Model-Based Synthesis of Incremental and Correct Estimators for Discrete Event Systems

Stéphanie Roussel[1] , **Xavier Pucel**[1] , **Valentin Bouziat**[1] and **Louise Travé-Massuyès**[2]
[1] ONERA / DTIS, Université de Toulouse, F-31055 Toulouse – France
[2] LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
{stephanie.roussel, xavier.pucel, valentin.bouziat}@onera.fr, louise@laas.fr

Abstract

State tracking, i.e. estimating the state over time, is always an important problem in autonomous dynamic systems. Run-time requirements advocate for incremental estimation and memory limitations lead us to consider an estimation strategy that retains only one state out of the set of candidate estimates at each time step. This avoids the ambiguity of a high number of candidate estimates and allows the decision system to be fed with a clear input. However, this strategy may lead to dead-ends in the continuation of the execution. In this paper, we show that single-state trackability can be expressed in terms of the simulation relation between automata. This allows us to provide a complexity bound and a way to build estimators endowed with this property and, moreover, customizable along some correctness criteria. Our implementation relies on the Sat Modulo Theory solver MONOSAT and experiments show that our encoding scales up and applies to real world scenarios.

1 Introduction

For autonomous systems, state estimation and tracking is a critical task because it strongly influences the decisions that are made and that can be essential to the life of the system. It provides the means to diagnose the faults and to react to the various hazards that can affect the system.

In this paper, we focus on discrete event systems [Zaytoon and Lafortune, 2013]. When the system state is partially observable, the number of candidate estimates can quickly become too large to be usable. In embedded or distributed systems, memory and communication limitations can also become a problem, even with symbolic representations techniques such as in [Torta and Torasso, 2007]. These limitations lead us to propose an estimation strategy that retains only one state out of the set of candidate estimates at each time step, as in [Bouziat *et al.*, 2018]. This can be seen as an extreme strategy but nevertheless efficient to feed the decision system with a clear input and consistent with several works that select a limited number of best candidates according to some preference criterion, for example probabilities like in [Williams and Nayak, 1996; Kurien and Nayak, 2000]. However, the more we limit the number of estimates, the more we may be confronted with the problem of dead-ends related to the fact that next observations may prove that previous estimates were wrong and there is no continuation in the estimated trajectory. Because backtracking [Kurien and Nayak, 2000] is not a viable solution with real time constraints, we propose to build single-state estimators that are guaranteed to avoid dead-ends. The ability of a system to support the construction of a single-state dead-end free estimator has been defined as single-state trackability in [Bouziat *et al.*, 2019].

Several properties have been investigated for discrete event systems. A system is (strongly) detectable if one can determine its state along some (all) trajectories of the system [Shu *et al.*, 2007]. Diagnosability [Lafortune *et al.*, 2018] and manifestability [Dague *et al.*, 2019] amount to strong detectability and detectability for faulty trajectories. Single-state trackability is somewhat orthogonal to these properties.

In this paper, we recall the necessary and sufficient condition for single-state trackability as it is derived in [Bouziat *et al.*, 2019] based on equality of regular languages. A first contribution is to show that this condition can be expressed in terms of the simulation relation between automata [Holík *et al.*, 2018]. This allows us to provide a complexity bound for the single-state trackability problem and a way to build estimators endowed with this property. A second contribution of the paper is to propose two customizable correctness criteria that lead the estimator to constrain the acceptable estimator states towards the real states of the system. A third contribution is an implementation that relies on the Sat Modulo Theory (SMT) solver MONOSAT for synthesising estimators. Intensive experiments show that our implementation is able to synthesize dead-end free estimators on real world scenarios.

The paper is organised as follows. Related work is discussed in Section 2. Section 3 introduces incremental single-state estimation and how it is performed in our framework. The property of single-state trackability is defined and characterized in Section 4 and the associated decision problem is proved to be in NP. Customisable correctness of single-state estimators is formalized in Section 5. Section 6 presents the SMT encoding for synthesizing single-state estimators given a system specification. Experimental results are presented in Section 7, and future work is discussed in Section 8.

2 Related Work

A related research domain is the well-known controller synthesis problem, as described in [Ramadge and Wonham, 1987]. Given a specification of the system, controller synthesis consists in finding a decision procedure that produces a command to apply to the system, given observations as input. Under partial observability, a trend of work proposes to synthesize finite-state controllers, which use a size-limited memory to record information related to the past. These were first introduced for POMDP [Meuleau *et al.*, 1999] and shown to be useful for non-deterministic planning [Bonet *et al.*, 2009; Pralet *et al.*, 2010]. Restricting the information recorded from the past is also one important feature of our approach. However, there are several differences compared to our incremental estimation approach. The above works design controllers guaranteed to satisfy some properties related to planning, in particular the evolution of the controlled system must terminate in a goal state. In our estimation framework, we are rather interested in correctness properties, i.e. guaranteeing that the estimated state is "as close as possible" to the real state of the system (see Section 5).

Controller synthesis has also been approached within the game theory framework. In particular, [Doyen and Raskin, 2011] consider observation-based strategies for two-player turn-based games. Similar to our approach, observation-based strategies rely on imperfect information on the past observation sequence, hence defining partially observable games. Such games occur in the synthesis of a controller that does not see the private state of the plant. The main difference with our approach is that game theory considers games as complete graphs. Such a hypothesis means that strategies cannot encounter dead-ends in the execution, *i.e.* there is always a feasible action from any state of the game.

3 Incremental Single-State Estimation

In this section, we present the process of state estimation for partially observable systems modelled by non-deterministic Moore machines with an empty input alphabet, and whose output alphabet represents the system observations.

Definition 1 (System)**.** *A system M is defined by a 5-tuple (S, s_0, O, Δ, obs) consisting of the following:*

- *a finite set of states S;*
- *an initial state $s_0 \in S$;*
- *a finite set of observations O (output alphabet in the Moore machine);*
- *a transition relation $\Delta \subseteq S^2$;*
- *an observation function $obs : S \to O$ mapping each state to its output.*

Without loss of generality, we consider that all the states of the system are reachable from the initial state s_0.

We define $cands$ as the function from $S \times O$ to 2^S such that for all state s in S, for all o in O, $cands(s, o)$ represents the set of successors of s that have observation o. Formally, $\forall s \in S, \forall o \in O, cands(s, o) = \{t | (s, t) \in \Delta \text{ and } obs(t) = o\}$.

We also define $nextObs$ the function $S \to 2^O$ such that $nextObs(s)$ is the set of observations that can be observed just after the system is in state s. Formally, $\forall s \in S, nextObs(s) = \{o | cands(s, o) \neq \emptyset\}$.

Notation. A state sequence seq is a list $(s_0, s_1, \dots, s_{n-1})$ where each s_i is a state in S; $|seq| = n$ is the length of the sequence and $seq[i] = s_i$ is the ith state in the sequence; $last(seq)$ designates the last state of seq; if s is a state, $seq \cdot s$ is the sequence of length $|seq| + 1$ that begins with seq and ends with s. Similarly, we define an observation sequence. We also extend the function obs to a state sequence: if seq is a state sequence, $obs(seq)$ is the observation sequence seq_{obs} such that $|seq_{obs}| = |seq|$ and $seq_{obs}[i] = obs(seq[i])$ for all $i \in [0, n-1]$.

Definition 2 (Language, observation language)**.** *The language associated with a system $M = (S, s_0, O, \Delta, obs)$ is the set of state sequences accepted by the system and starting with s_0. Formally $\mathcal{L}(M) = \{seq \in S^+ | seq[0] = s_0$ and $\forall i \in [1, |seq| - 1], (seq[i-1], seq[i]) \in \Delta\}$.*
The observation language is the language accepted by the system projected on the observations. Formally, $\mathcal{L}_{obs}(M) = \{obs(seq) | seq \in \mathcal{L}(M)\}$.

We now focus on the estimation part for systems defined above. Since we consider non-deterministic, partially observable systems, there may be several state sequences that explain a given observation sequence. We adopt an incremental approach to select a unique explanation, called an *estimation strategy*. Then, we show that for a given system, any estimation strategy can be represented by a finite state machine called an estimator.

Definition 3 (Estimation strategy)**.** *An* incremental single-state estimation strategy *for a system $M = (S, s_0, O, \Delta, obs)$ is a function $estim : S \times O \to S$ such that for all s in S, for all o in $nextObs(s)$, $estim(s, o)$ represents the estimated state of the system at time step n if it was estimated in state s at time step $n - 1$ and if o is observed at time step n.*
We impose the estimation strategy to be consistent both across time (i.e. $estim$ is a function) and with the system behaviour (i.e. $estim(s, o)$ belongs to $cands(s, o)$).

For conciseness purposes, we use "estimation strategy" to refer to "incremental single-state estimation strategy". Given a system and an estimation strategy, the estimation process takes place as follows.

Definition 4 (Estimated sequence)**.** *Let M be a system, seq_{obs} be an observation sequence in $\mathcal{L}_{obs}(M)$ and $estim$ an estimation strategy for M. The estimated sequence for seq_{obs} is the state sequence $\widehat{seq} \in \mathcal{L}(M)$ such that $\widehat{seq}[0] = s_0$ and for all i in $[1, |\widehat{seq}| - 1]$, $\widehat{seq}[i] = estim(\widehat{seq}[i-1], seq_{obs}[i])$.*

Example 1. *Let $M = (S, s_0, O, \Delta, obs)$ be the system represented in Fig. 1. We have $S = \{s_0, \dots, s_5\}$, $O = \{a, b\}$, Δ is represented by the arrows in the figure, $obs(s_0) = obs(s_3) = obs(s_5) = a$ and $obs(s_1) = obs(s_2) = obs(s_4) = b$.*

In this system only two pairs in $S \times O$ have more than one estimation candidate: (s_0, b) and (s_1, a). Let $estim_1$ be the estimation strategy such that $estim_1(s_0, b) = s_1$ and $estim_1(s_1, a) = s_3$, and the unique candidate is selected

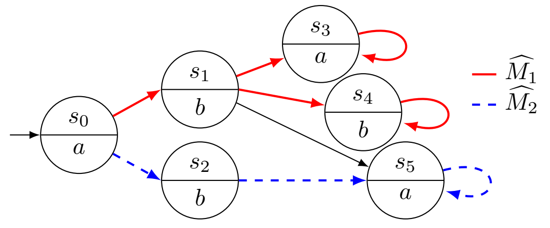

Figure 1: A simple system M, with one possible estimator $\widehat{M_1}$ depicted in solid red, and another $\widehat{M_2}$ in dashed blue.

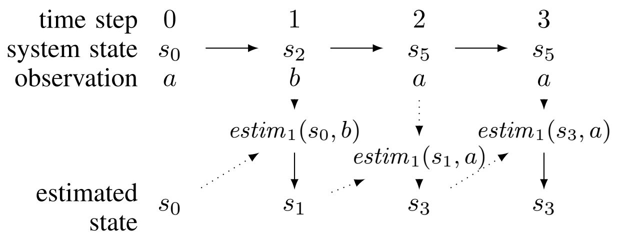

Figure 2: Estimation process in Example 1.

otherwise (for example, $cands(s_3, a) = \{s_3\}$ so necessarily $estim_1(s_3, a) = s_3$).

When the system produces the observation sequence (a, b, a, a), it can be explained by three state sequences in the system: (s_0, s_1, s_3, s_3), (s_0, s_1, s_5, s_5) and (s_0, s_2, s_5, s_5). With the strategy $estim_1$, we select the first one. The estimation process is illustrated in Figure 2 using this strategy.

In Example 1, one can observe that the state s_2 cannot be part of any estimated sequence, as the estimation strategy chooses a branch of the system states that does not contain s_2. The set of states that the estimation strategy actually uses is denoted $\widehat{S}$ and is formally defined by the set of states in S that are part of an estimation sequence associated with a sequence of observation in $\mathcal{L}_{obs}$.

An estimation strategy $estim$ for a system $M = (S, s_0, O, \Delta, obs)$ can also be represented by an *estimator*, *i.e.* a 5-tuple $\widehat{M} = (\widehat{S}, s_0, O, \widehat{\Delta}, obs)$ composed of the reachable states and transitions in the system with the strategy.

Definition 5 (Estimator)**.** *Let $M = (S, s_0, O, \Delta, obs)$ be a system and $estim$ an estimation strategy. The* estimator *induced by $estim$ is a deterministic Moore machine $\widehat{M} = (\widehat{S}, s_0, O, \widehat{\Delta}, obs)$ such that:*

- *$\widehat{S} \subseteq S$ is the set of states that belong to some estimated sequence for the given estimation strategy;*
- *$\widehat{\Delta}$ is the smallest subset of Δ such that $\forall \widehat{s} \in \widehat{S}, \forall o \in nextObs(\widehat{s})$, $(\widehat{s}, estim(\widehat{s}, o))$ belongs to $\widehat{\Delta}$.*

In Figure 1, $\widehat{M_1}$ is the estimator associated with the estimation strategy $estim_1$ introduced in Example 1.

Definition 5 indicates how for a given system, every estimation strategy can be represented by an estimator. Note that the reverse also holds: for a given system $M = (S, s_0, O, \Delta, obs)$, any deterministic Moore machine $\widehat{M} = (\widehat{S}, s_0, O, \widehat{\Delta}, obs)$ such that $\widehat{S} \subseteq S$ and $\widehat{\Delta} \subseteq \Delta$, is in fact an estimator that implements an estimation strategy $estim$ such that $estim(\hat{s}, o)$ is the unique $\hat{t}$ such that $(\hat{s}, \hat{t}) \in \widehat{\Delta}$ and $obs(\hat{t}) = o$, for $\hat{s} \in \widehat{S}$ and $o \in O$. In the rest of the paper, we use estimators to represent estimation strategies.

4 Single-State Trackability

4.1 Dead-ends

In this section, we first recall the notion of dead-end as defined in [Bouziat *et al.*, 2019]. During estimation, the estimator may choose a state sequence different from the one taken by the system. This can be particularly problematic when the following conditions happen: a) the system is in state s, the estimator estimates that it is in state $\widehat{s}$ with $\widehat{s} \neq s$; b) the system moves in state t and produces observation o; c) there exists no candidate $\widehat{t}$ in $\widehat{S}$ such that $obs(\widehat{t}) = o$ and $(\widehat{s}, \widehat{t})$ belongs to $\widehat{\Delta}$.

Definition 6 (Dead-end)**.** *A* dead-end *for an estimator $\widehat{M}$ of a system M is an observation sequence that belongs to $\mathcal{L}_{obs}(M)$ but does not belong to $\mathcal{L}_{obs}(\widehat{M})$.*
An estimator $\widehat{M}$ is dead-end free *if there is no dead-end for it, i.e. $\mathcal{L}_{obs}(M) = \mathcal{L}_{obs}(\widehat{M})$.*

Example 2. *We consider the example illustrated in Figure 1. Let $\widehat{M_2}$ be the estimator induced by the estimation strategy $estim_2$ defined by $estim_2(s_0, b) = s_2$ (no other choices necessary). The sequence of observations (a, b, b, b) belongs to $\mathcal{L}_{obs}(M)$ as it is produced by the state sequence (s_0, s_1, s_4, s_4) but it does not belong to $\mathcal{L}_{obs}(\widehat{M_2})$. This observation sequence is therefore a dead-end for $\widehat{M_2}$. Note that $\widehat{M_1}$ is dead-end free.*

In [Kurien and Nayak, 2000], dead-ends are handled by backtracking in time in order to modify previous estimation choices. The main drawback of this technique is the lack of control over both the computational time and memory. Moreover, some actions may have been taken based on the previous estimated states that cannot be undone. Backtracking in time and revising estimations may turn past actions sub-optimal or even inconsistent.

Another way to tackle dead-ends is to keep a fixed number k of past observations and reason on this window [Su *et al.*, 2014]. However this approach is also subject to dead-ends as the estimator and system paths may diverge earlier beyond the memorized window.

Our goal is to design estimators able to follow the execution of a system while keeping just a unique state in memory and never encountering dead-ends. Such deterministic incremental estimators may be feasible depending on the system specifications. We express that possibility by defining single-state trackability for discrete event systems.

4.2 Trackability Problem

In [Bouziat *et al.*, 2019], a system is considered as single-state trackable if there exists a dead-end free single-state incremental estimator for this system. They show that this is the case if and only if there exists an estimator with an observation language equal to that of the system. In this paper, we take the language equivalence as a definition of Single-State Trackability.

Definition 7 (Single-State Trackability)**.** *A system M is* single-state trackable *if there exists an estimator $\widehat{M}$ for M such that $\mathcal{L}_{obs}(M) = \mathcal{L}_{obs}(\widehat{M})$.*

Determining whether a system is single-state trackable or not is called the TRACKABILITY problem.

The problem of deciding if two languages are equivalent (also called trace-equivalence) is, in the general case, a PSPACE-complete problem [Hüttel and Shukla, 1996]. This leads to the definition of a rather complex algorithm for the TRACKABILITY problem in [Bouziat *et al.*, 2019]. This approach can be improved because the system and the estimator, for which language equivalence is considered, are not independent from each other. In fact, the estimator is a submachine of the system's machine. A contribution of this paper is to prove that in this case, the language equivalence problem is reduced to the simulation preorder problem, which is decidable in polynomial time [Shukla *et al.*, 1996].

We first adapt the definition of simulation to the case of a system and an estimator.

Definition 8 (Simulation for estimators)**.** *Let $M = (S, s_0, O, \Delta, obs)$ be a system and $\widehat{M} = (\widehat{S}, s_0, O, \widehat{\Delta}, obs)$ be an estimator for M. $\widehat{M}$ simulates M if there exists a relation $R \subseteq S \times \widehat{S}$ such that:*

$$(s_0, s_0) \in R \tag{1}$$

$$\forall (s, \widehat{s}) \in R, obs(s) = obs(\widehat{s}) \tag{2}$$

$$\forall (s, \widehat{s}) \in R, \forall t \in S \text{ s.t. } (s, t) \in \Delta, \quad \exists \widehat{t} \in \widehat{S} \text{ s.t. } (\widehat{s}, \widehat{t}) \in \widehat{\Delta} \text{ and } (t, \widehat{t}) \in R \tag{3}$$

Informally, if s is a state in S and $\widehat{s}$ a state in $\widehat{S}$, then $\widehat{s}$ simulates s (or $(s, \widehat{s})$ is in R) represents the fact that $\widehat{M}$ can estimate that the system is in state $\widehat{s}$ while it really is in state s without encountering a dead-end later in the execution.

Proposition 1. *Let M be a system and $\widehat{M}$ be an estimator for M. $\widehat{M}$ is dead-end free (or $\mathcal{L}_{obs}(M) = \mathcal{L}_{obs}(\widehat{M})$) if and only if $\widehat{M}$ simulates M.*

Proof. ($\Leftarrow$) $\mathcal{L}_{obs}(\widehat{M}) \subseteq \mathcal{L}_{obs}(M)$ as the states and transitions of $\widehat{M}$ are subsets of those of the system. Moreover, let $seq_{obs} \in \mathcal{L}_{obs}(M)$ be produced by $seq \in \mathcal{L}(M)$. We prove that for all $k < |seq|$, there exists a sequence $\widehat{seq} \in \mathcal{L}(\widehat{M})$ of length k such that $obs(seq) = obs(\widehat{seq})$ and $(seq[k], \widehat{seq}[k]) \in R$, which entails $\mathcal{L}_{obs}(\widehat{M}) \supseteq \mathcal{L}_{obs}(M)$.

($\Rightarrow$) We consider the following relation R: $\forall s \in S, \widehat{s} \in \widehat{S}$, $(s, \widehat{s}) \in R$ iff $\exists seq_{obs} \in \mathcal{L}_{obs}(M)$, $seq \in \mathcal{L}(M)$, $\widehat{seq} \in \mathcal{L}(\widehat{M})$ *s.t.* $obs(seq) = obs(\widehat{seq}) = seq_{obs}$, $s = last(seq)$ and $\widehat{s} = last(\widehat{seq})$. We show that R is a simulation relation. (1,2) By construction of R, we have $(s_0, s_0) \in R$, and for all $(s, \widehat{s}) \in R$, $obs(s) = obs(\widehat{s})$. (3) Let $(s, \widehat{s}) \in R$ and $t \in S$ such that $(s, t) \in \Delta$. By construction of R, $\exists seq_{obs} \in \mathcal{L}_{obs}(M)$, $seq \in \mathcal{L}(M)$, $\widehat{seq} \in \mathcal{L}(\widehat{M})$ *s.t.* $obs(seq) = obs(\widehat{seq})$, $s = last(seq)$ and $\widehat{s} = last(\widehat{seq})$. $(s, t) \in \Delta$ so $seq.t \in \mathcal{L}(M)$ and produces the sequence of observation $seq_{obs}.obs(t) \in \mathcal{L}_{obs}(M)$. By the equality of languages and definition of $\mathcal{L}_{obs}(\widehat{M})$, there exists $\widehat{seq}' \in \mathcal{L}(\widehat{M})$ *s.t.* $obs(\widehat{seq}') = seq_{obs}.obs(t)$. As $\widehat{M}$ is deterministic, seq_{obs} is generated by the same state sequence $\widehat{seq}$. So, $\widehat{seq}'$ is equal to $\widehat{seq}.\widehat{t}$ and $(\widehat{s}, \widehat{t}) \in \widehat{\Delta}$. By construction of R, $(t, \widehat{t}) \in R$. □

The following corollary presents a necessary condition for estimators. It indicates that any observation sequence that can follow a state s can also follow the state $\widehat{s}$ by which s is estimated. This corollary is used in Section 6 to increase the efficiency of estimator synthesis.

Corollary 1. *Let M be a system, $\widehat{M}$ a dead-end free estimator for M, s a state in S, $\widehat{s}$ a state in $\widehat{S}$, and R a simulation relation such that $(s, \widehat{s}) \in R$. For any observation sequence seq_{obs}, if seq_{obs} can follow s then seq_{obs} can follow $\widehat{s}$.*

Corollary 2. *The* TRACKABILITY *problem is in* NP.

Proof. Checking if $\widehat{M}$ is a dead-end free estimator for M is polynomial since it comes to checking whether $\widehat{M}$ simulates M, which can be encoded into HORN-SAT clauses as shown in [Shukla *et al.*, 1996]. □

5 Estimator Correctness

Correctness is the property of an estimator that estimates the actual system state. So far we provided a way to generate dead-end free estimators, which can be seen as a very weak form of correctness as a dead-end free estimator cannot be proven wrong by the system's observations. However, depending on the operational context, total correctness may be required or an incorrect pessimistic or optimistic estimation may be satisfying [Bouziat *et al.*, 2018; Couto *et al.*, 2020].

In this section, we model correctness as constraints and optimization criteria that allow some end-user to generate interesting estimators. Correctness constraints allow one to restrict the possible estimator states associated with some given system state. Correctness criteria let one associate a cost with each possible estimator, with the lowest cost being associated with the "most correct" estimators.

Definition 9 ((ϕ_1, ϕ_2)-correctness)**.** *Let M be a system, $\phi_1 \subseteq S$ and $\phi_2 \subseteq S$ be two sets of states of M, $\widehat{M}$ a dead-end free estimator for M, and R the associated simulation relation. $\widehat{M}$ is (ϕ_1, ϕ_2)-correct if whenever the system is in a state of ϕ_1, then the estimator is in a state of ϕ_2, i.e. $\forall (s, \widehat{s}) \in R$, if $s \in \phi_1$ then $\widehat{s} \in \phi_2$.*

In a symbolic model where states are represented by Boolean variables, correctness constraints can be represented by propositional formulae. They can also be deduced from a proof of reachability of an undesirable pair $(s, \hat{s})$ as produced in [Couto *et al.*, 2020]. With such constraints, one can require that whenever some critical fault is present in the system state, it is also present in the estimated state. This ensures the estimator is correct with respect to this critical fault.

Example 3. *Let us consider the system M presented in Example 1 and illustrated on Figure 1. Let f_1 be a fault only present in states $\phi^{f_1} = \{s_1, s_3, s_4\}$ and f_2 a fault only present in $\phi^{f_2} = \{s_5\}$. The estimator $\widehat{M_1}$ is (ϕ^{f_1}, ϕ^{f_1})-correct as all the states in ϕ^{f_1} are estimated by states in ϕ^{f_1}. However $\widehat{M_1}$ is not (ϕ^{f_2}, ϕ^{f_2})-correct as s_5 is estimated by s_3 which does not belong to ϕ^{f_2}.*

While it is very flexible, in some systems, (ϕ_1, ϕ_2)-correctness might be either too strong or too tedious to specify. For example, in Example 3, there exists no estimator that satisfies both (ϕ^{f_1}, ϕ^{f_1}) and (ϕ^{f_2}, ϕ^{f_2})-correctness. In order to relax this requirement, a first laborious approach is to enumerate the possible estimated states for each system state. Another approach consists in expressing a weaker form of correctness, for instance that (ϕ_1, ϕ_2)-correctness should be satisfied "as much as possible". In this regard we introduce a cost function that lets one model this kind of requirement.

Definition 10 (Correctness cost). *Let M be a system, $\widehat{M}$ an estimator for M, and R their simulation relation. Let $cost : S \times \widehat{S} \to \mathbb{N}$ be a cost function such that $cost(s, \widehat{s})$ represents how much we want to avoid estimating that the system is in state $\widehat{s}$ when its real state is s. The correctness cost for $\widehat{M}$ is $cost(\widehat{M}) = \sum_{(s,\widehat{s}) \in R} cost(s, \widehat{s})$*

Definition 11 (Correctness order). *Let M be a system, $\widehat{M_1}$ and $\widehat{M_2}$ be two dead-end free estimators for this system, and cost a cost function. $\widehat{M_1}$ is strictly more correct than $\widehat{M_2}$ with respect to cost if $cost(\widehat{M_1}) < cost(\widehat{M_2})$.*

There are several ways to define a cost for pairs of states. A straightforward one is to check if the real state of the system is correctly estimated. Such a cost $c_=$ is formally defined by $\forall (s, \widehat{s}) \in S^2, c_=(s, \widehat{s}) = 0$ if $s = \widehat{s}$, 1 otherwise. One could also consider a cost based on (ϕ_1, ϕ_2)-correctness: $\forall (s, \widehat{s}) \in S^2, c_{\phi_1,\phi_2}(s, \widehat{s}) = 0$ if $s \in \phi_1$ and $\widehat{s} \in \phi_2$, 1 otherwise. Symbolic models make it easier to specify cost functions. The Hamming distance ([Hamming, 1950]) is an example of a cost function for Boolean variables. Weighted sums, or lexicographical aggregation of costs can also be used to emphasize some critical faults for example.

Example 4. *Let us consider the system M presented in Example 1 and illustrated on Figure 1. We define the dead-end free estimator $\widehat{M_3}$ similar to $\widehat{M_1}$ except that $estim_3(s_1, a) = s_5$. We consider a correctness cost c such that $\forall i, c(s_i, s_i) = 0$, $c(s_1, s_2) = c(s_2, s_1) = 1$, $c(s_5, s_3) = 1$ and $c(s_3, s_5) = 2$. This cost represents that it is preferable to estimate that the system is in state s_3 when it really is in state s_5 than the opposite. We have $c(\widehat{M_1}) = 2$ and $c(\widehat{M_3}) = 3$. This means that $\widehat{M_1}$ is more correct than $\widehat{M_3}$ with respect to cost c.*

6 Estimator Synthesis

In this section, we describe a procedure to solve the TRACKABILITY problem. To decide if a system M is single-state trackable, we try to synthesize a dead-end free estimator for it. More precisely, we consider that a system M can be seen as a graph and that a dead-end free estimator $\widehat{M}$ is a sub-graph that satisfies several constraints, that can be encoded into SMT provided the underlying theory provides graph predicates. We specifically target the solver MONOSAT ([Bayless *et al.*, 2015]), that allows to handle classical SAT constraints but also graph constraints.

Dead-end free estimator synthesis. Let $M = (S, s_0, O, \Delta, obs)$ be a system and $\widehat{M} = (\widehat{S}, s_0, O, \widehat{\Delta}, obs)$ the dead-end free estimator we try to synthesize for M. If $\widehat{M}$ exists (*i.e.* M is single-state trackable), then states and transitions of $\widehat{M}$ are subsets of those of M and following Prop. 1, there exists a simulation relation R between the states of M and $\widehat{M}$.

We associate the system M with a directed graph G whose vertices are the states S and whose edges are the transitions Δ. Similarly, $\widehat{G}$ is the graph associated with an estimator $\widehat{M}$.

First, we instantiate the following Boolean decision variables:

- for each $s \in S$, $\mathbf{v}_s$ is true iff s belongs to $\widehat{S}$;
- for each $(s, t) \in \Delta$, $\mathbf{e}_{s,t}$ is true iff (s, t) belongs to $\widehat{\Delta}$;
- for each $(s, \widehat{s})$ in S^2, $\mathbf{r}_{s,\widehat{s}}$ is true iff $(s, \widehat{s})$ belongs to R.

We next instantiate the following constraints.

$$\forall (s, t) \in \Delta, \mathbf{e}_{s,t} \to \mathbf{v}_s \wedge \mathbf{v}_t \tag{4}$$

$$\forall \widehat{s} \in S, \mathbf{v}_{\widehat{s}} \leftrightarrow reachable(\widehat{G}, s_0, \widehat{s}) \tag{5}$$

$$\forall s \in S, \forall o \in nextObs(s), \sum_{t \in S \mid obs(t) = o} \mathbf{e}_{s,t} \leq 1 \tag{6}$$

$$\forall s \in S, \forall o \in nextObs(s), \mathbf{v}_s \to \bigvee_{t \in S \mid obs(t) = o} \mathbf{e}_{s,t} \tag{7}$$

$$\mathbf{r}_{s_0,s_0} \tag{8}$$

$$\forall s \in S, \bigvee_{\widehat{s} \in S, obs(s) = obs(\widehat{s})} \mathbf{r}_{s,\widehat{s}} \tag{9}$$

$$\forall (s, \widehat{s}) \in S^2 \text{ s.t. } obs(s) = obs(\widehat{s}), \forall t \in S \text{ s.t. } (s, t) \in \Delta,$$
$$\mathbf{r}_{s,\widehat{s}} \to \bigvee_{(\widehat{s},\widehat{t}) \in \Delta, obs(t) = obs(\widehat{t})} (\mathbf{r}_{t,\widehat{t}} \wedge \mathbf{e}_{\widehat{s},\widehat{t}}) \tag{10}$$

$$\forall \widehat{s} \in S, \mathbf{v}_{\widehat{s}} \leftrightarrow \bigvee_{s \in S \mid obs(s) = obs(\widehat{s})} \mathbf{r}_{s,\widehat{s}} \tag{11}$$

We first encode that the estimator is a well-built automaton. Constraint (4) states that if an edge is selected in $\widehat{G}$, its vertices are selected as well. Constraint (5) enforces the states of the estimator to be exactly the ones reachable from s_0 in the graph $\widehat{G}$. We use here the MONOSAT predicate *reachable*.

Constraints (6) and (7) force $\widehat{M}$ to be deterministic: for every state s and every successor observation o, exactly one outgoing transition leads to a state with observation o. We use two constraints, as in SAT/SMT the "at most one" cardinality constraint (6) cannot easily be combined with the implication of Constraint (7).

We next consider constraints related to the simulation relation. Constraint (8) states that the initial state simulates itself (see Definition 8). Constraint (9) encodes that every state of the system is simulated by at least one state of the estimator. Constraint (10) encodes the simulation relation from Definition 8 and also states that the edge associated with the simulation relation must belong to $\widehat{M}$[1].

Constraint (11) makes the simulation relation minimal as the estimator states must all simulate at least one system state.

Using a SMT solver let us use the predicate *reachable* in constraint (5). This constraint can be expressed in SAT clauses using for instance the *Floyd-Warshall* algorithm [Cormen *et al.*, 2001] with a complexity of $\mathcal{O}(|V|^3)$. MONOSAT achieves a complexity of $\mathcal{O}(|V| \cdot |E|)$ [Bayless *et al.*, 2015].

From Definition 8 and Corollary 1, the number of variables $\mathbf{r}_{s,\widehat{s}}$ can be reduced by creating variables only for pairs $(s, \widehat{s})$ for which $\widehat{s}$ accepts all the observation sequences of length 2 (or more) that s accepts.

Correct estimator synthesis. The encoding presented above allows to synthesize dead-end free estimators. It is possible to enrich it to synthesize correct estimators, as defined in Section 5. (ϕ_1, ϕ_2)-correctness can be encoded as a hard constraint the following way: $\forall s \in \phi_1, \forall \widehat{s} \in S \setminus \phi_2, \neg \mathbf{r}_{s,\widehat{s}}$. The second correctness requires a cost function c as input, and is encoded as the following pseudo Boolean criterion: $\mathbf{minimize}\ \sum_{(s,\widehat{s}) \in R} \mathbf{r}_{s,\widehat{s}} \cdot c(s, \widehat{s})$.

7 Experiments

Our implementation uses the SCALA programming language, and each experiment was performed on a computer with a Intel® Xeon® CPU E5-2699 v3 @ 2.30GHz processor and a limit of 16GB of memory.

Examples from the literature. We have encoded benchmarks from three system models from the literature:

- *valve controller* [Sampath *et al.*, 1995] that models a valve controller for heating, ventilation and air conditioning units with a valve that can be stuck closed or stuck open, a pump that can fail in modes on and off and a controller; each component has 4 possible states; this system is not trackable.
- *valve driver* [Williams and Nayak, 1996] that models a valve driver that has 4 modes, 6 command inputs and 3 command outputs; this system is trackable.
- *baggage transfer* [Pencolé *et al.*, 2018] that models a baggage transfer system composed of 2 conveyors, 1 piston, 1 controller and 4 sensors. No fault model is provided, so we enriched each component with a permanent fault model. By enabling or not the fault model for each of the 8 components, we managed to create 218 systems among 256, all of which are trackable.

Random examples. In order to test our prototype intensively, we randomly generated a set of systems as follows. First, we generated a set of small systems of 5 states, 3 observations, and 0.5 transition probability for any two states. Second, we aggregated them by Cartesian product (of states and observations), with a 0.1 probability that any two states are incompatible (in this case their pair is removed from the aggregated states). The benchmark contains systems obtained by aggregation of 4 and 5 small systems. Finally, we selected only "interesting" systems, such that:

1. the system is trackable;
2. the estimator has less than half the number of states of the system (so as to reject trivially trackable systems);
3. there exists a state s such that the system has no $(\{s\}, S \setminus \{s\})$-correct estimator.

This method generated systems from 10 to 1461 states, with more small systems than large ones.

[1]Note that Constraint (9) is redundant with Constraints (8) and (10), but improves the solver's performance.

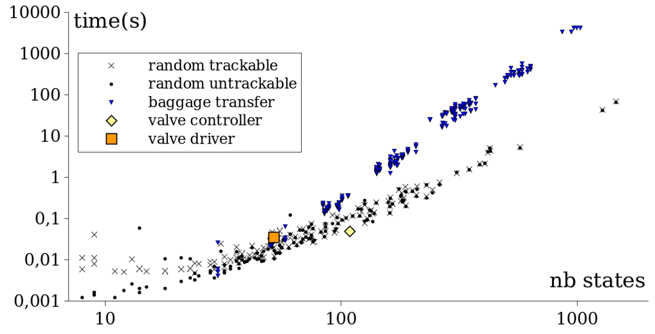

Figure 3: Computation time *w.r.t.* number of system states. Both scales are logarithmic.

Analysis. In all our experiments, most of the computation time is used for generating SMT clauses whereas the time required by MONOSAT to solve the SMT instance is negligible. This explains why the computation time is nearly the same for the trackable and untrackable versions of random examples. We have yet to properly assess the impact of a cost function on the solving time. Figure 3 shows that on random benchmarks, the solving time grows regularly in the number of states. The *valve driver* and *valve controller* systems require similar computation time as random systems. For the *baggage transfer* system, the computation time is higher, because of the large number of symmetries in the system.

8 Conclusion

This paper describes a new approach for checking single-state trackability as defined in [Bouziat *et al.*, 2019]. As a system is simulated by its estimator, we prove that the TRACKABILITY problem is in NP. We also define two customizable correctness types for modelling relevant estimators. We solve the TRACKABILITY problem by reducing it to a MONOSAT SMT query, along with correctness requirements. The approach is validated on a set of benchmarks both from the literature and from randomly generated problems.

Studying NP-completeness with and without correctness constraints is a natural next step in our work. Allowing for delayed estimation, or using a bounded number of memorized states at each time step could help apply this approach to a larger range of systems, including real-world systems, and will be considered in future work.

Symbolic representation of estimators with preferences as in [Bouziat *et al.*, 2018] is also a possible extension.

References

[Bayless *et al.*, 2015] Sam Bayless, Noah Bayless, Holger H. Hoos, and Alan J. Hu. SAT modulo monotonic theories. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pages 3702—3709, 2015.

[Bonet *et al.*, 2009] Blai Bonet, Hector Palacios, and Hector Geffner. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling*, pages 34–41, 2009.

[Bouziat *et al.*, 2018] Valentin Bouziat, Xavier Pucel, Stéphanie Roussel, and Louise Travé-Massuyès. Preferential discrete model-based diagnosis for intermittent and permanent faults. In *Proceedings of the 29th International Workshop on Principles of Diagnosis*, 2018.

[Bouziat *et al.*, 2019] Valentin Bouziat, Xavier Pucel, Stéphanie Roussel, and Louise Travé-Massuyès. Single state trackability of discrete event systems. In *Proceedings of the 30th International Workshop on Principles of Diagnosis*, 2019.

[Cormen *et al.*, 2001] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Transitive closure of a directed graph*, pages 993–1024. The MIT Press, 2001.

[Couto *et al.*, 2020] Diego Couto, Kévin Delmas, and Xavier Pucel. On the safety assessment of RPAS safety policy. *Proceedings of the 9th European Congress Embedded Real Time Software and Systems*, 2020.

[Dague *et al.*, 2019] Philippe Dague, Lulu He, and Lina Ye. How to be sure a faulty system does not always appear healthy?: Fault manifestability analysis for discrete event and timed systems. *Innovations in Systems and Software Engineering*, 2019.

[Doyen and Raskin, 2011] Laurent Doyen and Jean-François Raskin. Games with imperfect information: Theory and algorithms. In *Lectures in Game Theory for Computer Scientists*, pages 185—-212, 2011.

[Hamming, 1950] Richard W. Hamming. Error-detecting and error-correcting codes. *Bell System Technical Journal*, 29(2):147–160, 1950.

[Holík *et al.*, 2018] Lukáš Holík, Ondřej Lengál, Juraj Síč, Margus Veanes, and Tomáš Vojnar. Simulation algorithms for symbolic automata. In *Automated Technology for Verification and Analysis*, pages 109–125, 2018.

[Hüttel and Shukla, 1996] Hans Hüttel and Sandeep Shukla. On the Complexity of Deciding Behavioural Equivalences and Preorders. A Survey. *BRICS Report Series*, 3(39), 1996.

[Kurien and Nayak, 2000] James Kurien and P. Pandurang Nayak. Back to the future for consistency-based trajectory tracking. In *Proceedings of the 17th AAAI Conference on Artificial Intelligence*, pages 370–377, 2000.

[Lafortune *et al.*, 2018] Stéphane Lafortune, Feng Lin, and Christoforos N. Hadjicostis. On the history of diagnosability and opacity in discrete event systems. *Annual Reviews in Control*, 45:257–266, 2018.

[Meuleau *et al.*, 1999] Nicolas Meuleau, Leonid Peshkin, Kee-Eung Kim, and Leslie Pack Kaelbling. Learning finite-state controllers for partially observable environments. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, page 427–436, 1999.

[Pencolé *et al.*, 2018] Yannick Pencolé, Gerald Steinbauer, Clemens Mühlbacher, and Louise Travé-Massuyès. Diagnosing discrete event systems using nominal models only. In *28th International Workshop on Principles of Diagnosis*, pages 169–183, 2018.

[Pralet *et al.*, 2010] Cédric Pralet, Gérard Verfaillie, Michel Lemaître, and Guillaume Infantes. Constraint-based controller synthesis in non-deterministic and partially observable domains. In *Proceedings of the 19th European Conference on Artificial Intelligence*, pages 681–686, 2010.

[Ramadge and Wonham, 1987] Peter J. G. Ramadge and W. Murray Wonham. Supervisory control of a class of discrete event processes. *SIAM journal on control and optimization*, 25(1):206–230, 1987.

[Sampath *et al.*, 1995] Meera Sampath, Raja Sengupta, Stéphane Lafortune, Kasim Sinnamohideen, and Demosthenis Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on automatic control*, 40(9):1555–1575, 1995.

[Shu *et al.*, 2007] Shaolong Shu, Feng Lin, and Hao Ying. Detectability of discrete event systems. *IEEE Transactions on Automatic Control*, 52(12):2356–2359, 2007.

[Shukla *et al.*, 1996] Sandeep Shukla, Daniel Rosenkrantz, Harry Iii, and Richard Stearns. The polynomial time decidability of simulation relations for finite state processes: A HORNSAT based approach. *Satisfiability Problem: Theory and applications*, 1996.

[Su *et al.*, 2014] Xingyu Su, Yannick Pencolé, and Alban Grastien. Window-based diagnostic algorithms for discrete event systems: What information to remember. In *Proceedings of the 25th International Workshop on Principles of Diagnosis*, 2014.

[Torta and Torasso, 2007] Gianluca Torta and Pietro Torasso. An on-line approach to the computation and presentation of preferred diagnoses for dynamic systems. *AI Communications*, 20(2):93–116, 2007.

[Williams and Nayak, 1996] Brian C. Williams and P. Pandurang Nayak. A model-based approach to reactive self-configuring systems. In *Proceedings of the 13th AAAI Conference on Artificial Intelligence*, pages 971–978, 1996.

[Zaytoon and Lafortune, 2013] Janan Zaytoon and Stéphane Lafortune. Overview of fault diagnosis methods for discrete event systems. *Annual Reviews in Control*, 37(2):308–320, 2013.