Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 192 (2021) 407–416

Procedia Computer Science

www.elsevier.com/locate/procedia

25th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES 2021)

Fixing Nondeterminism in Large Discrete-Event Knowledge

Michele Dusi, Gianfranco Lamperti*

Department of Information Engineering, University of Brescia, 25123 Brescia, Italy

Abstract

Discrete-event knowledge (DEK) consists in a variety of automaton-based data structures which are generated by knowledge-based and engineering systems, including intelligent diagnosis systems. Model-based diagnosis methods for discrete-event systems (DESs) are typically supported by large automata that allow for the efficient generation of the candidate diagnoses when the DES is being operated. However, the generation of the DEK may involve a determinization step, where a nondeterministic finite automaton (NFA) is transformed into an equivalent deterministic finite automaton (DFA) by means of the classical Subset Construction algorithm. When this determinization is performed online and the NFA is large, the diagnosis engine may slow down to such an extent that the entire diagnosis task is jeopardized. This is why a novel determinization algorithm is proposed, called Quick Subset Construction, which, instead of generating from scratch the equivalent DFA, removes the nondeterminism by operating directly on the NFA. This way, the larger the NFA and the smaller the portion of nondeterminism involved, the faster Quick Subset Construction will be over Subset Construction. Massive experimentation on large automata has confirmed this result empirically.

Keywords: knowledge representation; discrete-event knowledge; model-based reasoning; finite automata; discrete-event systems; nondeterminism; determinization; uncertainty; subset construction; large-scale systems; conservative algorithms

1. Introduction

Knowledge-based and engineering systems exploit some sort of knowledge that is either provided by experts or is generated by automated techniques, like in model-based diagnosis [36, 19]. In artificial intelligence, model-based diagnosis was first proposed having in mind static systems, like combinational circuits [22, 23, 18, 15]. For diagnosing a time-varying system [39], a discrete-event system (DES) model can be adopted [11], this being either a Petri net [20, 10, 4, 12] or a net of communicating finite automata, one automaton for each component [3, 13, 33, 16, 21, 24, 17]. Although an automaton relevant to a DES component can represent just its nominal behavior [34], usually the communicating automaton also describes the faulty behavior of the component, as proposed in the seminal work by

* Gianfranco Lamperti. Tel.: +39-030-371-5491 ; fax: +39-030-380-014.
E-mail address: gianfranco.lamperti@unibs.it

Sampath et al. [37, 38]. Diagnosis a DES becomes a sort of abductive reasoning [32], based on the reconstruction of the behavior that entails a temporal observation generated by the DES being operated. To speed up the online diagnosis engine, knowledge compilation can be performed offline for generating a (temporal) diagnoser of the DES [38, 5, 9, 8], which allows for online diagnosis in linear time. Such a data structure is a sort of discrete-event knowledge (DEK) that is typically represented as a large (possibly huge) deterministic finite automaton (DFA). To make the compiled DEK deterministic, a determinization step may be required, where a nondeterministic finite automaton (NFA) is transformed into an equivalent DFA. For real DESs, the compilation of DEK may be prohibitive even if performed offline, as the number of the states grows exponentially with the number of the DES components. This is why incremental techniques have recently been proposed, where the DEK expands over time by exercising the diagnosis engine, thereby avoiding total knowledge compilation [6, 7]. The problem of fixing nondeterminism in DEK becomes even more critical when determinization is required online [28, 30], as the diagnosis engine may slow down to such an extent that the entire diagnosis task is jeopardized. The classical algorithm for determinization of finite automata is Subset Construction [35], which adopts a *functional* approach: given an NFA, it constructs from scratch an equivalent DFA. Since DEK may be very large, functional determinization may be less than optimal, especially so when the nondeterminism is confined to a tiny fraction of the NFA, in other words, when the DFA retains a large portion of the NFA where the transition function is deterministic already. For this reason, a novel determinization algorithm is presented in this paper, called Quick Subset Construction, which adopts an *inward* (rather than functional) approach. Instead of generating an equivalent DFA from scratch, as functional determinization does, inward determinization fixes the nondeterminism by operating directly on the NFA through specific repair actions, thereby leaving the rest of the NFA untouched. Consequently, the larger the NFA and the smaller the number of repair actions involved, the better the performance of inward determinization over functional determinization. Quick Subset Construction was not invented overnight: it is the ultimate result of a research conducted over several years, aimed at finding alternative algorithms for the determinization of the so-called *mutating automata* [26, 2, 27, 31, 25, 14] involved in model-based diagnosis of active systems [29]. Still, despite its origins, Quick Subset Construction is a general-purpose algorithm which can be regarded as a viable alternative to Subset Construction when inward determinization is expected to perform significantly better than classical functional determinization.

2. Finite Automata and Functional Determinization

A finite automaton (FA) is a computational model that can describe a wide range of systems. An FA includes a finite set of states, a set of actual states (ideally, just one), and a set of transitions that establish what the new set of actual states is when an input is applied. An FA can be exploited for recognizing a *regular language*. A string in the language is recognized when an accepting (final) state, is reached. An FA can be either *deterministic* (DFA) or *nondeterministic* (NFA). Formally, a DFA $\mathcal{D}$ is a 5-tuple, $\mathcal{D} = (\Sigma, D, T_{\mathrm{d}}, d_0, F_{\mathrm{d}})$, where Σ is the alphabet (a finite set of symbols), D is the set of states, T_{d} is the transition function, $T_{\mathrm{d}} : D_\Sigma \mapsto D$, where $D_\Sigma \subseteq D \times \Sigma$, d_0 is the initial state, and $F_{\mathrm{d}} \subseteq D$ is the set of final states. Determinism is based on that T_{d} maps a state-symbol pair into a *single* state. Likewise, an NFA $\mathcal{N}$ is a 5-tuple, $\mathcal{N} = (\Sigma, N, T_{\mathrm{n}}, n_0, F_{\mathrm{n}})$, where the fields have the same meaning as they have in the DFA, with the exception that the transition function is nondeterministic, $T_{\mathrm{n}} : N^{\varepsilon}_{\Sigma} \mapsto 2^N$, where $N^{\varepsilon}_{\Sigma} \subseteq N \times (\Sigma \cup \{\varepsilon\})$, with ε being the *empty* symbol, $\varepsilon \notin \Sigma$. Roughly, the regular language recognized by an FA is the set of strings generated by the cascade application of the transition function from the initial state to a final state, where the occurrences of ε (if an NFA) are immaterial. Two FAs are *equivalent* when they recognize the same regular language. A transition mapping a pair (s, ℓ) is said to be *marked* with ℓ, and is called an *ℓ-transition*. Let $\mathbb{N}$ be a set of states in an NFA $\mathcal{N}$ and let ℓ be a symbol in Σ. The *ε-closure* of $\mathbb{N}$ in $\mathcal{N}$ is the union of $\mathbb{N}$ and the set of states that are reached in $\mathcal{N}$ by a sequence of transitions exiting a state in $\mathbb{N}$, where all such transitions are marked by ε. The *ℓ-mapping* of $\mathbb{N}$ is the set ε-closure of the set of states reached by the ℓ-transitions exiting the states in $\mathbb{N}$. For each NFA there exists (at least) an equivalent DFA. One particular such DFA is generated by the Subset Construction algorithm, which, for this reason, is said to be SC-*equivalent* to the NFA. Determinization is notoriously convenient because processing a DFA is generally more efficient than processing an NFA. For instance, in lexical analysis, the recognition of a string based on a DFA is performed without backtracking [1].

The pseudocode of Subset Construction is listed in Algorithm 1 (lines 1–13), where $\mathcal{N}$ is the input NFA and $\mathcal{D}$ is the output (*SC*-equivalent) DFA. By construction, each state in $\mathcal{D}$ is identified by a subset of the states of $\mathcal{N}$. The

Algorithm 1: SUBSET CONSTRUCTION

```
input  : $\mathcal{N} = (\Sigma, N, T_n, n_0, F_n)$: an NFA
output: $\mathcal{D} = (\Sigma, D, T_d, d_0, F_d)$: the DFA that is $SC$-equivalent to $\mathcal{N}$
1  Generate the initial state $d_0$ of $\mathcal{D}$, where $\|d_0\| = \varepsilon$-closure$(\{n_0\}, \mathcal{N})$
2  Push $d_0$ onto the (empty) stack
3  repeat
4      Pop a state $d$ from the stack
5      foreach $\ell \in \Sigma$ such that $\langle n, \ell, n' \rangle \in T_n, n \in \|d\|$ do
6          $\mathbb{N} \leftarrow \ell$-mapping$(\|d\|, \mathcal{N})$
7          if there is no state $d'$ in $D$ such that $\|d'\| = \mathbb{N}$ then
8              Insert a new state $d'$ into $D$, where $\|d'\| = \mathbb{N}$, as well as into $F_d$ provided that $\|d'\| \cap F_n \neq \emptyset$
9              Push $d'$ onto the stack
10         else
11             Let $d'$ be the state in $D$ where $\|d'\| = \mathbb{N}$
12         Insert a new transition $\langle d, \ell, d' \rangle$ into $T_d$
13 until the stack is empty
```

subset of the states of $\mathcal{N}$ identifying a state d of $\mathcal{D}$ is denoted $\|d\|$, called the *extension* of d. The initial state d_0 of $\mathcal{D}$ is the ε-closure of the singleton $\{n_0\}$, where n_0 is the initial state of $\mathcal{N}$. The generation of $\mathcal{D}$ is supported by a stack. Each element in the stack is a state of $\mathcal{D}$ to be processed. Processing a state d means generating the transition function of d, that is, the set of transitions exiting d and entering another state d', where each transition is denoted $\langle d, \ell, d' \rangle$, with ℓ being the symbol marking the transition. Initially, the stack contains the initial state d_0 only (line 2). Then, SUBSET CONSTRUCTION pops and processes one state d from the stack at a time, by generating the transitions exiting d, until the stack becomes empty (lines 3–13). Each transition is generated by considering each symbol $\ell \in \Sigma$ marking a transition exiting a state $n \in \|d\|$ (line 5). For each symbol ℓ, a transition $\langle d, \ell, d' \rangle$ is created (line 12), where $\|d'\|$ is the ℓ-mapping of $\|d\|$ in $\mathcal{N}$. Also, if d' does not exist, then it is created and possibly qualified as final if it contains at least one state that is final in $\mathcal{N}$, and pushed onto the stack (lines 8 and 9). At the termination of the algorithm, $\mathcal{D}$ is the DFA SC-equivalent to $\mathcal{N}$.

Example 1. Shown on the left of Figure 1 is the diagrammatic representation of an NFA $\mathcal{N}$, where states and transitions are denoted by circles and arcs, respectively, with 0 being the initial state, whereas the (only) final state is 5. Nondeterminism of $\mathcal{N}$ is localized in the transition function mapping $(1, b)$, and in the ε-transition exiting the state 3. We have b-mapping$(\{1\}, \mathcal{N}) = \{3, 4\}$. Depicted next to $\mathcal{N}$ are the intermediate configurations of $\mathcal{D}$ generated by SUBSET CONSTRUCTION, where the boxes in gray represent the content of the stack in each configuration. The six states in $\mathcal{D}$ are named $d_0, d_1, d_2, d_{34}, d_4$, and d_5, where $\|d_{34}\| = \{3, 4\}$.

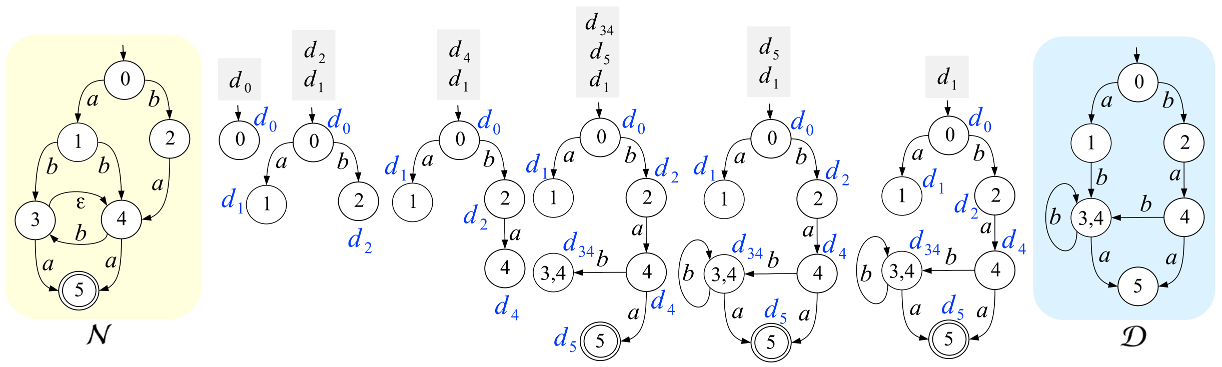

Fig. 1. An NFA $\mathcal{N}$ (left) and the intermediate configurations of the SC-equivalent DFA $\mathcal{D}$ generated by SUBSET CONSTRUCTION (right).

3. Inward Determinization of Finite Automata

Removing nondeterminism from an NFA by means of Subset Construction requires the generation of a DFA that is equivalent to the NFA. This means that the NFA is not fixed directly by specific repair actions, but indirectly, by generating a completely new DFA. To use a metaphor, it is like fixing the problem of a car by constructing an equivalent new car where the problem has been removed. If the DFA differs from the NFA considerably, this functional approach is reasonable, as there is no point in repairing an NFA by means of a (possibly costly) list of actions when the complete generation of the equivalent DFA turns out to be less expensive. With reference to our metaphor, if the cost for repairing a car is larger than the cost of an equivalent new car, there is no point in repairing the car. However, if the NFA is large and nondeterminism affects a tiny fraction of the transition function, chances are that the equivalent DFA can be obtained by a (relatively short) sequence of repair actions (metaphorically, it is worth fixing the car rather than buying a new one). Admittedly, automata are not cars and generating a DFA involves computational resources and software technology rather than physical materials and mechanical technology. Still, in specific circumstances, the construct/repair dilemma may be equally applicable to the problem of NFA determinization.

The idea is to design a non-functional algorithm that solves the problem of NFA determinization by intervening directly on the NFA rather than generating an equivalent DFA from scratch. This technique is called *inward determinization*. To this end, each point of nondeterminism in the NFA is considered and a specific repair action is applied in order to remove such nondeterminism. Nondeterminism occurs when a state is exited either by an ε-transition or by several transitions marked with the same label. In order to capture all the nondeterminism points, relevant states of the NFA are marked with a set of labels in $\Sigma \cup \{\varepsilon\}$, where Σ is the alphabet of the NFA. Specifically, the initial marking of the NFA is based on the following three rules:

1. If an ε-transition exits the initial state s_0 of the NFA, then s_0 is marked with ε.
2. If there is a transition $\langle s, \ell, s' \rangle$, where $\ell \neq \varepsilon$, and s' is exited by an ε-transition, then s is marked with ℓ.
3. If a state s is exited by two or more ℓ-transitions, where $\ell \neq \varepsilon$, then s is marked with ℓ.

If a label ℓ marks a state s, then (s, ℓ) is called a *singularity*.[1] A singularity marking the initial NFA indicates that the transition function mapping (s, ℓ) results in several states, in other words, nondeterminism arises. Hence, a repair action is required in order to fix this specific singularity. Fixing a singularity may cause the creation of new singularities, not included in the initial set (defined by the three rules above). Generally speaking, a newly created singularity (s, ℓ) indicates that the transition function mapping (s, ℓ) needs to be either generated or fixed.

An important issue in fixing an NFA by means of repair actions is concerned with the order in which singularities are considered. In fact, choosing the singularities randomly is bound to run into two major problems: the disconnection of the resulting DFA and/or the nontermination of the algorithm. In order to avoid both disconnection and nontermination, singularities are sorted based on the *distance* of the relevant states. The distance of a state s in an FA with initial state s_0 is the minimum number of transitions that connect s_0 with s. The distance of s is denoted $\delta(s)$; in particular, $\delta(s_0) = 0$. Since the distance only imposes a partial order on states, as several states may share same distance, singularities (s, ℓ) are totally ordered based not only on $\delta(s)$ but also on the natural (ascending) order of the identifiers s and ℓ (with the label ε being in first position). In other words, singularities (s, ℓ) are sorted based on three keys: $\delta(s)$, the identifier of s, and ℓ. The resulting (ordered) list of singularities is called the *singularity list*. Processing a singularity may cause the insertion/removal of transitions, with possible change in the distance of some states. Consequently, during processing, singularities may change their position within the singularity list.

Since each state d in the *SC*-equivalent DFA is marked with a nonempty subset of the states of the NFA, a distinction is in order between the identifier d and the subset of states marking d, denoted $\|d\|$. During processing, the current extension of a state d can change (before the termination of the algorithm), whereas the identifier d cannot. For formal reasons, we extend the notion of extension to an NFA state s, namely $\|s\| = \{s\}$. Also, given a set $\mathbb{S}$ of states in the automaton being processed, $\|\mathbb{S}\|$ denotes the union of the extensions of the states in $\mathbb{S}$, in other words, a subset of the states of the initial NFA.

[1] Since the collection of labels marking a state forms a set, no duplication of the same singularity may exist.

In order to avoid the disconnection of the automaton, the notion of *safety* of a state needs to be introduced. The problem arises when a transition $\langle d, \ell, d' \rangle$ might be removed. If so, it is of utmost importance to know whether d' will still be connected with the initial state or not.

Definition 1 (Safety)**.** *Let* (d, ℓ) *be a singularity in an FA* $\mathcal{D}$ *having initial state* d_0*, and let* $\mathbb{D} = \ell$*-mapping*$(d, \mathcal{D})$*. A state* $\bar{d} \in \mathbb{D}$ *is* safe *iff either* $\bar{d} = d_0$ *or there is a transition* $\langle d', \ell', \bar{d} \rangle$ *in* $\mathcal{D}$ *where* $(d', \ell') \neq (d, \ell)$ *and* $\delta(d') \leq \delta(d)$*; otherwise,* $\bar{d}$ *is* unsafe.

In the next section, an algorithm called Quick Subset Construction is presented, which progressively transforms an NFA into the SC-equivalent DFA by means of inward determinization.

4. Quick Subset Construction

Quick Subset Construction takes as input an NFA $\mathcal{N}$ and a copy $\mathcal{D}$ of $\mathcal{N}$ that is marked with the singularities. When the algorithm terminates, $\mathcal{D}$ turns out to be the DFA SC-equivalent to $\mathcal{N}$. The pseudocode of Quick Subset Construction is listed in Algorithm 2 (lines 1–30). Singularities are considered one by one in their order in a *singularity list*, and repair actions are applied depending on the actual *scenario*. Three scenarios exist: $\mathcal{S}_0$, $\mathcal{S}_1$, and $\mathcal{S}_2$.

Scenario $\mathcal{S}_0$ (lines 1–10) occurs for the singularity (ε, d_0), called *ε-singularity*, where d_0 is the initial state of $\mathcal{D}$. This means that d_0 is exited by one or more ε-transitions. If present, the ε-singularity is in first position within the singularity list; hence, it is the first singularity that is processed. Besides, since no singularity (d, ε) can be generated subsequently (neither for the initial state nor for any other state), the processing of the ε-singularity is placed upfront in the algorithm, before the main loop (lines 11–30), in which the other two scenarios are considered. The repair action for (d_0, ε) removes the ε-transitions exiting d_0 while extending the extension of d_0. First the sets $\mathbb{D}$, $\mathbb{N}$, and $\mathbb{U}$ are determined (line 2), where $\mathbb{D}$ is the ε-closure of d_0, $\mathbb{N}$ is the set of NFA states involved in $\mathbb{D}$, and $\mathbb{U}$ is the set of unsafe states in $\mathbb{D}$ (cf. Definition 1). Then, the extension of d_0 is extended to $\mathbb{N}$ and new singularities are created for d_0 (line 3). Specifically, for each transition $\langle s, \ell, s' \rangle$ in $\mathcal{N}$ such that $s \in \mathbb{N}$ and $\ell \neq \varepsilon$, a singularity (d_0, ℓ) is created, which indicates that the transition function mapping (d_0, ℓ) possibly needs to be adjusted. Next, the ε-transitions exiting d_0 are removed (line 4). Then, for each transition $\langle u, \ell, d \rangle$ in $\mathcal{D}$ where u is unsafe, $\ell \neq \varepsilon$, and d is not in the set of unsafe states (possibly outside the set $\mathbb{D}$), a transition $\langle d_0, \ell, d \rangle$ is created (lines 5–6). These new transitions are meant to prevent $\mathcal{D}$ from becoming disconnected owing to the subsequent removal of transitions. Then, every transition $\langle d, \ell, u \rangle$ exiting a state that is not unsafe (possibly outside the set $\mathbb{D}$) and entering an unsafe state is removed and, if $\ell \neq \varepsilon$, a singularity (d, ℓ) is created (lines 7–8). This is necessary to avoid dangling transitions resulting from the subsequent removal of the unsafe states. Eventually, the unsafe states and relevant transitions are removed from $\mathcal{D}$ (line 9), as well as the ε-singularity from the singularity list (line 10).

After the (eventual) processing of the ε-singularity, a loop is entered (lines 11–30) where, at each iteration, the next singularity (d, ℓ) is considered. The processing of the singularity may cause the creation of new singularities and the loop terminates when all singularities have been processed (the singularity list is empty). First, the set $\mathbb{N}$ is determined (line 12) as the ℓ-mapping of the extension of d in $\mathcal{N}$. Then, two scenarios are possible, namely either $\mathcal{S}_1$ or $\mathcal{S}_2$.

Scenario $\mathcal{S}_1$ (lines 13–18) occurs when no transition marked with ℓ exits d. Hence, the transition mapping (d, ℓ) needs to be created towards a state d' such that $\|d'\| = \mathbb{N}$. If d' already exists, then a transition $\langle d, \ell, d' \rangle$ is inserted into $\mathcal{D}$ (line 15); otherwise, d' is created, with $\|d'\| = \mathbb{N}$, as well as a transition $\langle d, \ell, d' \rangle$ (line 17). Since d' is a newly created state, relevant singularities need to be generated (line 18), so that the transition function mapping d' and relevant labels can be subsequently built.

Scenario $\mathcal{S}_2$ (lines 19–29) occurs when a transition existing d and marked with ℓ already exists.[2] The condition in line 19 serves two purposes: on the one hand, it avoids processing a singularity which does not change $\mathcal{D}$, on the other (and most importantly), in case of auto-transitions exiting d, it avoids the nontermination of the algorithm. In the processing, first the sets $\mathbb{D}$ and $\mathbb{U}$ are computed (line 20). Then a state d' is created (line 21), where $\|d'\| = \mathbb{N}$ (with $\mathbb{N}$ being the ℓ-mapping of $\|d\|$ generated in line 12). The creation of d' requires the generation of the relevant singularities also. Then, the transitions exiting d and marked with ℓ are removed in line 22 (they will be replaced by

[2] To some extent, scenario S_2 is reminiscent of scenario $\mathcal{S}_0$; consequently, the repair actions are somewhat similar.

Algorithm 2: Quick Subset Construction

```
input  : $\mathcal{N}$: an NFA,  $\mathcal{D}$: a copy of $\mathcal{N}$ being marked with the initial singularities
output: $\mathcal{D}$ is transformed into the DFA that is $\mathcal{SC}$-equivalent to $\mathcal{N}$
1  if there is a singularity $(\varepsilon, d_0)$, where $d_0$ is the initial state of $\mathcal{D}$, then      // Scenario $\mathcal{S}_0$ (lines 1-10)
2      $\mathbb{D} \leftarrow \varepsilon$-closure$(d_0, \mathcal{D})$,  $\mathbb{N} \leftarrow \|\mathbb{D}\|$,  $\mathbb{U} \leftarrow \{ d \mid d \in \mathbb{D}, d$ is unsafe $\}$
3      Extend $\|d_0\|$ to $\mathbb{N}$ and create the new singularities for $d_0$
4      Remove the $\varepsilon$-transitions exiting $d_0$
5      foreach transition $\langle u, \ell, d \rangle$ in $\mathcal{D}$ where $u \in \mathbb{U}, \ell \neq \varepsilon, d \notin \mathbb{U}$ do
6          Insert a transition $\langle d_0, \ell, d \rangle$ into $\mathcal{D}$
7      foreach transition $t = \langle d, \ell, u \rangle$ in $\mathcal{D}$ where $d \notin \mathbb{U}, u \in \mathbb{U}$ do
8          Remove the transition $t$ from $\mathcal{D}$ and, if $\ell \neq \varepsilon$, then create the singularity $(d, \ell)$
9      Remove from $\mathcal{D}$ all the states in $\mathbb{U}$ and the relevant entering/exiting transitions
10     Remove the singularity $(d_0, \varepsilon)$
11 while there is a singularity in $\mathcal{D}$, with $(d, \ell)$ being the first singularity to be processed, do
12     $\mathbb{N} \leftarrow \ell$-mapping$(\|d\|, \mathcal{N})$
13     if no transition marked with $\ell$ exits $d$ then      // Scenario $\mathcal{S}_1$ (lines 13-18)
14         if there is a state $d'$ in $\mathcal{D}$ where $\|d'\| = \mathbb{N}$ then
15             Insert the transition $\langle d, \ell, d' \rangle$ into $\mathcal{D}$
16         else
17             Create a state $d'$, with $\|d'\| = \mathbb{N}$, and insert a transition $\langle d, \ell, d' \rangle$ into $\mathcal{D}$
18             Create the singularities for $d'$
19     else if several transitions being marked with $\ell$ exit $d$ or $\langle d, \ell, d' \rangle$ is such that $\|d'\| \neq \mathbb{N}$ or $d'$ is exited by an
        $\varepsilon$-transition then      // Scenario $\mathcal{S}_2$ (lines 19-29)
20         $\mathbb{D} \leftarrow \ell$-mapping$(d, \mathcal{D})$,  $\mathbb{U} \leftarrow \{d \mid d \in \mathbb{D}, d$ is unsafe$\}$
21         Create a state $d'$, with $\|d'\| = \mathbb{N}$, as well as the relevant singularities
22         Remove the transitions exiting $d$ and marked with $\ell$
23         foreach transition $\langle u, \ell', d'' \rangle$ in $\mathcal{D}$ where $u \in \mathbb{U}, \ell' \neq \varepsilon, d'' \notin \mathbb{U}$ do
24             Insert the transition $\langle d', \ell', d'' \rangle$ into $\mathcal{D}$
25         foreach transition $t = \langle d'', \ell', u \rangle$ in $\mathcal{D}$ where $u \in \mathbb{U}, d'' \notin \mathbb{U}$ do
26             Remove the transition $t$ from $\mathcal{D}$ and, if $\ell' \neq \varepsilon$, insert the singularity $(d'', \ell')$
27         Remove from $\mathcal{D}$ all the states in $\mathbb{U}$ and the relevant entering/exiting transitions
28         Insert a new transition $\langle d, \ell, d' \rangle$ into $\mathcal{D}$
29         Possibly merge $d'$ with a state $\bar{d}$, provided that $\|d'\| = \|\bar{d}\|$
30     Remove the singularity $(d, \ell)$
```

a new transition in line 28). Next, for each transition $\langle u, \ell', d'' \rangle$ where u is unsafe, $\ell \neq \varepsilon$, and d'' is not among the unsafe states, a transition $\langle d', \ell', d'' \rangle$ is inserted (lines 23–24). Like in scenario $\mathcal{S}_0$, these new transitions are meant to prevent $\mathcal{D}$ from becoming disconnected owing to the subsequent removal of transitions performed in lines 25–26, where for each transition $\langle d'', \ell', u \rangle$ removed, with $\ell' \neq \varepsilon$, a singularity (d'', ℓ') is created. Then, all unsafe states and relevant transitions are removed (line 27). Afterwards, in line 28, a new transition $\langle d, \ell, d' \rangle$ is inserted into $\mathcal{D}$, where d' is the new state created in line 21. Eventually, if another state $\bar{d}$ turns out to have the same extension as d', then d' and $\bar{d}$ are merged into a single state (line 29). Specifically, $\bar{d}$ is preserved and d' is removed, while the transitions entering/exiting d' are redirected to/from $\bar{d}$, respectively; moreover, the singularities relevant to d' (the state removed) are inherited by $\bar{d}$ (the state preserved), by transforming each singularity (d', ℓ') into $(\bar{d}, \ell')$; also, new singularities are possibly created for $\bar{d}$ (if nondeterminism arises). The singularity (d, ℓ) is eventually removed (line 30) before considering the next singularity (if any) in line 11.

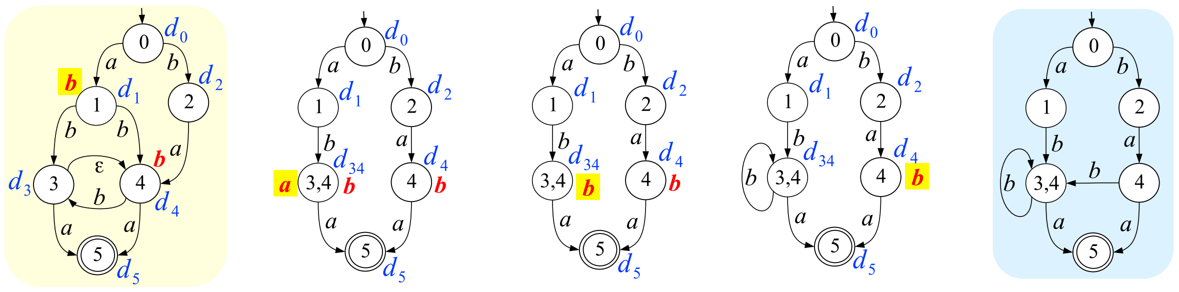

Fig. 2. Step-by-step transformation of an NFA (left) to the *SC*-equivalent DFA (right) by Quick Subset Construction.

The strong point of Quick Subset Construction over Subset Construction is the ability to confine its processing to the singularities rather than generating the entire DFA from scratch (*inward* rather than *functional* determinization). Intuitively, the convenience in using Quick Subset Construction depends on the number of singularities processed compared to the total number of transitions generated by Subset Construction to make up the equivalent DFA. This is captured by the notion of *impact*.

Definition 2 (Impact)**.** *Let $\mathcal{N}$ be an NFA and $\mathcal{D}$ the corresponding* SC*-equivalent DFA. The* impact $\Im$ *of the inward determinization of $\mathcal{N}$ is the ratio between the number σ of singularities processed by* Quick Subset Construction*, for transforming $\mathcal{N}$ into $\mathcal{D}$, and the number τ of transitions in $\mathcal{D}$, namely $\Im = \sigma/\tau$.*

Note how the impact is only a theoretical notion, as the processing of a singularity is in general more complex than the creation of a transition. For instance, $\Im = 0.5$ does not necessarily mean that the processing time of Quick Subset Construction will be half the time of Subset Construction. Still, the experimental results indicate a clear (practically linear) correlation between the impact and the processing time of Quick Subset Construction (cf. Section 5).

Example 2. Consider the NFA $\mathcal{N}$ displayed on the left of Figure 1, whose determinization was performed by Subset Construction in Example 1. We now apply Quick Subset Construction to $\mathcal{N}$, thereby obtaining the same DFA $\mathcal{D}$, displayed on the right of Figure 1, by means of inward (rather than functional) determinization. Outlined on the left of Figure 2 is a copy of $\mathcal{N}$ being marked with the initial singularities, namely (d_1, b) and (d_4, b). In fact, according to the classification defined in Section 3, the second rule applies, since the state d_1 is exited by two transitions marked by b. Moreover, the third rule applies also, as there are two transitions, namely $\langle d_1, b, d_3\rangle$ and $\langle d_4, b, d_3\rangle$, where d_3 is exited by an ε-transition. According to Algorithm 2, since the ε-singularity is missing, the processing starts at line 12, where $\mathbb{N} = \ell\text{-mapping}(\{1\}, \mathcal{N}) = \{3, 4\}$. Then, the scenario $\mathcal{S}_2$ comes into play, where $\mathbb{D} = \{d_3, d_4\}$ and $\mathbb{U} = \{d_3\}$, with the state d_4 being safe owing to the entering transition $\langle d_2, a, d_4\rangle$, where $\delta(d_2) = \delta(d_1) = 1$ (cf. Definition 1). Then, a state d_{34} is created, where $\|d_{34}\| = \{3, 4\}$, along with the singularities involving the symbols a and b (line 21). After removing the transitions exiting the state d_1 and marked by b (line 22), a transition $\langle d_{34}, a, d_5\rangle$ is inserted (line 24). The subsequent processing in line 26 removes the transition $\langle d_4, b, d_3\rangle$, while creating the singularity (d_4, b), which already exists. Eventually, after the insertion of the transition $\langle d_1, b, d_{34}\rangle$ (line 28), since no merging is applicable, the resulting automaton is outlined in second position in Figure 2. At the second iteration, the singularity to be processed is (d_{34}, a). However, since the condition in line 19 is not fulfilled, no processing is performed, thereby leading to the automaton in third position, the same as the previous one, without the singularity (d_{34}, a). At the third iteration, the relevant singularity is (d_{34}, b), which corresponds to the scenario $\mathcal{S}_1$, which leads to the insertion of the auto-transition $\langle d_{34}, b, d_{34}\rangle$ (automaton in fourth position). Eventually, the singularity (d_4, b) corresponds to the scenario $\mathcal{S}_1$, leading to the insertion of the transition $\langle d_4, b, d_{34}\rangle$. At this point, since no other singularity is created, the resulting automaton (shaded) on the right of Figure 2 is in fact the DFA *SC*-equivalent to $\mathcal{N}$, exactly the same DFA generated by Subset Construction in Example 1 (cf. Figure 1). According to Definition 2, the impact of the inward determinization of $\mathcal{N}$ is $\Im = \sigma/\tau = 4/8 = 0.5$. In other words, the number of singularities processed by Quick Subset Construction is half the number of transitions generated by Subset Construction.

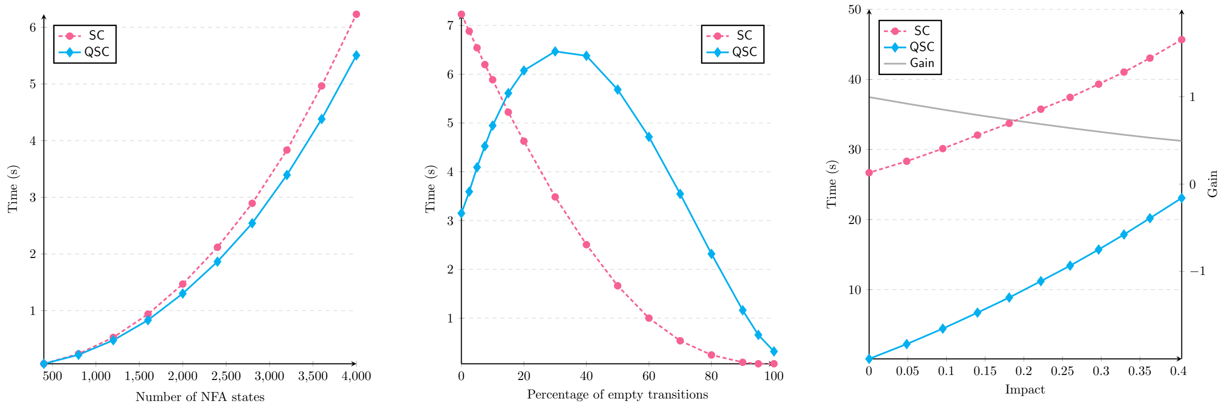

Fig. 3. Empirical results from the experiments: E_1 (left), E_2 (center), and E_3 (right).

5. Experimental Results

In order to evaluate functional versus inward determinization, both Subset Construction and Quick Subset Construction have been implemented in the C++17 programming language, under the GNU/Linux 5.4.0-42-generic x86_64 (Ubuntu 18.04.5 LTS) operating system, on a machine with Intel(R) Xeon(R) Gold 6140M CPU (2.30GHz) and 128 GB of working memory. To compare the two algorithms, a procedure for automatic generation of varying FAs has been developed. To make the comparison consistent, the two implementations share the same data structures for representing states, transitions, and automata. A large number of problem instances generated randomly have confirmed the correctness of Quick Subset Construction empirically, namely that the DFA resulting from inward determinization is identical to the DFA generated from scratch by functional determinization. Displayed in Figure 3 are the results of three specific experiments, named (from left to right) E_1, E_2, and E_3. Each experiment is composed of several configurations (one configuration for each point in the curves), where each configuration is composed of 100 test cases (thereby resulting in at least one thousand test cases for each experiment). Specifically, each point in the curve represents the average of the results of 100 test cases. In the experiment E_1, the NFAs were generated randomly with increasing size (from 400 to 4000 states). However, the NFAs in E_1 do not contain ε-transitions. The two curves in the plot indicate the (average) processing (CPU) time (in seconds) for the two algorithms. Note how Quick Subset Construction always outperforms Subset Construction, albeit not overwhelmingly so. However, when ε-transitions are inserted into the NFA, results are different. In the experiment E_2 (center of Figure 3), the size of the NFAs is kept constant (10000 states), while varying the percentage of ε-transitions, form 0 to 100. The curves indicate that Quick Subset Construction outperforms Subset Construction only up to a threshold (around 18% of ε-transitions), after which the time of Quick Subset Construction continues growing up to a point (about 30% of ε-transitions), thereby descending afterwards. This experiment clearly shows that inward determinization is convenient only within a limited range of ε-transitions in the NFA. The circumstances in which inward determinization actually outperforms functional determinization are connected with the notion of impact (cf. Definition 2), as shown in the experiment E_3 (right of Figure 3). The two algorithms are compared based on the impact, which in this experiment ranges from almost zero to about 0.4, where the size of the NFA is invariant (10000 states). Unsurprisingly, the processing time of Quick Subset Construction grows linearly with the impact (from 0.12 seconds to 23 seconds). Interestingly, the processing time of Subset Construction also is linear, but from 26 to 45 seconds. In other words, the smaller the impact, the more convenient it is to use Quick Subset Construction rather than Subset Construction, a confirmation of the idea behind inward determinization. To quantify this convenience in empirical terms, we have defined the notion of *gain*.

Definition 3 (Gain)**.** *Let t and t_q denote the processing time of* Subset Construction *and* Quick Subset Construction*, respectively, for the same NFA. The* gain $\mathcal{G}$ *is a number in the range* $[-1 \,..\, 1]$ *indicating the relative portion of processing time either saved (if positive) or wasted (if negative) by* Quick Subset Construction *over* Subset Construction*,*

$$\mathcal{G} = \frac{t - t_q}{\max(t, t_q)}. \tag{1}$$

Based on Definition 3, the results of experiment E_3 (right of Figure 3) indicate the gain also (gray curve, right axis), which is always positive, ranging from 0.995 to 0.4948. So, when the impact is very low, Quick Subset Construction allows for saving 99.5% of the processing time. Then, the gain decreases linearly with an increasing impact. With impact 0.4, the gain is still consistent, namely almost 0.5, thus showing the robustness of inward determinization.

6. Conclusion

Large DEK may require determinization in order to be exploited efficiently, which consists in removing the uncertainty caused by the nondeterminism. To this end, the classical Subset Construction algorithm can be used, which generates an equivalent DFA from scratch, namely by functional determinization. However, when the determinization needs to be performed online, like in model-based diagnosis of DESs, the processing time becomes an issue, possibly jeopardizing the working task, e.g. the diagnosis engine. This is why a technique called inward determinization has been proposed in this paper, which aims at fixing the nondeterminism directly within the NFA by means of repair actions. An algorithm based on this technique, called Quick Subset Construction, has been implemented and compared with Subset Construction. Experimental results clearly indicate that Quick Subset Construction always outperforms Subset Construction as long as ε-transitions are not involved in the NFA. Quick Subset Construction keeps being advantageous when the percentage of ε-transitions is limited to a certain range. Most of all, the results indicate that Quick Subset Construction outperforms Subset Construction in varying degree, depending on the topological distance between the NFA and the SC-equivalent DFA, which is somehow measured by the impact. In a nutshell, the smaller the impact, the more efficient Quick Subset Construction over Subset Construction.

References

[1] Aho, A., Lam, M., Sethi, R., Ullman, J., 2006. Compilers – Principles, Techniques, and Tools. second ed., Addison-Wesley, Reading, MA.

[2] Balan, S., Lamperti, G., Scandale, M., 2014. Incremental subset construction revisited, in: Neves-Silva, R., Tshirintzis, G., Uskov, V., Howlett, R., Jain, L. (Eds.), Smart Digital Futures. IOS Press, Amsterdam. volume 262 of *Frontiers in Artificial Intelligence and Applications*, pp. 25–37.

[3] Baroni, P., Lamperti, G., Pogliano, P., Zanella, M., 1999. Diagnosis of large active systems. Artificial Intelligence 110, 135–183. doi:10.1016/S0004-3702(99)00019-3.

[4] Basile, F., 2014. Overview of fault diagnosis methods based on Petri net models, in: Proceedings of the 2014 European Control Conference, ECC 2014, pp. 2636–2642. doi:10.1109/ECC.2014.6862631.

[5] Bertoglio, N., Lamperti, G., Zanella, M., 2019a. Intelligent diagnosis of discrete-event systems with preprocessing of critical scenarios, in: Czarnowski, I., Howlett, R., Jain, L. (Eds.), Intelligent Decision Technologies 2019. Springer, Singapore. volume 142 of *Smart Innovation, Systems and Technologies*, pp. 109–121. doi:10.1007/978-981-13-8311-3_10.

[6] Bertoglio, N., Lamperti, G., Zanella, M., 2019b. Temporal diagnosis of discrete-event systems with dual knowledge compilation, in: Holzinger, A., Kieseberg, P., Weippl, E., Tjoa, A.M. (Eds.), Machine Learning and Knowledge Extraction. Springer, Berlin. volume 11713 of *Lecture Notes in Computer Science*, pp. 333–352. doi:10.1007/978-3-030-29726-8_21.

[7] Bertoglio, N., Lamperti, G., Zanella, M., Zhao, X., 2019c. Twin-engined diagnosis of discrete-event systems. Engineering Reports 1, 1–20. doi:10.1002/eng2.12060.

[8] Bertoglio, N., Lamperti, G., Zanella, M., Zhao, X., 2020a. Diagnosis of temporal faults in discrete-event systems, in: Giacomo, G.D., Catala, A., Dilkina, B., Milano, M., Barro, S., Bugarín, A., Lang, J. (Eds.), 24th European Conference on Artificial Intelligence (ECAI 2020). IOS Press, Amsterdam. volume 325 of *Frontiers in Artificial Intelligence and Applications*, pp. 632–639. doi:10.3233/FAIA200148.

[9] Bertoglio, N., Lamperti, G., Zanella, M., Zhao, X., 2020b. Temporal-fault diagnosis for critical-decision making in discrete-event systems, in: Cristani, M., Toro, C., Zanni-Merk, C., Howlett, R., Jain, L. (Eds.), Knowledge-Based and Intelligent Information & Engineering Systems:

Proceedings of the 24th International Conference KES2020. Elsevier. volume 176 of *Procedia Computer Science*, pp. 521–530. doi:10.1016/j.procs.2020.08.054.
[10] Cabasino, M.P., Giua, A., Seatzu, C., 2010. Fault detection for discrete event systems using Petri nets with unobservable transitions. Automatica 46, 1531–1539.
[11] Cassandras, C., Lafortune, S., 2008. Introduction to Discrete Event Systems. second ed., Springer, New York.
[12] Cong, X., Fanti, M., Mangini, A., Li, Z., 2018. Decentralized diagnosis by Petri nets and integer linear programming. IEEE Transactions on Systems, Man, and Cybernetics: Systems 48, 1689–1700.
[13] Debouk, R., Lafortune, S., Teneketzis, D., 2000. Coordinated decentralized protocols for failure diagnosis of discrete-event systems. Journal of Discrete Event Dynamic Systems: Theory and Applications 10, 33–86.
[14] Dusi, M., Lamperti, G., 2020. Conservative determinization of translated automata by Embedded Subset Construction, in: Czarnowski, I., Howlett, R., Jain, L. (Eds.), Intelligent Decision Technologies 2020. Springer, Singapore. volume 193 of *Smart Innovation, Systems and Technologies*, pp. 49–61. doi:10.1007/978-981-15-5925-9_5.
[15] El Fattah, Y., Dechter, R., 1995. Diagnosing tree-decomposable circuits, in: Fourteenth International Joint Conference on Artificial Intelligence (IJCAI 1995), Montreal, Quebec. pp. 1742–1748.
[16] Grastien, A., Cordier, M., Largouët, C., 2005. Incremental diagnosis of discrete-event systems, in: Nineteenth International Joint Conference on Artificial Intelligence (IJCAI 2005), Edinburgh, UK. pp. 1564–1565.
[17] Grastien, A., Haslum, P., Thiébaux, S., 2012. Conflict-based diagnosis of discrete event systems: theory and practice, in: Thirteenth International Conference on Knowledge Representation and Reasoning (KR 2012), Association for the Advancement of Artificial Intelligence, Rome, Italy. pp. 489–499.
[18] Hamscher, W., 1991. Modeling digital circuits for troubleshooting. Artificial Intelligence 51, 223–271.
[19] Hamscher, W., Console, L., de Kleer, J. (Eds.), 1992. Readings in Model-Based Diagnosis. Morgan Kaufmann, San Mateo, CA.
[20] Jiroveanu, G., Boel, R., Bordbar, B., 2008. On-line monitoring of large Petri net models under partial observation. Journal of Discrete Event Dynamic Systems 18, 323–354.
[21] Kan John, P., Grastien, A., 2008. Local consistency and junction tree for diagnosis of discrete-event systems, in: Eighteenth European Conference on Artificial Intelligence (ECAI 2008), IOS Press, Amsterdam, Patras, Greece. pp. 209–213.
[22] de Kleer, J., 1984. How circuits work. Artificial Intelligence 24, 205–280.
[23] de Kleer, J., Williams, B., 1987. Diagnosing multiple faults. Artificial Intelligence 32, 97–130.
[24] Kwong, R., Yonge-Mallo, D., 2011. Fault diagnosis in discrete-event systems: incomplete models and learning. IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics 41, 118–130.
[25] Lamperti, G., 2020. Temporal determinization of mutating finite automata: Reconstructing or restructuring. Software: Practice and Experience 50, 335–367. doi:10.1002/spe.2776.
[26] Lamperti, G., Scandale, M., 2013. From diagnosis of active systems to incremental determinization of finite acyclic automata. AI Communications 26, 373–393. doi:10.3233/AIC-130574.
[27] Lamperti, G., Scandale, M., Zanella, M., 2016. Determinization and minimization of finite acyclic automata by incremental techniques. Software: Practice and Experience 46, 513–549. doi:10.1002/spe.2309.
[28] Lamperti, G., Zanella, M., Zhao, X., 2018a. Abductive diagnosis of complex active systems with compiled knowledge, in: Thielscher, M., Toni, F., Wolter, F. (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Sixteenth International Conference (KR 2018), AAAI Press, Tempe, Arizona. pp. 464–473.
[29] Lamperti, G., Zanella, M., Zhao, X., 2018b. Introduction to Diagnosis of Active Systems. Springer, Cham. doi:10.1007/978-3-319-92733-6.
[30] Lamperti, G., Zanella, M., Zhao, X., 2020. Diagnosis of deep discrete-event systems. Journal of Artificial Intelligence Research 69, 1473–1532. doi:10.1613/jair.1.12171.
[31] Lamperti, G., Zhao, X., 2018. Decremental subset construction, in: Czarnowski, I., Howlett, R., Jain, L. (Eds.), Intelligent Decision Technologies 2017. Springer International Publishing. volume 72 of *Smart Innovation, Systems and Technologies*, pp. 22–36. doi:10.1007/978-3-319-59421-7_3.
[32] McIlraith, S., 1998. Explanatory diagnosis: conjecturing actions to explain observations, in: Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR 1998), Morgan Kaufmann, S. Francisco, CA, Trento, I. pp. 167–177.
[33] Pencolé, Y., 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.
[34] Pencolé, Y., Steinbauer, G., Mühlbacher, C., Travé-Massuyès, L., 2018. Diagnosing discrete event systems using nominal models only, in: Zanella, M., Pill, I., Cimatti, A. (Eds.), 28th International Workshop on Principles of Diagnosis (DX'17), Kalpa Publications in Computing. pp. 169–183. doi:10.29007/1d2x.
[35] Rabin, M., Scott, D., 1959. Finite automata and their decision problems. IBM Journal of Research and Development 3, 114–125. doi:10.1147/rd.32.0114.
[36] Reiter, R., 1987. A theory of diagnosis from first principles. Artificial Intelligence 32, 57–95.
[37] Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., Teneketzis, D., 1995. Diagnosability of discrete-event systems. IEEE Transactions on Automatic Control 40, 1555–1575.
[38] Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., Teneketzis, D., 1996. Failure diagnosis using discrete-event models. IEEE Transactions on Control Systems Technology 4, 105–124.
[39] Struss, P., 1997. Fundamentals of model-based diagnosis of dynamic systems, in: Fifteenth International Joint Conference on Artificial Intelligence (IJCAI 1997), Nagoya, Japan. pp. 480–485.