DPO-SYNT: Discrete Control Synthesis for Partially-Observed Systems ⋆

Xiang Yin * **Maxwell Morrison** * **Siyuan Sheng** *
Stéphane Lafortune *

* *Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109, USA. (e-mail:{xiangyin,morrimax,ssyssy,stephane}@umich.edu)*

Abstract: This paper describes DPO-SYNT, a C++ based software toolbox for property enforcement in partially-observed Discrete Event Systems. DPO-SYNT implements a recently developed uniform framework for synthesizing maximally-permissive supervisors and optimal sensor activation policies. It can handle the enforcement/synthesis problems for a large variety of properties, including safety, opacity, diagnosability, and detectability, in a uniform manner.

Keywords: Synthesis Tools, Discrete Event Systems, Partial Observation, Supervisory Control, Sensor Activation.

1. INTRODUCTION

The DPO-SYNT software toolbox is designed for the synthesis of *supervisors* and *sensor activation policies* for partially-observed Discrete Event Systems (DES) modeled by finite-state automata. It is based upon the recently developed framework presented in [Yin and Lafortune 2015, 2016b,a, 2017] that handles the supervisory control problem and the sensor activation problem in a uniform manner. DPO-SYNT can be used to solve a large class of supervisory control problems under the partial observation setting, including but not restricted to, the standard supervisory control problem, the active diagnosis problem, the discrete stabilization problem, and the opacity enforcement problem. For all of the above mentioned problems, the non-blockingness requirement can also be guaranteed. When the property under consideration is safety, DPO-SYNT guarantees the minimal behavior achieved by solving a range control problem. DPO-SYNT can also handle the sensor activation policy synthesis problem in a similar manner. This allows the use of DPO-SYNT to synthesize optimal sensor activation policies for the purposes of supervisory control, fault diagnosis, and fault prognosis. DPO-SYNT implements the algorithms in [Yin and Lafortune 2015, 2016b,a, 2017] in an explicit manner, without using symbolic methods. As such, it is a useful educational and research tool for systems of small or moderate size.

DPO-SYNT can be downloaded from
`https://gitlab.eecs.umich.edu/M-DES-tools/DPO-SYNT`

1.1 Related Software Tools

Several software tools have been developed for solving analysis and synthesis problems for DES modeled as finite-state automata (FSA). Among them, we mention `DESUMA` [Ricker et al. 2006], `SUPREMICA` [Åkesson et al. 2006], `TCT` [Feng and Wonham 2006], `LibFAUDES` [Moor et al. 2008], `DESLAB` [Clavijo et al. 2012], `DESPOT` [Leduc 2015], and `IDES` [Rudie 2006]. Compared with the above mentioned tools, DPO-SYNT has the following distinguishing features: (1) it solves a large class of control problems in addition to the standard supervisory control problem; (2) it handles the issue of non-blockingness under the partial observation setting; (3) it synthesizes optimal sensor activation policies. These capabilities extend the state-of-the-art in FSA-based DES tools.

2. DESCRIPTION OF DPO-SYNT

2.1 Theoretical Foundations

The main computational object in DPO-SYNT is a DES modeled as a deterministic FSA, as a special case of a labeled transition system. DPO-SYNT can handle two types of synthesis problems: the *supervisory control problem* and the *dynamic sensor activation problem*. We briefly review the theoretical foundations of DPO-SYNT; the reader is referred to the publications [Yin and Lafortune 2015, 2016b,a, 2017] for more technical details.

In our recent work [Yin and Lafortune 2016b,a], a two-stage approach for synthesizing *non-blocking* and *property-enforcing* supervisors was proposed. The class of properties under consideration are so-called *Information-State-based* (or IS-based) properties. Specifically, an IS-based property is a predicate $\varphi : 2^X \to \{0, 1\}$, where X is the set of states of the FSA. It was shown by Yin and Lafortune [2016b] that many important system-theoretic properties such as safety, K-diagnosability, and opacity, can be formulated as IS-based properties. The key of the uniform computational approach is the construction of two discrete structures called the *All Enforcement Structure* (AES) and the *Non-Blocking All Enforcement Structure* (NB-AES). Based on the (NB-)AES, a maximally-permissive (non-blocking) supervisor satisfying φ can be synthesized.

⋆ This work was partially supported by NSF grants CCF-1138860 (Expeditions in Computing project ExCAPE: Expeditions in Computer Augmented Program Engineering), CNS-1421122, and CNS-1446298.

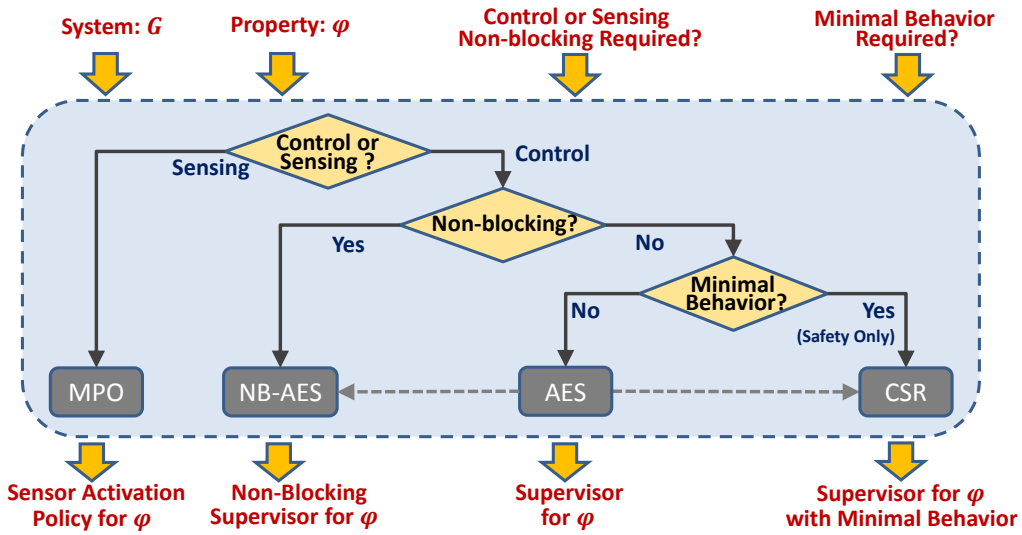

Fig. 1. The architecture of DPO-SYNT

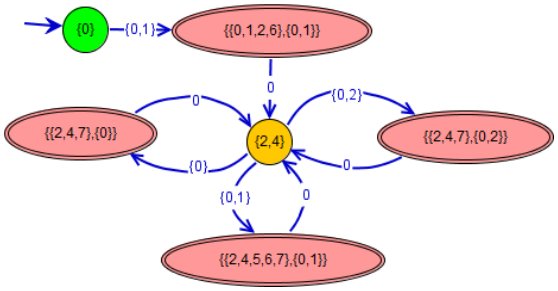

Fig. 2. Example of a complete NB-AES

A similar approach was also developed recently in [Yin and Lafortune 2015] for the synthesis of optimal sensor activation policies. The key structure in this synthesis problem is called the *Most Permissive Observer* (MPO), which embeds all valid sensor activation policies in its structure. Based on the MPO, an optimal sensor activation policy can be synthesized.

2.2 The Tool

To use DPO-SYNT, first, we need to provide the plant automaton, which is implemented by class `FSM`. The `FSM` format is depicted in Table 1; each state is then mapped to an integer in the C++ toolbox. Second, we need to specify the IS-based property $\varphi : 2^X \to \{0, 1\}$ that is to be enforced. Each information state, which is a set of states of the plant automaton, is encoded as a Boolean variable without enumerating all state names as a set. In the current version of DPO-SYNT, we have included three different types of IS-based properties, namely, safety, state distinguishability, and opacity. For example, for the safety requirement, we need to specify the set of *illegal states*, while for the state disambiguation problem, we need to specify the set of *state pairs* that we need to distinguish. One is also allowed to define any custom IS-based property in DPO-SYNT by defining a *truth table* on 2^X.

Next, we need to select whether we want to solve a supervisory control problem or a sensor activation problem. Under the selection of the supervisory control problem for instance, one can further select whether or not the non-blockingness condition is required. If one is only interested in enforcing safety without non-blockingness, then DPO-SYNT can also guarantee the minimal behavior achieved by the resulting supervisor; this is done by computing the *Control Simulation Relation* (CSR) developed in [Yin and Lafortune 2017].

⟨number_of_states⟩ ⟨number_of_events⟩

⟨state_1⟩ ⟨m or u⟩ /*marked or not*/
⋮
⟨state_n⟩ ⟨m or u⟩

⟨event_1⟩ ⟨c or u⟩ ⟨o or u⟩ /*controllability & observability*/
⋮
⟨event_n⟩ ⟨c or u⟩ ⟨o or u⟩

⟨event state⟩ ⟨event state⟩ ... /*transitions from state_1*/
⋮
⟨event state⟩ ⟨event state⟩ ... /*transitions from state_n*/

Table 1. The `FSM` format in DPO-SYNT

The basic architecture of DPO-SYNT is summarized in Figure 1. The default output of DPO-SYNT is either an optimal sensor activation policy, which is realized by a Bipartite Dynamic Observer (BDO), or a maximally-permissive supervisor, which is realized by a Bipartite Transition System (BTS). The BPO and the BTS are discrete structures that can be viewed as a form of bipartite labeled transition system. One can also choose to export the AES, the NB-AES, or the MPO, which are constructed in the intermediate steps. The similarity between the AES and the MPO allows the implementation of these two structures, as well as the BTS and the BDO, with the same data structure in DPO-SYNT.

DPO-SYNT is implemented in C++ with approximately 5000 lines of code (current version). DPO-SYNT is command-line-based and it does not have a GUI. However, command `CONVERT` is provided to transform an FSA

in DPO-SYNT FSM format to the `.fsm` format used in `DESUMA`. This allows drawing an FSA using the layout and export capabilities of `DESUMA`. Note that both the AES and the MPO are bipartite structures with two different types of states; this feature is preserved by using marked states in the conversion, namely, marking is used to distinguish between the two different types of states. An example of AES is shown in Figure 2, in which each state without a double circle is called a Y-state (where the supervisory controller acts) and each state with a double circle is called a Z-state (where the system reacts). Note that the structure in Figure 2 is also a BTS, since the AES is a special class of BTS.

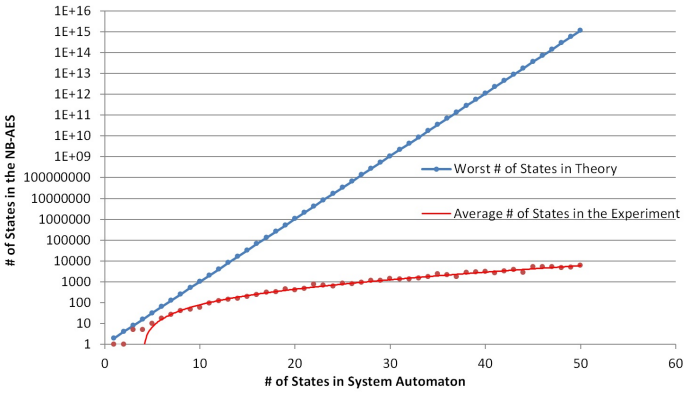

Fig. 3. Average number of states in the NB-AES

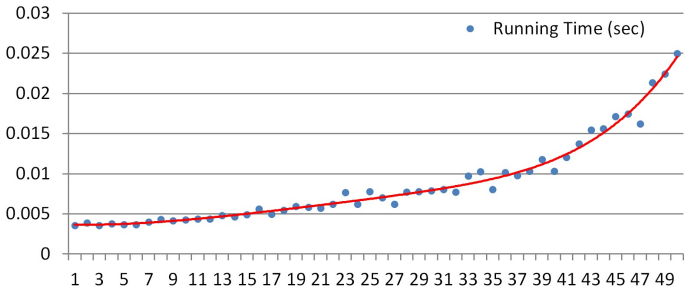

Fig. 4. Average running time of the synthesis algorithm (y-axis) in terms of number of system states (x-axis)

2.3 Scalability

We discuss computational complexity in terms of the number of states of the plant automaton, $|X|$. In the worst case, the resulting BTS or BDO in solving a (control or sensor activation) synthesis problem may contain $2^{|X|}$ states. However, this theoretical upper bound may not be achieved in practice. To better understand this exponential growth and test the limits of the current explicit implementation in DPO-SYNT, we performed scalability tests of DPO-SYNT on an 8 GB Lenovo Laptop with a 2.5-GHz Intel core i7.

We fixed the number of events as $|E| = 10$ and varied the number of states from $|X| = 1$ to $|X| = 50$. We considered the standard non-blocking supervisor synthesis problem for a safety requirement. For each $|X| \in \{1, \ldots, 50\}$, we randomly generated 20 system models using the following parameters: the out-degree of each state is $|X|/2$, 50% of the events are controllable, 50% of the events are observable, 20% of the states are marked states, and 20% of the states are illegal. Then we constructed the NB-AES for each randomly generated automaton and computed the average number of states of the NB-AESs for each $|X|$. Also, we computed the average running time of the entire synthesis algorithm for each $|X|$.

The experimental results are summarized in Figures 3 and 4. Specifically, Figure 3 shows the average number of states in the NB-AES compared with the theoretical upper bound. Figure 4 shows the average running time of the entire synthesis algorithm when the number of states in the system increases. These experimental results show that, in these tests on random plant models, the complexity of the synthesis algorithm is much smaller than its theoretical upper bound. This was due to the fact that for our randomly generated automata, the exponential upper bound when constructing the observer was not achieved. However, we also found that the running time increases significantly when the number of events increases. The current version of DPO-SYNT can handle systems with about 100 states and 20 events. Note that the current implementation of DPO-SYNT does not use symbolic methods. It is well known that using symbolic methods can significantly improve the scalability of reactive synthesis algorithms. Indeed, several DES tools have BDD-based implementations, such as `SUPREMICA`, `TCT`, and the recent supervisor synthesis tool SynthSMV [1] [Rawlings 2016]. Implementing a new version of DPO-SYNT using symbolic methods is an important future direction.

REFERENCES

Åkesson, K., Fabian, M., Flordal, H., and Malik, R. (2006). Supremica - an integrated environment for verification, synthesis and simulation of discrete event systems. In *8th International Workshop on Discrete Event Systems*, 384–385.

Clavijo, L., Basilio, J., and Carvalho, L. (2012). DESLAB: A scientific computing program for analysis and synthesis of discrete-event systems. In *11th International Workshop on Discrete Event Systems*, 349–355.

Feng, L. and Wonham, W. (2006). TCT: A computation tool for supervisory control synthesis. In *8th International Workshop on Discrete Event Systems*, 388–389.

Leduc, R. (2015). DESpot. URL `http://www.cas.mcmaster.ca/~leduc/DESpot.html`.

Moor, T., Schmidt, K., and Perk, S. (2008). libFAUDES an open source C++ library for discrete event systems. In *9th International Workshop on Discrete Event Systems*, 125–130.

Rawlings, B. (2016). *Discrete Dynamics in Chemical Process Control and Automation*. Ph.D. thesis, Carnegie Mellon University.

Ricker, L., Lafortune, S., and Genc, S. (2006). DESUMA: A tool integrating GIDDES and UMDES. In *8th International Workshop on Discrete Event Systems*, 392–393.

Rudie, K. (2006). The integrated discrete-event systems tool. In *8th International Workshop on Discrete Event Systems*, 394–395.

Yin, X. and Lafortune, S. (2015). A general approach for solving dynamic sensor activation problems for a class

[1] `https://bitbucket.org/blakecraw/synthsmv/`

of properties. In *54th IEEE Conference on Decision and Control*, 3610–3615.

Yin, X. and Lafortune, S. (2016a). Synthesis of maximally permissive supervisors for partially observed discrete event systems. *IEEE Transactions on Automatic Control*, 61(5), 1239–1254.

Yin, X. and Lafortune, S. (2016b). A uniform approach for synthesizing property-enforcing supervisors for partially-observed discrete-event systems. *IEEE Transactions on Automatic Control*, 61(8), 2140–2154.

Yin, X. and Lafortune, S. (2017). Synthesis of maximally-permissive supervisors for the range control problem. *IEEE Transactions on Automatic Control*. DOI: 10.1109/TAC.2016.2644867.