Formal Synthesis from Control Programs

Vladimir Sinyakov, Antoine Girard

Formal Synthesis from Control Programs

Vladimir Sinyakov and Antoine Girard

***Abstract*— We introduce a new way of specifying rich behaviors for discrete-time dynamical systems called control programs. Essentially, a control program consists of a set of elementary control tasks with a scheduler. A control task is described by a discrete-time hybrid automaton and a termination semantics, specifying if the task must terminate in finite time or if it is allowed to run forever. The scheduler provides a set of rules that is used to sequence the control tasks. Control programs also have external inputs, which makes it possible to specify how a system must react to instructions provided by a human user or by another system. We define the set of executions that are accepted by the control program. Then, we consider the problem of synthesizing a controller for a dynamical system such that the closed-loop behavior is an execution of the control program. Building on our recent work on formal synthesis from specifications given by hybrid automata, we propose two algorithms for computing controllers based on contracting and expanding fixed-point computations. The first algorithm computes the maximal controllable set but needs to reach the fixed-point to provide a valid controller. The second algorithm may not converge to the maximal controllable set but provides a valid controller at each iteration. We illustrate our methodology with an autonomous vehicle control example.**

I. Introduction

The theory of hybrid dynamical systems has been developing during the past decades motivated, in part, by their ability to represent cyber-physical systems: models of digital devices interacting with "continuous" physical world. The formal methods approach suggests considering these dynamical systems as *transition systems* which are defined by a transition relation, describing how states evolve under the effect of inputs (see e.g. [26]). The *specification* describes the desired behavior of the system, as a set of trajectories (i.e. sequences of states and inputs), satisfying a certain criterion. Behavior specification languages that describe rich spacial and temporal properties of transition systems include, in particular, finite state automata [8], [20], [9], Linear Temporal Logic formulas, Büchi and Rabin automata [27], [6], [13], [2]. The specification may also be given by another transition system: the closed-loop system is then required to be related in some sense to this specification system (e.g. [26], [19], [29], [15]). In our previous work [25], we required the closed-loop system to *alternatingly simulate* the specification system given as a discrete-time hybrid automaton, with the possibility to consider an additional control objective specifying whether a set of terminal states should be reached in finite time (reachability termination semantics) or if the system can run forever (safety termination semantics). In both cases, non-terminal blocking states should be avoided.

One of the prospective approaches in formal methods is called symbolic control: continuous components (e.g. defined by differential or difference equations) are abstracted over a finite set of states (symbols), each symbol corresponding to infinitely many continuous states. The resulting discrete dynamical system is called *symbolic abstraction* and can be used for controller synthesis (see e.g. [21], [30], [4], [16], [12], [22]). In [25], a symbolic control approach was proposed to synthesize controllers from specifications given as discrete-time hybrid automata.

In this paper, we present a new specification language for complex behaviors, which we call *control programs*, and provide algorithms for control synthesis from these specifications.

In Section III, we define control programs consisting of a set of elementary *control tasks* with a *scheduler*. Each control task, in turn, is described by a discrete-time hybrid automaton with reachability or safety termination semantics. Such control problems given by control tasks were investigated in [25]. The scheduler provides a set of rules that is used to sequence the control tasks. An important feature of control programs and control tasks is the presence of *external inputs*: these input parameters affect the dynamics of the specification systems and represent interactions with human users or other unmodeled systems. This two-level construction, which describes the desired behavior of the system, has similarities with the formalism of maneuver automata [7], [11], [5]. However, our formalism is richer in the sense that it makes it possible to use different termination semantics for tasks and to define more complex scheduling policies. One may observe that control programs define specifications beyond reachability and safety properties on discrete-time hybrid autotomata, which were considered in [25]. The detailed analysis of expressivity of control programs and how it relates to other specification languages is beyond the scope of this paper where we focus on the introduction of control programs and algorithmic aspects of synthesis from control program specification.

This is the topic of Section IV, where we introduce the notion of *program controller* that essentially enforces the closed-loop system to produce trajectories accepted by the considered control program. In particular, it solves the control problem of each task, guarantees the possibility of transition between tasks according to the scheduler, and ensures the satisfaction of the termination condition for the

V. Sinyakov and A. Girard are with Université Paris-Saclay, CNRS, CentraleSupélec, Laboratoire des signaux et systèmes, 91190, Gif-sur-Yvette, France. e-mail: firstname.lastname@l2s.centralesupelec.fr

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 725144).

program. Program controllers follow the structure of the corresponding control program as they may be automatically constructed from a set of task controllers if a certain *schedulability condition* is satisfied (see Theorem 1). This structure potentially allows one to utilize different algorithms for reachability and safety computations for each individual task. We then present algorithms for synthesizing program controllers: the first one computes the maximal controllable set and the corresponding controller via contracting fixed-point iteration of controllable set candidates while the second one is an anytime algorithm which produces an expanding sequence of controllable sets where the computations may be stopped at each iteration.

Finally, in Section V we present an illustrative example for our framework: an autonomous vehicle control problem where the controllable car is able to either follow or takeover the preceding vehicle and the accepted behaviors are those where the car follows or takes over indefinitely while avoiding collisions with other vehicles.

Notations: $\mathbb{N} = \{0, 1, 2, \dots, \}$ is the set of natural numbers; $|A|$ is the cardinality of a set A; for a set $A \subseteq X_1 \times \cdots \times X_m$, the projection onto X_i $(1 \leq i \leq m)$ is a set $\mathrm{proj}_{X_i} A = \{x_i \mid \exists x = (x_1, \dots, x_m) \in A,\ x_j \in X_j,\ 1 \leq j \leq m\}$.

II. Preliminaries on transition systems

We consider transition systems, which allow us to model in a unique framework continuous or hybrid systems as well as their discrete abstractions (see e.g. [26], [2]):

Definition 1: A *transition system* S is a tuple (X, U, Y, Δ, H), where X is a set of states; U is a set of inputs; Y is a set of outputs; $\Delta : X \times U \rightrightarrows X$ is a set-valued transition map; $H : X \to Y$ is an output map.

An input $u \in U$ is *enabled* in a state $x \in X$, denoted $u \in \mathrm{enab}_\Delta(x)$, if and only if $\Delta(x, u) \neq \emptyset$. If $\mathrm{enab}_\Delta(x) = \emptyset$ then x is called a *blocking state*, otherwise it is a *non-blocking state*. The set of non-blocking states is denoted nbs_Δ. S is said to be *deterministic* if for all $x \in \mathrm{nbs}_\Delta$, for all $u \in \mathrm{enab}_\Delta(x)$, $|\Delta(x, u)| = 1$.

Definition 2: A sequence $(x_k, u_k)_{k=0}^{k=K}$, where $K \in \mathbb{N} \cup \{\infty\}$, $x_k \in X$, $u_k \in U$, for $0 \leq k \leq K$, is called a *trajectory* of S if and only if $x_{k+1} \in \Delta(x_k, u_k)$, for $0 \leq k < K$.

A trajectory is *maximal* if $K = \infty$ or else if $\Delta(x_K, u_K) = \emptyset$. A trajectory is *complete* if $K = \infty$. The set of trajectories of S is called the *behavior* of S, denoted $\mathcal{B}(S)$.

III. Control programs

In this section, we present a formalism for specifying the intended behavior of a system. We introduce the notion of control programs, which form a high-level language, making it possible to specify a rich set of behaviors. Control programs have themselves inputs to enable interaction with external users and systems. Intuitively, a control program consists of a set of elementary control tasks, with a scheduler describing how these tasks must be sequenced. The semantics of the control tasks is very similar to that of discrete-time hybrid automata, enriched with two types of termination semantics. While such elementary control tasks were thoroughly studied in our previous paper [25], the focus of this paper is on the definition of more complex control programs from control tasks and on the formal synthesis of controllers implementing these programs.

A. Control tasks

Control tasks are elementary components that are used to define a control program. They are formally specified as follows:

Definition 3: A *control task* is a tuple

$$\mathbb{T} = (X, P, V, G, W_f, s)$$

consisting of

- a system state space X;
- a finite set of modes P;
- a finite set of external inputs V;
- a transition relation G: $X \times P \times V \rightrightarrows X \times P$;
- a terminal set $W_f \subseteq X \times P \times V$;
- a task semantics $s \in \{$"reachability", "safety"$\}$.

We define a transition system $S_\mathbb{T}$ associated to task $\mathbb{T}$ as follows:

$$S_\mathbb{T} = (X \times P, V, X, G, \mathrm{proj}_X).$$

To avoid confusion we will call *task state space* $Z = X \times P$ as opposed to *system state space* X.

Control tasks are associated to sets of executions:

Definition 4: An *execution* of $\mathbb{T}$ is a triple of state, mode and input sequences $(x_k, p_k, v_k)_{k=0}^{k=K}$, where $K \in \mathbb{N} \cup \{\infty\}$ such that:

1) $(x_k, p_k, v_k)_{k=0}^{k=K} \in \mathcal{B}(S_\mathbb{T})$;
2) one of the following condition holds:
   - if $s =$ "reachability", then $K \in \mathbb{N}$ and $(x_K, p_K, v_K) \in W_f$;
   - if $s =$ "safety", then either $K \in \mathbb{N}$ and $(x_K, p_K, v_K) \in W_f$, or $K = \infty$.

The set of executions of $\mathbb{T}$ is called the *language* of $\mathbb{T}$ and is denoted $\mathcal{L}(\mathbb{T})$. Two tasks having the same language are called *equivalent*.

The notion of a control task is reminiscent of that of a (discrete-time) hybrid automaton [10], [14], [24], [18] where the transition relation G captures the invariants, flows, guards and resets simultaneously. The external inputs make it possible to specify how the system should react to instructions received from a human user or from another system. Compared to hybrid automata, control tasks additionally make it possible to specify a terminal set and two distinct semantics for termination. In the reachability semantics, it is expected that the execution of the control task reaches the terminal set in finite-time. In the safety semantics, the execution may additionally last an infinite amount of time. Thus, intuitively, control tasks allow us to specify behaviors that can be formulated as safety or reachability properties on hybrid automata [28], [1].

B. From control tasks to control programs

Control programs allow us to orchestrate a set of control tasks and can be defined inductively as follows:

Definition 5: A *control program* $\mathbb{P}$ is one of the following:

- $\mathbb{P} = \mathbb{T}$ where $\mathbb{T} = (X, P, V, G, W_f, s)$ is a control task. Then, $\mathbb{P}$ has system state space X, set of modes P, set of external inputs V, and terminal set W_f.
- $\mathbb{P} = (\mathcal{P}, \mathcal{R}, W_{f,0})$ where
  - $\mathcal{P} = \{\mathbb{P}_i,\ i \in L\}$ is a finite set of control programs with the same system state space X, sets of modes P_i (with $P_i \cap P_j = \emptyset$ for $i \neq j$), sets of external inputs V_i, and terminal sets $W_{f,i} \subseteq X \times P_i \times V_i$, $i \in L$.
  - $\mathcal{R} = \{R_{i,j} :\ X \times P_i \times V_i \rightrightarrows P_j \times V_j,\ i, j \in L\}$ is a *scheduler*;
  - $W_{f,0} \subseteq \cup_{i \in L} W_{f,i}$ is a terminal set.

  $\mathbb{P}$ has system state space X, set of modes $P = \cup_{i \in L} P_i$, set of external inputs $V = \cup_{i \in L} V_i$ and terminal set $W_{f,0}$.

Executions of a control program $\mathbb{P}$, defined in the second item of Definition 5 consist in a concatenation of executions of control subprograms in $\mathcal{P}$. The scheduler $\mathcal{R}$ defines the transitions between subprograms.

We define the maps $R^*_{i,j} : X \times P_i \times V_i \rightrightarrows P_j \times V_j$, $i, j \in L$, which result from a finite sequence of scheduling actions: $(p', v') \in R^*_{i,j}(x, p, v)$ if and only if there exists a finite sequence $(i_k, p_k, v_k)_{k=0}^{k=l}$ with $l \geq 1$, $i_k \in L$, $p_k \in P$, $v_k \in V$ such that $(i_0, p_0, v_0) = (i, p, v)$, $(i_l, p_l, v_l) = (j, p', v')$ and $(p_{k+1}, v_{k+1}) \in R_{i_k, i_{k+1}}(x, p_k, v_k)$, for all $k \in \{0, \ldots, l-1\}$. We formulate the following assumption:

Assumption 1: For all $i, j \in L$, the map $R^*_{i,j}$ satisfies:

1) for all $(x, p, v) \notin W_{f,i}$, $R^*_{i,j}(x, p, v) = \emptyset$;
2) for all $(x, p, v) \in W_{f,i}$, $\big(\{x\} \times R^*_{i,j}(x, p, v)\big) \cap W_{f,0} = \emptyset$.

The first item of the assumption states that scheduling actions can only occur upon termination of a subprogam. The second item states that it is not possible to reach the terminal set directly through scheduling actions.

Remark 1: If $\big(\{x\} \times R_{i,j}(x, p, v)\big) \cap W_{f,j} = \emptyset$ for all $(x, p, v) \in W_{f,i}$ then $R^*_{i,j} = R_{i,j}$.

The set of executions of $\mathbb{P}$ is called the *language* of $\mathbb{P}$, denoted $\mathcal{L}(\mathbb{P})$, and is formally defined as follows:

Definition 6: Let $\mathbb{P}$ be a control program:

- If $\mathbb{P} = \mathbb{T}$, then $\mathcal{L}(\mathbb{P}) = \mathcal{L}(\mathbb{T})$.
- If $\mathbb{P} = (\mathcal{P}, \mathcal{R}, W_{f,0})$, an *execution* of $\mathbb{P}$ is a triple of state, mode and input sequences $(x_k, p_k, v_k)_{k=0}^{k=K}$ where $K \in \mathbb{N} \cup \{\infty\}$, and such that there exists a sequence $(i_j, K_j)_{j=0}^{j=J}$ where $i_j \in L$, $K_j \in \mathbb{N}$, $J \in \mathbb{N} \cup \{\infty\}$ such that:
  1) $(x_k, p_k, v_k)_{k=0}^{k=K_0} \in \mathcal{L}(\mathbb{P}_{i_0})$;
  2) if $J > 0$, for all $j \in \{0, \ldots, J-1\}$, $K_j < K_{j+1}$ and there exists $(x_k, \overline{p}_k, \overline{v}_k)_{k=K_j}^{k=K_{j+1}} \in \mathcal{L}(\mathbb{P}_{i_{j+1}})$, such that for all $K_j + 1 \leq k \leq K_{j+1}$, $p_k = \overline{p}_k$, $v_k = \overline{v}_k$ and $(\overline{p}_{K_j}, \overline{v}_{K_j}) \in R^*_{i_j, i_{j+1}}(x_{K_j}, p_{K_j}, v_{K_j})$;
  3) the following conditions hold:
     - if $J \in \mathbb{N}$ then $K_J = K$;
     - if $K \in \mathbb{N}$ then $(x_K, p_K, v_K) \in W_{f,0}$.

Similar to control tasks, we say that control programs are *equivalent* if they have the same language.

We define the transition system $S_{\mathbb{P}}$ associated to a program $\mathbb{P}$ as $S_{\mathbb{T}}$ if $\mathbb{P} = \mathbb{T}$ or otherwise as

$$S_{\mathbb{P}} = (X \times P, V, X, G_{\mathbb{P}}, \mathrm{proj}_X)$$

where for all $x \in X$, $i \in L$, $p \in P_i$, $G_{\mathbb{P}}(x, p, v) = \emptyset$ if $v \notin V_i$, or if $v \in V_i$:

$$G_{\mathbb{P}}(x, p, v) = G_{\mathbb{P}_i}(x, p, v) \cup \Big(\bigcup_{j \in L} G_{\mathbb{P}_j}(\{x\} \times R^*_{i,j}(x, p, v)) \Big).$$

From Definition 6, we have that $\mathcal{L}(\mathbb{P}) \subseteq \mathcal{B}(S_{\mathbb{P}})$.

Definition 7: A control program is in *expanded form* if $\mathbb{P} = (\mathcal{P}, \mathcal{R}, W_{f,0})$ where all elements of $\mathcal{P}$ are control tasks.

Proposition 1: Any control program $\mathbb{P}$ has an equivalent control program $\mathbb{P}'$ in expanded form.

Proof: If $\mathbb{P} = \mathbb{T}$ where $\mathbb{T} = (X, P, V, G, W_f, s)$ is a control task, then one can define the control program $\mathbb{P}' = (\{\mathbb{T}\}, \mathcal{R}, W_f)$ where $\mathcal{R} = \{R\}$, $R(x, p, v) = \emptyset$ for all $x \in X$, $p \in P$, $v \in V$. $\mathbb{P}'$ is in expanded form and from Definition 6, it follows that $\mathcal{L}(\mathbb{P}') = \mathcal{L}(\mathbb{P})$.

If $\mathbb{P} = (\mathcal{P}, \mathcal{R}, W_{f,0})$ where $\mathcal{P} = \{\mathbb{P}_i,\ i \in L\}$, we reason inductively on the structure of the program. Let us assume that for all $i \in L$, $\mathbb{P}_i$ has an equivalent control program $\mathbb{P}'_i$ in expanded form. Let $\mathbb{P}'_i = (\mathcal{P}_i, \mathcal{R}_i, W_{f,i})$, where $\mathcal{P}_i = \{\mathbb{T}_{i'}, i' \in L_i\}$ and $\mathcal{R}_i = \{R^i_{i',j'},\ i', j' \in L_i\}$. Note that all elements of $\mathcal{P}_i$ are control tasks. Then, let $L' = \bigcup_{i \in L} L_i$ and let us define $\mathbb{P}' = (\mathcal{P}', \mathcal{R}', W_{f,0})$ where $\mathcal{P}' = \{\mathbb{T}_{i'}, i' \in L'\}$, and $\mathcal{R}' = \{R'_{i',j'},\ i', j' \in L'\}$ where the map $R'_{i',j'}$ is defined as follows for $i', j' \in L'$. Let $i, j \in L$ such that $i' \in L_i$ and $j' \in L_j$, let $(x, p, v) \in X \times P_{i'} \times V_{i'}$. If $i = j$ then let $R'_{i',j'}(x, p, v) = R^i_{i',j'}(x, p, v) \cup \big(R_{i,j}(x, p, v) \cap (P_{j'} \times V_{j'})\big)$. If $i \neq j$ then let $R'_{i',j'}(x, p, v) = R_{i,j}(x, p, v) \cap (P_{j'} \times V_{j'})$. All elements of $\mathcal{P}'$ are control tasks so $\mathbb{P}'$ is in expanded form. Moreover, it follows from Definition 6 that $\mathcal{L}(\mathbb{P}') = \mathcal{L}(\mathbb{P})$. ∎

Let us remark that the proof of Proposition 1 is constructive and allows us for any control program $\mathbb{P}$ to compute inductively an equivalent control program $\mathbb{P}'$ in expanded form. $\mathbb{P}'$ is called the *expansion* of $\mathbb{P}$. It is also important to note that if Assumption 1 is satisfied for $\mathbb{P}$ and for all programs $\mathbb{P}'_i$, $i \in L$, then it is also satisfied for the expansion $\mathbb{P}'$.

Control programs in expanded form describe how control tasks must be sequenced. It has therefore some similarities with the notion of maneuver automaton [7], [11], which is used to describe how a set of elementary motion primitives can be sequenced. More closely related is the work of [5], where the motion primitives are specified by hybrid automata. However, in general, maneuver automata do not allow the choice of the next maneuver to depend on how the previous maneuver terminates, which is possible for control programs using the scheduler $\mathcal{R}$ and the user inputs v. Moreover, maneuver automata assumes that all maneuver terminates in finite time, which excludes the use of control tasks with

safety semantics. Finally, contrarily to control programs, maneuver automata are not defined inductively.

IV. Synthesis from control programs

In this section, we formalize the synthesis problem. Let us consider a system $S = (X, U, Y, F, H)$ and a control program $\mathbb{P}$ with set of modes P, set of inputs V and terminal set $W_{f,0} \subseteq X \times P \times V$, specifying the intended behavior of S. In the following, we use the notation $v \in W_{f,0}(x,p)$, for $(x,p,v) \in W_{f,0}$.

A. Problem formulation

We consider a *program controller* given by a pair of set-valued maps θ: $X \times P \times V \rightrightarrows U$ and π: $X \times P \times X \times V \rightrightarrows P$. Controller (θ, π) is said to be *compatible* with S if for all $x \in X$, $p \in P$, $v \in V$,

$$\theta(x,p,v) \subseteq \mathrm{enab}_F(x) \text{ and}$$
$$\forall x' \in F(x, \theta(x,p,v)),\ \pi(x,p,x',v) \neq \emptyset.$$

Then, the dynamics of the closed-loop system S_{cl} is given by:

$$\left\{ \begin{array}{lcl} x_{k+1} & \in & F(x_k, \theta(x_k,p_k,v_k)), \\ p_{k+1} & \in & \pi(x_k,p_k,x_{k+1},v_k). \end{array} \right. \qquad (S_{cl})$$

and formally described by the transition system

$$S_{cl} = (X \times P, V, X, \Delta_{cl}, \mathrm{proj}_X)$$

where for all $x \in X$, $p \in P$, $v \in V$,

$$\Delta_{cl}(x,p,v) = \left\{ (x',p') \;\middle|\; \begin{array}{l} x' \in F(x,\theta(x,p,v)) \\ p' \in \pi(x,p,x',v) \end{array} \right\}.$$

Let us remark that if (θ, π) is compatible with S then $v \in \mathrm{enab}_{\Delta_{cl}}(x,p)$ if and only if $\theta(x,p,v) \neq \emptyset$.

The synthesis of controllers from control programs can be formalized as follows:

Problem 1: Synthesize a program controller (θ, π) compatible with S and a controllable set $Z_c \subseteq X \times P$ such that for every $(x_0,p_0) \in Z_c$, for every maximal trajectory $(x_k,p_k,v_k)_{k=0}^K$ of S_{cl}, one of the following condition holds:

1) $(x_k,p_k,v_k)_{k=0}^K \in \mathcal{L}(\mathbb{P})$;
2) $(x_k,p_k,v_k)_{k=0}^K$ is a maximal trajectory of $S_{\mathbb{P}}$, $K \in \mathbb{N}$ and $W_{f,0}(x_K,p_K) \cup \mathrm{enab}_{G_{\mathbb{P}}}(x_K,p_K) \neq \emptyset$.

If Problem 1 is solved, then it follows that for every maximal trajectory $(x_k,p_k,v_k)_{k=0}^K$ of S_{cl}, where for all $0 \leq k \leq K$, $v_k \in W_{f,0}(x_k,p_k) \cup \mathrm{enab}_{G_{\mathbb{P}}}(x_k,p_k)$, we have $(x_k,p_k,v_k)_{k=0}^K \in \mathcal{L}(\mathbb{P})$. In other words, it means that if at all times, the external inputs v_k belong to the inputs enabled by the control program $\mathbb{P}$, then the closed-loop system produces a trajectory which is accepted by the program.

We call a program controller for $\mathbb{P}$ *maximal* if the corresponding controllable set Z_c is maximal by inclusion, i.e. if Z_c' is a controllable set for some other program controller then $Z_c' \subseteq Z_c$. We say that a pair (θ, π) is a *task controller* $\mathbb{T}$, if it solves Problem 1 for $\mathbb{P} = \mathbb{T}$. Let us remark that the synthesis of task controllers can be done using a symbolic control approach as presented in our previous work [25]. Therefore, in the following, we will assume that we have a method to synthesize task controllers and we show how these controllers can be glued in order to obtain program controllers.

B. Schedulability condition

In this section, we provide a sufficient condition for the synthesis of program controllers:

Theorem 1: Consider a program $\mathbb{P} = (\mathcal{P}, \mathcal{R}, W_{f,0})$. Suppose we have program controllers (θ_i, π_i) and sets $Z_{c,i}$ solving Problem 1 for programs $\mathbb{P}_i$, $i \in L$. Let us assume that the following schedulability condition holds for all $(x,p,v) \in \cup_{i \in L} W_{f,i} \setminus W_{f,0}$:

$$(\cup_{j \in L} \{x\} \times R^*_{i,j}(x,p,v)) \cap \mathrm{dom}(\theta_j) \neq \emptyset. \qquad (1)$$

Let $Z_c = \bigcup_{i \in L} Z_{c,i}$ and (θ, π) be given for $(x,p,v) \in X \times P \times V$, $x' \in X$ by

- if $(x,p,v) \in \mathrm{dom}(\theta_i)$

$$\theta(x,p,v) = \theta_i(x,p,v)$$
$$\pi(x,p,x',v) = \pi_i(x,p,x',v)$$

- else, if $(x,p,v) \in W_{f,i} \setminus W_{f,0}$

$$\theta(x,p,v) = \theta_j(x,\overline{p},\overline{v})$$
$$\pi(x,p,x',v) = \pi_j(x,\overline{p},x',\overline{v})$$

  where $(x,\overline{p},\overline{v}) \in (\{x\} \times R^*_{i,j}(x,p,v)) \cap \mathrm{dom}(\theta_j)$
- else, $\theta(x,p,v) = \emptyset$ and $\pi(x,p,x',v) = \emptyset$.

Then, (θ, π) and Z_c solve Problem 1 for program $\mathbb{P}$.

Proof. The fact that (θ, π) is compatible with S is a consequence of (θ_i, π_i) being compatible with S. Then, consider a maximal trajectory $(x_k,p_k,v_k)_{k=0}^K$ of S_{cl} with $(x_0,p_0) \in Z_c$. By construction of (θ, π), we get that if $K = \infty$ then $(x_k,p_k,v_k)_{k=0}^K \in \mathcal{L}(\mathbb{P})$. Let us now assume that $K \in \mathbb{N}$. Then from the definitions of (θ, π) and of $G_{\mathbb{P}}$, we get that $(x_k,p_k,v_k)_{k=0}^K \in \mathcal{B}(S_{\mathbb{P}})$. Moreover, there are three posibilities for (x_K,p_K,v_K).

First, if $(x_K,p_K,v_K) \in W_{f,0}$, then $(x_k,p_k,v_k)_{k=0}^K \in \mathcal{L}(\mathbb{P})$.

Assuming that $(x_K,p_K,v_K) \notin W_{f,0}$, let $i \in L$ be such that $p_K \in P_i$. If $(x_K,p_K,v_K) \in W_{f,i} \setminus W_{f,0}$, then by the schedulability condition and the definition of (θ, π) we get $(x_K,p_K,v_K) \in \mathrm{dom}(\theta)$ which contradicts the maximality of $(x_k,p_k,v_k)_{k=0}^K$.

Consider now the third posibility: $(x_K,p_K,v_K) \notin W_{f,i}$. Since (θ_i, π_i) solves Problem 1 for program $\mathbb{P}_i$, we get $G_{\mathbb{P}_i}(x_K,p_K,v_K) = \emptyset$, and $W_{f,i}(x_K,p_K) \cup \mathrm{enab}_{G_{\mathbb{P}_i}}(x_K,p_K) \neq \emptyset$. It follows that $G_{\mathbb{P}}(x_K,p_K,v_K) = \emptyset$ which implies that $(x_k,p_k,v_k)_{k=0}^K$ is a maximal trajectory of $S_{\mathbb{P}}$. Moreover, if $\mathrm{enab}_{G_{\mathbb{P}_i}}(x_K,p_K) \neq \emptyset$ then it follows that $\mathrm{enab}_{G_{\mathbb{P}}}(x_K,p_K) \neq \emptyset$. If $W_{f,i}(x_K,p_K) \neq \emptyset$ then by the schedulability condition we get that $W_{f,0}(x_K,p_K) \cup \mathrm{enab}_{G_{\mathbb{P}}}(x_K,p_K) \neq \emptyset$. Hence, (θ, π) and Z_c solve Problem 1 for program $\mathbb{P}$. □

C. Algorithms

In this section, we present fixed-point type algorithms (see Algorithm 1 below) for controller synthesis based on Theorem 1. The termination Let $\mathrm{Combine}$ be the procedure, described in Theorem 1, of constructing program controller (θ, π) and the respective controllable set Z_c from subprogram controllers and controllable sets. There are two versions: "maximal" and "anytime" synthesis which we discuss below. For finite systems S, the termination of both versions is guaranteed by Theorem 1. For infinite systems, abstraction based methods can be used to get finite symbolic abstractions of S and of all tasks found in the subprograms of $\mathbb{P}$.

For the algorithms to work we make the following assumption: for every subprogram $\mathbb{P}_i \in \mathcal{P}$ of a program $\mathbb{P} = (\mathcal{P}, \mathcal{R}, W_{f,0})$, we have $W_{f,i}(x,p) \subseteq \mathrm{enab}_{G_{\mathbb{P}_i}}(x,p)$.

Remark 2: The assumption above does not limit generality since one may add an additional mode p^i_{bad} to subprogram $\mathbb{P}_i$ such that for all $x \in X$, (x, p^i_{bad}) is blocking and $W_{f,i}(x, p^i_{\mathrm{bad}}) = \emptyset$. The transitions $(x, p^i_{\mathrm{bad}}) \in G_{\mathbb{P}_i}(x,p,v)$ are added for all $(x,p,v) \in W_{f,i}$. After such modification the program satisfies the assumption but the language $\mathcal{L}(\mathbb{P})$ remains unchanged.

1) Maximal synthesis: In this algorithm we first initialize individual subprogram controllers by solving the respective synthesis subproblems. Then on each iteration we construct so called *effective terminal sets* $\hat{W}_{f,i}$ by removing the states $w = (x,p,v)$ which do not satisfy the schedulability condition. The subprogram controllers are then updated using the new candidate effective terminal sets which are contracting from iteration to iteration until a fixed-point is reached. Function Solve calls itself recursively to perform computations of subprogram controllers utilizing the supplied effective terminal sets instead of the actual terminal sets of the respective subprograms. For subprograms that are tasks, function Solve is given by two fixed-point algorithms: one for safety and one for reachability. For details on specification abstractions and task controller computations we refer to the previous paper [25].

If condition (1) is satisfied for more restrictive controller, it is also satisfied for a more permissive one. By induction, all states z which are removed in the algorithm are uncontrollable by any program controller. Thus, the controller which is constructed in this algorithm is maximal.

2) Anytime synthesis: In this algorithm the effective terminal sets are initialized to the respective subsets of $W_{f,0}$. Then on each iteration they are expanded by adding the states which satisfy the schedulability condition. The subprogram controllers are then updated with respect to the new effective terminal sets.

One may observe that the schedulability condition holds for all $(x,p,v) \in \cup_{i \in L} \hat{W}_{f,i} \setminus W_{f,0}$ at every iteration. Therefore, the computation may be stopped at any iteration to obtain a valid program controller.

The controllable set produced by this algorithm may be not maximal. Indeed, if the algorithm terminated after M iterations and a state (x_0, p_0) is controllable then an execution with no more than M subprogram switching moments may be enforced from it.

Remark 3: An analysis of fixed-point algorithms in [25] suggests that the complexity of task controller computation is $O(|X||P|^2|V|^2|U||T|)$ where T is the maximal number of

Algorithm 1: $(\theta, \pi, Z_c) := \mathrm{Solve}(S,\ \mathbb{P},\ A)$

Input: Finite transition system S, control program $\mathbb{P} = (\mathcal{P}, \mathcal{R}, W_{f,0})$ where $\mathcal{P} = \{\mathbb{P}_i, i \in L\}$ and $\mathbb{P}_i = (\mathcal{P}_i, \mathcal{R}_i, W_{f,i})$, algorithm $A \in$ {"maximal", "anytime"}

Output: Controller (θ, π) for program $\mathbb{P}$ compatible with S and the controllable set Z_c

begin
- **foreach** $i \in L$ **do**
  - **if** $A =$ *"maximal"* **then**
    - $\hat{W}_{f,i} := W_{f,i}$;
  - **else**
    - $\hat{W}_{f,i} := W_{f,i} \cap W_{f.0}$;
  - $\hat{\mathbb{P}}_i := (\mathcal{P}_i, \mathcal{R}_i, \hat{W}_{f,i})$;
  - $(\theta_i, \pi_i, Z_{c,i}) := \mathrm{Solve}(S,\ \hat{\mathbb{P}}_i,\ A)$;
- *fixedPoint* := *false*;
- **while** $\neg$*fixedPoint* **do**
  - *fixedPoint* := *true*;
  - **foreach** $i \in L$ **do**
    - $W := \emptyset$;
    - **if** $A =$ *"maximal"* **then**
      - $W_{\max} := \hat{W}_{f,i} \setminus W_{f,0}$;
    - **else**
      - $W_{\max} := W_{f,i} \setminus \hat{W}_{f,i}$;
    - **foreach** $w = (x,p,v) \in W_{\max}$ **do**
      - **if** $(\cup_j \{x\} \times R^*_{i,j}(x,p,v)) \cap \mathrm{dom}\ \theta_j = \emptyset$ **then**
        - **if** $A =$ *"maximal"* **then**
          - $W := W \cup \{w\}$;
          - *fixedPoint* := *false*;
      - **else**
        - **if** $A =$ *"anytime"* **then**
          - $W := W \cup \{w\}$;
          - *fixedPoint* := *false*;
    - **if** $A =$ *"maximal"* **then**
      - $\hat{W}_{f,i} := \hat{W}_{f,i} \setminus W$;
    - **else**
      - $\hat{W}_{f,i} := \hat{W}_{f,i} \cup W$;
    - $\hat{\mathbb{P}}_i := (\mathcal{P}_i, \mathcal{R}_i, \hat{W}_{f,i})$;
    - $(\theta_i, \pi_i, Z_{c,i}) := \mathrm{Solve}(S,\ \hat{\mathbb{P}}_i,\ A)$;
- $(\theta, \pi, Z_c) := \mathrm{Combine}(\{(\theta_i, \pi_i, Z_{c,i}),\ i \in L\})$;
- **return** (θ, π, Z_c);

transitions corresponding to a state-input pair of S (see also [3]). Without loss of generality, one may consider the program in the expanded form since the total number of modes and transitions for it is the same as for the original program. Thus, we may estimate the complexity of Algorihm 1 as $O(|X|^2|P|^3|V|^3|U||T|)$ since at least one of the effective terminal sets is reduced (increased) on each iteration of the outer loop of the maximal synthesis (anytime synthesis, respectively). The program controller may be represented by a boolean array of $|X||P|^2|V|^2|U||T|$ elements. The choice of the algorithm depends on the available resources and the size of the problem. For a very large problem, anytime controller synthesis may be preferred as it allows to obtain at least some solution. For a smaller problem, a maximal controller is preferred as it is generally more permissive.

Algorithm 1 could be parallelized to a certain extent. Particularly, the controller synthesis algorithms for each subprogram $\mathbb{P}_i$ could run in parallel. This, however, may increase the number of iterations of the main loop.

V. Example

A. The system

Let us consider a model of several vehicles moving in one lane on a two-lane infinite straight road. The controllable vehicle is described by the following discrete-time transition system:

$$\begin{array}{rcl} x^0_{k+1} & = & x^0_k + \beta(u_{2,k}, v^0_k, h_k)T_0, \\ v^0_{k+1} & = & \chi\left(v^0_k + \alpha(u_{1,k}, v^0_k)T_0,\ [0, v^0_{\max}]\right), \\ h_{k+1} & = & u_{2,k}. \end{array}$$

Here x^0 is the position, v^0 is velocity, and h is the current lane of the controllable vehicle. The controllable parameter $u_{1,k}$ is the torque applied to the wheels, $u_{1,k} \in [u_{\min}, u_{\max}]$, and the controllable parameter $u_{2,k}$ is the lane at the next time instant.

$$\begin{array}{rcl} \alpha(u, v) & = & u - M^{-1}(f_0 + f_1 v + f_2 v^2), \\ \beta(u, v, h) & = & \begin{cases} v, & u = h, \\ \frac{1}{T_0}\sqrt{(vT_0)^2 - r^2}, & u \neq h, v \geq v_*, \end{cases} \\ \chi(v, [v_1, v_2]) & = & \min\{\max\{v, v_1\}, v_2\}. \end{array}$$

The vector of parameters $f = (f_0, f_1, f_2) \in \mathbb{R}^3_+$ describes road friction and vehicle aerodynamics whose numerical values are taken from [17].

We assume that every vehicle has the length d_{car} and is responsible for not colliding with the vehicles in front of them. Next, we assume that the controllable vehicle has the information about the next two vehicles in front of it (vehicle 1 and vehicle 2 respectively) which are described by the equations ($j = 1, 2$):

$$\begin{array}{rcl} x^j_{k+1} & = & x^j_k + v^j_k T_0, \\ v^j_{k+1} & = & \chi\left(v^j_k + \alpha(w^j_k, v^j_k)T_0,\ [0, v_{\max}]\right). \end{array}$$

Here x^j is the position and v^j is the velocity of the uncontrollable vehicle j. It is assumed that $x^1 < x^2$. The uncertain parameters w^j represent the torque applied to the wheels of the uncontrollable vehicles.

By introducing the signed relative distances $d^1 = x^0 - x^1$ and $d^2 = x^1 - x^2$, we represent the system of 3 vehicles as a 6-dimensional transition system:

$$\begin{array}{rcl} d^1_{k+1} & = & d^1_k + (\beta(u_{2,k}, v^0_k, h_k) - v^1_k)T_0, \\ d^2_{k+1} & = & \chi\left(d^2_k + (v^1_k - v^2_k)T_0,\ (-\infty, -d_{\text{car}}]\right), \\ v^0_{k+1} & = & \chi\left(v^0_k + \alpha(u_{1,k}, v^0_k)T_0,\ [0, v^0_{\max}]\right), \\ v^1_{k+1} & = & \chi\left(v^1_k + \alpha(w^1_k, v^1_k)T_0,\ [0, v_{\max}]\right), \\ v^2_{k+1} & = & \chi\left(v^2_k + \alpha(w^2_k, v^2_k)T_0,\ [0, v_{\max}]\right), \\ h_{k+1} & = & u_{2,k}. \end{array}$$

One may observe that d^2 always stays negative due to the above assumption about collision avoidance.

Let z denote the state variable: $z = (d^1, d^2, v^0, v^1, v^2, h)$. The system may be written in a compact form as follows:

$$z_{k+1} = f(z_k, u_k, w_k).$$

The equation above hold for $z_k \notin Z_{\text{jump}}$ where

$$Z_{\text{jump}} = \{z \mid d^1 \geq d_{\text{car}},\ d^1 + d^2 \leq -d_{\text{car}},\ h < 1\}.$$

Set Z_{jump} describes the state space configurations where the controllable vehicle took over vehicle 1 and returned to the first lane. For configurations $z_k \in Z_{\text{jump}}$, the system is reinitialized as follows:

$$\begin{array}{rcl} d^1_{\text{jump}} & = & d^1_k + d^2_k, \\ d^2_{\text{jump}} & \in & (-\infty, -d_{\text{car}}], \\ v^0_{\text{jump}} & = & v^0_k, \\ v^1_{\text{jump}} & = & v^2_k, \\ v^2_{\text{jump}} & \in & [0, v^1_{\max}], \\ h_{\text{jump}} & = & h_k. \end{array}$$

The state z_{k+1} is then determined by

$$z_{k+1} = f(z_{\text{jump}}, u_k, w_k)$$

where $z_{\text{jump}} = (d^1_{\text{jump}}, d^2_{\text{jump}}, v^0_{\text{jump}}, v^1_{\text{jump}}, v^2_{\text{jump}}, h_{\text{jump}})$.

We now define the state space of the considered transition system: $Z = (-\infty, +\infty) \times (-\infty, -d_{\text{car}}] \times [0, v^0_{\max}] \times [0, v_{\max}] \times [0, v_{\max}] \times [0, 2]$.

For the simulations below we choose the following parameters: $T_0 = 0.1\ s$, $M = 1370\ kg$, $f_0 = 51.0709\ N$, $f_1 = 0.3494\ Ns/m$, $f_2 = 0.4161\ Ns^2/m^2$, $d_{\text{car}} = 2\ m$, $r = 2\ m$, $v^0_{\max} = 30\ m/s$, $v_{\max} = 20\ m/s$, $v_* = 20\ m/s$, $u_{\min} = w_{\min} = -20\ m/s^2$, $u_{\max} = w_{\max} = 10\ m/s^2$.

B. The specification

The specification is given by the control program depicted below. Here $\mathbb{T}_1$ is a safety task "cruise control" and $\mathbb{T}_2$

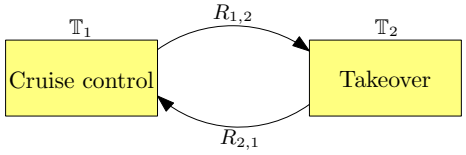

Fig. 1: Control program graphical representation.

is a reachability task "takeover". Note that $\mathbb{P}$ cannot be

represented by a single hybrid automaton with a safety or a reachability specification or by a single task.

The hybrid automaton for $\mathbb{T}_1$ has two modes: p^1 and p^2 ("track velocity" and "avoid collision"). In the mode p^1, vehicle 0 has to track the desired velocity v^*, which is provided by an external user. If the distance between vehicle 0 and vehicle 1 is too short, mode p^2 is activated in order to avoid collision. When a longer distance is restored, mode p^1 is reactivated. The discrete dynamics of the hybrid automaton is shown in Figure 2.

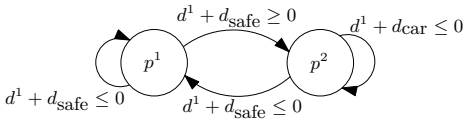

Fig. 2: Graphical representation of discrete dynamics for $\mathbb{T}_1$.

The associated continuous dynamics is then given by

$$\begin{cases} v^0_{k+1} \geq v^0_k + C, & \text{if } v^0_k \leq v^*_k - \varepsilon, \\ v^0_{k+1} \leq v^0_k - C, & \text{if } v^0_k \geq v^*_k + \varepsilon \end{cases}$$

for $p_{k+1} = p^1$ and

$$v^0_{k+1} \leq v^0_k - C, \text{ if } v^0_k \geq v^*_k + \varepsilon$$

for $p_{k+1} = p^2$. Input parameter v^* ("target velocity") in the specification belongs to the following set:

$$V = \{\tfrac{1}{N_m} v^0_{\max}, \ \ldots, \ \tfrac{N_m - 1}{N_m} v^0_{\max}\}.$$

The terminal set for $\mathbb{T}_1$ is given by

$$W_{f,1} = \{(z, p, v^*) \mid d^1 \leq -d_{\text{car}}, \ v^1 \leq v^*, \ h < 1\}.$$

For the simulation we choose the following numerical values for the parameters of this specification: $N_m = 6, \quad \varepsilon = 3 \ m/s, \quad C = 0.1 \ m/s, \quad d_{\text{safe}} = 50 \ m.$

The hybrid automaton $(z', p') \in G_{\mathbb{T}_2}(z, p)$ for task $\mathbb{T}_2$ is defined by the following constraints on the continuous dynamics: $z' \in Z$ and by the discrete variable evolution which is depicted on Fig. 3. This automaton has three modes: p^3, p^4, and p^5 ("change to lane 2", "takeover vehicle 1", and "change to lane 1", respectively).

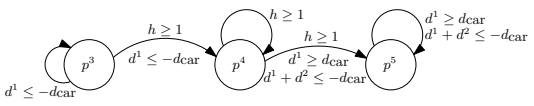

Fig. 3: Graphical representation of discrete dynamics for $\mathbb{T}_2$.

The terminal set for $\mathbb{T}_2$ is given by

$$W_{f,2} = \{(z, p, v^*) \mid d^1 \geq d_{\text{car}}, \ d^1 + d^2 \leq -d_{\text{car}}, \ h < 1, \ p = p^5\}.$$

Finally, the reset maps $R_{1,2}$ and $R_{2,1}$ are defined as follows

$$\begin{aligned} R_{1,2}(z, p, v^*) &= p^3, \\ R_{2,1}(z, p) &= \begin{cases} \{p^1\} \times V, & d^1 + d^2 \leq -d_{\text{safe}}, \\ \{p^2\} \times V, & d^1 + d^2 > -d_{\text{safe}}. \end{cases} \end{aligned}$$

C. Numerical results

The simulations were done in MATLAB R2019a using Intel Core i7 4.20GHz CPU, 32 GB RAM. To compute the controller we utilized symbolic abstractions of the system and the specifications [25] and compositionality reasoning similar to [23]. Particularly, we divide the 6-dimensional transition system into two subsystems:

$$\begin{aligned} d^1_{k+1} &= d^1_k + (\beta(u_{2,k}, v^0_k, h_k) - v^1_k) T_0, \\ v^0_{k+1} &= \chi\left(v^0_k + \alpha(u_{1,k}, v^0_k) T_0, \ [0, v^0_{\max}]\right), \\ v^1_{k+1} &= \chi\left(v^1_k + \alpha(w^1_k, v^1_k) T_0, \ [0, v_{\max}]\right), \\ h_{k+1} &= u_{2,k} \end{aligned}$$

and

$$\begin{aligned} d^2_{k+1} &= \chi\left(d^2_k + (v^1_k - v^2_k) T_0, \ (-\infty, -d_{\text{car}}]\right), \\ v^1_{k+1} &= \chi\left(v^1_k + \alpha(w^1_k, v^1_k) T_0, \ [0, v_{\max}]\right), \\ v^2_{k+1} &= \chi\left(v^2_k + \alpha(w^2_k, v^2_k) T_0, \ [0, v_{\max}]\right). \end{aligned}$$

Task $\mathbb{T}_1$ is unaffected by this change because it does not impose restriction on the dynamics of d^2 and v^2 and due to system S structure. For $\mathbb{T}_2$ the terminal set for the first subsystem is then given by the following:

$$\{(d^1, v^0, v^1, h, p, v^*) \mid d^1 \geq d_{\text{car}}, \ d^1 \leq d^*, \ h < 1, \ p = p^5\}.$$

The transitions from p^4 to p^5 and from p^5 to p^5 are changed accordingly. The second subsystem is used essentially to estimate the maximal number of time steps for takeover maneuver for given values of d^2, v^1, v^2 at the start of the maneuver. To do this we evaluate the number of time steps needed by the second subsystem to reach $\{(d^2, v^1, v^2) \mid d^2 \geq -d^* - d_{\text{car}}\}$. For numerical computations we utilized the value $d^* = 20 \ m$.

The total computation runtime for the maximal synthesis is 2.5 hours of which 1% is spent on abstracting the system, 31% on computing task abstractions, and 68% on the maximal program controller synthesis algorithm. The synthesis algorithm terminates after 2 iterations.

On Figure 4 we compare the controllable set of task $\mathbb{T}_2$ to the respective controllable region of program $\mathbb{P}$. Figure 5 depicts the simulated trajectory of the system. Here the top figure shows the relative distance d^1 depending on time t. The second figure shows the velocities of the three vehicles. Finally, the bottom figure depicts the current mode and task of the realized program execution. For this trajectory the desired velocity is set by the user to 25 m/s for the first 30 seconds and to 5 m/s afterwards. The controllable vehicle takes over 3 vehicles in front of it because it cannot maintain the desired velocity. After the first 30 seconds it stays in the cruise control mode.

VI. Conclusion

We introduced a new control specification language called control programs. A control program is decomposable into control tasks which are elementary control problems. Each task is defined by a transition system, which must be simulated by the closed-loop system, and a termination semantic, which is reminiscent of safety and reachability property of temporal logic. The provided algorithms for computing

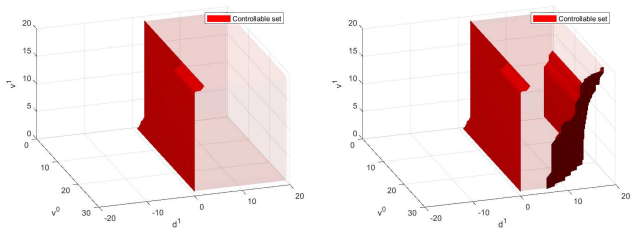

Fig. 4: Controllable region comparison for $\mathbb{T}_2$ and $\mathbb{P}$ ($h < 1$, $p = p^5$, $d^2 = -100\ m$).

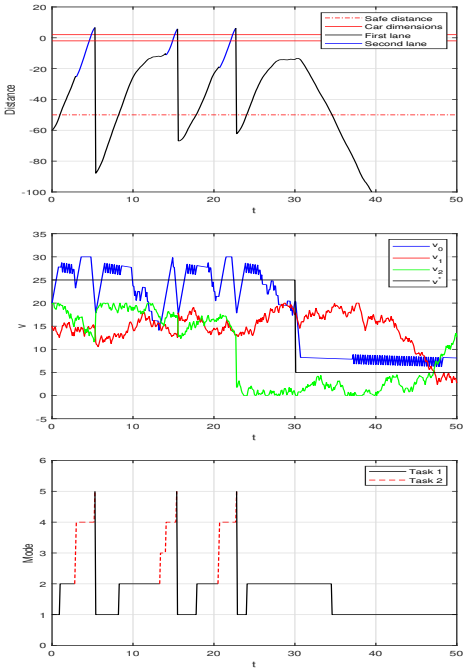

Fig. 5: Distance d^1, velocities v^0, v^1, v^2 and v^*, and the current program modes and tasks for the simulated trajectory.

program controllers limit the complexity of synthesis by polynomial time in their data. Theorem 1 may act as an instrument to reuse tasks and subprograms in a similar way one reuses functions in a programming language. We also present an example related to autonomous driving which illustrates how the formalism may be used in applications.

References

[1] E. Asarin, O. Bournez, T. Dang, O. Maler, and A. Pnueli. Effective synthesis of switching controllers for linear systems. *Proceedings of the IEEE*, 88(7):1011–1025, 2000.

[2] C. Belta, B. Yordanov, and E. A. Gol. *Formal Methods for Discrete-Time Dynamical Systems*. Springer, 2017.

[3] D. Berwanger. Graph games with perfect information. *MPRI*, 2012/13.

[4] S. Coogan and M. Arcak. Efficient finite abstraction of mixed monotone systems. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, pages 58–67. ACM, 2015.

[5] J. Eilbrecht and O. Stursberg. Optimization-based maneuver automata for cooperative trajectory planning of autonomous vehicles. In *Proceedings European Control Conference, ECC 2018*, Limassol, Cyprus, June 12-15 2018.

[6] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas. Temporal logic motion planning for dynamic robots. *Automatica*, 45(2):343–352, 2009.

[7] E. Frazzoli, M. A. Dahleh, and E. Feron. Maneuver-based motion planning for nonlinear systems with symmetries. *IEEE transactions on robotics*, 21(6):1077–1091, 2005.

[8] A. Girard, G. Gössler, and S. Mouelhi. Safety controller synthesis for incrementally stable switched systems using multiscale symbolic models. *IEEE Transactions on Automatic Control*, 61(6):1537–1549, 2016.

[9] E. A. Gol, M. Lazar, and C. Belta. Language-guided controller synthesis for linear systems. *IEEE Transactions on Automatic Control*, 59(5):1163–1176, 2014.

[10] T. A. Henzinger. The theory of hybrid automata. In *Verification of Digital and Hybrid Systems*, pages 265–292. Springer, 2000.

[11] D. Heß, M. Althoff, and T. Sattel. Formal verification of maneuver automata for parameterized motion primitives. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference*, pages 1474–1481. IEEE, 2014.

[12] J. Liu and N. Ozay. Finite abstractions with robustness margins for temporal logic-based control synthesis. *Nonlinear Analysis: Hybrid Systems*, 22:1–15, 2016.

[13] J. Liu, N. Ozay, U. Topcu, and R. M. Murray. Synthesis of reactive switching protocols from temporal logic specifications. *IEEE Transactions on Automatic Control*, 58(7):1771–1785, 2013.

[14] J. Lygeros, K. H. Johansson, S. N. Simić, J. Zhang, and S. S. Sastry. Dynamical properties of hybrid automata. *IEEE Transactions on Automatic Control*, 48(1):2–17, 2003.

[15] N. Y. Megawati and A. van der Schaft. Abstraction and control by interconnection of linear systems: A geometric approach. *Systems & Control Letters*, 105:27–33, 2017.

[16] P.-J. Meyer, A. Girard, and E. Witrant. Safety control with performance guarantees of cooperative systems using compositional abstractions. *IFAC-PapersOnLine*, 48(27):317–322, 2015.

[17] P. Nilsson, O. Hussien, Y. Chen, A. Balkan, M. Rungger, A. D. Ames, J. W. Grizzle, N. Ozay, H. Peng, and P. Tabuada. Preliminary results on correct-by-construction control software synthesis for adaptive cruise control. *53rd IEEE Conference on Decision and Control*, pages 816–823, 2014.

[18] A. Platzer. Differential dynamic logic for hybrid systems. *Journal of Automated Reasoning*, 41(2):143–189, 2008.

[19] G. Pola, A. Borri, and M. D. Di Benedetto. Integrated design of symbolic controllers for nonlinear systems. *IEEE Transactions on Automatic Control*, 57(2):534–539, 2011.

[20] G. Pola and M. D. Di Benedetto. Control of cyber-physical-systems with logic specifications: A formal methods approach. *Annual Reviews in Control*, 2019.

[21] G. Reißig. Computing abstractions of nonlinear systems. *IEEE Transactions on Automatic Control*, 56(11):2583–2598, 2011.

[22] G. Reissig, A. Weber, and M. Rungger. Feedback refinement relations for the synthesis of symbolic controllers. *IEEE Transactions on Automatic Control*, 62(4):1781–1796, 2016.

[23] A. Saoud, A. Girard, and L. Fribourg. Contract based design of symbolic controllers for interconnected multiperiodic sampled-data systems. In *57th IEEE Conference on Decision and Control*, 2018.

[24] C. Seatzu, D. Gromov, J. Raisch, D. Corona, and A. Giua. Optimal control of discrete-time hybrid automata under safety and liveness constraints. *Nonlinear Analysis: Theory, Methods & Applications*, 65(6):1188–1210, 2006.

[25] V. Sinyakov and A. Girard. Formal Controller Synthesis from Specifications Given by Discrete-Time Hybrid Automata. preprint, `https://hal.archives-ouvertes.fr/hal-02361404`, Nov. 2019.

[26] P. Tabuada. *Verification and Control of Hybrid Systems: a symbolic approach*. Springer, 2008.

[27] P. Tabuada and G. J. Pappas. Linear time logic control of discrete-time linear systems. *IEEE Transactions on Automatic Control*, 51(12):1862–1877, 2006.

[28] C. J. Tomlin, J. Lygeros, and S. S. Sastry. A game theoretic approach to controller design for hybrid systems. *Proceedings of the IEEE*, 88(7):949–970, 2000.

[29] H. Vinjamoor and A. J. van der Schaft. The achievable dynamics via control by interconnection. *IEEE Transactions on Automatic Control*, 56(5):1110–1117, 2010.

[30] M. Zamani, G. Pola, M. Mazo, and P. Tabuada. Symbolic models for nonlinear control systems without stability assumptions. *IEEE Transactions on Automatic Control*, 57(7):1804–1809, 2012.