(O)$ (center), and $\mathbf{Bhv}(\wp(\Sigma))$ (right).

5 EXPERIMENTAL RESULTS

The implementation of a prototype software system, coded in the Haskell programming language (Thompson, 1999), that embodies the lazy diagnosis method dealt with in this paper, was performed, as well as the implementation of the (busy) diagnosis method previously proposed for a-posteriori diagnosis (Lamperti and Zanella, 2003), the Diagnostic Engine (*DE*). As explained in Section 3, *DE* involves no circular pruning. Rather, it processes in one step the whole observation, in order to obtain the prefix space. Then, it invokes the Subset Construction algorithm (Hopcroft et al., 2006) to determinize the whole prefix space into the (whole) index space. Next, it performs a reconstruction of the behavior driven by the index space and, finally, it decorates the behavior in order to draw candidate diagnoses. Also the implemented *LDE*, after having reconstructed the condensed behavior corresponding to the whole observation, decorates it and draws candidate diagnoses.

In order to compare the performances of *LDE* and *DE*, hundreds of experiments were run based on observations with different sizes and different overlays between their extensions and the language of the behavior space. Such experiments have confirmed that the savings in memory allocation brought by *LDE*, as far as the prefix space and the index space are concerned, increase with the size of the observation and, given the same observation, decrease with the growing of the extent of the overlay. Interestingly, the execution time of *LDE* was shorter than that of *DE* in all experiments, with a saving in time having the same trend as the saving in space.

Table 1: *LDE* vs. *DE*

Nodes	Space		Time	
	DE	*LDE*	*DE*	*LDE*
1	7	7	0.06	0.04
2	22	19	0.06	0.04
3	64	19	0.08	0.06
4	178	115	0.08	0.10
5	438	225	0.18	0.16
6	900	410	0.50	0.46
7	2286	931	4.70	1.30
8	5258	1976	39.84	3.94
9	10738	4093	293.60	10.90
10	20597	8476	1758.28	52.26

Shown in Table 1 are the size of the memory allocation and the execution time of the two methods, corresponding to the number of nodes in the involved observations. Memory allocation (space) is the maximum value of the sum of nodes and arcs of both the prefix space and the index space, while the execution time is the CPU time in seconds. The table refers to 10 experiments led on a five-component seven-link active system with totally disconnected observations including from 1 to 10 nodes, where the observation including $n+1$ nodes is obtained by adding a new node to the observation including n nodes ($n \in [0..9]$).

6 CONCLUSIONS

This paper presents a lazy approach to diagnosis of active systems with uncertain observations. It primarily aims to reduce the size of the memory space needed by the indexing automata for observation handling. The lazy generation of the index space has been

obtained by adapting an algorithm for incremental determinization (Lamperti et al., 2008), by specializing it to cope with acyclic automata. Experimental evidence supports the theoretical claim of space reduction, and shows also a saving in computation time.

Pruning an uncertain observation based on an incremental history reconstruction can be generalized to domains other than active systems. The task of transforming an uncertain observation graph into an automaton is not faced in (Grastien et al., 2005), where it is assumed that the uncertain observation is represented by an automaton from the very beginning. However, such an automaton could be pruned based on a lazy technique similar to ours.

Other approaches in the literature have tried to improve efficiency, above all for carrying out the monitoring task, for which real time constraints hold, including (Qiu and Kumar, 2006), whose computational effort is linear. However, such an approach considers certain observations only and, at the moment, we cannot devise how much the degrees as well as the kinds of uncertainties that may affect an observation can change such figures.

REFERENCES

Baroni, P., Lamperti, G., Pogliano, P., and Zanella, M. (1998). Diagnosis of active systems. In *Thirteenth European Conference on Artificial Intelligence – ECAI'98*, pages 274–278, Brighton, UK.

Baroni, P., Lamperti, G., Pogliano, P., and Zanella, M. (1999). Diagnosis of large active systems. *Artificial Intelligence*, 110(1):135–183.

Grastien, A., Cordier, M., and Largouët, C. (2005). Incremental diagnosis of discrete-event systems. In *Sixteenth International Workshop on Principles of Diagnosis – DX'05*, pages 119–124, Monterey, CA.

Hopcroft, J., Motwani, R., and Ullman, J. (2006). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA, third edition.

Lamperti, G. and Zanella, M. (2002). Diagnosis of discrete-event systems from uncertain temporal observations. *Artificial Intelligence*, 137(1–2):91–163.

Lamperti, G. and Zanella, M. (2003). *Diagnosis of Active Systems – Principles and Techniques*, volume 741 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publisher, Dordrecht, NL.

Lamperti, G. and Zanella, M. (2004). A bridged diagnostic method for the monitoring of polymorphic discrete-event systems. *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, 34(5):2222–2244.

Lamperti, G. and Zanella, M. (2006). Flexible diagnosis of discrete-event systems by similarity-based reasoning techniques. *Artificial Intelligence*, 170(3):232–297.

Lamperti, G. and Zanella, M. (2008). Observation-subsumption checking in similarity-based diagnosis of discrete-event systems. In *Eighteenth European Conference on Artificial Intelligence – ECAI'2008*, pages 204–208, Patras, G.

Lamperti, G., Zanella, M., Chiodi, G., and Chiodi, L. (2008). Incremental determinization of finite automata in model-based diagnosis of active systems. In Lovrek, I., Howlett, R., and Jain, L., editors, *Knowledge-Based Intelligent Information and Engineering Systems*, volume 5177 of *LNAI*, pages 362–374. Springer.

Pencolé, Y. and Cordier, M. (2005). A formal framework for the decentralized diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence*, 164:121–170.

Qiu, W. and Kumar, R. (2006). Decentralized failure diagnosis of discrete event systems. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, 36(2):384–395.

Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., and Teneketzis, D. (1995). Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 40(9):1555–1575.

Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., and Teneketzis, D. (1996). Failure diagnosis using discrete-event models. *IEEE Transactions on Control Systems Technology*, 4(2):105–124.

Thompson, S. (1999). *Haskell – The Craft of Functional Programming*. Addison-Wesley, Harlow, UK.