Directed Controller Synthesis of Discrete Event Systems: Taming Composition with Heuristics

Daniel Ciolek[1], Victor Braberman[1,3], Nicolás D'Ippolito[1,2,3] and Sebastián Uchitel[1,2,3]

Abstract— **This paper presents a Directed Controller Synthesis (DCS) technique for discrete event systems. This DCS method explores the solution space for reactive controllers guided by a domain-independent heuristic. The heuristic is derived from an efficient abstraction of the environment based on the componentized way in which complex environments are described. Then by building the composition of the components on-the-fly DCS obtains a solution by exploring a reduced portion of the state space. This work focuses on untimed discrete event systems with safety and co-safety (i.e. reachability) goals. An evaluation for the technique is presented comparing it to other well-known approaches to controller synthesis (based on symbolic representation and compositional analyses).**

I. Introduction

Discrete Event Systems (DES) are discrete-state, event-driven dynamic systems that react to the occurrence of events. DES arise in many domains including robotics, logistics, manufacturing, and communication networks. These applications require control and coordination to ensure that the system goals are achieved.

The field of synthesis of controllers for DES was introduced by Ramadge and Wonham [1] for controlling systems within a given set of constraints. In this setting, the environment (also called plant) and goals are specified using a formal language, and a procedure generates a correct by construction controller (or supervisor). The environment is usually modeled with state machines whose event sets are partitioned into controllable and uncontrollable events. A controller must achieve its goals by dynamically disabling some of the controllable events.

We address control problems for behavior models expressed as deterministic Labeled Transition Systems (LTS) and parallel composition ($\|$) defined broadly as a synchronous product. This setting allows to ease the modeling of complex environments by describing its behavior compositionally. However, constructing the complete behavior of the environment via composition produces an exponential explosion. For this reason we want to avoid the computation of the composition ahead of time, building it on-the-fly instead (hopefully obtaining a solution by exploring a reduced portion of the state space).

1. Departamento de Computación, Universidad de Buenos Aires, Argentina
2. Department of Computing, Imperial College, London, UK
3. CONICET

This work waspartially supported by grants ANPCYT PICT 2012-0724, ANPCYT PICT 2013-2341, ANPCYT PICT 2014-1656, ANPCYT PICT 2015-0638, ANPCYT PICT 2015-1718, UBACYT 2014 N0384BA, UBACYT 2014 N0036BA, CONICET PIP 2012 N0596CO, CONICET PIP 2014 N0688CO, CONICET PIP 2015 N0931CO.

Hereinafter we present the Directed Controller Synthesis (DCS) of DES for safety and co-safety (i.e. reachability) goals. Informally, we want controllers to *always* avoid safety violations while *eventually* ensuring co-safety objectives. DCS explores the solution space for reactive controllers guided by a domain-independent heuristic. The main contribution of this paper is the heuristic derived from an abstraction of the environment that exploits the componentized way in which complex environments are described. We highlight that componentization not only simplifies modeling, but also allows for preprocessing procedures that can improve the applicability of controller synthesis techniques.

II. Related Work

Traditional controller synthesis techniques can be classified in various ways. In Table I we show a classification of software tools found in the literature that are related to DCS.

Monolithic approaches to controller synthesis work with the complete state space but, since it is exponential with respect to the size of the components, explicit representations are impractical. For this reason, Binary Decision Diagrams (BDD) [6] are usually used to keep a symbolic representation of the state space, in an attempt to cope with the state explosion problem.

Contrarily to monolithic approaches, compositional approaches to controller synthesis build a controller for each component, such that when composed achieve the system requirements. Although this allows to greatly reduce the impact of the state explosion problem it also restricts its application to simple goals, such as safety, since it is not easy to guarantee composability for richer goals.

Techniques that deal only with safety objectives look for maximal (i.e. least restrictive) controllers [7], since safety properties can many times be trivially satisfied by restricting the system to an unproductive zone. Maximality comes at a cost in complexity given that it requires to exhaustively explore the state space. Non-maximal approaches can perform a narrower on-the-fly exploration, but this requires richer goals such as co-safety and liveness to direct the search.

Tool	1	2	3	4	5	6
CIRCA [2]	✗	✗	✓	✗	✗	✓
DCS	✗	✗	✓	✗	✓	✗
MBP [3]	✗	✓	✗	✓	✓	✗
MTSA [4]	✗	✗	✗	✓	✓	✗
Supremica [5]	✓	✗	✗	✗	✓	✗

TABLE I
Tools classification: (1) compositional (2) symbolic (3) on-the-fly (4) liveness (5) component-based (6) timed

Approaches based on a timed automaton model can take advantage of the additional information provided by times in order to cut the exploration. However, they are not applicable to problems without meaningful time information.

Non-maximal solutions are the standard in the area of Automated Planning, a branch of artificial intelligence. In planning the state space is usually represented explicitly, and the exploration is performed on-the-fly guided by heuristics [8]. Informed search procedures – like A* [9] – are used to perform a goal-directed exploration, generally obtaining a solution by inspecting a reduced portion of the state space. However, in planning the input is not based on components and it has been oriented mainly towards nonreactive environments, which are insufficient in the setting in which controller synthesis is applied.

Inspired by planning, informed search procedures have been introduced in model checking to accelerate the search for an error [10], [11]. Despite the positive results of these techniques, their application to controller synthesis has been scarcely explored [12]. In this paper we present an extension to our previous work on the Modal Transition System Analyser (MTSA) [4] that now includes the DCS algorithm, which explores the state space on-the-fly focusing on safety and co-safety goals for nonmaximal controllers.

III. Background

Definition 1 (Labeled Transition Systems). A *Labeled Transition System* (LTS) is a tuple $E = (S, A, \rightarrow, s_0)$, where S is a finite set of states, A is its alphabet, $\rightarrow \subseteq (S \times A \times S)$ is a transition relation, and $s_0 \in S$ is the initial state.

Notation 1. Let E and M be LTSs such that:

- $E = (S_E, A_E, \rightarrow_E, s_0)$, with s and s' states of S_E
- $M = (S_M, A_M, \rightarrow_M, t_0)$, with t and t' states of S_M

Notation 2 (Step). We denote $(s, \ell, s') \in \rightarrow$ by $s \xrightarrow{\ell} s'$.

Definition 2 (Parallel Composition). The *Parallel Composition* ($\|$) is a symmetric operator such that it yields an LTS $E\|M = (S_E \times S_M, A_E \cup A_M, \rightarrow_{E\|M}, \langle s_0, t_0\rangle)$, where $\rightarrow_{E\|M}$ is a relation that satisfies the following rules:

$$\frac{s \xrightarrow{\ell}_E s'}{\langle s,t\rangle \xrightarrow{\ell}_{E\|M} \langle s',t\rangle}\ {\scriptstyle \ell\in(A_E\setminus A_M)} \qquad \frac{t \xrightarrow{\ell}_M t'}{\langle s,t\rangle \xrightarrow{\ell}_{E\|M} \langle s,t'\rangle}\ {\scriptstyle \ell\in(A_M\setminus A_E)}$$

$$\frac{s \xrightarrow{\ell}_E s', t \xrightarrow{\ell}_M t'}{\langle s,t\rangle \xrightarrow{\ell}_{E\|M} \langle s',t'\rangle}\ {\scriptstyle \ell\in(A_E\cap A_M)}$$

Given a partition of the alphabet in controllable and uncontrollable actions, a controller M must achieve its goal G by disabling some of the controllable actions in E. We say that a controller M satisfies a goal G in an environment E if and only if every trace accepted by $E\|M$ satisfies G.

Definition 3 (Trace). A trace of an LTS E is a sequence of labels $\pi = \ell_0, \ell_1 \dots$ of A_E, for which there exists a sequence of states $s_0, s_1, \dots$ of S_E such that $\forall i \geq 0 \,.\, s_i \xrightarrow{\ell_i}_E s_{i+1}$.

Definition 4 (Safety). We say that a trace π satisfies a *safety* goal G_S given as a set of labels, if and only if a label of G_S never occurs in π.

Definition 5 (Co-Safety). We say that a trace π satisfies a *co-safety* goal G_C given as a set of labels, if and only if there is at least one occurrence of a label of G_C in π.

Notation 3. States of the composition are ordered pairs, however we may need to refer to the *step* relation for any ordering. For this purpose we may use the following notation:

$$\{s,t\} \xrightarrow{\ell}_{E\|M} \{s',t'\} \Leftrightarrow \langle s,t\rangle \xrightarrow{\ell}_{E\|M} \langle s',t'\rangle \vee \langle s,t\rangle \xrightarrow{\ell}_{E\|M} \langle t',s'\rangle \vee \langle t,s\rangle \xrightarrow{\ell}_{E\|M} \langle s',t'\rangle \vee \langle t,s\rangle \xrightarrow{\ell}_{E\|M} \langle t',s'\rangle$$

When using the set notation we relax the convention that s and s' belong to E while t and t' belong to M since the order of the states is made irrelevant.

Notation 4. For brevity we denote that a set of states $s_0, s_1, \dots, s_n$ belongs to a set q, that is to say that $\forall i \,.\, 0 \leq i \leq n \Rightarrow s_i \in q$, as $[s_1, s_2, \dots, s_n]_q$.

IV. On-the-fly Exploration

In this section we discuss the main algorithm that constitutes DCS. Given a heuristic function that estimates the distance from a state to a goal, DCS looks for a controller by computing the parallel composition on-the-fly guided by the heuristic. The procedure is a modification of Best First Search (a classical informed search procedure) adapted to account for uncontrollable events. We keep a priority queue of *open* states, ordered by their estimated distance to a goal, initialized only with the initial state. At each iteration the algorithm takes the most promising state from the queue and expands a child state following its best unexplored event. The expanded state is evaluated using the heuristic, and it is then inserted in the *open* queue (the value of the state is set as the estimate of its best unexplored event).

For states containing only controllable events we need only one successful successor, while for states containing only uncontrollable events all their successors have to be able to reach a goal. Thus we allow a fully controllable state to remain in the *open* queue as long as it has unexplored events to broaden the search (i.e. competing with its descendants). Mixed states – with both controllable and uncontrollable events – are treated as in the uncontrollable case. This is because mixed states represent a race condition between the controller and the environment, which can always be won by the environment. Thereby the validity of the controller cannot depend on the result of race conditions. This treatment of controllable and uncontrollable nodes is similar to AND/OR search procedures [13].

When a state is recognized either as a co-safety goal or a safety error, it is marked and this information is propagated back to its ancestors until the propagation is interrupted and a state reopened. The propagation of an error is interrupted when a controllable ancestor with unexplored events is reached. Whereas, the propagation of a goal is interrupted when an uncontrollable ancestor with unexplored events is reached. The process is repeated until the initial state is marked as a goal or an error. In the former the states reachable from the initial state form a controller, while in the latter there is no controller.

Since DCS explores controllable events in the order given by the heuristic. When an event reaches a goal it excludes from the exploration the less promising controllable events available from the same state. Therefore, the controller obtained is not maximal, since it has at most one controllable event enabled from each state. In [14] these kind of controllers are referred as directors.

Observe that while there is an obvious advantage in exploring the most promising controllable event first, there is no such advantage in exploring the best uncontrollable event since all such events must lead to a goal. Thus we could avoid the computation of the heuristic for fully uncontrollable states. Instead we use the reverse ranking and expand the least desirable uncontrollable event first. We do this in an attempt to find an error as fast as possible and consequently being able to close the state avoiding futile exploration.

In the worst case, the heuristic misguides the search and the algorithm explores the complete state space. With the additional cost of computing the heuristic for each state, the complexity of the algorithm is worse than its monolithic counterpart. That is, DCS could take an exponential amount of time. In spite of that, if the heuristic accurately guides the exploration with a small computational overhead, great savings could be obtained.

V. Abstraction and Heuristic

In this section we present the abstraction we use and the heuristic it induces. The main goal of the heuristic is to provide an estimate of the distance from a state to a goal without computing the parallel composition. However, in order to provide informative estimates the effects of synchronization need to be taken into account. For this reason we build an abstraction that, once it reaches a point where a state is covered, it never drops a state. The abstraction behaves as if, after a transition, not only a new state is covered but also the source state is not left. This progressively distances the abstraction from the real environment (as it collects states), yet it keeps information about the causal relationship between events. We then use the length of the paths that reach a co-safety goal in the abstraction as an estimate of the real distance to the goal.

A. LTS Abstraction and Composition

We begin by defining the *LTS abstraction* for a single LTS, that yields an LTS whose reachable region forms *a sequence of sets of states*.

Definition 6 (LTS Abstraction)**.** The abstraction of an LTS E is an LTS $\tilde{E} = (2^{S_E}, 2^{A_E}, \rightsquigarrow_E, \{s_0\})$, where $\rightsquigarrow_E$ is the relation that satisfies:

$$\begin{aligned} q\overset{\mathcal{A}}{\rightsquigarrow}_E q' \Leftrightarrow &\big(\forall s, s', a \,.\, [s]_q \wedge s\xrightarrow{a}_E s' \Rightarrow [s, s']_{q'} \wedge a{\in}\mathcal{A}\big)\wedge \\ &\big(\forall s' \,.\, [s']_{q'} \Rightarrow [s']_q \vee \exists s, a \,.\, [s]_q \wedge a{\in}\mathcal{A} \wedge s\xrightarrow{a}_E s'\big) \end{aligned}$$

Observe that in the abstraction the states are sets of the original states, and also the labels are sets of the original labels. Furthermore, there is only one set of events available from each set of states, that is, the maximal set of enabled events. At first glance considering the power set of states may seem the worst case, but the abstraction applies a rule of monotonic growth. That is, the reachable sets of states are restricted to sets that contain all the traversed states from the initial state. Thus the abstraction size is polynomial with respect to the number of states in the components.

We now define the *abstracting composition*, an operation for composing these abstracted LTSs. Informally, this operation works by applying a relaxed synchronization rule that follows that of ($\|$), but considering sets of states.

Definition 7 (Abstracting Composition)**.** The *Abstracting Composition* ($\mathbin{\wr\wr}$) is a symmetric operator in the spirit of ($\|$) that yields an abstract LTS $\tilde{E}\mathbin{\wr\wr}\tilde{M} = (S_{\tilde{E}} \cup S_{\tilde{M}}, A_{\tilde{E}} \cup A_{\tilde{M}}, \rightsquigarrow_{\tilde{E}\wr\wr\tilde{M}}, \tilde{s}_0 \cup \tilde{t}_0\})$ and $\rightsquigarrow_{\tilde{E}\wr\wr\tilde{M}}$ is the relation that satisfies:

$$\begin{aligned} &q \overset{A}{\rightsquigarrow}_{\tilde{E}\wr\wr\tilde{M}} q' \Leftrightarrow \\ &\quad\big(\forall s,s',t,t',a \,.\, [s,t]_q \wedge \langle s,t\rangle \xrightarrow{a}_{E\|M} \langle s',t'\rangle \Rightarrow [s',s,t',t]_{q'} \wedge a{\in}A\big)\wedge \\ &\quad\big(\forall s' \,.\, [s']_{q'} \Rightarrow [s']_q \vee \exists s,t,t',a \,.\, [s,t]_q \wedge a{\in}A \wedge \{s,t\} \xrightarrow{a}_{E\|M} \{s',t'\}\big) \end{aligned}$$

Note that we do not need to compute $E\|M$ since we only check locally that states s and t synchronize following the rules in Def.2.

Property 1. Given LTSs E and M the number of sets of states reachable in $\tilde{E}\mathbin{\wr\wr}\tilde{M}$ is, in worst case, $|S_E| + |S_M|$.

Proof. The proof is immediate given that once every state in $S_E \cup S_M$ is included in a set of states q, no other set of states can be reached. Thus, in the worst case each step contributes only one fresh state, leading to $|S_E|+|S_M|$ steps before reaching the last possible fresh state. $\square$

Despite looking very dissimilar there is a strong relation between a model and its abstraction. In particular if π is a trace of $E\|M$, π is a path contained in $\tilde{E}\mathbin{\wr\wr}\tilde{M}$.

Definition 8 (Contained Path)**.** We say that a sequence of labels $\pi{=}\ell_0, \ell_1 \ldots$ is a *contained path* in $\tilde{E}\mathbin{\wr\wr}\tilde{M}$ if and only if there is a trace $\Pi{=}L_0, L_1 \ldots$ of $\tilde{E}\mathbin{\wr\wr}\tilde{M}$ such that $\forall i \,.\, \ell_i \in L_i$.

We may abuse notation and denote that for a label ℓ_i there exists a set L_i such that $\ell_i \in L_i \wedge q \overset{L_i}{\rightsquigarrow}_{\tilde{E}\wr\wr\tilde{M}} q'$ using directly ℓ_i as the label of the transition as follows: $q \overset{\ell_i}{\rightsquigarrow}_{\tilde{E}\wr\wr\tilde{M}} q'$.

Note that by Def. 7 every trace of $E\|M$ is a path contained in $\tilde{E}\mathbin{\wr\wr}\tilde{M}$. However, not every contained path can be mapped to a trace in $E\|M$. Therefore after computing the abstraction some paths will relate to traces while others will not, yet a priori we have no way to distinguish between these cases. We can easily filter some paths that are obviously uninteresting. For this purpose we define the *abstracting paths* to more precisely capture the notion of path that will help us estimate the distance to a goal.

Definition 9 (Abstracting Path)**.** We say that a sequence of labels $\ell_0, \ell_1, \ldots$ is an *abstracting path* of $\tilde{E}\mathbin{\wr\wr}\tilde{M}$ if and only if there is a sequence of states $s_0, s_1, \ldots$ of $S_E \cup S_M$ and a sequence of sets of states $q_0, q_1, \ldots$ of $S_{\tilde{E}\wr\wr\tilde{M}}$ such that:

$$\begin{aligned} &\forall i \,.\, ([s_i]_{q_i} \wedge q_i \overset{\ell_i}{\rightsquigarrow}_{\tilde{E}\wr\wr\tilde{M}} q_{i+1} \wedge \exists t,t' \,.\, \{s_i, t\} \xrightarrow{\ell_i}_{E\|M} \{s_{i+1}, t'\})\vee \\ &\quad(s_i{=}s_{i+1} \wedge \ell_i{=}\tau) \end{aligned}$$

This definition allows not only for valid intra-LTS steps, but it also allows for inter-LTS steps as long as they are potential synchronization steps. Inter-LTS steps are important since an isolated component may not be able to reach a goal, but it may be necessary for an intermediate synchronization step. Thus, they allow for more accurate estimates. Observe that we add a special event τ to "delay" a step from a state $[s_i]_{q_i}$ to events available from $[s_i]_{q_{i+1}}$. This allows to consider paths that skip necessary synchronization steps, while still considering them in the estimation. This definition filters the paths that are not solely constituted by these intra-LTS and inter-LTS steps. However, it remains that not every abstracting path is a trace, yet we use the length of these paths as an estimate of the distance to a co-safety goal.

Property 2. Given LTSs E and M every trace π of $E \| M$ is an *abstracting path* of $\tilde{E} \between \tilde{M}$.

Proof. The proof is straightforward, since given that $\pi = \ell_0, \ell_1, \ldots$ is a trace of $E \| M$ there exists a sequence of states:

$$\langle s_0, t_0 \rangle \xrightarrow{\ell_0}_{E\|M} \langle s_1, t_1 \rangle \xrightarrow{\ell_1}_{E\|M} \cdots$$

Then by Def. 7 there exists a sequence of sets of states $q_0, q_1, \ldots$ such that $\forall i \,.\, [s_i, t_i]_{q_i}$ and

$$q_0 \overset{\ell_0}{\rightsquigarrow}_{\tilde{E} \between \tilde{M}} q_1 \overset{\ell_1}{\rightsquigarrow}_{\tilde{E} \between \tilde{M}} \cdots$$

Therefore by Def. 9 π is an *abstracting path* of $\tilde{E} \between \tilde{M}$. □

We compute the estimate simply by taking the length of an *abstracting path* that reaches a co-safety goal. Paths that reach a safety violation are considered to have an infinite (∞) distance to the goal. Interestingly, this can never overestimate the distance to the goal, but it can underestimate it. Heuristics that do not overestimate the distance to the goal are called admissible and have been studied in the literature since they enjoy useful properties (e.g. early error detection).

Lemma 1 (Admissibility)**.** The length of an *abstracting path* that reaches a co-safety goal is an admissible heuristic, that is, it does not overestimate the distance to the goal.

Proof. In the worst case a path π reaching a goal is a trace of the environment, thus its length estimates exactly the distance to the goal. In any other case π is not a valid sequence of events of the environment, that is to say is an artifact of the weakened composition rules of the abstraction. Hence π may skip some necessary intermediate steps required to actually reach the goal in the environment. Therefore, the length of π underestimates the distance to the goal. □

B. Heuristic Computation

In this section we show how we build the abstraction and compute the heuristic estimates for the events available from a given state. For computing the heuristic we need all the *abstracting paths* reaching a goal or an error from an initial state. For this we build a graph whose paths are all such *abstracting paths*, which is naturally induced from Def. 9. However, instead of explicitly representing the special event τ, we represent the "distance" between states by a positive integer that we call *generation*.

Definition 10 (Generation)**.** Given an abstracted LTS $\tilde{E}$ we define the *generation* of a state s of $S_{\tilde{E}}$ as the index g of the *sequence of sets of states* of $\tilde{E}$ in which the state s appears for the first time, denoted s^g. More formally,

$$s^g \Leftrightarrow g = \max_i \{\, i \mid \forall j \,.\, 0 \le j < i \Rightarrow \exists \ell_j \,.\, q_j \overset{\ell_j}{\rightsquigarrow}_{\tilde{E}} q_{j+1} \wedge s \notin q_j \}$$

The *generation* is useful since given a transition from a state s_i into s_j with generations n and m respectively, we can deduce the "distance" between them by subtracting $|m - n|$.

Definition 11 (Abstracting Path Graph)**.** Given LTSs E and M, the *Abstracting Path Graph* of $\tilde{E} \between \tilde{M}$ is a graph $\mathcal{G}(\tilde{E} \between \tilde{M}) = (\mathcal{V}, \mathcal{E}, gen)$ where:

- $\mathcal{V} = S_E \cup S_M$, is a set of vertices formed by states
- $\mathcal{E} = \{(s_i, \ell, s_j) \mid \exists q, q', t, t' \,.\, [s_i, t]_q \wedge q \overset{\ell}{\rightsquigarrow}_{\tilde{E} \between \tilde{M}} q' \wedge \{s_i, t\} \xrightarrow{\ell}_{E\|M} \{s_j, t'\}\}$, is a set of labeled edges
- $gen : V \to \mathbb{N}$, is a function mapping states to their respective *generations*

Informally, two vertices of the graph are connected by an edge if and only if there is a step, connecting the corresponding states, that is part of an *abstracting path*.

In Fig. 1 we present a procedure that given a state $\langle s_{E_0}, \ldots, s_{E_n} \rangle$ of the composition $E_0 \| \ldots \| E_n$, and sets of events desired to *reach* and to *avoid*, it returns the information required to compute a heuristic ranking:

1) *errors*, a set of states that do not conduce to a goal.
2) *goals*, a set of transitions connected by a *reach* event.
3) *edges* of the *abstracting path graph*.
4) *generations* of the states in the graph.

The procedure works by iteratively building the *sequence of sets of states* produced by the abstraction (Def. 7) and, at the same time, the *abstracting path graph* (Def. 11). At each iteration a new set q_g is considered, where g indicates the current *generation*. We then compute the set of *ready* transitions, that is, the steps available from q_g following the relaxed synchronization rule from Def. 7. Observe that once a transition is processed it is never considered again. Also note that states are considered errors until they show potential to reach a goal (i.e. deadlock states are treated as errors). Since the procedure processes each transition once its complexity is $\Theta(|\rightsquigarrow_{\tilde{E}_0 \between \ldots \between \tilde{E}_n}|)$, which considering the weakened synchronization rules, can be bigger than the actual transitions in the individual components. In the worst case, the complexity cannot surpass the connection of every state by every event, that is to say that its upper-bound complexity is $O\big((\sum_{i=0}^{n} |A_{E_i}|)(\sum_{i=0}^{n} |S_{E_i}|)^2\big)$.

Once we have built the *abstracting path graph*, obtaining a ranking of the events enabled from the state $\langle s_A, \ldots, s_N \rangle$ is straightforward. The states reached by the *goals* transitions are set with an estimated distance to a goal of 0, while the states in the *errors* set are set with an ∞ estimated distance. Then, we propagate these estimates from the states to its parents following the *edges* backwards. For each step we increase the estimated distance between parent and child according to the information stored in the *generations* map. For a state with multiple children we keep as an estimate the

```
buildAbstraction( $\langle s_{E_0}, \dots, s_{E_n} \rangle, reach, avoid$ )
  $errors = q_0 = \bigcup_{i=0}^{n} s_i$
  $goals = processed = edges = generations = \emptyset$
  for each $s \in q_0$ do $generations[s] = 0$
  $ready = \{ (s, \ell, s') \mid \exists q'.\ q_0 \overset{\ell}{\rightsquigarrow}_{\tilde{E}_0 \parallel \dots \parallel \tilde{E}_n} q' \wedge (s, \ell, s') \notin edges \}$
  $g = 1$
  while $ready \neq \emptyset$ do
    for each $(s, \ell, s') \in ready$ do
      $errors = errors \setminus \{s\}$
      $processed = processed \cup \{s\}$
      $edges = edges \cup \{(s, \ell, s')\}$
      if $\ell \in reach$ then
        $goals = goals \cup \{(s, \ell, s')\}$
      else if $s' \notin processed$ then
        $errors = errors \cup \{s'\}$
        $generations[s'] = g$
        if $\ell \notin avoid$ then
          $q_{g+1} = q_g \cup \{s'\}$
    $ready = \{ (s, \ell, s') \mid \exists q'.\ q_{g+1} \overset{\ell}{\rightsquigarrow}_{\tilde{E}_0 \parallel \dots \parallel \tilde{E}_n} q' \wedge (s, \ell, s') \notin edges \}$
    $g = g + 1$
  return $\langle errors, goals, edges, generations \rangle$
```

Fig. 1. LTS abstracting composition building procedure

minimum of its children's, since we are interested in reaching the goal as fast as possible. When this back-propagation ends, all the enabled events have an estimated distance to the goal, thus it is just a matter of sorting them with respect to these values. However, the same event could have multiple estimated values since we allow not only intra-LTS transitions but also inter-LTS transitions; in these cases we keep the best (minimum) estimate.

Summarizing, the procedure computes – through the construction of the graph – the length of the *abstracting paths* starting at a given initial state and reaching a co-safety goal. The algorithm concentrates on the best estimates, which are used to guide the on-the-fly exploration. Thus, it effectively computes an admissible heuristic, since it can never overestimate the distance to the goal (Lemma 1).

VI. Evaluation

In this section we report on an evaluation of DCS. We compare DCS with three other approaches running on an Intel i7-3770 with 8GB of RAM:

1) Monolithic explicit state representation, previously implemented in MTSA[1] (similar to CTCT [15]).
2) Monolithic symbolic state representation using BDD, implemented in MBP [3].
3) Compositional explicit state representation implemented in Supremica (SUP) [5].

The aim of the evaluation is to assess the gains in scalability with the use of the informed search procedure. For this reason, we want case studies with the capability to scale up to higher complexities. Fortunately, both MBP and Supremica come bundled with the specification for one of the most traditional examples in controller synthesis: Transfer Line (TL); which was first introduced by Wonham [15]. The fact that the models were written by the tools authors' is important because it reduces the impact of a threat to validity that would be an ill-designed model. However, in order to be able to scale the problem up we had to extend the models, yet we maintain the same structure.

Given the complexity of: writing identical specifications in the different formal languages, finding examples that can be scaled up and presenting the results; we opt for concentrating in varying three independent parameters for the TL. This gives us a big enough number of environments with different topologies to evaluate the techniques. Still, this represents another threat to validity, since ideally the techniques should be compared with a more diverse set of inputs.

The TL consists of series of machines $M_1, M_2, \dots, M_n$ connected by buffers $B_1, B_2, \dots, B_n$ and ending in a special machine called Test Unit (TU). A machine M_i takes work pieces from the buffer B_{i-1} (with the exception of machine M_1 that takes the work pieces from the outside). After an undetermined amount of time, the working machine M_i outputs a processed work piece through buffer B_i. Finally, when a work piece reaches the TU it can be accepted and taken out of the system or it can be rejected and placed back in buffer B_1 for reprocessing. The only controllable events in this case study are the taking of work pieces. An error ensues if a machine tries to take a work piece from an empty buffer or if it tries to place a processed work piece in a full buffer. One of the goals for the controller is to *avoid* the events that lead to errors, the other goal is to *reach* a state where a processed work piece can be accepted or rejected. We do not require the controller to achieve accepted pieces as acceptance and rejection are not decided by the controller.

The case study can be scaled in three directions:

1) $Machines$ (M): number of interconnected machines.
2) $Workload$ (W): maximum number of work pieces a machine can process simultaneously.
3) $Capacity$ (C): capacity of the buffers.

In Table II we consider some "small scale" cases, in which we consider all the combinations of the parameters for the following values: M in $[4, 5, 6]$, W in $[1, 2, 3]$, C in $[1, 2, 3]$. We report the time in seconds required by each tool to synthesize a controller with a Time Out (TO) of 30 minutes.

As it can be seen in the table, the number of states in the environment grows very quickly and it soon takes the tools to their limits. Nonetheless, DCS is able to solve all the problems in less than one second. MTSA soon runs Out of Memory (OM), while MBP, despite also working with a full representation of the environment, solves many of the large instances. Whereas Supremica outperforms the other tools in the simpler cases but falls short on the complex ones.

In a second experiment we assess the boundaries of the technique using "big scale" cases shown in Fig. 2. We remove the other techniques from the evaluation since they cannot tackle these problems. For ease of presentation we report on a subset of the combinations of the parameters where W and C take the same values. Note that W and C affect the number of states in the Machine and Buffer LTSs respectively, while M relates to the number of components.

[1]Available at https://bitbucket.org/dciolek/mtsa

M	W	C	States	MTSA	DCS	MBP	SUP
4	1	1	$1.5e^4$	1.5	0.01	0.07	**0.01**
4	1	2	$3.5e^4$	5.5	0.02	0.21	**0.01**
4	1	3	$7.2e^4$	29.8	0.03	0.41	**0.03**
4	2	1	$7.7e^4$	9.1	0.01	0.11	**0.01**
4	2	2	$1.8e^5$	74.5	**0.01**	0.86	0.04
4	2	3	$3.6e^5$	OM	**0.01**	5.95	0.18
4	3	1	$2.3e^5$	OM	**0.01**	0.08	0.02
4	3	2	$5.6e^5$	OM	**0.01**	0.87	0.03
4	3	3	$2.2e^6$	OM	**0.02**	11.58	0.45
5	1	1	$1.2e^5$	33.3	0.01	0.14	**0.01**
5	1	2	$3.5e^5$	OM	**0.02**	1.43	0.09
5	1	3	$8.6e^5$	OM	**0.05**	3.53	0.72
5	2	1	$8.8e^5$	OM	**0.01**	0.25	0.09
5	2	2	$2.6e^6$	OM	**0.02**	12.27	0.62
5	2	3	$6.7e^6$	OM	**0.03**	118	62
5	3	1	$3.7e^6$	OM	**0.01**	0.26	0.22
5	3	2	$1.1e^7$	OM	**0.02**	13.28	0.43
5	3	3	$2.8e^7$	OM	**0.06**	263.72	163
6	1	1	$9.7e^5$	OM	0.22	0.47	**0.09**
6	1	2	$3.5e^6$	OM	**0.18**	14.66	0.77
6	1	3	$1.0e^7$	OM	**0.21**	37.91	TO
6	2	1	$1.1e^7$	OM	**0.02**	1.75	0.13
6	2	2	$4.0e^7$	OM	**0.07**	200	79
6	2	3	$1.2e^8$	OM	**0.25**	1397	TO
6	3	1	$5.9e^7$	OM	**0.03**	1.23	0.17
6	3	2	$2.2e^8$	OM	**0.04**	228	177
6	3	3	$6.7e^8$	OM	**0.22**	TO	TO

TABLE II
SMALL SCALE TRANSFER LINE RESULTS

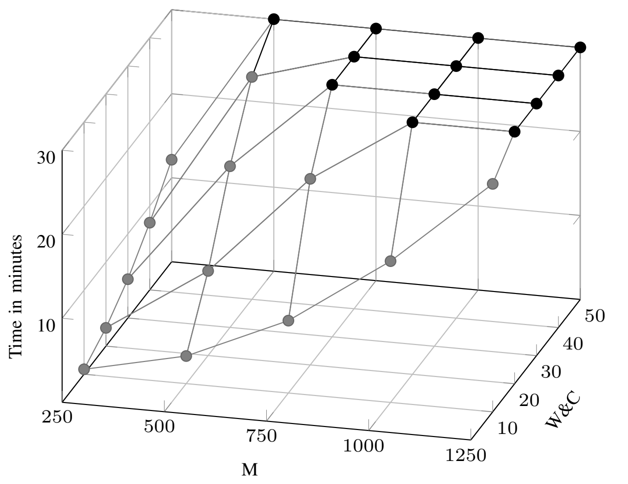

Fig. 2. Big scale TL results (TOs are depicted with black dots)

Thus, the evaluation still assess the performance as the number and complexity of components increase independently.

Summarizing, we have evaluated DCS against other approaches to controller synthesis. DCS is promising given that it is able to cope with increasing complexities beyond the capabilities of similar tools. The largest case tackled by DCS considers an environment with $8.9e^{2734}$ states (solved in $1728s$) generating a controller with 2503 states.

VII. Conclusions and future work

In this paper we present DCS, a method that looks for a controller exploring the state space on-the-fly guided by a domain-independent heuristic. The method performs better than other approaches both in time and memory for the selected case study, despite its worst case complexity. The approach forgoes maximality since reachability goals allow for a directed search in a reduced portion of the state space.

The risk of DCS is that it could fail to properly guide the exploration, triggering an unnecessary large exploration of the state space. Thus, despite our efforts to keep the approach as general as possible, further experiments are required in order to properly asses the generality of the technique. For this reason we plan to work on automatic translations to simplify the comparison with other tools.

These preliminary results show the potential of DCS and hence we intend to extend the technique. Our primary concern is to deal with general liveness goals, since the domain of application of controller synthesis usually have these kind of requirements.

We highlight that a compact representation of the problem is essential for extracting useful guidance for the informed search procedure. Thus, when dealing with DES, elements such as components and synchronization provide vital information to take into account. Furthermore, even when the *LTS abstraction* achieves remarkable results, there are other heuristics that could be used (instead or in combination) that could perform better. The field of directed controller synthesis is almost uncharted, and we believe it can lead to a leap in the applicability of controller synthesis techniques.

References

[1] P. J. Ramadge and W. M. Wonham, "Supervisory Control of a Class of Discrete Event Processes," *SIAM Journal on Control and Optimization*, vol. 25, 1987.

[2] R. P. Goldman, D. J. Musliner, and M. J. S. Pelican, "Exploiting Implicit Representations in Timed Automaton Verification for Controller Synthesis," in *Proc. 5th Int. Workshop of Hybrid Systems: Computation and Control*, HSCC, 2002.

[3] A. Gromyko, M. Pistore, and P. Traverso, "A Tool for Controller Synthesis via Symbolic Model Checking," in *Proceeding of the 8th Int. Workshop on Discrete Event Systems*, 2006.

[4] N. D'Ippolito, D. Fischbein, M. Chechik, and S. Uchitel, "MTSA: The Modal Transition System Analyser," in *Proc. 23rd IEEE/ACM Int. Conf. on Automated Software Engineering*, ASE, 2008.

[5] S. Mohajerani, R. Malik, S. Ware, and M. Fabian, "Compositional synthesis of discrete event systems using synthesis abstraction," in *Control and Decision Conf.*, CCDC, 2011.

[6] S. B. Akers, "Binary Decision Diagrams," *IEEE Transactions in Computers*, vol. 27, 1978.

[7] Wonham, W. Murray and Ramadge, Peter J., "On the Supremal Controllable Sublanguage of a Given Language," *SIAM Journal on Control and Optimization*, vol. 25, no. 3, 1987.

[8] B. Bonet and H. Geffner, "Planning as Heuristic Search," *Artificial Intelligence*, vol. 129, 2001.

[9] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *SIGART Bulletin*, no. 37, 1972.

[10] S. Edelkamp, A. Lluch-Lafuente, and S. Leue, "Directed Explicit Model Checking with HSF-SPIN," in *SPIN*, vol. 2057 of *Lecture Notes in Comp. Sci.*, 2001.

[11] S. Kupferschmid, J. Hoffmann, H. Dierks, and G. Behrmann, "Adapting an AI Planning Heuristic for Directed Model Checking," in *Model Checking Software*, vol. 3925 of *Lecture Notes in Comp. Sci.*, 2006.

[12] V. Alimguzhin, F. Mari, I. Melatti, I. Salvo, and E. Tronci, "On-the-Fly Control Software Synthesis," in *Model Checking Software*, vol. 7976 of *Lecture Notes in Comp. Sci.*, 2013.

[13] R. Simon and R. C. T. Lee, "On the Optimal Solutions to AND/OR Series-Parallel Graphs," *Journal of the ACM*, vol. 18, 1971.

[14] J. Huang and R. Kumar, "Directed Control of Discrete Event Systems for Safety and Nonblocking," *IEEE Trans. Automation Science & Engineering*, vol. 5, 2008.

[15] W. Wonham, "Notes on Control of Discrete-Event Systems." Dep. of Electrical and Comp. Engineering, University of Toronto, 1999.