Formal Synthesis of Safe Stop Tactical Planners for an Automated Vehicle

Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 53-4 (2020) 445–452

Formal Synthesis of Safe Stop Tactical Planners for an Automated Vehicle ⋆

Jonas Krook *,** **Roozbeh Kianfar** * **Martin Fabian** **

* *Zenuity, Gothenburg, Sweden*
(e-mail: firstname.lastname@zenuity.com*)*
** *Department of Electrical Engineering,*
Chalmers University of Technology, Gothenburg, Sweden
(e-mail: {krookj,fabian}@chalmers.se*)*

Abstract: Automated vehicles need a safe back-up solution in the presence of system degradations since a driver cannot be expected to take control on short notice. In the event of a degradation, the vehicle is required to reach a minimal risk condition via a minimal risk maneuver. The activation of such maneuvers is safety critical, and a correct implementation of the tactical planner that takes the activation decision is paramount. One way to ensure correctness is to employ formal methods since they can provide proofs thereof. Earlier, a tactical planner was formally verified to activate a minimal risk maneuver if and only if a failure occurs. Formal verification has some drawbacks, so this paper investigates the applicability of using the tools Supremica and TuLiP to synthesize correct-by-construction tactical planners. These two tools amend some of the verification's drawbacks, but also introduce their own.

Keywords: Automated Driving, Supervisory Control Theory, Reactive Synthesis.

1. INTRODUCTION

This paper builds upon previous work where formal verification was applied during development of a planner for an automated vehicle (Krook et al., 2019). Instead of manual implementation followed by verification, this paper investigates how the two different formal synthesis methods *Supervisory Control Theory* (SCT) and *Reactive Synthesis* (RS) (Ramadge and Wonham, 1989; Kupferman et al., 2000) can be used in an industrial setting to automatically create safe and correct-by-construction software. These methods are applied to the same problem as in the earlier work, and are evaluated against each other, and against the earlier result.

Krook et al. (2019) apply *Linear Temporal Logic* (LTL) model checking (Baier and Katoen, 2008) in a concurrent fashion to develop a state machine, unfortunately called *supervisor*, that acts as a tactical planner (Michon, 1985) for an automated vehicle. This planner will be referred to as the *Manual Planner*. Its task is to coordinate two path planners and ascertain that a safe stop trajectory planner is activated and stops the car in a safe place if and only if a failure occurs. Krook et al. (2019) model key aspects of the system's software modules and abstracted vehicle dynamics in Promela and model checks it with Spin (Holzmann, 2003). As the requirements are formally verified during the development process, faults are avoided and the final software is correct with respect to the requirements.

Although formal verification is a versatile tool when designing, developing, or testing software, it has drawbacks. For instance, when Zita et al. (2017) used the formal verification tool Supremica to verify a lane change software module, a fault was detected and an offending trace was presented in the user interface. Such traces can be difficult to analyze and understand, and they do not give any suggestion on how to correct the fault.

Furthermore, formal verification often requires a separate model in a different syntax from the implementation [1] (Baier and Katoen, 2008). This can be problematic for several reasons. If the formal model is an abstracted version of the software, then it may be difficult to convert an offending trace back into something meaningful in terms of the implementation.

Additionally, the formal model and the software both need to be updated to reflect changes in the other. If they are not, any correctness proof is invalid. The updating can be done manually or automatically, where manual modeling of the software can introduce modeling errors, while automatic modeling might be difficult to implement. The development of the Manual Planner suffered from the latter point (Krook et al., 2019). Although the software implementation was done in a subset of a high level language, it was difficult to automate the modeling step, and the formal modeling was done manually.

By using formal synthesis, the hope is to free the time spent on implementation and verification of software and put it to use on development of requirements and models, and on validation of requirements. The assumption is that such a shift of effort would make the vehicle safer and the

⋆ This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

[1] As is the case in both former works presented above.

10.1016/j.ifacol.2021.04.059

development more efficient, as the development effort can then be focused on building the right and safe product. To facilitate such a shift, there is a need to understand the type of problems to which formal synthesis can be applied, and how subsystems and requirements shall be modeled in a useful way. This is not trivial; for instance, Krook et al. (2018) found that automatically fixing the fault found by Zita et al. (2017) based on the verification model did not work because of how the problem was modeled.

In this paper we investigate how the SCT tool Supremica (Malik et al., 2017) and the RS tool TuLiP (Filippidis et al., 2016) can formally synthesize correct-by-construction tactical planners. We further investigate how the process of applying these synthesis methods compare to the process of formal verification with Spin. In the end, we are interested in what methods are suitable for the development of provably safe tactical planners in an automotive setting. Therefore, we compare the synthesis results of Supremica and TuLiP (referred to as the *Supervisory Planner* and the *Reactive Planner*, respectively), both with each other and the Manual Planner, to find benefits and drawbacks of each method. There are a lot of formal synthesis tools available and the conclusions from Supremica and TuLiP cannot be extrapolated to all of them. However, in addition to specific results, our comparison indicates some general properties of the SCT and RS fields.

For instance, it seems like neither Supremica nor TuLiP can take a model with detailed vehicle dynamics and synthesize a generic tactical planner that is not dependent on absolute position. By considering other works on formal synthesis it seems like this is a general result for the two fields (Wongpiromsarn et al., 2013; Korssen et al., 2018; Ramezani et al., 2019). The work by Nilsson et al. (2016) seems to avoid the issue, but the result depends on relative coordinates and bounds on maximal values of the continuous states.

2. PRELIMINARIES

2.1 Supervisory Control Theory

Supervisory Control Theory (Ramadge and Wonham, 1989; Cassandras and Lafortune, 2010) is a model-based approach for control of *Discrete Event System* (DES). A DES is a dynamic system that can be characterized by a set of states whose transitions are triggered by occurrences of *events*. Given a DES to be controlled, the *plant* G, and a *specification* K describing the desired behavior, a control entity, called *supervisor* S, can be automatically synthesized to dynamically restrict the behavior of the plant, such that the closed-loop system satisfies the specification.

The supervisor and the plant form an asymmetric *closed-loop* system where the supervisor restricts the event generation of the plant. As the plant generates events enabled by the supervisor, the supervisor observes the generated sequences of events and determines, at each state, which of the currently possible events should be enabled. Some of the events cannot be disabled by the supervisor. These events are *uncontrollable*. The supervisor is required to be *controllable*, i.e. it must never try to disable an uncontrollable event. By disabling controllable events, the supervisor can confine the plant to a subset of its possible states so that the closed-loop system only visits states that are considered "good" by the specification. At the same time, the supervisor guarantees that from any closed-loop state, the system can always continue to a desired marked state, it is *non-blocking*. Finally, the supervisor minimizes the behavior that is removed, it is *maximally permissive*.

Basically, supervisor synthesis is an iterative removal of states and/or transitions of an initially calculated supervisor "candidate". Practically, this candidate is calculated from $G||K$ where $||$ denotes the full synchronous composition operator (Cassandras and Lafortune, 2010). The iterative algorithm removes from $G||K$ the states that break the controllability and/or the non-blocking properties. Iteration is necessary since enforcing one property may break the other. The iteration will eventually reach a fix-point, and what then is obtained is the maximally permissive supervisor. In Supremica, one way to model this supervisor candidate is to use an *Extended Finite Automaton* (EFA).

EFAs are automata extended with bounded discrete variables, and updates defining logical conditions over those variables. Let $V = \langle v_1, v_2, \ldots, v_n \rangle$ be an ordered set of variables, with each variable v_i associated to a finite discrete domain, $dom(v_i) = \{\hat{v_i}^1, \hat{v_i}^2, \ldots, \hat{v_i}^j\}$, having an initial value $\hat{v_i}^\circ \in dom(v_i)$, and a set of marked values $\hat{V}_i^m = \{\hat{v_i}^{m_1}, \hat{v_i}^{m_2}, \ldots, \hat{v_i}^{m_k}\} \subseteq dom(v_i)$. Define the domain of V as $dom(V) = dom(v_1) \times dom(v_2) \times \cdots \times dom(v_n)$.

Definition 1. An Extended Finite Automaton (EFA) is an 8-tuple:

$$E = \langle L, V, \Sigma, \rightarrow, L^i, L^m, \hat{V}^\circ, \hat{V}^m \rangle$$

where L is a finite set of locations; V is as above. Σ denotes a finite set of events; $L^i \subseteq L$ and $L^m \subseteq L$ denote the set of initial locations and marked locations, respectively; $\hat{V}^\circ = \{\hat{v}_i^\circ : i = 1, \ldots, n\}$ is the set of initial values; $\hat{V}^m = \{\hat{V}_i^m : i = 1, \ldots, n\}$ is the set of marked values; $\rightarrow \subseteq L \times \Sigma \times \Pi \times L$ is the extended transition relation where Π denotes the set of updates, which are formulas consisting of variables, integer constants, Boolean literals, as well as propositional logic and discrete arithmetic connectives. A *state* of an EFA is a tuple $(\ell, \hat{V})$ where $\ell \in L$ and $\hat{V} \in dom(V)$. Hence, the initial states are defined as $Q^i = L^i \times \hat{V}^\circ$ and the marked states as $Q^m = L^m \times \hat{V}^m$.

A transition between locations ℓ, ℓ' with event $\sigma \in \Sigma$ and update $p \in \Pi$ is written as $\ell \xrightarrow{\sigma:p} \ell'$. The transition can be fired if E is at location ℓ and the update p evaluates to true; consequently, E changes its location to ℓ' while updating the variables in p; variables not in p remain unchanged.

For SCT, the alphabet Σ of an EFA is partitioned into the disjoint sets of the controllable, Σ_c, and uncontrollable, Σ_u, event sets; $\Sigma = \Sigma_c \cup \Sigma_u$. An uncontrollable event $e_u \in \Sigma_u$ is prefixed by ! as a convention.

2.2 Reactive Synthesis

The Reactive Synthesis problem is to automatically synthesize a controller, referred to as a *reactive module*, that satisfies the desired *guarantees* ϕ_s, under the *assumptions* of the environment ϕ_e. In other words, the reactive module

satisfies the formula $\phi_e \rightarrow \phi_s$ (Bloem et al., 2014). One way to model the RS problem is to consider the reactive module and the environment as adversaries that play a finite-state game and take turns to provide input to each other (Wongpiromsarn et al., 2011). Then, an iterative process can be adopted to find a fix-point of a subset of states and transitions that solves the RS problem. The states, transitions, inputs and outputs can be modeled by a *Kripke structure*.

Definition 2. A Kripke Structure is a tuple

$$M = \langle S, I, R, AP, L_{AP} \rangle,$$

where S is a set of states; $I \subseteq S$ is a set of initial states; $R \subseteq S \times S$ is a transition relation; AP is a set of atomic propositions; and $L_{AP} : S \rightarrow 2^{AP}$ is a labeling function that defines the atomic propositions that are true in each state. AP is divided into two disjoint subsets AP_e and AP_s, representing the propositions of the environment and the reactive module, respectively.

The atomic propositions in AP_e are seen as the inputs to the reactive module, while AP_s are its outputs. The atomic propositions in AP can be composed of relational operators on functions of discrete finite domain variables as they can be considered either true or false, given a certain valuation in a certain state.

In TuLiP, the environment and the requirements on the reactive module are modeled with LTL. LTL formulas can be evaluated over infinite runs on a Kripke structure. In addition to standard propositional logic operators, LTL includes temporal operators (Pnueli, 1977). The temporal operators $'$ (next), $\Box$ (always), and $\Diamond$ (eventually) are used in this paper. A run π of a Kripke structure M is an infinite sequence of states $\{\pi_0, \pi_1, \dots\}$, where $\pi_0 \in I$ and $(\pi_i, \pi_{i+1}) \in R$. Let $\pi[i]$ represent the infinite run starting from state π_i. Let θ be an LTL formula, and ψ be an atomic proposition. The satisfiability of an LTL formula τ by π is given inductively:

- $\pi \vDash \tau$ iff $\pi[0] \vDash \tau$
- $\pi[i] \vDash \psi$ iff $\psi \in L_{AP}(\pi_i)$
- $\pi[i] \vDash \neg\tau$ iff $\pi[i] \nvDash \tau$
- $\pi[i] \vDash \tau \vee \theta$ iff $\pi[i] \vDash \tau$ or $\pi[i] \vDash \theta$
- $\pi[i] \vDash \tau'$ iff $\pi[i+1] \vDash \tau$
- $\pi[i] \vDash \Box\tau$ iff $\pi[k] \vDash \tau$ for all $k \geq i$
- $\pi[i] \vDash \Diamond\tau$ iff $\pi[k] \vDash \tau$ for some $k \geq i$

The Kripke Structure M satisfies a formula τ if every possible run π satisfies the formula.

Given an LTL formula φ modeling the environment and the requirements, TuLiP synthesizes a reactive module such that it satisfies φ, if such reactive module exists. In RS, such an LTL formula is called the *specification*, and the subformulas modeling the environment and requirements are called *environment* and *system* specifications, respectively. This paper uses *system* in a wider sense, and to be consistent, the formalization of the requirements will be referred to as the specification [2]. The LTL formula φ has to be in GR(1) form (Piterman et al., 2006):

[2] See the work by Ramezani et al. (2019) for the overlapping nomenclature of SCT and RS.

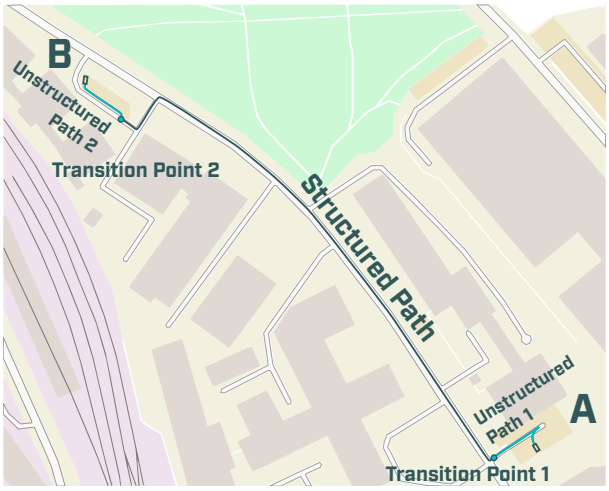

Fig. 1. A map showing a principal transport mission.

$$\varphi \triangleq \big((\psi^e_{init} \wedge \bigwedge_{\psi^e_{safe,i} \in \Psi^e_{safe}} \Box\psi^e_{safe,i} \wedge \bigwedge_{\psi^e_{live,i} \in \Psi^e_{live}} \Box\Diamond\psi^e_{live,i}) \rightarrow (\psi^s_{init} \wedge \bigwedge_{\psi^s_{safe,i} \in \Psi^s_{safe}} \Box\psi^s_{safe,i} \wedge \bigwedge_{\psi^s_{live,i} \in \Psi^s_{live}} \Box\Diamond\psi^s_{live,i})\big), \quad (1)$$

where ψ^e_{init} and ψ^s_{init} contain all the initial conditions of the environment and the reactive module, respectively. Ψ^e_{safe} and Ψ^e_{live} are the sets of *safety* and *liveness* assumptions on the environment. Ψ^s_{safe} and Ψ^s_{live} are the sets of safety and liveness guarantees of the synthesized planner. Note that TuLiP interprets the implication in (1) as a *strict realizability* implication (Klein and Pnueli, 2011).

3. SCENARIO

The KTH *Research Concept Vehicle* (RCV) is used as a target platform for the synthesized tactical planners. The RCV is a custom-built, fully electric and drive-by-wire concept vehicle, hosted by the Integrated Transport Research Lab at KTH, for validating and demonstrating research results (Kokogias et al., 2017). This section describes a use case that the RCV shall solve, and the system components that are available to do so.

3.1 Transport mission

The scenario considered is a transport mission where the RCV is initially parked in a parking spot in parking lot A, and receives a mission goal where it needs to drive to and park in a goal parking spot in parking lot B (Fig. 1). To do this, it first has to plan a path connecting the two parking lots via the road network. This path is called the *structured path* (SP). The start and end points of the SP are called transition point 1 and 2 (Tp1 and Tp2), respectively. Then the RCV needs to generate a path from a point in A to Tp1, called *unstructured path 1* (UP1). When the RCV arrives at parking lot B it needs to construct a path from Tp2 on the road to the goal parking spot in B, called *unstructured path 2* (UP2). The purpose of this paper is to synthesize and evaluate two different tactical planners that coordinate these tasks. One, called the Supervisory Planner, based on an SCT supervisor, and another, called the Reactive Planner, based on an RS reactive module.

3.2 Safe stop

The task of the tactical planner is to complete the transport mission safely. One purpose for having them is also to relieve the driver of driving tasks. However, a driver who is not participating in the driving tasks (or might not even be physically present) cannot reliably take over control of the vehicle in case of safety critical software system failures (Rudin-Brown and Parker, 2004; Merat et al., 2014). Therefore, the RCV needs a safe backup solution when such a system degradation occurs. One such backup solution is a *minimal risk maneuver*. When a safety critical system becomes degraded, the task of a minimal risk maneuver is to swiftly bring the automated vehicle to a *minimal risk condition*, which is supposed to be a safe state where the risk of harm or damage is minimal (European Commission, 2019). For instance, if an automated vehicle no longer 'knows' where it is on an otherwise empty low-speed road because of a GPS system failure, then it needs to stop, or it may drive off the road. In that scenario, being stopped on the road would be the minimal risk condition, and braking would be a minimal risk maneuver.

3.3 Architecture

Available to the tactical planner to complete the transport mission safely are the subsystems shown in the system architecture of Fig. 2. The figure shows how different subsystems are connected and in which way information flows. The different subsystems provide the following services:

- **Localization:** The localization subsystem uses a GPS sensor to position the RCV in a global map. It is also responsible for sending the goal position to the tactical planner. If the GPS sensor experiences a failure the localization subsystem cannot perform these tasks anymore and indicates this to the tactical planner.
- **Structure Area Path Planner (SPP)** and **Unstructured Area Path Planner (UPP):** These two path planners make plans in the road network (structured area) and in parking lots (unstructured areas), respectively. The tactical planner sends a start and goal location, and the path planner responds with a path connecting the locations, or a failure message. The SPP also provides the transition points (Tp1 and Tp2) between parking lots and the road network. The SPP and UPP are implemented based on the works by Bender et al. (2014) and Kutzer (2016), respectively.
- **Trajectory Planner (TP):** The trajectory planner receives a path from the tactical planner and generates *trajectories* of speeds and yaw angles by considering the vehicle dynamics and physical limits of the RCV (Kokogias et al., 2017). The trajectory planner also plans a stop at the end of the current path.
- **Safe Stop Trajectory Planner (SSTP):** The safe stop trajectory planner has a set of safe stop trajectories (SST) that are candidate minimal risk maneuvers that bring the RCV to a stop once they are activated. The set of trajectories is evaluated against the current position in the map, and the trajectory that is considered best is selected as the minimal risk maneuver (Svensson et al., 2018). The number of feasible trajectories is always communicated to the tactical planner.

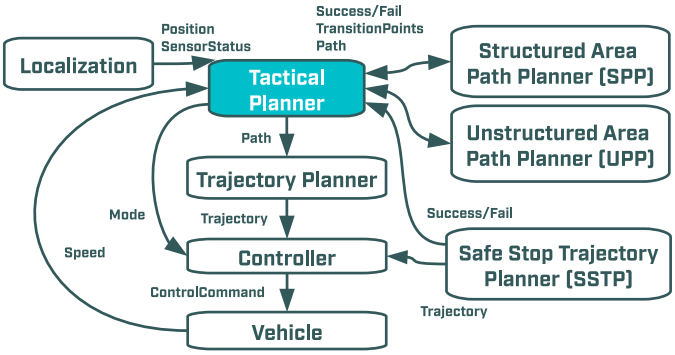

Fig. 2. The RCV architecture.

- **Controller:** The controller receives commands from three sources: TP, SSTP, and the tactical planner. The nominal operation is to execute the trajectories from TP, but depending on the mode sent by the tactical planner the controller either executes the minimal risk maneuver, or applies the brakes as hard as possible, referred to as *Automatic Emergency Braking* (AEB).

3.4 Requirements

The requirements that a tactical planner for the RCV must fulfill are based on the above subsections and the original requirements from Krook et al. (2019). There are mainly four requirements, presented here in natural language. The details of these requirements are discussed in Section 4.2 when they are formalized.

(i) The RCV shall reach the goal or a minimal risk condition, or emergency stop with AEB.
(ii) The RCV shall always eventually stop.
(iii) An SST is only allowed to be activated if failures are detected.
(iv) AEB is only allowed to be activated if no SST is available since the aggressive deceleration and stop position that results from AEB is considered less safe than the effects of a minimal risk maneuver.

Krook et al. (2019) proposed 7 requirements and also checked that the goal could be reached with the proposed planner. These 8 properties were called *missionComplete*, *stopInTheEnd*, *allPathsKnown*, *driveOnlyOnPaths*, *safeStop*, *unsafeStop*, *failure*, and *failToReachGoal*. Four of these are omitted: *driveOnlyOnPaths* refers to current position, which is abstracted away; the *allPathsKnown* requirement is replaced with requirements that specify that paths must be known before being activated; *failure* is removed because the sensor might fail very close to the goal, and then it is unnecessary to require activation of the safety systems; finally, the check *failToReachGoal* follows from the remaining four requirements.

4. RESULTS

This section presents the results of the modeling and syntheses, limited to the last path, UP2. The complete models cannot fit in this paper and are therefore supplementary material (Krook, 2020).

To the largest extent possible, the locations and automata in Supremica and the variables and values in TuLiP have

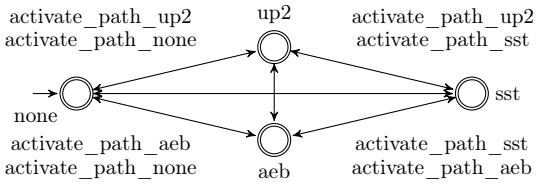

Fig. 3. The plant `active_path`.

been given the same names. For instance, the Supremica plant called `active_path` (Fig. 3) has the four locations "none", "up2", "sst", and "aeb", while the TuLiP variable `active_path` has the domain "none", "up2", "sst", and "aeb". Supremica allows updates on transitions to refer to automata and location names as if they were variables and domains (Definition 1), so the update `active_path` $\neq$ "none" in Fig. 4 means that the transition can only be taken if the automaton `active_path` in Fig. 3 is not in location "none".

The variables that are used in TuLiP are 'owned' by the environment or the Reactive Planner. The ownership is important and similar in effect to controllability in Supremica. The variables `goal`, `safe_stopped`, `emergency_stopped`, `sensor_failure`, `driving`, and `upp_2_response` are part of the environment model and are all initially false, except for `upp_2_response` which is "none". `active_path` and `up_2_available` are owned by the Reactive Planner, and they are initially "none" and false, respectively.

4.1 Vehicle, trajectory planner, and controller

When the Manual Planner was formally verified with Spin it was verified against a simple model of the RCV dynamics and the controller; a simple discrete point mass model in one dimension along the length of the paths (of arbitrary length), where braking distances for different speeds were pre-calculated. The trajectory planner passed the paths through from the tactical planner to the controller. As long as the controller had a path available and the RCV had not reached the end of the path, it was forced to travel forward such that it could stop before the end of the current path. Otherwise the controller would block all progress.

The Manual Planner can be implemented to operate on paths of arbitrary lengths, and then be verified against a path of arbitrary, but constant, length. Although that method does not give a proof for all different path lengths, it can be argued by induction that the result is general.

The synthesis methods evaluated in this paper have trouble to do the same. If a specific path length is chosen, then the Supervisory Planner and Reactive Planner have a dependence on specific positions along the path. Since the transport mission, and the paths needed to complete it, are not known a priori, this fixed-length path approach is not reasonable. Paths with variable lengths would solve that problem, but if the length of the paths are allowed to vary, the state space explodes, making the synthesis intractable.

The approach for the syntheses is to use a higher abstraction level for the RCV dynamics and controller. Instead of using a model of the dynamics and the controller,

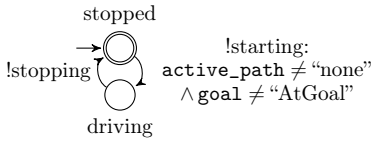

Fig. 4. Plant `state` describing when the RCV may drive.

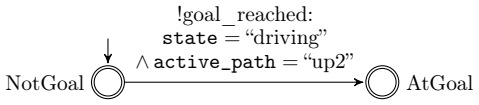

Fig. 5. Plant `goal` describing how the goal is reached.

a sort of assume-guarantee contract is provided to the syntheses. This contract is similar in SCT and RS, but have key differences because of the different semantics. Basically, both contracts state that the RCV only moves after the controller has received a path, and that the end of the path is only reached if the RCV is moving. There is also no possibility that the controller drives past the end point. However, when a path is supplied, the contract states in SCT that the RCV *might*, and in RS that the RCV eventually *does*, reach the end of the path. The plants for Supremica are shown in figures 4 and 5. The environment model for TuLiP is the following (where the set memberships at the end of the formulas indicate which part of the model in (1) they belong to):

$$\texttt{driving}' \rightarrow (\texttt{active\_path} \neq \text{“none”}) \in \Psi^e_{safe} \quad (2)$$

$$\texttt{goal} \rightarrow \neg\texttt{driving} \in \Psi^e_{safe} \quad (3)$$

$$(\neg(\texttt{driving} \wedge \texttt{active\_path} = \text{“up2”}) \wedge \neg\texttt{goal}) \rightarrow \neg\texttt{goal}' \in \Psi^e_{safe} \quad (4)$$

$$(\texttt{active\_path} = \text{“up2”} \wedge \neg\texttt{sensor\_failure}) \rightarrow \texttt{goal} \in \Psi^e_{live} \quad (5)$$

Another part of the assume-guarantee contracts is the effect of the sensor failure. If the global localization fails, the controller cannot guarantee that the end of the path is reached. For SCT it is part of the semantics that the uncontrollable event !goal_reached in the plant `goal` in Fig. 5 might not fire, so the effect of the sensor failure is implicitly included. For RS on the other hand it must be explicitly stated in (5) that `goal` is only guaranteed to be reached if the sensor has not encountered any failures.

4.2 Requirements

The four requirements from Section 3.4 all have simple formalizations. Figures 6 and 7 show the SCT-specifications for reaching one of the end points, and that activating the safe stop in absence of failures is forbidden, respectively. The specification for activating AEB is analogous to Fig. 7. The marking in Fig. 4 specifies that the RCV shall stop. These specifications closely resemble the original requirements used in the verification, but since the semantics differ they do not mean exactly the same. For instance, the marking in the plant `state` expresses that a supervisor may not restrict the RCV from stopping, while the original requirement expresses that the RCV must be stopped infinitely often.

The requirements for RS are in principle the same as for the Manual Planner, but GR(1) does not support

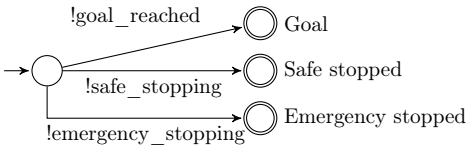

Fig. 6. Specification stating that the goal, safe stop, or emergency stop shall be reached.

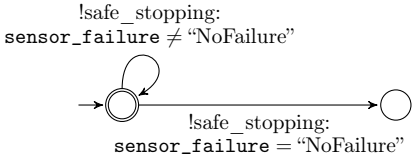

Fig. 7. Specification `safe_stop_req` stating that activation of the safe stop in absence of errors leads to a blocking state.

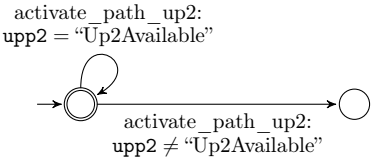

Fig. 8. Specification `request_path` stating that activation of UP2 when it is unavailable leads to a blocking state.

$\Diamond\Box$ so the formalization of requirement (i) must differ from *missionComplete*, $\Box\Diamond\texttt{goal} \vee \Diamond\Box\texttt{safe\_stopped} \vee \Diamond\Box\texttt{emergency\_stopped}$. For instance, once `safe_stopped` is reached the RCV shall not leave it, and that can be enforced by the two formulas $\texttt{safe\_stopped} \rightarrow \neg\texttt{driving} \in \Psi^e_{safe}$ and $(\neg\texttt{driving} \wedge \texttt{safe\_stopped}) \rightarrow \texttt{safe\_stopped}' \in \Psi^e_{safe}$; once the RCV has safely stopped it cannot be moving, and if it is not moving it cannot leave the safe stop. `emergency_stopped` can be made terminal analogously. Requirements (i)-(iv) thus become:

$$\texttt{goal} \vee \texttt{safe\_stopped} \vee \texttt{emergency\_stopped} \in \Psi^s_{live} \quad (6)$$

$$\neg\texttt{driving} \in \Psi^s_{live} \quad (7)$$

$$\texttt{safe\_stopped} \rightarrow (\texttt{sensor\_failure} \vee \texttt{up\_2\_failed}) \in \Psi^s_{safe} \quad (8)$$

$$\texttt{emergency\_stopped} \rightarrow (\texttt{sensor\_failure} \vee \texttt{up\_2\_failed}) \in \Psi^s_{safe} \quad (9)$$

Some extra requirements are needed that describe how the planner interacts with the environment. For instance, a path must be acquired before it can be activated. Fig. 8 shows how this is specified in SCT, and in RS it becomes:

$$\texttt{upp\_2\_response} = \text{“success”} \rightarrow \texttt{up\_2\_available}' \in \Psi^s_{safe} \quad (10)$$

$$\texttt{upp\_2\_response} \neq \text{“success”} \rightarrow \texttt{up\_2\_available}' \leftrightarrow \texttt{up\_2\_available} \in \Psi^s_{safe} \quad (11)$$

$$\texttt{active\_path} = \text{“up2”} \rightarrow \texttt{up\_2\_available} \in \Psi^s_{safe} \quad (12)$$

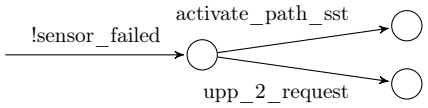

Fig. 9. A choice between two controllable events.

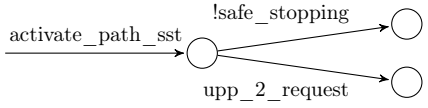

Fig. 10. A choice between firing a controllable event or not.

4.3 Synthesis Result

The synthesized planners are too big to display. The Supervisory Planner has 116 states which are difficult to inspect. The Reactive Planner has 73 states with some clear clusters. For instance, one cluster contains states where the creation of UP2 failed. Both synthesis methods require less than a second to synthesize the planners.

Visual inspection of these synthesized planners shows that they are similar, but still difficult to compare. The Supervisory Planner has many more allowed behaviors and only change one 'variable' between states. The Supervisory Planner allows all behaviors that the Reactive Planner allows, but not vice versa. For instance, the Reactive Planner only sets the path to "none" when one of "goal", "safe stop", or "emergency stop" is reached, but the Supervisory Planner allows the path to be "none" whenever the RCV is stopped. This complicates comparisons of the two planners, but does also have implications for applications. If the problem is best solved with a tactical planner that decides which action to take, then the Supervisory Planner requires more specifications to limit options.

The semantics and maximal permissiveness of SCT furthermore makes the Supervisory Planner difficult to implement as a planner; the Supervisory Planner protects from bad states, but gives no indication which controllable event that brings the RCV to a marked state. An event generator is needed that chooses what to do, but the implementation of such generator is not always clear. For instance, it is obvious that after a sensor failure the SST shall be activated (Fig. 9), but it is more difficult to select the correct action when the choice is to fire a controllable event or wait for an uncontrollable event (Fig. 10). Obviously, after the SST is activated there is no point in requesting UP2, but it is unclear how a generic event generator would come to that conclusion.

The Reactive Planner does not require an event generator. It is guaranteed to bring the system to a desired state, which makes it easy to implement. However, when there is a choice between two actions, TuLiP has to choose one, and the effects of that can be deceptive. For instance, the Reactive Planner fulfills requirement (iv) in that the AEB is only activated if no SST is available. However, in fact, this is not specified, as is evident from (9) and the following liveness assumption:

$$\texttt{active\_path} = \text{“aeb”} \rightarrow \texttt{emergency\_stopped} \in \Psi^e_{live}. \quad (13)$$

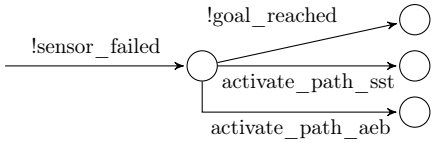

Fig. 11. The maximal permissiveness allows both SST and AEB after a sensor failure.

Formulas (9) and (13) allow AEB activation when an SST is available. Requirement (iv) is formalized the same way in Supremica, but looking at the fragment of the Supervisory Planner in Fig. 11 it is evident that it is not formalized correctly. So, if it is feasible to inspect the synthesis result, then it is possible to see this problem in the Supervisory Planner but *not always* in the Reactive Planner.

Additionally, inspection of the results is important because it can be difficult to understand exactly what is being specified. For instance, during the formalization of the requirements it was discovered that the Reactive Planner stopped the RCV by setting the path to "none". This is not how he RCV works; it has a stopping distance when driving. This was solved by the specification:

$$(\texttt{active\_path} = \text{“up2”} \wedge \texttt{driving}') \rightarrow \texttt{active\_path}' \neq \text{“none”} \in \Psi^s_{safe}.$$

However, in large models and with subtle corner cases, it might be difficult to find such instances without extensive inspection or testing, which defeats one goal of using formal methods, which is as a mean to overcome the hurdles of inspection and testing.

5. CONCLUSION

In this paper, an existing manually implemented and formally verified tactical planner for an automated vehicle is re-implemented using Supervisory Control Theory and Reactive Synthesis. Specifically, Supremica is used for the synthesis of an SCT Supervisory Planner, and TuLiP is used for the synthesis of an RS Reactive Planner. This paper investigates how these two tools can be used to synthesize tactical planners fulfilling the same requirements, and what adaptations of the model and requirements that are required to achieve that.

Since models and requirements were available from before, it was hypothesized that the synthesis effort would be low. However, this hypothesis assumes that the models and requirements can be used without much rework. This is not generally the case, since formal verification does not need a strict separation between the environment and the planner, while synthesis, especially in the case of TuLiP, needs separation to make sense. Also, during verification, the requirements are only used to verify some selected properties, and they are not necessarily complete; some requirements are implicitly defined in the tactical planner's implementation. So, although the requirements and model are available, it can be difficult to separate the environment model from the tactical planner model and to extract implicit requirements from the model of the tactical planner. This lack of separation slows the process.

Furthermore, the verification model was modeled in Promela, an imperative modeling language, whereas Supremica uses automata and TuLiP uses LTL. The Promela model must be translated to automata and LTL before synthesis, and this translation slowed the process. In the case of Supremica the event-based automata further complicate the modeling since 'variable' values must be translated to events. Possibly, the approach developed by Filippidis et al. (2015) might be faster when the initial model is written in Promela and LTL is used in synthesis.

The synthesis process makes it fast and easy to introduce new requirements, but it is under the proviso that the requirements are consistent. However, in practice, it may be very difficult to determine why a newly introduced requirement makes the system inconsistent, especially when designing large and complicated systems. Here the available synthesis tools provide no direct support and only present an empty result when the requirements are inconsistent. Supremica has the advantage of being able to simulate the uncontrolled system; such simulations can give clues as to where the issue lies. To us it is unknown whether this would be an approach that could be feasible to implement in an RS tool.

One benefit of employing synthesis is that a forced separation of the environment model and the requirements makes it very clear which assumptions that have been made and what is guaranteed, a distinction that can be difficult to discern in formal verification, and almost impossible in source code. Knowledge of the exact assumptions is important since they define the safe operational domain in which the tactical planner can operate. Especially TuLiP is good for this since the separation is forced into assume-guarantee statements, but Supremica also provides good separation into controllable and uncontrollable events.

One significant difference between the syntheses results and the verification is the model of the controller and the vehicle dynamics. The Manual Planner has a connection to a discrete point-mass model of the RCV, but the Supervisory Planner and the Reactive Planner only observe stopped/driving and 'know' that supplying a path eventually leads to the goal. In this application the higher-level abstraction is acceptable, but if requirements of the type 'do not crash' are desired, the abstraction pushes a lot of responsibility on correctness onto the controller. The compromise between general planners and specific requirements can be seen elsewhere. Synthesis with a close connection to continuous dynamics might require states for every driving location as shown in the works by Wongpiromsarn et al. (2013), while more generic supervisors might lack a direct connection to the continuous dynamics as shown in the works by Korssen et al. (2018). Nilsson et al. (2016) present two methods that are both generic and concrete, but the feasibility of the solutions depends on a maximal sensor range, and such constraints are not applicable for the synthesis in this paper.

REFERENCES

Baier, C. and Katoen, J.P. (2008). *Principles of Model Checking (Representation and Mind Series)*. The MIT Press.

Bender, P., Ziegler, J., and Stiller, C. (2014). Lanelets: Efficient map representation for autonomous driving. In *IEEE Intelligent Vehicles Symposium Proceedings (IV)*, 420–425. doi:10.1109/IVS.2014.6856487.

Bloem, R., Ehlers, R., Jacobs, S., and Könighofer, R. (2014). How to handle assumptions in synthesis. In K. Chatterjee, R. Ehlers, and S. Jha (eds.), *Proceedings 3rd Workshop on Synthesis*, volume 157 of *Electronic Proceedings in Theoretical Computer Science*, 34–50. Open Publishing Association, Vienna, Austria.

Cassandras, C.G. and Lafortune, S. (2010). *Introduction to Discrete Event Systems, 2nd Edition*. Springer.

European Commission (2019). Guidelines on the exemption procedure for the EU approval of automated vehicles.

Filippidis, I., Dathathri, S., Livingston, S.C., Ozay, N., and Murray, R.M. (2016). Control design for hybrid systems with TuLiP: The temporal logic planning toolbox. In *IEEE Conference on Control Applications (CCA)*, 1030–1041.

Filippidis, I., Murray, R.M., and Holzmann, G.J. (2015). A multi-paradigm language for reactive synthesis. In *Proceedings Fourth Workshop on Synthesis, SYNT 2015, San Francisco, CA, USA, 18th July 2015*, 73–97. doi:10.4204/EPTCS.202.6.

Holzmann, G. (2003). *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, first edition.

Klein, U. and Pnueli, A. (2011). Revisiting synthesis of GR(1) specifications. In S. Barner, I. Harris, D. Kroening, and O. Raz (eds.), *Hardware and Software: Verification and Testing*, 161–181. Springer, Berlin, Heidelberg.

Kokogias, S., Svensson, L., Pereira, G.C., Oliveira, R., Zhang, X., Song, X., and Mårtensson, J. (2017). Development of platform-independent system for cooperative automated driving evaluated in GCDC 2016. *IEEE Transactions on Intelligent Transportation Systems*, PP(99), 1–13. doi:10.1109/TITS.2017.2684623.

Korssen, T., Dolk, V., Van De Mortel-Fronczak, J., Reniers, M., and Heemels, M. (2018). Systematic model-based design and implementation of supervisors for advanced driver assistance systems. *IEEE Transactions on Intelligent Transportation Systems*, 19(2), 533–544. doi:10.1109/TITS.2017.2776354.

Krook, J. (2020). krooken/RCV-Synthesis: RCV-Synthesis v1.0.1. doi:10.5281/zenodo.3695638. URL `https://doi.org/10.5281/zenodo.3695638`.

Krook, J., Svensson, L., Li, Y., Feng, L., and Fabian, M. (2019). Design and formal verification of a safe stop supervisor for an automated vehicle. In *2019 International Conference on Robotics and Automation (ICRA)*, 5607–5613. doi:10.1109/ICRA.2019.8793636.

Krook, J., Zita, A., Kianfar, R., Mohejerani, S., and Fabian, M. (2018). Modeling and synthesis of the lane change function of an autonomous vehicle. *IFAC-PapersOnLine*, 51(7), 133–138. doi:https://doi.org/10.1016/j.ifacol.2018.06.291. 14th IFAC Workshop on Discrete Event Systems (WODES).

Kupferman, O., Madhusudan, P., Thiagarajan, P., and Vardi, M. (2000). Open systems in reactive environments: Control and synthesis. In *CONCUR 2000 — Concurrency Theory*, volume 1877 of *Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg.

Kutzer, K. (2016). *Path Planning in Unstructured Environments: A Real-time Hybrid A* Implementation for Fast and Deterministic Path Generation for the KTH Research Concept Vehicle*. Master's thesis, KTH, Royal Institute of Technology.

Malik, R., Åkesson, K., Flordal, H., and Fabian, M. (2017). Supremica – an efficient tool for large-scale discrete event systems. *IFAC-PapersOnLine*, 50(1), 5794 – 5799. 20th IFAC World Congress.

Merat, N., Jamson, A.H., Lai, F.C., Daly, M., and Carsten, O.M. (2014). Transition to manual: Driver behaviour when resuming control from a highly automated vehicle. *Transportation Research Part F: Traffic Psychology and Behaviour*, 27, 274 – 282. doi:10.1016/j.trf.2014.09.005.

Michon, J.A. (1985). A critical view of driver behavior models: What do we know, what should we do? In L. Evans and R.C. Schwing (eds.), *Human Behavior and Traffic Safety*, 485–524. Springer US, Boston, MA. doi:10.1007/978-1-4613-2173-6_19.

Nilsson, P., Hussien, O., Balkan, A., Chen, Y., Ames, A.D., Grizzle, J.W., Ozay, N., Peng, H., and Tabuada, P. (2016). Correct-by-construction adaptive cruise control: Two approaches. *IEEE Transactions on Control Systems Technology*, 24(4), 1294–1307. doi:10.1109/TCST.2015.2501351.

Piterman, N., Pnueli, A., and Sa'ar, Y. (2006). Synthesis of Reactive(1) designs. In E. Emerson and K. Namjoshi (eds.), *Verification, Model Checking, and Abstract Interpretation*, volume 3855 of *Lecture Notes in Computer Science*, 364–380. Springer.

Pnueli, A. (1977). The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, 46–57.

Ramadge, P.J.G. and Wonham, W.M. (1989). The control of discrete event systems. *Proceedings of the IEEE*, 77(1), 81–98.

Ramezani, Z., Krook, J., Fei, Z., Fabian, M., and Åkesson, K. (2019). Comparative case studies of reactive synthesis and supervisory control. In *2019 18th European Control Conference (ECC)*, 1752–1759. doi:10.23919/ECC.2019.8795696.

Rudin-Brown, C.M. and Parker, H.A. (2004). Behavioural adaptation to adaptive cruise control (ACC): implications for preventive strategies. *Transportation Research Part F: Traffic Psychology and Behaviour*, 7(2), 59 – 76. doi:10.1016/j.trf.2004.02.001.

Svensson, L., Masson, L., Mohan, N., Ward, E., Brenden, A.P., Feng, L., and Törngren, M. (2018). Safe stop trajectory planning for highly automated vehicles: An optimal control problem formulation. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, 517–522. doi:10.1109/IVS.2018.8500536.

Wongpiromsarn, T., Topcu, U., and Murray, R.M. (2013). Synthesis of control protocols for autonomous systems. *Unmanned Systems*, 01(01), 21–39.

Wongpiromsarn, T., Topcu, U., Ozay, N., Xu, H., and Murray, R. (2011). TuLiP: A software toolbox for receding horizon temporal logic planning. In *Proceedings of the 14th International Conference on Hybrid Systems: Computation and Control*, 313–314.

Zita, A., Mohajerani, S., and Fabian, M. (2017). Application of formal verification to the lane change module of an autonomous vehicle. In *13th IEEE Conference on Automation Science and Engineering (CASE)*, 932–937. doi:10.1109/COASE.2017.8256223.