Abstract—A plethora of driver convenience and safety automation systems are being introduced into production vehicles, such as electronic stability control, adaptive cruise control, lane keeping, and obstacle avoidance. Assuring the seamless and safe integration of each new automation function with existing control functions is a major challenge for vehicle manufacturers. This challenge is compounded by having different suppliers providing software modules for different control functionalities. In this paper, we report on our preliminary steps to address this problem through a fresh perspective combining formal methods, control theory, and correct-by-construction software synthesis. In particular, we begin the process of synthesizing the control software module for adaptive cruise control from formal specifications given in Linear Temporal Logic. In the longer run, we will endow each interacting software module with an assume-guarantee specification stating under which environment assumptions the module is guaranteed to meet its specifications. These assume-guarantee specifications will then be used to formally prove correctness of the cyber-physical system obtained when the integrated modules interact with the physical dynamics.

I. INTRODUCTION

Adaptive Cruise Control (ACC) is a driver assistance system designed to provide improved convenience and comfort with respect to conventional cruise control systems. When there is no preceding vehicle in sight, an ACC-equipped vehicle behaves just like one with conventional cruise control, i.e., it will maintain a constant speed set by the driver. When a preceding vehicle is detected and is driving at a speed slower than the preset speed, an ACC-equipped vehicle changes its control objective to maintaining a safe headway (range) or time headway instead. These two modes are commonly known as the speed (cruise) mode, and distance (following) mode, respectively, and the preceding vehicle is commonly called the lead vehicle.

ACC has been available on production vehicles since the mid-1990s. Many of the early ACC systems shut off below a given threshold speed. More recently, full-range (stop-and-go) ACC is available that can bring a vehicle to a full stop and then launch from standstill, and is thus capable of dealing with congested urban traffic. In the last several years, some auto makers (e.g., Volvo and Cadillac) have introduced an automated full-braking function to leverage hardware already installed for ACC, tiptoeing the red-line between comfort/convenience systems and active safety systems. This trend is likely to continue as more automated vehicle control functions are introduced on production vehicles.

In [1], National Highway Traffic Safety Administration (NHTSA) announced its policy on the development of automated vehicles, and defined five levels of vehicle automation: Level 0 (no automation), Level 1 (function-specific automation, such as ACC or electronic stability control), Level 2 (combined function automation, such as ACC plus lane keeping), Level 3 (limited self-driving), and finally Level 4 (full self-driving). All new production cars in the US are already at Level 1, because electronic stability control has been required for all new cars in the US since 2012. A major challenge for control engineers is that as more automation functions are introduced (either required or as options), their seamless integration with existing control functions must be guaranteed. This requirement is important both to ensure vehicle safety and for tort/liability considerations.

This paper is the first step towards the broader objective of synthesizing correct-by-construction control software for automation Level 2 and beyond. By correct-by-construction we mean control software that is guaranteed to meet its formal specifications given certain assumptions on the physical plant and implementation platform. The formal specifications considered in this paper are given in Linear Temporal Logic, a common specification language for software systems. Here, we begin the task of synthesizing correct-by-construction control software for ACC in the special case that the lead vehicle’s speed is constant. Two synthesis methods are used, one performing set computation directly on the continuous domain, and a second based on finite-state abstractions. The two resulting correct-by-construction controllers are compared by running simulations in Simulink and on a 16 degree of freedom model in CarSim. In subsequent work, we will address stop-and-go ACC with variable lead-vehicle speed, correct-by-construction control software for lane keeping, and correct-by-construction control software when ACC and lane keeping are simultaneously active.

II. ADAPTIVE CRUISE CONTROL

For ACC systems, the most important function is to maintain a safe and comfortable range from the preceding vehicle. Whether a range is safe or not is based on metrics

\cite{1}
such as time-to-collision or deceleration needed to avoid a crash. Whether a range is comfortable or not is subjective, and may consider additional factors such as value and rate of change of range, time-headway, and jerk.

A. Past Work

Adaptive cruise control started as an extension of conventional cruise control (CCC), which was how it was described in relevant ISO [2] and SAE [3] standards. In these early design concepts, ACC was a phase or mode of the overall control system, and used existing CCC hardware architecture: in the ACC mode, the speed command to the CCC servo-loop was adjusted to achieve the desired range control objective (e.g., [4]). This nested control architecture resulted in slower response (in comparison to when throttle and brake are controlled directly), which was observed in prototype ACC vehicles in field tests [5]. Comparison between the velocity-command approach and acceleration command approach was analyzed in [6].

The potential of ACC-equipped vehicles for improving traffic flow and safety has been studied extensively since the 1990s [7][8]. In addition to traffic flow and safety, string stability [9][10][11], congestion [12], fuel economy [13] and integration with crash avoidance [14] have also been studied. An extensive survey on ACC designs, including the underlying control concepts, is given in [15]. In recent years, MPC has been widely used in ACC design [16].

Although driver assistance and safety modules such as ACC have been investigated for many years, the emphasis on the correctness of a module’s software implementation is much more recent. Safety verification of evasive maneuvers for autonomous vehicles is addressed in [17]. By computing the reachable set of each vehicle under different types of uncertainties and disturbances, safety is ensured whenever the reachable sets for different vehicles are disjoint. The computation of reachability sets for hybrid systems is known to be expensive and in [18] verification is done through counterexample-guided search. Rather than working with a detailed model of the system to be verified, an abstraction is used. Verifying the abstraction leads to counterexamples that may be spurious, i.e., they may not be true behaviors of the real system. However, since reachability analysis on the abstraction is cheaper, a two-step approach is taken: 1) the abstract model is used to obtain counterexamples; 2) the counterexamples are proved or disproven on a detailed model of the system to be verified. The previous approaches to verification fall in the class of model checking, given a specification one checks (by reachability computation or counterexample-guided search) if the specification is met. A different approach, considered in [19], is theorem proving. Here, one writes the assumptions about the system and its environment in a convenient logic and proves a theorem stating that the desired specification follows from the assumptions. A related approach is the use of satisfiability (SAT) and satisfiability modulo theory (SMT) solvers instead of a customized logic as was done in [20]. Unlike all the aforementioned approaches, we do not verify an existing software module. Instead, we synthesize a software module that is guaranteed to satisfy the specification by construction, hence the term correct-by-construction.

B. Simplified Model For Constant Lead Vehicle Speed

The vehicle is modeled as a (lumped) point mass \( m \) moving along a straight line. The net action of braking and engine torque applied to the wheels is lumped as a net force \( F_w \) acting on the mass of the vehicle, while the combined aerodynamic and rolling resistance is gathered into a net force \( F_r \).

\[
\dot{m}v = F_w - F_r.
\]

In the above equation, \( F_w \) is viewed as the control input and is assumed to be bounded by

\[
-0.3mg \leq F_w \leq 0.2mg,
\]

where \( g \) is the gravitational constant. Such a bound is consistent with non-emergency braking and acceleration, and thus with the “driver convenience” notion of ACC. The term \( F_r \) is represented by \( F_r = f_0 + f_1v + f_2v^2 \) [7]. We limit the admissible velocities to a bounded set \( \mathcal{V} = [v_{\text{min}}, v_{\text{max}}] \) with \( v_{\text{min}} \geq 0 \).

To include a lead vehicle in the system description, we use a hybrid system model with two discrete modes \( M_1 \) and \( M_2 \), called no lead car and lead car mode, respectively. The lead car mode \( M_2 \) has an additional continuous state \( h \) which measures the headway to the lead car. The continuous dynamics of \( M_1 \) are those of \( \{1\} \) while the continuous dynamics in mode \( M_2 \) contain an additional equation describing the dynamics of the headway:

\[
\dot{h} = v_L - v
\]

The two modes have different state spaces, mode \( M_1 \) has state space \( \mathcal{V} \) while mode \( M_2 \) has state space \( \mathcal{V} \times \mathcal{H} \), where \( \mathcal{H} = [0, h_{\text{max}}] \) for an upper limit \( h_{\text{max}} \) on the radar range.

In practice, the system is in \( M_2 \) if there is a car within the radar range, and in \( M_1 \) otherwise. Switching between the two states is governed by lane changes of lead cars, which are modeled using reset maps \( R_{1,2} : \mathcal{V} \rightarrow 2^{\mathcal{V} \times \mathcal{H}} \), \( R_{2,1} : \mathcal{V} \times \mathcal{H} \rightarrow \mathcal{V} \) and \( R_{2,2} : \mathcal{V} \times \mathcal{H} \rightarrow 2^{\mathcal{V} \times \mathcal{H}} \).

\[
R_{1,2}(v) = \{(v, h) : \exists h \in \mathcal{H}\}, \quad R_{2,1}(v, h) = \{v\}, \quad R_{2,2}(v, h) = \{(v, h) : \exists h \in \mathcal{H}\}.
\]

Here \( R_{1,2} \) models a transition from the no lead car mode \( M_1 \) to the lead car mode \( M_2 \), where the headway is initialized to some \( h \in \mathcal{H} \). Similarly, \( R_{2,2} \) models situations where the radar reading suddenly changes as a result of lane changes undertaken by cars in front.

The switching is assumed to be non-deterministic, except for the case when \( h \) reaches \( h_{\text{max}} \) in mode \( M_2 \), in which case a forced transition to mode \( M_1 \) occurs. The complete hybrid model is shown schematically in Fig. [1].

\footnote{In SI units, \( m = 1370 \text{ kg}, f_0 = 3.8 \times 10^{-3} \times mg \text{ N}, f_1 = 2.6 \times 10^{-5} \times mg \text{ N/m}, f_2 = 0.4161 \text{ N/s/m}^2.}
The velocity $v$ is defined as

$$\dot{v} = F_w - f_0 - f_1 v - f_2 v^2$$

Fig. 1: Hybrid system model.

Sedan’’ model in CarSim. We also restrict the domain to $v_{min} = 0$ m/s, $v_{max} = 35$ m/s, $h_{min} = 0$ m, $h_{max} = 300$ m, where the latter models the radar range.

For simplicity, we make the following assumptions:

(A.1) The velocity $v_L$ of the lead car is known and constant, i.e., $\dot{v}_L = 0$. In the following we assume $v_L = 12$ m/s.

(A.2) There is at most one lead car within the radar range at time $t$, i.e., a desired velocity $v$ vehicle satisfies $G_h$. The set $G_h$ is defined as $\{v, h : h/v \geq \tau_{des}, v \leq v^+, h \in H\}$.

Here $v^-$ and $v^+$, define the interval of tolerated deviation around the desired speed $v_{des}$. The set $G_h$ is depicted in Fig. 2. The sets $S_1$ and $S_2$ are used to express the constraints that need to be satisfied at all times, i.e., the distance constraint and input constraint. These sets are given by

$$S_1 = \{(v, h) : h/v \geq 1, v \in V, h \in H\},$$

$$S_2 = \{F_w : F_w \in [-0.3mg, 0.2mg]\}.$$  

In summary, the set of atomic propositions is given by

$$\{M_1, M_2, G_1, G_2, S_1, S_2\}.$$  

The specifications considered in this paper can be expressed using the atomic propositions, the propositional logic conjunction “$\land$” and negation “$\neg$”, and the temporal operators always “$\Box$”. Disjunction “$\lor$” and implication “$\Rightarrow$” can be constructed from conjunction and negation. Eventually “$\diamond$” can be constructed from negation and always.

We interpret LTL formulas over infinite sequences $(\mu, \xi, \nu)$ where the signal $\mu : N \rightarrow \{M_1, M_2\}$ specifies the current mode of the system and $\xi : N \rightarrow \mathbb{R}^2$ is a sample-and-hold trajectory of (5) given the input signal $\nu : N \rightarrow \mathbb{R}$ generated by the ACC. We refer to the triple of such sequences $(\mu, \xi, \nu)$ as a behavior of the closed-loop system, i.e., (5) controlled by the ACC.

A behavior $(\mu, \xi, \nu)$ is said to satisfy $\Box p$ or $\diamond p$ if the atomic proposition $p$ holds true at every time step or at some future time step(s), respectively. A closed-loop system satisfies an LTL formula $\varphi$ if every system behavior $(\mu, \xi, \nu)$ satisfies $\varphi$. For reasons of space, we do not provide the syntax and semantics of LTL and refer the interested reader to [22]. Instead, we continue with two examples. Consider the simple formulas $\varphi_1 = \Box(M_1 \land G_1 \land S_2)$ and $\varphi_2 = \Diamond(G_2 \land S_2)$. A behavior satisfies $\varphi_1$ if $\mu_t = M_1$ and $(\xi_t, \nu_t) \in G_1 \times S_2$ holds for all $t \in N$. A behavior satisfies $\varphi_2$ if $\mu_t = M_2$ and $\xi_t \in G_2$ holds for some time $t \in N$.

The ACC specification can be described by the LTL formula $\psi$:

$$\Box((M_1 \lor S_1) \land S_2) \land (\bigwedge_{i=1}^2 \Box M_i \implies \Diamond G_i).$$

The first term $\Box((M_1 \lor S_1) \land S_2)$ ensures that the input constraint and the distance to the lead vehicle constraint are always met. Note that the distance constraint only needs to be satisfied in mode $M_2$, hence the disjunction $M_1 \lor S_1$. We
specify with the second term \(\Box (\bigwedge_{i=1}^{2} \Box M_i \implies \Diamond \Box G_i)\) that the desired velocity/time headway \(G_i\) should eventually be reached and kept invariant, given that the system is in mode \(M_i\).

Notice that the specification gives no guarantees of reaching the desired time headway and velocity, when the system switches between \(M_1\) and \(M_2\) infinitely often. This is in compliance with the ISO standards [2], which does not specify what the system should guarantee in such a scenario. Moreover, when the controller for mode \(M_i\) is constructed, as long as the system is in \(M_i\), it guarantees progress towards \(G_i\). This property is desirable and it holds by construction because the controller cannot know if a mode change is going to occur in the future or not.

D. Problem statement

The goal of this paper is to synthesize a controller for the hybrid system in Fig. 1 with reset maps given by \(A,1\) so that every behavior of the closed-loop system satisfies the specification \(\psi\) given in \(9\). However, if the lane-change behavior of the lead car is not properly constrained, e.g. if a car with very low speed may cut right in front of an ACC-equipped car traveling at higher speed, no controller can prevent collisions, hence \(\psi\) cannot be satisfied.

Motivated by this fact, the formal controller synthesis problem can now be stated as follows:

Problem 1: Assume (A.1) and (A.2) hold. Synthesize a controller and a control domain \(D \subseteq V \times H\) so that every behavior \((\mu, \xi, \nu)\) of the closed-loop system satisfies \(\psi\) if the values of \(R_{1,2}(v)\) are restricted to \(D\) for all \((v, h) \in V \times H\).

In what follows, we present two approaches to this problem. Both approaches seek to synthesize controllers with control domains that are as large as possible. In general, finding a maximal \(D\) can be hard [23]. The presented approaches are approximate in the sense that although the synthesized controllers are provably-correct, the synthesized control domains are not necessarily maximal.

III. Solution by Computations on the Continuous State Space

In this section a solution strategy based on set computations on the continuous domain \(V \times H\) is presented. We proceed by separately synthesizing controllers for the two modes \(M_1\) and \(M_2\), such that, respectively, the specifications

\[\Diamond \Box G_1 \land \Box \Diamond S_2\]
\[\Box \Diamond G_2 \land \Diamond \Box (S_1 \land S_2)\]

are satisfied when there is no mode switching. Then, we show how these two controllers can be implemented together to satisfy the original specification in \(6\) under mode switching.

A. Linearized Model

The solution strategy presented in this section relies on computation of reachable sets. Such computations are hard to perform for general nonlinear systems, we therefore linearize the polynomial system \(3\) around a nominal velocity \(\bar{v}\) to obtain

\[
\dot{v} = \frac{1}{m}(\bar{F}_w - \bar{f}_0 - \bar{f}_1 v),
\]
\[
\dot{h} = \bar{v} - v,
\]

for \(\bar{f}_0 = f_0 - f_2 \bar{v}^2\) and \(\bar{f}_1 = f_1 + 2f_2 \bar{v}\). Assuming that a control input \(\bar{F}_w\) has been computed for the linearized system \(12\), the modified control \(F_w = \bar{F}_w + f_2(v - \bar{v})^2\) can be used to make the nonlinear system behave in accordance with the linear system. Effectively, this linearization procedure moves the nonlinearity from the system dynamics to appropriate bounds on the input. To ensure that \(F_w\) stays inside the required range, we conservatively restrict \(\bar{F}_w\) to the interval

\[-0.3mg \leq \bar{F}_w \leq 0.2mg - \gamma,\]

where \(\gamma = \max_{\nu \in [v_{\text{min}}, v_{\text{max}}]} f_2(v - \bar{v})^2\). The correction term \(\gamma\) is small for typical parameter values and the level of conservativeness is therefore not very severe. By construction, the following now holds.

Proposition 1: For any \((v_0, h_0)\), \((v_1, h_1)\) \(\in V \times H\), if there exists an input \(\bar{F}_w : [0, T] \to [-0.3mg, 0.2mg - \gamma]\) that steers the state of \(12\) from \((v_0, h_0)\) to \((v_1, h_1)\) in time \(T\), then there exists an input \(F_w : [0, T] \to [-0.3mg, 0.2mg]\) that steers the state of \(3\) from \((v_0, h_0)\) to \((v_1, h_1)\) in time \(T\).

It is therefore sufficient to study reachability for the linearized system, which is described in the subsequent section.

B. Backwards-Time Reachability for Linear Systems

A polyhedral set, or polyhedron, is a set defined by linear inequalities. In this section we outline how the following problem can be solved numerically:

Problem: Given a final polyhedral set \(X_1 = \{x \in \mathbb{R}^n : L_2 x \leq l_2\}\), a time-continuous invariant system

\[\Xi : \quad \dot{x} = Ax + Bu + K,\]

a time step \(\Delta T\), and (possibly state-dependent) linear constraints on the input represented as \(H_x x + H_u u \leq h_{xu}\), find an initial set \(\text{Pre}_\Xi(X_1) \subset \mathbb{R}^n\) such that for all \(x_0 \in \text{Pre}_\Xi(X_1)\), there exists a constant control \(u_0\) such that

1) the control constraint \(H_x x_0 + H_u u_0 \leq h_{xu}\) is satisfied,
2) the final set \(X_1\) is reached at time \(\Delta T\) by applying \(u_0\), i.e. \(A\Delta T x_0 + B_d u_0 + K_d \in X_1\), where \(A_d = e^{A\Delta T}\), \(B_d = \int_{0}^{\Delta T} e^{A\tau} Bd\tau\) and \(K_d = \int_{0}^{\Delta T} e^{A\tau} Kd\tau\) define the time-discretized version \(\Xi_d\) of \(\Xi\).

The constraint 1) above can be expressed as \(L_2 (A_d x_0 + B_d u_0 + K_d) \leq l_2\) and combined with 2) the joint constraints on initial state and input can be written as

\[
\begin{bmatrix}
L_2 A_d \\
H_x \\
H_u \\
0
\end{bmatrix}
\begin{bmatrix}
x_0 \\
u_0
\end{bmatrix}
\leq
\begin{bmatrix}
l_2 \\
-L_2 K_d
\end{bmatrix}.
\]

Let \(L\) be the left-hand side matrix and \(l\) the right-hand side vector in \(15\), which together define a polyhedron \(P = \{(x, u) \in \mathbb{R}^{n+m} : [x^T \ u^T]^T \leq l\}\) in \(x-u\)-space containing all possible combinations of initial states and inputs that reach \(X_1\). To separate out the set of all
initial states, we project $P$ onto its $x$-coordinates. If $\Pi_x$ is the $x$-projection operator, then $P_{\text{pre}}(X_1) = \Pi_x(P)$. The projection of a polyhedron can be computed with for example the Multi-Parametric Toolbox (MPT) [24].

C. Finding and Steering to a Controlled-Invariant Set

In this section we outline how to solve a problem of type

$$\Diamond \Box G \land \Box S$$

by performing reachability calculations.

An equivalent condition to $\Diamond \Box G$ for deterministic systems is $\Diamond G \land (G \rightarrow \Box G)$, i.e. ensure that $G$ is eventually reached and that the system never leaves $G$ after reaching it for the first time. To find a strategy for staying in $C$ we search for a controlled-invariant set $C$ contained in $G$, where controlled-invariance is defined as follows.

Definition 1: A set $C$ is controlled-invariant for the discrete-time system $\Xi_d$ if for any $x_0 \in C$, there exists an input $u$ such that $(x_0, u)$ satisfies the input constraint and such that $A_d x_0 + B_d u + K_d \in C$.

In the following, we propose two methods for finding a controlled-invariant set $C$ contained inside a given set $G$. Both methods rely on finding fixed points for the operator $K_G(C) = P_{\text{pre}}(C) \cap G$.

The inside-out Algorithm 1 starts with the whole set $G$ and gradually shrinks it until only the controlled-invariant part remains. This algorithm converges to the maximal controlled-invariant set contained in $G$ [25], [26]. However, it may not terminate and manual termination gives an over-approximation of the maximal controlled-invariant set in $G$. On the other hand, if a ‘small’ controlled-invariant set $C^*$ can be found inside $G$, the inside-out Algorithm 2 [27] grows $C^*$ as much as possible inside of $G$. The initial ‘small’ controlled-invariant set $C^*$ can for example be some natural equilibrium point of the system. The advantage of Algorithm 2 is that the iteratively updated set $\tilde{C}$ is controlled-invariant in every iteration. It can therefore be terminated early and still produce a non-maximal controlled-invariant set inside of $G$.

Algorithm 1 Outside-in controlled-invariant set computation.

function CONINVIO($\Xi, G$)
    $\tilde{C} := G$, $C := \emptyset$
    while $\tilde{C} \neq C$
        $C := \tilde{C}$, $\tilde{C} := P_{\text{pre}}(C) \cap G$
    return $C$

Finally, when a controlled-invariant set $C \subseteq G$ has been found, we find a list of sets $C_2, \ldots, C_I$, such that $C_I$ is reachable from $C_{i+1}$, using Algorithm 3. By ensuring that all sets $C_i$ are contained in the safe set $S$ and denoting $C_I = C$, the control strategy defined as “when in $C_i$ go to $C_{\text{max}(1,i-1)}$ in finite time” will satisfy (10). This control strategy can be implemented using any method capable of generating input that takes the plant to a set defined by linear inequalities. A natural choice is to use Model-Predictive Control (MPC) with a quadratic cost criterion for the discrete-time system $\Xi_d$, which will result in a piecewise-constant control signal for the continuous system. The domain $D$ of this controller is $\bigcup_{i=1}^{I} C_i$.

Algorithm 2 Inside-out controlled-invariant set computation.

function CONINVIO($\Xi, G, C^*$)
    $C := C^*$, $C := \emptyset$
    while $\tilde{C} \neq C$
        $C := \tilde{C}$, $\tilde{C} := P_{\text{pre}}(C) \cap G$
    return $C$

Algorithm 3 Find a chain of sets in $G$ that reaches $C_1$.

function CONTROLCHAIN($\Xi,C_1,S,I$)
    for $i:=2,1$ do
        $C_i := P_{\text{pre}}(C_{i-1}) \cap S$
    return $C_2, \ldots, C_I$

D. Application to ACC problem

In this section we construct controllers $K_1$ and $K_2$ for the two modes $M_1$ and $M_2$ such that the specifications (10) and (11) are satisfied, for parameters $r_{\text{des}} = 1.3$ s, $v^- = 28$ m/s, $v^+ = 34$ m/s. The sets of initial conditions for which these controllers satisfy their respective specifications, denoted $D_1 \subset V$ and $D_2 \subset V \times H$, will define the necessary restrictions on the reset maps of the hybrid system. We use a time discretization $\Delta T = 0.5$ s.

In mode $M_1$, the requirement on $K_1$ is $\Diamond \Box G_1$, which can be interpreted as conventional (non-adaptive) cruise control. It is possible to use the method presented in Section III-C to synthesize $K_1$. However, for simplicity, we choose to use MPC to design a $K_1$ which satisfies (10) for any trajectory starting in $D_1 = V$.

Turning to controller design for mode $M_2$, we now apply the solution strategy in Section III-C to find a controller $K_2$ that satisfies (11). If $\Xi$ denotes the linearized dynamics (12), we proceed as follows: 1) call CONINVIO($\Xi,G_2$) or CONINVIO($\Xi,G_2,C^*$) to find a controlled-invariant set $C_1$ contained in $G_2$, 2) call CONTROLCHAIN($\Xi,C_1,S_1,I$) to find a chain of sets $C_2, \ldots, C_I$ in $S_1$ such that $C_I$ is reachable from $C_{\text{max}(1,i-1)}$.

For this example on a 3.4 Ghz iMac, Algorithms 1 and 2 finish in 15 and 17 milliseconds, respectively, and return an over- and under-approximation of the same controlled-invariant set $C_1 \subset G_2$. The call to Algorithm 3 takes 20 milliseconds on the same computer and yields two additional sets $C_2$ and $C_3$ such that $C_{\text{max}(1,i-1)}$ is reachable from $C_1$.

These sets are depicted in Fig. 3.

Let $K_2$ be a controller that implements the control strategy when in $C_1$, go to $C_{\text{max}(1,i-1)}$. By construction, $K_2$ then satisfies (11) for initial conditions in $D_2 = \bigcup_i C_i$.

Proposition 2: A controller that uses the controller $K_1$ when in mode $M_1$ and the controller $K_2$ when in mode $M_2$ solves Problem 1 with control domain $D = D_2$.

Proof: Let $\Gamma(v,h)(D_1)$ be the lifting of $D_1$ to $V \times H$, that is, $\Gamma(v,h)(D_1) = V \times D_1$. The result follows from the facts that $R_{2,1}(D_2) \subseteq D_1$ and $D_2 \subseteq \Gamma(v,h)(D_1) \cap D_2$. $\blacksquare$
C regions. We denote the corresponding sets in \( Q \) \( G \) space. Therefore the sets based on a discrete abstraction of the system (3).

The computation is that enforces the desired behavior (9) on the system (3) using of states, \( U \)

Controller Synthesis

In this section we describe how we compute a controller that enforces the desired behavior (9) on the system (3) using the MATLAB toolbox PESSOA [28]. The computation is based on a discrete abstraction of the system (3).

The abstraction is computed by a discretization of the state space, input space and time. The result is a finite state transition system \( \Sigma = (Q, U, \delta) \), where \( Q \) is the set of states, \( U \) is the set of inputs and \( \delta : Q \times U \rightarrow 2^Q \) is the transition relation. Note that \( Q \) forms a grid on the state space. Therefore the sets \( G_1, G_2 \) and \( S_1, S_2 \) can easily be mapped to \( Q \) by computing the grid points that fall inside the regions. We denote the corresponding sets in \( Q \) as \( G_{1,a}, G_{2,a} \) and \( S_{1,a}, S_{2,a} \). To be able to satisfy the input constraints we set \( U = S_{2,a} \).

A. Controller Synthesis

We use \( \langle \psi \rangle \) to denote the states for which there exists a control strategy that enforces \( \psi \). From the set \( \langle \psi \rangle \) we can easily construct a strategy. For reasons of space, however, we will only discuss the computation of \( \langle \psi \rangle \) at an intuitive level.

All the controller synthesis algorithms in PESSOA are based on fixed-point solutions to certain types of games on finite graphs. Termination of these algorithms is guaranteed by the finiteness of the set of states \( Q \). The interested reader is referred to [29] for further details on the algorithmic solution of such games.

To find \( \langle \psi \rangle \), we introduce Algorithm 4. Here, PESSOA first handles the safety part of the specification, i.e. \( \square S_{1,a} \), by solving a safety game via Algorithm 5. The result is a set-valued controller that determines the largest set of inputs that can be applied to each state while enforcing the safety specification. We then restrict the inputs of the abstraction to the inputs given by this controller and obtain a new system that will be used to compute the remaining specification via Algorithm 4. Since we assume that the system can change

**Algorithm 4 Computation of \( \langle \psi \rangle \).**

```plaintext
function SYNTHESIZE(\( \Sigma \), \( G_{1,a} \), \( G_{2,a} \))
    \( X := Q \), \( X := \emptyset \)
    while \( \bar{X} \neq X \) do
        \( X := \bar{X}, \bar{X} := \bigcap_{i=1}^{2} (\square G_{i,a} \land \square X) \)
    return \( X \)
```

from any mode to any other mode at any time, the resulting controller should be able to enforce \( \diamond\square G_{i,a} \) for \( i = \{1, 2\} \) at all times.

The first iteration of the algorithm computes \( \bigcap_{i=1}^{2} (\diamond\square G_{i,a}) \). These are the states from which there exist controllers that enforce \( \diamond\square G_1 \) and \( \diamond\square G_2 \). However, since the modes can change nondeterministically, this is not enough. These controllers also have to make sure that, the system always remains in the set \( \bigcap_{i=1}^{2} (\diamond\square G_{i,a}) \). Hence, in the next iteration we add this safety constraint and compute \( \bigcap_{i=1}^{2} (\diamond\square G_{i,a} \land \square \bigcap_{i=1}^{2} (\diamond\square G_{i,a})) \). We continue the same procedure until a fixed-point is reached.

Note that at each iteration of Algorithm 4, we compute a controller which enforces \( \diamond\square G_{i,a} \land \square K_{i-1} \), where \( K_{i-1} \) is the set of states computed in iteration \( i-1 \). This is a “reach and stay while stay” specification which is supported in PESSOA. Again first, PESSOA handles the safety part of the specification and then it computes the domain of the controller for the “reach and stay” part via Algorithm 6

In these algorithms, we denote the predecessor operator by, \( Pre_{\Sigma} : 2^Q \rightarrow 2^Q \), where \( Pre_{\Sigma}(Q') = \{ q \in Q \mid \exists u \in U : \delta(q, u) \in Q' \} \).

**Algorithm 5 Computation of \( \square K \).**

```plaintext
function STAY(\( \Sigma \), \( K \))
    \( X := Q \), \( X := \emptyset \)
    while \( \bar{X} \neq X \) do
        \( X := \bar{X}, \bar{X} := Pre_{\Sigma}(X) \land K \)
    return \( X \)
```

**Algorithm 6 Computation of \( \diamond\square K \).**

```plaintext
function REACHSTAY(\( \Sigma \), \( K \))
    \( X := Q \), \( X := \emptyset \), \( Y := Q \)
    while \( \bar{Y} \neq Y \) do
        \( Y := \bar{Y} \)
        while \( \bar{X} \neq X \) do
            \( X := \bar{X}, \bar{X} := (Pre_{\Sigma}(X) \land K) \cup Pre_{\Sigma}(Y) \)
            \( \bar{Y} := Y, \bar{Y} := \emptyset \)
        return \( Y \)
```

B. Solving the ACC problem

When using PESSOA to synthesize a controller, the first step is the computation of a discrete abstraction \( \Sigma \) of (3).
Fig. 4: The domain of the controller enforcing (9) on the system (3), i.e. $[\psi]$. Each point represents a discrete state. 90% of the points in the $h$-coordinate have been removed for visibility.

The computation is based on the algorithms described in [30] and supported by PESSOA. For our case, we chose the state space and input space discretization parameters to be $\eta = 0.5$ and $\mu = 0.2$, respectively. Moreover, we use $\Delta T = 0.5$ for the sampling time and the same problem parameter values as in Section III-D. Afterwards, we run the algorithms described in the previous subsection and obtain a controller that enforces the formula $\psi$ given in (9) on the system (3).

It took 30.65 seconds to synthesize the controller on a 3.4 GHz iMac. Fig. 4 illustrates the domain of the controller synthesized in PESSOA.

The controller synthesized by PESSOA is given by a large look-up table which is stored on the hard disk in terms of a binary decision diagram (BDD). Based on the current mode and state of the system, the BDD provides all inputs that are valid so that the closed-loop system satisfies the specification. It is straightforward to implement the controller on an embedded device, since it basically reduces to querying the BDD at each sampling time. The time to query the BDD is at least two magnitudes smaller than the sampling time and therefore does not constitute a problem.

V. EVALUATION AND COMPARISON IN CARSIM

The two controllers designed in Sections III and IV have been tested in a Simulink environment using the polynomial model (5). Fig. 5a shows simulation results in the following scenario: at $t = 0$ the speed of the ACC-equipped car is 5 m/s and there is a lead car 250 m away which leaves the lane at $t = 50$ s. A new car cuts 100 meters in front at $t = 100$ s and leaves the lane at $t = 150$ s. As can be seen, the formal requirements on input, velocity and time headway are satisfied for the two modes.

The controller from Section III has also been tested in CarSim, as are shown in Fig. 5b. A video is also available at http://web.eecs.umich.edu/cpswiki/public_media/.

VI. DISCUSSION ON ASSUMPTIONS

We made some simplifying assumptions with regard to the lead car speed (A.1) and the number of cars within the radar range (A.2). Next, we discuss implications of these assumptions and how they can be relaxed.

A cruise controller that has practical value must be able to handle a lead car that changes speed. Relaxing the assumption (A.1) introduces environmental uncertainty into the system and increases the dimension of the state space from two to three. The theory of approximate finite-state abstractions, upon which PESSOA is based, already supports adversarial environments such as a lead car with changing velocity. Similarly, for the invariant-set based approach, the notions of a controlled-invariant set and reachability can be replaced with those of a robustly controlled-invariant set and “robust” reachability for systems with uncertainty. The increase in the dimension of the model, from two to three states, will render the synthesis of a controller in PESSOA or based on invariant-sets more challenging from a computational point of view.

On the other hand, the assumption (A.2) is rather technical and it is adopted to avoid the pathological cases where lead cars cut into the lane infinitely often in a way to prevent maintaining the desired time headway while always in mode $M_2$. The synthesized controllers are still correct when this assumption is relaxed to either (i) $R_{2,2}(v, h) = D$ for all $(v, h) \in V \times H$ and there are finitely many resets, or (ii) $R_{2,2}(v, h) = G_2$ for all $(v, h) \in V \times H$.

VII. CONCLUSIONS

In this paper, we formalized the ACC problem using a hybrid dynamical system model with two modes and an LTL specification. Then, we presented two solution approaches to synthesize correct-by-construction control software for ACC. Both approaches rely on a fixed-point-based characterization of the LTL specification, one computing such fixed-points directly on the continuous state-space, the other on a finite-state abstraction of the nonlinear dynamics. Each approach has certain advantages and disadvantages: (i) Termination of fixed-point-based algorithms on continuous state-spaces is not guaranteed whereas termination is always guaranteed when working with finite-state abstractions. (ii) Approximate finite-state abstractions can be computed directly for nonlinear dynamics, whereas the current implementation of the invariant set based approach requires linearization of the dynamics and error bounds between the actual model and the linearized model. (iii) It is currently not possible to handle quantitative objectives in PESSOA, whereas the invariant-set based approach readily gives the designer the flexibility to do optimizations to shape the transient behavior or to encode “soft” constraints within MPC.

It can happen that an ACC system cannot maintain safety within specified limits on braking (e.g., (2)), such as in the event of a car cutting dangerously close into a lane. In this case, the ACC system is expected to alert the driver through automated emergency braking and/or other feedback cues, and hand control of the vehicle back to the driver. An advantage of the presented formal methods is that the safe set (i.e., control domain $D$) is explicitly computed and thus conditions for passing control to an emergency braking module are clearly defined.

ACKNOWLEDGEMENT

The work is supported by NSF Contract #CNS-1239037.
Fig. 5: Simulation results for (a) both controllers in Simulink and (b) the controller from Section III in Simulink and CarSim. The plots show, from top to bottom: speed, headway, time headway, and normalized applied wheel force.

REFERENCES