Optimal Symbolic Controllers Determinization for BDD storage. ⋆

Ivan S. Zapreev, Cees Verdier, Manuel Mazo Jr. *

* *Center for Systems and Control, Technical University of Delft, The Netherlands (e-mails: I.Zapreev@tudelft.nl, M.Mazo@tudelft.nl, C.F.Verdier@tudelft.nl)*

Abstract: Controller synthesis techniques based on symbolic abstractions appeal by producing correct-by-design controllers, under intricate behavioural constraints. Yet, being relations between abstract states and inputs, such controllers are immense in size, which makes them futile for embedded platforms. Control-synthesis tools such as PESSOA, SCOTS, and CoSyMA tackle the problem by storing controllers as binary decision diagrams (BDDs). However, due to redundantly keeping multiple inputs per-state, the resulting controllers are still too large. In this work, we first show that choosing an optimal controller determinization is an NP-complete problem. Further, we consider the previously known controller determinization technique and discuss its weaknesses. We suggest several new approaches to the problem, based on greedy algorithms, symbolic regression, and (muli-terminal) BDDs. Finally, we empirically compare the techniques and show that some of the new algorithms can produce up to $\approx 85\%$ smaller controllers than those obtained with the previous technique.

Keywords: control law, determinism, embedded systems, data compression, genetic algorithms

1. INTRODUCTION

Controller synthesis techniques based on symbolic models, such as e.g. Tabuada (2009); Rungger et al. (2013); Liu et al. (2013), are becoming increasingly popular. One of the key advantages of these techniques is that they allow for synthesising correct-by-construction controllers of general nonlinear systems under intricate behavioural requirements. However, the downside of the synthesised controllers is their size as, in essence, they are huge tables mapping abstract state-space elements into input-signal values. Even for toy examples, the produced controllers can reach a size of several megabytes. In real-life applications however, they can be several orders of magnitude larger. The latter prohibits them from being used on embedded micro-controllers which typically have very limited memory resources. This state-space explosion is the consequence of: (1) the number of abstract system states and inputs which are exponential in the number of dimensions and inverse-polynomial in the discretisation values; and (2) storing multiple valid input signals per abstract state.

There are numerous tools, implementing or incorporating control synthesis, such as PESSOA, SCOTS, CoSyMA, TuLiP, see Mazo Jr et al. (2010), Rungger and Zamani (2016), Mouelhi et al. (2013), and Wongpiromsarn et al. (2011) correspondingly. Internally, they either use an explicit control law representation in a table form or employ Reduced Ordered Binary Decision Diagrams, introduced by Bryant (1986) and called RO-BDDs or simply BDDs, in an attempt to optimise the memory needed to store the synthesised control law. RO-BDDs are canonical, efficiently manipulable, and in many cases allow for compact data representation. However, their size is strongly dependent on the variables' ordering and the problem of finding an optimal one is known to be NP-complete, as shown by Bollig and Wegener (1996). To fight that issue, tools such as SCOTS and Pessoa use the state of the art RO-BDD library CUDD, see Somenzi (2015), which implements numerous efficient variable ordering optimisation heuristics. Yet for practical applications, synthesised BDD controllers can still easily reach hundreds of megabytes.

To our knowledge, there have been just a few attempts made to find compact but practical representations of (symbolically produced) control laws. Except for using BDDs, we are only aware of another two approaches. The first one, suggested by Staudt (1998), uses piece-wise linear functions, also known as linear in segments (LIS) functions, to approximate control functions of the form $g : \mathbb{R} \to \mathbb{R}$. The approximation is considered for scalar control functions of one argument only. The main motivation for LIS is to reduce the memory footprint of implementing controllers at the cost of some on-line computations, which nonetheless are fast to perform. However, this approach does not directly scale to multiple dimensions or allows to resolve multiple-input's non-determinism.

Another technique to reduce the control-law size, we shall refer to as `LA` (Local Algorithm), was proposed by Girard (2012b). It borrows ideas from algebraic decision diagrams (ADDs), see Bahar et al. (1993), for compact function representation and exploits the non-determinism inherent to safety controllers. The considered controllers are multi-valued maps $g : \mathbb{R}^n \rightrightarrows \mathbb{N}$. The suggested approach attempts to optimise the controller size determinizing the control law by choosing one of the possible control signals for each of the state-space points. In the selection of such unique inputs, `LA` maximizes the size of state-space neighbourhoods employing the same input with the expected outcome of minimizing an ADD representation of the resulting control function. However, the minimality of the ADD representation cannot be guaranteed in general by this approach, which leads us to investigate if better compression approaches may be viable.

In this paper, we first prove that the problem of choosing a size-optimal controller determinization is NP-complete. We do that assuming the BDD controller representation, but the result can be easily generalised. Next, we suggest two new determinization approaches : `GA` (Global Algorithm) - based on a greedy algorithm for the minimum set-cover selection problem, see Karp (1972); `SR` - a hybrid of

⋆ This work is supported by the STW-EW grant as a part of the CADUSY project #13852.

ADD-based and symbolic regression techniques, powered by genetic programming, see Koza (1994); Willis et al. (1997). `GA` attempts to minimise the BDD size by maximising the number of controller states having the same input signal. It differs from `LA` in that, when choosing a common input for a set of states, it looks at the state-space globally, without considering the actual state positions. `SR` (Symbolic Regression) aims at bridging the intrinsic limitations of `LA` and `GA` by using "arbitrary" (polynomial and sigmoid in our case) functions as controller representations. This way we realise the Kolmogorov's Li and Vitnyi (2008) view on data compression [1]. Further, we combine `LA` and `GA` into a hybrid approach called `LGA` (Local-Global Algorithm). The idea here is that the determinization is done as in `LA` but, if multiple common inputs are possible, the preference is given to the one suggested by `GA`. In addition, we consider `B`-prefixed version of `LGA` (`BLGA`) which attempts for a better compression by using BDD variable reordering to produce abstract state indexes.
We perform an empirical evaluation on a number of examples from the literature. Our results show that compression-wise [2] there is no absolute best approach. However, `LGA` seems, on most cases, to be providing the best compression. The `SR` approach, while only providing better compressions in few examples, may be most promising when looking at actual embedded deployments, if it could be pushed to remove any use of BDDs, and their overhead on actual implementations.

2. PRELIMINARIES

2.1 Minimum set cover

The minimum set cover problem (`MSC`) is formulated as:

Problem 2.1. (`MSC`). Given a set X and a cover $\{S_j\}_{j\in I}$, i.e. $X \subseteq \bigcup_{j\in I} S_j$, where $|X|, |I| < \infty$, find the smallest subcover $I^* \subseteq I : X \subseteq \bigcup_{j\in I^*} S_j$.

Both, the decision and selection versions of `MSC`, are known to be NP-complete. The first approximate poly-nomial-time solution for `MSC` was given by Karp (1972).

2.2 Symbolic regression

Symbolic regression is a type of regression analysis that searches for analytical expressions best fitting a given dataset of numerical data, both in terms of accuracy and simplicity. We apply this technique in order to find the smallest analytical expressions best fitting symbolic-model-based control-law functions, ensuring for the smallest control law representation. One of the most popular means for symbolic regression is genetic programming, see Koza (1992) (GP). In this work, similar to Whigham et al. (1995), we employ grammar guided genetic programming algorithms (GGGP) to find multi-dimensional analytical expressions fitting the controller's data. In fact, the genetic process follows Verdier and Mazo (2017) except for that the real-value parameter tuning is done with CMA-ES Hansen and Ostermeier (2001). To speed up the CMA-ES procedure, we use sep-CMA-ES which has a linear time and space complexity Raymond and Nikolaus (2008).

2.3 Binary Decision Diagrams

Binary Decision Diagrams (BDDs), represented with rooted directed acyclic graphs were introduced by Bryant (1986), as a compact representation for boolean functions $F : \{0,1\}^n \to \{0,1\}$. Given F with a list of arguments $\{v_i\}_{i=0}^n$, also called BDD variables or just variables, the BDD of F results from the Shannon expansion thereof. The order of arguments in the signature of F has clearly no impact on F itself, but it has a drastic impact on the size of the resulting BDD. Finding a size-optimal BDD variable ordering was shown, in Bollig and Wegener (1996), to be NP-complete. Yet, there are multiple polynomial heuristics, Scholl et al. (1999), that can find a semi-optimal variable ordering. One of the most popular thereof is sifting, Rudell (1993), and its variants. Given a fixed variable order, each BDD has a canonical minimum-size representation, called Reduced Ordered BDD (RO-BDD). Assuming the bottom-up BDD traversal, an RO-BDD can be obtained by the following poynomial-time algorithm, for more details see Section 4.2 of Bryant (1986):
(1) Combine terminal nodes with equal values
(2) Eliminate nodes with equivalent [3] children
(3) Combine nodes with pairwise equivalent children
Multi Terminal BDDs (MTBDDs) extend BDDs in that tree's terminal nodes allow for arbitrary labels, thus useful to encode functions of the form $F : \{0,1\}^n \to U$, with $|U| < \infty$. The BDD reduction algorithm is naturally extendible towards MTBDD which thus have the canonical RO-MTBDD form. For an (MT)BDD M, we define $R(.)$ as a reduction function producing the RO-(MT)BDD $R(M)$. Algebraic Decision Diagrams (ADDs), introduced by Bahar et al. (1993), are a synonym of MTBDDs. The current state of the art implementation for RO-(MT)BDDs is provided by the CUDD package Somenzi (2015).

3. PROBLEM STATEMENT

Consider a (possibly non-linear) discrete time control system of the form:

$$x(k+1) = f(x(k), u(k)),\ x(k) \in \mathcal{X} \subseteq \mathbb{R}^n,\ u(k) \in \mathcal{U} \subseteq \mathbb{R}^m.$$

Symbolic approaches, see e.g. Tabuada (2009), automatically synthesize controllers in the form of discrete state transition systems. Furthermore, the resulting controllers can often be reduced to a look-up table, see Reissig et al. (2016), prescribing for each point of the state-space a set of applicable inputs guaranteeing that the control specification is satisfied. Such synthesized controllers usually take the form of the combination of a (finite) set-valued map $g : \mathcal{S} \rightrightarrows \mathcal{V}$, and quantization maps $\mathtt{q}_x : \mathcal{X} \to \mathcal{S}$, $\mathtt{q}_u : \mathcal{U} \to \mathcal{V}$ reducing the originally infinite state and input sets to finite sets (usually defining a grid), i.e. $\mathcal{S} \subset \mathcal{X}$, $\mathcal{S} \subset \mathcal{X}$, $|\mathcal{S}| < \infty$, $|\mathcal{V}| < \infty$. Moreover, the usual approach is to quantize each dimension of $\mathcal{X}$ and $\mathcal{U}$ independently, i.e. $\mathtt{q}_x(x) = (\mathtt{q}_x^1(x_1), \ldots, \mathtt{q}_x^n(x_n))$, $\mathtt{q}_u(u) = (\mathtt{q}_u^1(u_1), \ldots, \mathtt{q}_u^n(u_n))$, where each of the $\mathtt{q}_x^i : \mathcal{X}_i \subset \mathbb{R} \to \mathcal{S}_i$, such that $\mathcal{S} = \mathcal{S}_1 \times \ldots \times \mathcal{S}_n$, and similarly for the input quantizer. This results in controller implementations selecting at each time step $u(k) \in g(\mathtt{q}_x(x(k)))$, see for details of such controllers Reissig et al. (2016). Most often, the controllers synthesized do not provide a valid input for some subset $\mathcal{S}_\emptyset \subset \mathcal{S}$. We define the set $\mathcal{S}_c := \mathcal{S} \setminus \mathcal{S}_\emptyset$. We may assume that there is some element $\mathtt{nc} \in \mathcal{V}$ denoting a "no-input", and thus we can define $\mathcal{S}_\emptyset := g^{-1}(\mathtt{nc})$.
A symbolic controller $g \subseteq \mathcal{S} \times \mathcal{V}$, by indexing the countable sets $\mathcal{S}_i$ and $\mathcal{V}_i$, can alternatively be interpreted as a relation $g \subset \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0}$. Consider $\mathcal{B} := \{0,1\}$, and let us define a fixed-length base-2 bit encoding for non-negative integers $bits : \mathcal{K} \to \mathcal{B}^b$ for some $\mathcal{K} \subset \mathbb{Z}_{\geq 0}$, $|\mathcal{K}| < \infty$, and $b := \lceil log_2(max(\mathcal{K})) \rceil$. For $k = (k_1, k_2) \in g \subset \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0}$, mapping the bit vector $(bits(k_1), bits(k_2))$ to a boolean 1 defines a BDD encoding of g. Similarly, one can construct an MTBDD encoding of g by mapping $bits(k_1)$ to k_2.

[1] Instead of storing the control law as an explicit map, we search for a symbolic function that for a given state computes the input value.
[2] Up to the found optimal BDD variable reordering.
[3] "Equivalent" means: Representing the same binary function.

Relating elements of $\mathcal{S}$ or $\mathcal{V}$ with $\mathbb{Z}_{\geq 0}$ can be done with an indexing function, typically defined as:

$$f_b(k_a,\ldots,k_b) := \sum_{i=a}^{b} k_i \cdot \left(\prod_{j=a}^{i-1} 2^{|bits(N_j)|}\right), \text{ or} \quad (1)$$

$$f_s(k_a,\ldots,k_b) := \sum_{i=a}^{b} k_i \cdot \left(\prod_{j=a}^{i-1} N_j\right) \quad (2)$$

Here, $N_j{:=}|\mathcal{S}_j|$ for $j{\in}\overline{1,n}$, and $N_j{:=}|\mathcal{V}_j|$ for $j{\in}\overline{n{+}1,n{+}m}$; $|bits(N_j)|$ is the data-type size needed to enumerate intervals in j. Equations 2 and 1 are both used in SCOTSv2.0. The former is employed in its interface classes (`UniformGrid` and `SymbolicSet`), as it delivers smaller indexes. The latter is used for BDD encoding as it avoids bit sharing between distinct dimension interval indices.

In the present we consider the following minimisation problem aimed at finding the smallest controller determinization of a given controller g:

Problem 3.1. (`OD`). Find the best determinization g^* of a controller g optimizing: $g^* = argmin_{\tilde{g}\in\mathcal{F}} |enc(\tilde{g})|$, where

$$\mathcal{F} := \{\tilde{g} : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0} |$$
$$\forall s \in Dom(g) : ((\tilde{g}(s) \in g(s)) \wedge (|\tilde{g}(s)| = 1))\},$$

$enc(.)$ encodes controllers into RO-(MT)BDDs, and $|.|$ provides the (MT)BDD size.

In theoretical derivations, as in Kwiatkowska et al. (2006), we define $|.|$ to be the number of (MT)BDD nodes. In practice, $|.|$ is the number of bits used to store the (MT)BDD by the CUDD package in the best-found, variable reordering.

Theorem 1. The `OD` problem is NP-complete (NP-C).

Proof. See the proof in Zapreev et al. (2018)

4. LA ON MTBDDS

Girard (2012b) suggests a controller-size minimisation technique, which we call `LA`, that uses ideas from MTBDDs to represent the controller function in the form of a binary tree. The approach does dimension-wise binary splitting of the controller's state-space bounding box. The areas with no-inputs are considered to allow for any input. For the areas with common inputs possible a single input is selected non-deterministically. A branch in the tree represents a state-space area with all states having common inputs (stored in terminal nodes). The determinization aims at choosing single inputs in a way minimising the depth of the tree branches. The latter is equivalent to reductions as in steps (1) and (2) of the RO-BDD construction (c.f. Section 2.3), but not (3). Girard (2012b) showed that `LA` can lead to drastic size reductions, e.g., for "the simple thermal model of a two-room building" example the original controller required 1.000.000 data units, whereas in the tree format it went down to 27. Yet, in its original form this approach: *(i)* does not preserve the controller's domain – neglecting basic data of safe initial states; *(ii)* employs a fixed state-space splitting algorithm – not using controller's structural features; *(iii)* uses simple binary trees which are less efficient than MTBDDs, due to the latter compression abilities by variable reordering and their canonical reduced form. This motivates extending the approach towards MTBDDs.

`LA` can be adapted to quantised state-spaces, since:

(i) For dimension $i{\in}\overline{1,n}$ and $s_i{\in}\mathcal{S}_i$, the bit sequence $bits(s_i)$, defines a binary-tree path to s_i in $\mathcal{S}_i$.

(ii) For $s{\in}\mathcal{S}$, a binary-tree path to s in $\mathcal{S}$, is defined by alternating the bits of sequences $bits(s_1),\ldots,bits(s_n)$.

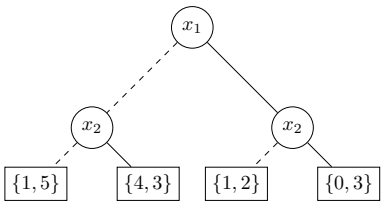


Fig. 1. An example MTBDD

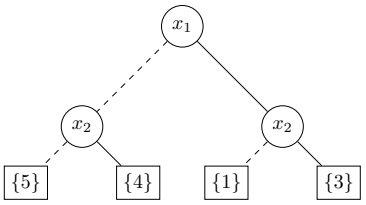
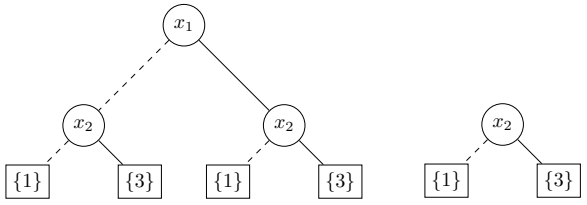


Fig. 2. A non-reducible determinization



(a) After determinization (b) After reduction

Fig. 3. A reducible determinization

The latter, using bounded-length bit sequences as in Section 3, allows to encode the `LA`'s binary tree as an MTBDD. The size reductions obtained for the original `LA` are then a subset of those we get using MTBDDs [4], as we can: *(i)* obtain RO-MTBDDs, utilising all the reduction steps *(ii)* find a more efficient variable ordering. Let us now show that `LA` does not allow to utilise the full power of the MTBDD reductions due to its pure spacial orientation.

Consider an MTBDD encoding of some `LA`'s binary tree, in its original variable ordering, see Figure 1. `LA` traverses an MTBDD trying to find common inputs, stored in terminal nodes, for all of its sub-trees. A sub-tree with a common input can then be trivially reduced to a single terminal node. In this case however, there are no non-trivial sub-trees with common inputs, so `LA` has to *non-deterministically* choose one (arbitrary) input value per terminal node. This results in 16 possible determinization variants, most of which are non-reducible, see e.g. Figure 2, but one giving a significant reduction, see Figure 3.

In this paper, we suggest alternatives and hybrid approaches to overcome this potential shortcoming of `LA`, see Section 5. Furthermore, to preserve information on safe initial states, we shall consider a modification of `LA` which forbids assignment of "any input" to "no-input" grid cells.

5. DETERMINIZATION ALGORITHMS

The newly suggested determinization algorithms have various underlying ideas: `GA` tries to maximise the number of states with the same input, and minimise the number of different inputs as a whole, both in an attempt to maximise the chances for (MT)BDD reductions; `LGA` combines complementary ideas of `LA` and `GA` to reduce the number of non-deterministic choices to be taken in the former one; `SR` attempts to find an analytical expression fitting the controller points on the largest part of its domain to reduce the number of distinct control mode areas to be stored;

5.1 Global Approach

The `GA` approach is summarised by the following steps:

[4] Even with the original variable ordering.

(i) Obtain sets: C – domain state indexes, I – input indexes, and $\{C_j\}_{j\in I}$ – states for the given inputs;

(ii) Solve the `MSC`, see Section 2.1, for $\{C_j\}_{j\in I}$, getting an ordered, more common inputs first, list of indexes I^*;

(iii) Iterate over all $i \in I^*$ and $\forall x \in C_i$, still having input i, remove all other inputs.

`GA` differs from `LA` by looking at the state-space globally regardless of its' elements location. It maximizes the number of terminal nodes with identical labels (inputs), generally leading to a reduction in the number of used labels, which should facilitate MTBDD reductions.

5.2 Local-Global Approach

Recall the MTBDD-based `LA` algorithm discussed in Section 4. We showed that such determinization procedure can suffer from sub-optimal non-deterministic resolutions when multiple input choices are available in some regions. `LGA` combines `LA` with `GA` in an attempt to improve the resulting reductions by minimising this uncertainty. In essence, the `LGA` approach proceeds as `LA` up to the moment a non-trivial set of inputs, common for a grid area, is found; then the input is chosen according to the priority-descending order of inputs, as done in the `LA` algorithm.

5.3 BDD-index based Local-Global Approach

RO-(MT)BDDs achieve significant size reductions only if a "good" variable ordering is found, see Section 2.3. Given the (MT)BDD encoding, see Equation 1 of Section 3, the variable reordering swaps grid-cell index bits realising a limited[5] form of cell re-indexing. The latter has a common-input-value clustering effect on the $g \subset \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0}$ function viewed in RO-BDD indexes as opposed to those of SCOTSv2.0. To use this to our benefit, we suggest a version of `LGA`, called `BLGA`, using the RO-BDD indexes.

5.4 Symbolic Regression

For the `SR` algorithm, a set of candidate controllers is evolved using a combination of GGGP and sep-CMA-ES, *c.f.* references in Section 2.2, using $i_{\max}$ individuals (i.e. candidate solutions) for N generations. GGGP is used to evolve the functional structure of the controller based on a grammar and sep-CMA-ES to optimize the parameters. Given a candidate controller $g_{\texttt{SR}} : \mathbb{R}^n \to \mathbb{R}^m$, the fitness function F with respect to a finite set S is defined as:

$$F(g_{\texttt{SR}}, \mathcal{S}_c) = \frac{|\{s \in \mathcal{S}_c \mid \mathtt{q}_u(g_{\texttt{SR}}(s)) \in g(s)\}|}{|\mathcal{S}_c|}.$$

In order to reduce the computation time, the set of states $\mathcal{S}_c$ is down-sampled to a set with a maximum of λ elements. The reproduction involves selecting individuals based on tournament selection and the genetic operators crossover and mutation, in which parts of the individuals are exchanged or randomly altered respectively. More in depth descriptions of the used GGGP and sep-CMA-ES algorithms can be found in Verdier and Mazo (2017) and Raymond and Nikolaus (2008) respectively. After a maximum amount of generations the individual with the highest fitness is selected. For the resulting controller, it is verified for which states $s \in \mathcal{S}_c$ it holds that $\mathtt{q}_u(g_{\texttt{SR}}(s)) \in g(s)$. For the remaining states the inputs are determinized using `GA`, `LA` or `LGA`. Finally, all states and corresponding new input indexes are again stored in a BDD. Details on the used grammar and `SR` parameters can be found in Zapreev et al. (2018).

[5] Swapping bits affects all indexes; bits can not change value.

6. EMPIRICAL EVALUATION

6.1 Case studies

All of the considered case-studies, but the last one, are taken from the standard distribution of SCOTSv2.0: `Aircraft` - a DC9-30 aircraft landing maneuver, see Reissig et al. (2016); `Vehicle` - a path planning problem for an autonomous vehicle, see Zamani et al. (2012) and Reissig et al. (2016); `DCDC` - a boost DC-DC converter with a reach-and-stay voltage specification, see Girard (2012a); `DCDC rec 1/2` - the same as `DCDC` but enforces a recurrence specification for two targets; `DCM` - a DC motor with a reach-and-stay velocity specification, see Mazo Jr et al. (2010).The symbolic BDD controller sizes were varied by modifying the models' input-/state-space discretisation parameters.

6.2 Software details

For the evaluation, we have realised the following software:

- A C++11 based LibLink library[6] for Mathematica 11, see Inc. (2017), allowing to load and store BDD-based symbolic controllers of SCOTSv2.0.
- A C++11 based single-threaded (due to CUDD) application implementing `LA`, `GA`, `LGA`, and `BLGA`.
- A Mathematica 11 package implementing the `SR` approach. This realisation is natively multi-threaded and allows for a best utilisation of the CPU cores.

6.3 Experimental setup

We have measured: *(i)* determinization run-time in seconds as reported by the tools; *(ii)* size of the determinized controllers in bytes, when stored to the file system. `SR` is probabilistic and therefore each of its experiments was repeated 5 times. All other approaches are deterministic and thus their experiments were repeated only once. Overall, we have considered the algorithms on various size models, varying the discretisation parameters, and thus changing:

(1) The number of model inputs:
   (a) `GA`, `LA`, `LGA`, `BLGA`
   (b) `SR` on `LGA` determinized controllers
(2) The number of model states:
   (a) `GA`, `LA`, `LGA`, `BLGA`

`SR` was only done on `LGA` determinized controllers because it: *(i)* did not scale well with the growing number of inputs; *(ii)* if feasible, shall be capable of reducing deterministic controllers. The experiments were done on two machines: *(A)* MacBook Pro with: Intel i5 CPU (4 cores) 2.9 GHz; 8 GB 2133 Mhz RAM; MacOS v10.12.6; *(B)* PC with: Intel Xeon CPU (8 cores) E5 − 1660 v3 3.00GHz; 32 GB 2133 MHz RAM; Ubuntu 16.04.3 LTS. The type $(1.a)$ experiments ran on machine *(A)*; $(1.b)$ and (2) on (B). Given, a significant difference in software realization (Mathematica v.s. C++11, multi v.s. single threaded), running `SR` on faster multicore machine, and that controllers' determinization is an offline job, our run-time data: *(i)* is only dedicated to show the approaches' feasibility; *(ii)* can only hint the actual performance differences between `SR` and others. This is why also the run-time for `LA`, `GA`, `LGA`, and `BLGA` is not averaged over multiple re-runs.

6.4 Results

Table 1 presents the core experimental data for models obtained by varying the number of inputs. Here, column: "SCOTS" lists information for the original controllers; "Time" is the algorithm's run-time in seconds; "`A-SR`" and "`M-SR`"stand for the average and maximum `SR` values over 5 repetitions; and "Fit %" is the fitness percentage of the `SR` controller's symbolic part. To compare the

[6] We preferred LibLink over WSTP due to faster data-exchange.

Table 1. Core experimental data

	SCOTS		LA		LGA	A-SR	M-SR
	#inputs	#Bytes	#Bytes	Time	#Bytes	Time	Fit %
Aircraft	20	2878481	150459	121.81	150316	1065, 50	48.56
	57	9563407	193590	159.61	193055	1547, 92	43.55
	112	8533274	236753	183.62	235273	1949, 12	40.58
Vehicle	49	21972	10462	1.38	9821	572, 77	32.31
	169	28537	11956	1.79	11047	614, 85	27.15
	441	54692	19430	2.86	17357	770, 81	13.96
	729	52447	18435	3.41	15793	954, 53	18.14
	1087	60757	18939	4.04	16338	1455, 87	24.82
DCM	2001	4951	830	2.04	371	458, 61	33.14
	10001	11957	1000	13.3	420	639, 11	33.14
	20001	24206	1166	34.65	410	742, 10	24.43
	30001	19161	1306	63.45	441	951, 10	33.14
	40001	13772	1308	94.00	449	975, 50	19.82
	50001	12921	1252	143.13	448	1121, 98	33.14
DCDC	2	4532	786	0.75	786	431, 15	94.19
	45	5218	1025	1.57	1025	440, 99	94.19
	89	5350	1030	2.38	1030	351, 11	94.18
	134	5272	1036	3.31	1035	450, 36	94.17
	178	5266	1036	4.11	1035	368, 17	94.15
	223	5300	1037	5.09	1036	426, 01	93.55
DCDC r1	2	4247	773	0.78	773	448, 00	97.35
	45	6009	915	1.48	915	417, 70	97.35
	89	5615	921	2.15	921	422, 11	97.35
	134	5768	936	2.96	930	359, 65	97.35
	178	5781	936	3.64	930	409, 91	97.35
	223	5714	936	4.48	930	428, 49	95.13
DCDC r2	2	2243	791	0.73	828	439, 04	95.08
	45	3685	934	2.15	937	428, 78	94.51
	89	3638	939	3.66	943	395, 83	95.12
	134	3565	949	4.98	949	361, 49	94.99
	178	3531	949	6.43	949	456, 36	94.82
	223	3549	950	8.13	950	408.70	92.91

compressing power of the approaches, for an algorithm ω and a case study ν we define size compression as: C^{ν}_{ω}:=$(1 - |B^{\nu}_{\omega}|/|B^{\nu}|) * 100$, where B^{ν} and B^{ν}_{ω} stand for the original and ω-determinized BDD sizes. Comparing algorithms "ω_1 v.s. ω_2" is done by computing a difference $\Delta^{\nu}_{\omega_1,\omega_2}$:=$C^{\nu}_{\omega_1} - C^{\nu}_{\omega_2}$. Clearly, $\Delta^{\nu}_{\omega_1,\omega_2} > 0$ means ω_1 being better than ω_2 on ν. Taking into account the `A-SR` experiment repetitions, we define [7]: $C^{\nu}_{\texttt{A-SR}} := E\,[C^{\nu}_{\texttt{SR}}]$.

Figure 4 contains two compression comparison sets: *(i)* `GA`, `LGA`, `BLGA` v.s. `LA`; and *(ii)* `A-SR`, `M-SR` v.s. `LGA` [8]. The plot features mean compressions and the standard deviation thereof. We conclude the next compression ranking of the algorithms: *1.* `LGA`, *2.* `BLGA`, *3.* `LA`, *4.* `GA`, *5.* `M-SR` *6.* `A-SR`.

Figure 5 summarises the execution times for the set-up of Table 1. Relative to `LA`, on average: `GA` is ≈ 0.8 times faster; `LGA` is comparable; `BLGA` is ≈ 1.1 times slower; `A-SR` is ≈ 180 times slower but has a huge deviation of ≈ 174. The latter is due to probabilistic nature of `SR`. Note that, `A-SR` is multi-threaded and was run on a faster machine than the single-threaded `LA`. So the actual performance difference between the algorithms is more significant.

Additionally, we compared `GA`, `LGA`, `BLGA` and `LA` on up to 52 Mb size BDD controllers, obtained by varying the number of system states. These experiments only strengthened the algorithms' ranking conclusions implied by Figure 4. We omit further detail on that, to save space.

To conclude, we present Figure 6 summarising the compression of `LGA` relative to `LA` on all of the 67 considered BDD controllers. Per case-study ν the compression is computed as: $C^{\nu}_{\texttt{LGA, LA}}$:=$(1 - |B^{\nu}_{\texttt{LGA}}|/|B^{\nu}_{\texttt{LA}}|) * 100$. The plot on the left of Figure 6 shows the discretized distribution of $C^{\nu}_{\texttt{LGA, LA}}$, the plot on the right shows its mean and standard deviation. Notice that, on average, `LGA` produces $\approx 14\%$ smaller controllers than `LA`, in the best case `LGA` was capable of delivering up to $\approx 85\%$ smaller controllers.

7. CONCLUSIONS

In this work, we have considered the problem of size-optimal BDD controllers determinisation (`OD`), which we show to be NP-complete. Up until now, the only heuristic approach to solve `OD` was proposed by Girard (2012b) and

[7] The mean value over 5 experiment repetitions of `SR` on ν.

[8] Since `SR` was applied to the `LGA`-determinized controllers.

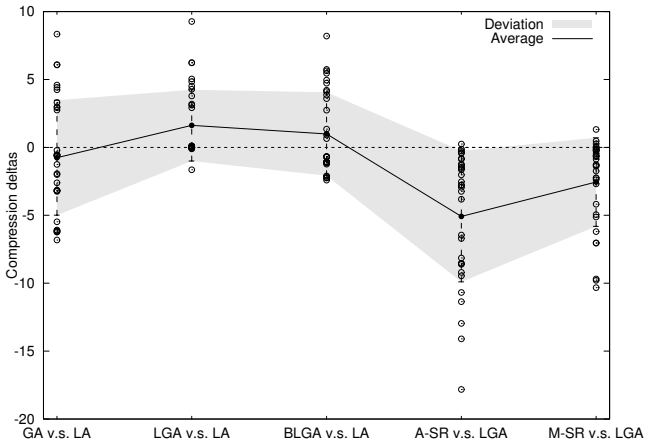


Fig. 4. Plotting $\Delta^{\nu}_{\texttt{GA,LA}}$, $\Delta^{\nu}_{\texttt{LGA,LA}}$, $\Delta^{\nu}_{\texttt{BLGA,LA}}$, $\Delta^{\nu}_{\texttt{A-SR,LGA}}$, $\Delta^{\nu}_{\texttt{M-SR,LGA}}$

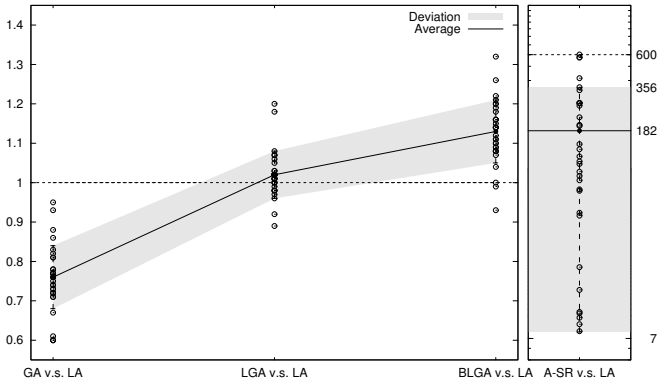


Fig. 5. Execution times ratio relative to `LA`

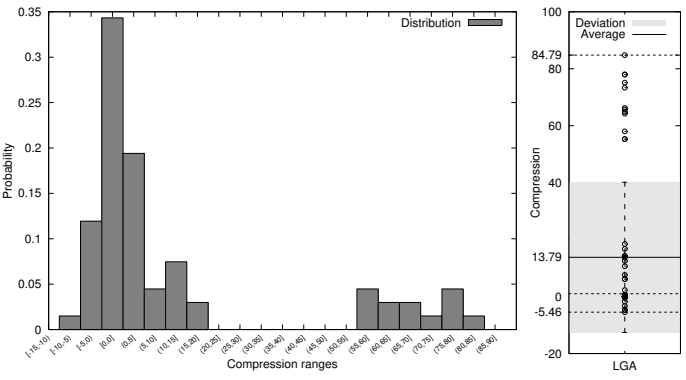


Fig. 6. $C^{\nu}_{\texttt{LGA, LA}}$ distribution, deviation and average

was based on representing the controller function as a binary tree. We have shown how that such an approach, which we call `LA`, can be extended to use the more size-efficient RO-(MT)BDDs data structure. In addition, we have identified examples where `LA` is sub-optimal due to only considering controller's local properties. A global approach (`GA`), based on the minimum set cover problem solution algorithm, was proposed to remedy this. Further, a hybrid of `GA` and `LA`, called `LGA`, was suggested to incorporate the strengths of both approaches. To exploit the clustering of internal BDD indexes, we have come up with a BDD-index based version of `LGA`, called `BLGA`. Finally, we made an attempt of substituting the BDD-based control-law representations by functions generated using the symbolic regression (genetic-algorithm powered) approach, we refer to as `SR`.

All of the devised approaches were compared in compressing power and run-time by means of an extended empirical evaluation. The compression ranking of the algorithms turns out to be: *1.* `LGA`, *2.* `BLGA`, *3.* `LA`, *4.* `GA`, *5.* `SR`. The run times of `LA`, `GA`, `LGA`, and `BLGA` are of the same order but `SR` is at least one to two orders of magnitude slower.
In principle, `SR` could allow us to eliminate BDDs completely, leading to potentially smaller functional expressions and prevent using BDD-data accessing code that, as for CUDD, is difficult (and size expensive) to port to embedded hardware. We did not manage to achieve that due to: *(i)* our `SR` realization not being powerful enough, see low fitness values in Table 1; *(ii)* using BDDs for storing the controller's support, due to a decision to preserve controller's domain. For now, we shall note that `SR` still looks promising for getting small and practical controllers. However, symbolic controllers seem to have structure that is not easy for `SR` to achieve a 100% fitness on. So more research is needed to be done in this direction.

REFERENCES

Bahar, R.I., Frohm, E.A., Gaona, C.M., Hachtel, G.D., Macii, E., Pardo, A., and Somenzi, F. (1993). Algebraic decision diagrams and their applications. In *ICCAD proceedings*, 188–191. IEEE Computer Society Press, Los Alamitos, CA, USA.

Bollig, B. and Wegener, I. (1996). Improving the variable ordering of obdds is np-complete. *IEEE Trans. Comput.*, 45(9), 993–1002.

Bryant, R.E. (1986). Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8), 677–691.

Girard, A. (2012a). Controller synthesis for safety and reachability via approximate bisimulation. *Automatica*, 48(5), 947 – 953.

Girard, A. (2012b). Low-complexity switching controllers for safety using symbolic models*. *IFAC Proceedings Volumes*, 45(9), 82 – 87.

Hansen, N. and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.*, 9(2), 159–195.

Inc., W.R. (2017). Mathematica, Version 11.1. Champaign, IL.

Karp, R.M. (1972). Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher (eds.), *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York.*, The IBM Research Symposia Series, 85–103. Plenum Press, New York.

Koza, J.R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, USA.

Koza, J.R. (1994). Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, 4(2), 87–112.

Kwiatkowska, M., Norman, G., and Parker, D. (2006). *Symmetry Reduction for Probabilistic Model Checking*, 234–248. Springer Berlin Heidelberg, Berlin, Heidelberg.

Li, M. and Vitnyi, P.M. (2008). *An Introduction to Kolmogorov Complexity and Its Applications.* Springer Publishing Company, Incorporated, 3 edition.

Liu, J., Ozay, N., Topcu, U., and Murray, R.M. (2013). Synthesis of reactive switching protocols from temporal logic specifications. *IEEE Trans. Automat. Contr.*, 58(7), 1771–1785.

Mazo Jr, M., Davitian, A., and Tabuada, P. (2010). *PESSOA: A Tool for Embedded Controller Synthesis*, 566–569. Springer Berlin Heidelberg, Berlin, Heidelberg.

Mouelhi, S., Girard, A., and Gössler, G. (2013). Cosyma: A tool for controller synthesis using multi-scale abstractions. In *HSCC proceedings*, 83–88. ACM, New York, NY, USA.

Raymond, R. and Nikolaus, H. (2008). A simple modification in cma-es achieving linear time and space complexity. In *Parallel Problem Solving from Nature – PPSN X: 10th International Conference, Dortmund, Germany, September 13-17, 2008. Proceedings*, 296–305. Springer Berlin Heidelberg, Berlin, Heidelberg.

Reissig, G., Weber, A., and Rungger, M. (2016). Feedback Refinement Relations for the Synthesis of Symbolic Controllers. *IEEE Trans. Automat. Control*, 62(4), 1781–1796.

Rudell, R. (1993). Dynamic variable ordering for ordered binary decision diagrams. In *ICCAD proceedings*, ICCAD '93, 42–47. IEEE Computer Society Press, Los Alamitos, CA, USA.

Rungger, M., Mazo Jr, M., and Tabuada, P. (2013). Specification-guided controller synthesis for linear systems and safe linear-time temporal logic. In *HSCC proceedings*, 333–342. ACM, New York, NY, USA.

Rungger, M. and Zamani, M. (2016). SCOTS: A tool for the synthesis of symbolic controllers. In *HSCC proceedings*, 99–104.

Scholl, C., Becker, B., and Brogle, A. (1999). Solving the multiple variable order problem for binary decision diagrams by use of dynamic reordering techniques. Technical report, Albert-Ludwigs-University, Freiburg, D 79110 Freiburg im Breisgau, Germany.

Somenzi, F. (2015). CUDD: CU Decision Diagram Package. Electronically available at: `http://vlsi.colorado.edu/ fabio/CUDD/`.

Staudt, V. (1998). Compact representation of mathematical functions for control applications by piecewise linear approximations. *Electrical Engineering*, 81(3), 129–134.

Tabuada, P. (2009). *Verification and Control of Hybrid Systems: A Symbolic Approach.* Springer Publishing Company, Incorporated, 1st edition.

Verdier, C. and Mazo, M. (2017). Formal controller synthesis via genetic programming. *IFAC-PapersOnLine.* (To Appear).

Whigham, P.A. et al. (1995). Grammatically-based genetic programming. In *The workshop on genetic programming: from theory to real-world applications*, 33–41.

Willis, M.J., Hiden, H.G., Marenbach, P., McKay, B., and Montague, G.A. (1997). Genetic programming: an introduction and survey of applications. In *International Conference On Genetic Algorithms In Engineering Systems: Innovations And Applications*, 314–319.

Wongpiromsarn, T., Topcu, U., Ozay, N., Xu, H., and Murray, R.M. (2011). Tulip: A software toolbox for receding horizon temporal logic planning. In *HSCC proceedings*, 313–314. ACM, New York, NY, USA.

Zamani, M., Pola, G., Mazo Jr, M., and Tabuada, P. (2012). Symbolic models for nonlinear control systems without stability assumptions. *IEEE Trans. Automat. Contr.*, 57(7), 1804–1809.

Zapreev, I., Verdier, C., and Mazo Jr, M. (2018). Optimal Symbolic Controllers Determinization for BDD storage. An extended arXiv version: `http://arxiv.org/abs/1803.07369`.