Estimator-Based Reactive Synthesis under Incomplete Information

Rüdiger Ehlers
University of Bremen and DFKI GmbH
Bremen, Germany

Ufuk Topcu
University of Pennsylvania
Philadelphia, Pennsylvania, United States

ABSTRACT

Lack of complete run-time information about the environment behavior significantly increases the computational complexity and limits the applicability of *practical* reactive synthesis methods, e.g., synthesis from generalized reactivity(1) specifications. We tackle this difficulty by splitting incomplete-information controller synthesis into *estimator construction* and *complete-information synthesis* steps. The estimator, which executes in parallel to the controller, establishes approximations of the unobserved variables that are salient for the synthesis step. It essentially provides an abstraction from the belief space of the controller, whose exponential growth often plagues incomplete-information synthesis, by keeping track of only the properties of relevance for the specification engineer and the scenario under consideration.

We formalize an estimator notion for controller synthesis, and present a framework in which such estimators work in concert with controllers reacting partly to the estimator outputs to realize given temporal logic specifications. In order to limit the size and structure of the estimators, we focus on *positional* estimators in computation. Moreover, we demonstrate how such estimators are well-suited to be used in the context of generalized reactivity(1) synthesis. We illustrate the use of the estimator-based synthesis method on a running example motivated by intelligent transportation systems.

1. INTRODUCTION

Cyber-physical systems have controllers whose requirements can be specified concisely and completely, which opens up the opportunity to apply automated controller synthesis. Especially in cases in which one or more optimization criteria are given, an automated synthesis procedure allows to explore the space of feasible solutions and to pick the controller in a well-informed manner. Recent work in the area of practical reactive synthesis demonstrates that for smaller specifications, contemporary synthesis algorithms can already be applied [2, 9]. At the same time, new concepts are developed to deal with larger specifications and thus to extend the applicability of reactive synthesis even further [16, 19].

A drawback of most of these approaches is however that they cannot deal with *incomplete information*, meaning that not all information about the environment is available to the controller at runtime. For cyber-physical systems, this is a common case: measured values can deviate from real values and often the system does not have sensors for every physical property of the environment that occurs in the specification. While for some synthesis problems of already high complexity (e.g., synthesis from specifications in general linear-time temporal logic), incomplete information does not raise the complexity of the problems further [10], this fact does typically not hold for synthesis problems of lower complexity (hence of practical interest). In particular, for generalized reactivity(1) synthesis, which is a synthesis approach that is valued in control and robotics applications for its good scalability, incorporating incomplete information raises its time complexity from singly-exponential to doubly-exponential. A doubly-exponential complexity eliminates the simple symbolic encoding of the generalized reactivity(1) synthesis problem, and thus substantially weakens its scalability. This fact leads to the question of how we can avoid one of the exponential factors in the rate of complexity growth in the synthesis problem and keep the scalability of singly-exponential synthesis approaches even for settings with incomplete information.

In this paper, we present an approach to cyber-physical system controller synthesis under incomplete information that offers a singly-exponential time complexity and thus side-steps the complexity increase imposed by incomplete information. This reduction in complexity is accomplished by computing an *estimator* for the unobserved variables before performing the actual synthesis step. An estimator is an automaton that uses all information that the system can observe to derive sound bounds on the unobserved variables. It executes in parallel to the controller and provides it with information about the possible values of the unobserved variables. As such, estimators abstract the *belief space* of a controller. The belief space represents all physical system states that the controller must consider to be possible at a point in time in order to operate correctly according to the modeling of the physical environment. Belief spaces are commonly of size exponential in the number of physical system states and responsible for the high complexity of synthesis under incomplete information. Our estimators abstract from this

HSCC '15, April 14 - 16, 2015, Seattle, WA, USA

ACM 978-1-4503-3433-4/15/04 ... $15.00.
http://dx.doi.org/10.1145/2728606.2728626.

belief space by only keeping track of the properties defined by the specification engineer and that are of relevance for the scenario.

Example: As an example, consider a car that is equipped with an automatic distance keeping assistant. The car has a sensor that measures the approximate distance to the leader car and a sensor for its own speed. At every point in time, some minimum distance is to be maintained, and the acceleration and speed of both cars are bounded by some constants. Updating the distance between the cars is subject to noise due to imperfect steering and influence of wind. Suppose that it is the objective of the follower car to drive close to the leader car without risking to go below the minimum allowable distance. This task is difficult as the follower car only has a noisy distance sensor. Using terms from reactive synthesis, the actual distance between the cars can be considered to be *unobserved* as there is no precise measurement available. Equally, the speed of the car ahead is unobserved, and information about possible speed values can only be obtained by deriving them from the evolution of the measured distances and the follower car's speed values.

We can automatically obtain a controller for the scenario in this example by first synthesizing an estimator that always keeps track of the minimal and maximal possible distances and the minimum and maximum speed values of the leader car. In addition to the sensed values, the estimator also has the chosen acceleration of the controller to be synthesized as input. Using these bounds on the actual physical quantities, the synthesis step can then be performed, and the resulting controller is implemented in parallel to the estimator.

In order for the estimator to compress the belief space to a representation that can be handled in practical synthesis approaches, we restrict it to base its next estimate only on the last and current input and output of the system and the previous estimate, and call such estimators *positional*. While this approach reduces the precision in some applications, we can avoid the exponential blow-up due to the belief space in this way. To use an estimator in synthesis, the controller specification has to be concerned with observable or estimated variables only. In this way, variables that are not input to the controller (and are hence *unobservable*) vanish completely from consideration in the main synthesis step and we can treat the problem as one of complete information at that point.

In our example, instead of specifying that the minimum distance between the cars never drops below some value, we would instead require the controller to ensure that the estimator always outputs that its minimum estimate is above the critical value. As the estimation is always conservative, the intended specification follows.

Our estimator computation technique is not tied to a particular synthesis approach. For the demonstration of its applicability, we will however use it in the context of generalized reactivity(1) synthesis, where we discretize the continuous values of the physical domain into finite ranges. This modification allows us to apply well-established synthesis tools using binary decision diagrams (BDD) as symbolic reasoning engines for a proof of concept. To improve the scalability of the synthesis process and the estimator computation further, we also present a technique to perform *state space compression* that uses relationships between observed and estimated variables in order to reduce the size of the BDD representation of the estimator.

1.1 Related Work

In control design, estimators in various forms have played a central role in cases in which states of the system under control are not directly measured [11]. More relevant to our work are the notions of observability and estimators for systems with discrete-valued variables. For example, observability in finite-state automata with limited direct measurements was studied in [13, 4]. The latter reference also provided encodings of logic-based dynamic observers. Similarly, deterministic finite state observers acting in conjuction with full-state feedback controls were used in [17]. The notion of "immediate observability" in [12] limits the information used in estimation similarly to the positional estimators we discuss. Estimation of bounds on discrete variable values in cases in which the continuous variables are available for direct measurement was subject to [6]. Finally, observers for hybrid systems, composed of a location observer and a continuous observer, were studied in [1].

The problem of synthesizing controllers that operate under incomplete information is well-researched. While there are a number of models, including partially observable Markov decision processes [15] and two-player games with merely non-deterministic choices for the players, we focus on the latter model for performing synthesis in this paper. A classical result in this area is that solving games in which only one of the players can fully observe the play in the game has an exponential time complexity, even for very simple game types [14]. Thus, the complexity induced by computing a *belief set* in the game is unavoidable.

Estimators have already been used in the scope of distributed systems [8], where they observe the interface of a distributed system and compute estimates for its internal state. The estimators considered in this context were precise, and thus subject to state space explosion. Consequently, they are of little use to reduce the complexity of a controller synthesis problem for cyber-physical systems.

Velner and Rabinovich [18] considered the problem of noisy input in controller synthesis from a theoretical perspective. They identified decidable and undecidable cases of the reactive synthesis problem where input is incorrect in some steps and the controller may or may not observe this. They did not consider additional sources of incomplete information, such as measurements that yield values that are close to the real ones.

1.2 Structure of the Paper

We start by recalling some preliminaries in the next section. Then, we give a description of estimators in Section 3. In Section 4, we continue by describing the car following scenario that we introduced above and will use as case study for estimator-based synthesis. We show how to compute estimators in Section 5, perform synthesis using them in Section 6 and describe a state space reduction technique to further increase scalability in Section 7. The experiments performed on the car following scenario are given Section 8 followed by the concluding remarks in Section 9.

2. PRELIMINARIES

We now overview the main concepts needed in the subsequent sections.

Sets and words: Let AP be a set of *atomic propositions*. A word over AP is a finite or infinite sequence $w = w_0 w_1 \dots$ with $w_i \subseteq \mathsf{AP}$ for every $i < |w|$. Words can represent the execution of a *controller* or an *estimator*, which we define in the next section. Both are also called *finite state machines* (FSM). The proposition set AP can be partitioned into the input and output propositions of the FSM. We treat valuations of the propositions in AP to $\{\mathbf{false}, \mathbf{true}\}$ interchangably to sets that represent only the propositions that are meant to be **true**. For simplicity, we use elements from $2^X \times 2^Y$ and from $2^{X \cup Y}$ for disjoint sets X and Y interchangably. For $v \in 2^{X \cup Y}$, we denote by $v|_X$ the elements in $v \cap X$ and by $v|_Y$ the elements in $v \cap Y$.

Specifications: In order to reason about the behavior of an FSM, we can compare it against a *specification*. Formally, specifications are subsets of $(2^{\mathsf{AP}})^\omega$, i.e., sets of (infinite) words that are considered to satisfy the specification. Logics such as linear temporal logic (LTL) are commonly used to represent such subsets of $(2^{\mathsf{AP}})^\omega$ concisely. In addition to standard propositional logic, LTL uses temporal operators such as X ("next"), G ("globally") and F ("eventually") to allow reasoning over the valuation of the propositions along different places in a word. For example, the property $\mathsf{G}(r \rightarrow \mathsf{F}g)$ holds along all words in which an occurrence of r in the word is always followed by an occurrence of g at some later point. A full definition of LTL that includes all operators and not only the ones that we use in this paper can be found in the literature on this topic (see, e.g., [3]). In addition to LTL, we also use a *strong implication operator* $\rightarrow_s$. Intuitively, a property $\psi \rightarrow_s \psi'$ holds along a word if either ψ' holds or ψ is violated **before** ψ' can be observed not to hold. A formal definition of this concept has been given by Bloem et al. [3].

Finite-state machines: Finite-state machines are defined as tuples $\mathcal{M} = (S, \mathsf{AP}_I, \mathsf{AP}_O, s_0, \delta)$ with the (finite) set of states S, the set of input propositions AP_I, the set of output propositions AP_O, the initial state $s_0 \in S$, and the (partial) transition function $\delta : S \times 2^{\mathsf{AP}_I} \rightharpoonup S \times 2^{\mathsf{AP}_O}$. Given an *input stream* $w^I = w_0^I w_1^I \dots \in (2^{\mathsf{AP}_I})^\omega$, we say that $\mathcal{M}$ produces an output stream $w^O = w_0^O w_1^O \dots \in (2^{\mathsf{AP}_O})^\omega$ along with a *run* $\pi = \pi_0 \pi_1 \dots \in S^\omega$ if $\pi_0 = s_0$ and for all $i \in \mathbb{N}$, we have $\delta(\pi_i, w_i^I) = (\pi_{i+1}, w_i^O)$. The corresponding word (also called *trace*) of $\mathcal{M}$ is then $w = (w_0^I, w_0^O)(w_1^I, w_1^O) \dots$. A trace can also be finite if δ does not offer a next state and next output for some combination of state and input. For machines whose traces should satisfy some specification, finite traces are considered if all extensions of the finite trace trivially satisfy the specification. The set of all (infinite) traces of a machine is called its *langauge*, denoted as $\mathcal{L}(\mathcal{M})$. The type of finite-state machines used in this paper are also called *Mealy machines* in the literature.

Controller synthesis: Given a specification φ (e.g., in LTL) over a set $\mathsf{AP} = \mathsf{AP}_I \cup \mathsf{AP}_O$ of atomic propositions, the *reactive synthesis problem* is to compute a finite-state machine $\mathcal{M} = (S, \mathsf{AP}_I, \mathsf{AP}_O, s_0, \delta)$ all of whose words satisfy φ. A specification is called *realizable* if there exists such an FSM. A slight variation of the realizability and synthesis problems is to add *incomplete information*: φ then ranges over $\mathsf{AP} = \mathsf{AP}_I \cup \mathsf{AP}_O \cup \mathsf{AP}_H$, where AP_H is the set of *hidden* variables. The correctness requirement in this case is that for all words $w = w_0 w_1 \dots \in (2^{\mathsf{AP}_I \cup \mathsf{AP}_O})^\omega$ produced by $\mathcal{M}$ and all sequences of hidden variable valuations $w^H = w_0^H w_1^H \dots \in (2^{\mathsf{AP}_H})^\omega$, we must have that $(w_0 \cup w_0^H)(w_1 \cup w_1^H) \dots$ satisfies φ.

Generalized reactivity(1) synthesis: Generalized reactivity(1) synthesis [3], which is commonly abbreviated as GR(1) synthesis, is a synthesis approach with a comparably low computational complexity and good scalability in practice. It supports specifications of the form

$$\varphi = (\varphi_i^a \wedge \varphi_s^a \wedge \varphi_l^a) \rightarrow_s (\varphi_i^g \wedge \varphi_s^g \wedge \varphi_l^g). \tag{1}$$

The elements on the left-hand side of the strict implication are called the *assumptions* while the right-hand part of the implication forms the *guarantees*. All parts φ_γ^a and φ_γ^g for $\gamma \in \{i, s, l\}$ are conjunctions of properties in LTL. The parts φ_i^a and φ_i^g represent initialization assumptions and guarantees, which are formulas that are free of temporal operators. The parts φ_s^a and φ_s^g represent safety assumptions and guarantees, which are of the form $\mathsf{G}(\psi)$, where the only temporal operator that can occur is X and may not be nested. Finally, the parts φ_l^a and φ_l^g represent liveness assumptions and guarantees, which are of the form $\mathsf{GF}(\psi)$, where the constraint imposed on ψ is the same as in the previous case.

To simplify the notation, we assume henceforth that φ_i^a and φ_i^g represent a unique variable valuation to $\mathsf{AP}_I \cup \mathsf{AP}_O$ and that the finite-state machine to be synthesized from φ does not need to produce the initial valuation. All techniques presented in the following would still work without this assumption, though the notation would need to be more complicated.

Due to the special structure of the safety properties, we can represent them in an alternative form as sets of tuples over $2^{\mathsf{AP}_I \cup \mathsf{AP}_O}$. Let $\mathsf{G}\psi$ be such a safety constraint. We can replace the constraint by the set of tuples (x, x') for which a word starting with xx' satisfies ψ. As the temporal operators may not be nested in ψ, the other elements in the word do not matter. Taking the conjunction between safety constraints then amounts to taking the intersection of the respective sets. Note that for constraints in φ_s^a, which typically do not refer to AP_O in the scope of an X operator, the constraints can be represented as sets of elements in $2^{\mathsf{AP}_I \cup \mathsf{AP}_O} \times 2^{\mathsf{AP}_I}$.

One source of the high efficiency of the GR(1) synthesis approach is the fact that if there exists an implementation for the specification, then there exists one in which the set of states is defined as $S = 2^{\mathsf{AP}_I \cup \mathsf{AP}_O} \times \{1, \dots, n\} \times \{1, \dots, m\}$, where $n \geq 1$ is the number of liveness assumptions and $m \geq 1$ is the number of liveness guarantees. At the same time, a state (s, i, j) is always reached via transitions for the input proposition valuation $s|_{\mathsf{AP}_I}$ and the output proposition valuation $s|_{\mathsf{AP}_O}$. If no liveness assumptions or guarantees are present, we can then also represent the transition function of such an FSM by a set of transitions between the states, i.e., by a set $\rho \subseteq 2^{\mathsf{AP}_I \cup \mathsf{AP}_O} \times 2^{\mathsf{AP}_I \cup \mathsf{AP}_O}$.

Encoding integers: While we assume all propositions to be boolean in this paper, this restriction does not mean that the propositions do not encode values from a more complex type. In particular, we will encode *integer variables* into boolean variables. Given some integer variable v with the domain $\{k, \dots, n-1\}$, we need $\lceil \log_2(n-k) \rceil$ propositions to encode a value. If, for example, v is supposed to be the output of a finite-state machine, the propositions would all be contained in AP_O. We can then use linear equations over the variables to encode, for example, safety constraints. As an example, $\mathsf{G}(v \neq v')$ means that before and after a tran-

sition of an FSM, the variable v should not have the same value. We use primed variables to represent the variable's valuation in the next step as the LTL operator X is reserved for boolean values. We use equations of the form $a = b \pm c$ as shorthand for $(a + c \geq b) \wedge (a \leq b + c)$.

Parallel composition: Finite-state machines can run in parallel in order to jointly perform a certain control task. Given a set of input propositions $\mathsf{AP}_I = \mathsf{AP}_{I,1} \cup \mathsf{AP}_{I,2}$ to both machines together, two FSMs $\mathcal{M}_1 = (S_1, \mathsf{AP}_{I,1}, \mathsf{AP}_{O,1}, s_{0,1}, \delta_1)$ and $\mathcal{M}_2 = (S_2, \mathsf{AP}_{I,2}, \mathsf{AP}_{O,2}, s_{0,2}, \delta_2)$ produce a word $w = w_0 w_1 \ldots \in (2^{\mathsf{AP}_I \cup \mathsf{AP}_{O,1} \cup \mathsf{AP}_{O,2}})^\omega$ and two traces π_1 and π_2 such that for all $i \in \mathbb{N}$, we have $\delta_1(\pi_{1,i}, w_i|_{\mathsf{AP}_{I,1}}) = (\pi_{1,i+1}, w_i|_{\mathsf{AP}_{O,1}})$ and $\delta_2(\pi_{2,i}, w_i|_{\mathsf{AP}_{I,2}}) = (\pi_{2,i+1}, w_i|_{\mathsf{AP}_{O,2}})$. We require that $\mathsf{AP}_{O,1}$ and $\mathsf{AP}_{O,2}$ do not overlap and that only one of the FSMs can read from the other one, so that we cannot have $\mathsf{AP}_{I,1} \cap \mathsf{AP}_{O,2} \neq \emptyset$ and $\mathsf{AP}_{I,2} \cap \mathsf{AP}_{O,1} \neq \emptyset$ at the same time. A specification for the overall setting ranges over $\mathsf{AP}_I \cup \mathsf{AP}_{O,1} \cup \mathsf{AP}_{O,2}$ plus possibly some hidden propositions, which are taken into account as explained above.

3. ESTIMATORS FOR CONTROLLER SYNTHESIS

Estimators collect information about the state of an environment that is partially unobservable. They use available inputs (such as noisy measurements) and derive information about the environment's state using the description of its possible evolutions.

The motivation for computing estimators is due to their usefulness in a reactive synthesis process. If the requirements of a controller to be constructed can be specified only over the observable input and output of it and an estimator, we have effectively eliminated all unobservable variables in the controller specification. This modification allows us to treat the synthesis problem as one under complete information instead of one over incomplete information, which greatly reduces complexity.

Let us now formalize the notion of an estimator. For the scope of this section, we fix AP_{obs} to be a set of observable variables, AP_{est} to be a set of *estimation* variables, and AP_{hid} to be a set of *hidden variables*. We assume that AP_{obs} can be decomposed into the action set AP_{act} and the input set AP_{inp}. Furthermore, $\rho_e \subseteq (2^{\mathsf{AP}_{obs} \cup \mathsf{AP}_{hid}})^2$ and $\rho_s \subseteq (2^{\mathsf{AP}_{obs} \cup \mathsf{AP}_{hid} \cup \mathsf{AP}_{est}})^2$ are environment and estimator safety specifications, respectively. Both are in the form of sets of tuples that describe the allowed transitions. We also let $x_0 \subseteq \mathsf{AP}_{obs} \cup \mathsf{AP}_{est} \cup \mathsf{AP}_{hid}$ be an initial variable valuation.

Definition 1. *We call a finite-state machine $\mathcal{M} = (S, \mathsf{AP}_{obs}, \mathsf{AP}_{est}, \delta, s_0)$ an* estimator *if for every word $w = w_0 w_1 w_2 \ldots \in \mathcal{L}(\mathcal{M})$ and every sequence $h = h_0 h_1 h_2 \ldots \in 2^{\mathsf{AP}_{hid}}$ of hidden variable valuations, it holds that $x_0(w_0 \cup h_0)(w_1 \cup h_1) \ldots \models (\rho_e \rightarrow_s \rho_s)$.*

In the definition of a specification for an estimator, ρ_e intuitively encodes how the environment can evolve, and ρ_s describes what information about the environment the estimator is required to compute. Note that Definition 1 does not describe requirements over the size (or structure) of the estimator or which valuations to AP_{est} it prefers whenever there is more than one possible valuation at a time. The following definitions detail both of these points.

We start by introducing *history-freedom*. Consider an estimator $\mathcal{M}$ reading the observable part of an environment's evolution. For an estimator to be history-free, we require it to be able to work correctly even if the estimator suddenly starts to obtain the observable part of a different evolution of the environment as its input stream. This means that the estimates it produces should be correct for the second evolution after switching the estimator's input. The only requirement on the two evolutions for the switching to make sense is that they match on their observable variables' values at the switching point. More formally, we can define history-freedom as follows:

Definition 2. *Let $\mathcal{M}$ be an estimator and $r = r_0 r_1 r_2 \ldots \in 2^{\mathsf{AP}_{obs} \cup \mathsf{AP}_{hid}}$ and $r' = r'_0 r'_1 r'_2 \ldots \in 2^{\mathsf{AP}_{obs} \cup \mathsf{AP}_{hid}}$ be two arbitrary evolutions of the environment, i.e., such that $(r_i, r_{i+1}) \in \rho_e$ and $(r'_i, r'_{i+1}) \in \rho_e$ for every $i \in \mathbb{N}$. We pick some cut points j and $j' \in \mathbb{N}$ such that $r_j|_{\mathsf{AP}_{obs}} = r_{j'}|_{\mathsf{AP}_{obs}}$ and assemble r and r' to $\tilde{r} = \tilde{r}_0 \tilde{r}_1 \ldots = r_0 r_1 \ldots r_{j-2} r_{j-1} r'_{j'} r'_{j'+1} r'_{j'+2} \ldots$.*

Let $w = w_0 w_1 \ldots$ be the run of $\mathcal{M}$ for $\tilde{r}$. If, regardless of the initial choice of r, r', j, and j', we have that w correctly estimates r up to reaching step j, and correctly estimates r' later, we say that $\mathcal{M}$ is history-free. *Formally, this requirement amounts to having $((w_i \cup r_i|_{\mathsf{AP}_{hid}}), (w_{i+1} \cup r_{i+1}|_{\mathsf{AP}_{hid}})) \in \rho_s$ for all $i < j$, and having $((w_i \cup r'_{i-j'+j}|_{\mathsf{AP}_{hid}}), (w_{i+1} \cup r_{i-j'+j+1}|_{\mathsf{AP}_{hid}})) \in \rho_s$ for all $i \geq j$.*

Definition 3. *We call an estimator $\mathcal{M} = (S, \mathsf{AP}_{obs}, \mathsf{AP}_{est}, \delta, s_0)$* positional *if (1) $\mathcal{M}$ is history-free, (2) we have $S = 2^{\mathsf{AP}_{obs} \cup \mathsf{AP}_{est}}$, and (3) for every $(s', y) = \delta(s, x)$ for some $s, s' \in S, x \subseteq \mathsf{AP}_{obs}, y \subseteq \mathsf{AP}_{est}$, we have $s' = x \cup y$.*

Since a positional estimator has $S = 2^{\mathsf{AP}_{obs} \cup \mathsf{AP}_{est}}$ as the state space, the size of $\mathcal{M}$ is fixed and limited. The fact that positional estimators are history-free allows the estimator to forget how an estimate was derived once it has been made. This property gives a foundation for the estimator to recover gracefully in cases in which the environment deviates from ρ_e temporarily, as it ensures that the precise history of the environment's evolution is only of limited importance to the estimator.

The estimator specification ρ_s can be quite coarse. For example, it may state that, at every point in time, $(min\_b \leq b)$ and $(max\_b \geq b)$ must hold for some integer variable b encoded into AP_{hid} and some integer variables max_b and min_b of the same domain in the estimation propositions. An estimator can simply always output the minimal and maximal values in the domain of b. Clearly, such crude estimates are not desired, and we focus on *optimal estimators* instead.

Definition 4. *Given a partial order $\leq_E$ over valuations in AP_{est}, an estimator $\mathcal{M} = (S, \mathsf{AP}_{obs}, \mathsf{AP}_{est}, \delta, s_0)$ is* optimal *if, for every input sequence $w^I = w^I_0 w^I_1 w^I_2 \ldots \in (2^{\mathsf{AP}_{obs}})^\omega$ that satisfies ρ_e and for the word $w = w_0 w_1 w_2 \ldots$ induced by $\mathcal{M}$ for w^I, there is no other estimator that induces a word $w' = w'_0 w'_1 w'_2 \ldots$ that is a better approximation. We say that w' is a better approximation than w if $w \neq w'$ and for all $j \in \mathbb{N}$, $w'_j|_{\mathsf{AP}_{est}} \leq_E w_j|_{\mathsf{AP}_{est}}$.*

We say that $\mathcal{M}$ is an optimal positional estimator if there is no other positional estimator that offers a better approximation for some input sequence.

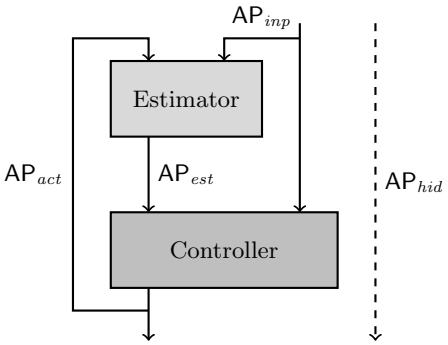

Figure 1: Using an estimator in combination with a controller. Edges denote how proposition values are propagated between the compontents. The hidden variables AP_{hid} can be read by neither the estimator nor the controller though the (original) controller specification may depend on them.

To conclude the problem statement, let us discuss how we can use an estimator to simplify a synthesis problem under incomplete information. Such simplification essentially provides the motivation for computing estimators. The overall architecture for the resulting system is depicted in Figure 1: we want to synthesize a controller for a setting with hidden propositions that works correctly when supplied with information from a given estimator. The following proposition formalizes the idea.

PROPOSITION 1. *Let $\mathcal{M}$ be an estimator, φ be an LTL specification over $\mathsf{AP}_{obs} \cup \mathsf{AP}_{hid}$, and φ' be an LTL specification over $\mathsf{AP}_{obs} \cup \mathsf{AP}_{est}$ such that $(\varphi' \wedge \mathsf{G}\,\rho_e \wedge \mathsf{G}\,\rho_s) \rightarrow \varphi$.*

If we can obtain a controller that reads AP_{inp} and AP_{est}, writes AP_{act}, and satisfies φ' starting from $x_0 \setminus \mathsf{AP}_{hid}$ when running in parallel to $\mathcal{M}$ along the sequences of AP_{obs} that induce infinite words by $\mathcal{M}$, then the parallel composition also satisfies φ starting from x_0 along all inputs sequences that satisfy $\mathsf{G}\,\rho_e$.

PROOF. Since $\mathcal{M}$ is guaranteed to satisfy $\mathsf{G}\rho_s$ along all observable sequences that satisfy $\mathsf{G}\rho_e$, we know that if φ' holding together with $\mathsf{G}\rho_e$ (whose satisfaction is ensured by the environment) and $\mathsf{G}\rho_s$ (whose satisfaction is ensured by the estimator) implies φ, then any FSM for φ' composed with $\mathcal{M}$ must also satisfy φ on a word that satisfies $\mathsf{G}\rho_e$ and for which $\mathcal{M}$ produces an infinite trace. Since we require $\mathcal{M}$ to do so for any trace that satisfies $\mathsf{G}\rho_e$, the claim follows. □

Note that neither the estimator FSM nor the modified specification φ' in Proposition 1 deals with the hidden variables. Thus, these variables have effectively been removed from the problem under consideration. This modification allows us to apply standard reactive synthesis techniques for problems under complete information. In order to account for the definition of $\mathcal{M}$, φ' can be altered to require the controller to be synthesized to simulate the behavior of $\mathcal{M}$ along with its correct reaction to them.

Note that in a reasonable estimator specification, we have that, for all $x \subseteq \mathsf{AP}_{obs} \cup \mathsf{AP}_{hid} \cup \mathsf{AP}_{est}$ and all $\bar{x} \subseteq \mathsf{AP}_{obs} \cup \mathsf{AP}_{hid}$, the set $\{y \subseteq \mathsf{AP}_{est} \mid (x, \bar{x} \cup y) \in \rho_s\}$ is non-empty, as otherwise an estimator may be required to deadlock in some cases. Henceforth, we assume ρ_s to have this property. Likewise, we assume ρ_e to be deadlock-free.

In Proposition 1, we did not require an estimator to be optimal, as the proposition is only concerned with soundness and not with completeness. However, the better an estimator is, the higher the chances are that the combination of φ' and $\mathcal{M}$ is found to be realizable by the synthesis procedure. It thus makes sense to focus on computing optimal estimators for using them in a synthesis process. Optimal estimators are not necessarily unique, however, raising the question which estimator should be chosen among the optimal ones. We will show in Section 5 that a preference relation $\leq_E$ with the following property will ensure the uniqueness of an optimal positional estimator.

DEFINITION 5. *Let ρ_s be an estimator specification over $\mathsf{AP}_{obs} \cup \mathsf{AP}_{est} \cup \mathsf{AP}_{hid}$ and $\leq_E$ be a preference relation among $2^{\mathsf{AP}_{est}}$. We say that $\leq_E$ and ρ_s are* monotone *if $\leq_E$ forms a lattice over $2^{\mathsf{AP}_{est}}$ and for all $X \subseteq \mathsf{AP}_{obs} \cup \mathsf{AP}_{est} \cup \mathsf{AP}_{hid}$ and $Y \subseteq \mathsf{AP}_{obs} \cup \mathsf{AP}_{hid}$, we have that $\{Y' \subseteq \mathsf{AP}_{est} \mid (X, Y \cup Y') \in \rho_e\}$ is a lattice together with $\leq_E$. The maximal element of this lattice is the same as of the lattice induced by $\leq_E$ over $2^{\mathsf{AP}_{est}}$.*

4. RUNNING EXAMPLE: CAR FOLLOWING

We demonstrate the use of estimators in reactive synthesis on a discrete model for the car following case study. The state of the environment is represented by:

- the distance between the cars, $dist \in \{0, \ldots, 85\}$,
- the speed of the leader car, $speedLeader \in \{0, \ldots, 15\}$,
- and the speed of the follower car, $speedFollower \in \{0, \ldots, 15\}$.

The output of the controller to be synthesized consists only of the acceleration (*accFollower*) of the follower car, which has a range of $\{-2, -1, 0, 1, 2\}$.

Both the speed of the leader car and the distance between the cars are considered to be hidden variables, while the follower car, whose controller we want to synthesize, has a sensor to measure its speed. The follower receives an imperfect measurement $distObserved \in \{0, \ldots, 85\}$ of the distance between the cars.

In summary, we have that AP_{hid} consists of boolean variables to encode the integer variables *dist* and *speedLeader*, while AP_{inp} encodes *speedFollower* and *distObserved*. The proposition set AP_{act} deals with *accFollower* as the only controllable integer variable.

The basis for computing an estimator is the known relationship between the environment variables. The environment specification ρ_e encodes the following restrictions between current and next variable values during each time step.

1. The speed of the follower car is updated according to its chosen acceleration:
$$speedFollower' = speedFollower + accFollower.$$
2. The speed of the leader car changes only gradually:
$$speedLeader' = speedLeader \pm 2.$$

3. The distance between the cars is updated according to their speed values:

$$distance' = distance + \frac{speedLeader + speedLeader'}{2} - \frac{speedFollower + speedFollower'}{2} \pm 1.$$

4. The observed distance can only deviate by at most 2 from the actual distance:

$$distance' = distObserved' \pm 2.$$

Note that we saved the explicit modeling of the leader car's acceleration by using the formulation from the second point above. The added noise (± 1) in the third point accounts for imprecision in motion and approximating the integral of the speed over time linearly. The first and third properties above are slightly simplified by not defining what happens when the values of *speedFollower*$'$ and *distance*$'$ need to be out of their domains in order to satisfy the constraint. For *speedFollower*, we encode ρ_e such that the variable has *saturation semantics*, so negative values are rounded to 0, and values greater than 15 are changed to 15. Thus, the maximum speed of the two cars is 15. Similarly, *distance*$'$ is saturated between 0 and 85. Here, however, we choose to let the value 85 represent all actual physical values of ≥ 85. Thus, a distance of 85 can stay the same in a time step regardless of the speed of both cars (as the real distance may be much larger), and at a transition at which the distance drops below 85, the value of *distance*$'$ may be higher than the one defined by the equation from the third point above.

We choose to let the estimator always output lower and upper bounds on the real distance between the cars and the speed of the leader car. Formally, AP_{est} encodes the four variables *minDist*, *maxDist*, *minSpeedLeader*, and *maxSpeedLeader* that have the same domains as their underlying precise values. In order to enforce that the estimator always updates these approximations with suitable values after a transition, we encode the following constraints on the values of these variables into ρ_s:

5. $minDist' \leq distance' \wedge maxDist' \geq distance'$.

6. $minSpeedLeader' \leq speedLeader' \wedge maxSpeedLeader' \geq speedLeader'$.

Finally, we state the controller specification. Intuitively, we would require the leader to always keep a distance between some lower and some upper bound (that is smaller than 85). However, due to the noisy distance update, the leader car effectively has a speed range of $\{-1, \ldots, 16\}$ at its disposal and can thus outrun the follower. Consequently, requiring the distance between the cars to be in some range other than $\{0, \ldots, 85\}$ results in a specification that cannot be fulfilled. We instead require the controller to ensure the following constraints:

7. $distance' \geq 5 \vee (speedFollower = 0 \wedge speedFollower' = 0)$.

8. $distance' < 85 \vee (speedFollower' \geq 15)$.

Therefore, the only case in which the minimum distance of 5 may be exceeded is when the follower car already came to a stand-still and does not accelerate. In such a case, having a distance smaller than 5 is intuitively not the fault of the follower car. We apply the same idea for the upper bound: given a nominal speed of the platoon of 8, while the follower is trying to catch up by using a speed of 15, we allow the distance to exceed a value of 85.

In order to remove the hidden propositions from the controller specification, we strengthen the specification to the following form:

9. $(minDistance' \geq 5) \vee (speedFollower = 0 \wedge speedFollower' = 0)$.

10. $(maxDistance' < 85) \vee (speedFollower' \geq 15)$.

Thus, we require the controller for the follower car to also *know* that it meets its specification (as *minDistance* is always smaller or equal to the real distance by the estimator specification and *maxDistance* satisfies a similar condition). Note that this modification satisfies the precondition of Proposition 1, so we know that, if we find a controller for this modified specification that works correctly in combination with the estimator, the estimator and controller together satisfy the original specification in an environment that follows ρ_e.

Note that the estimates of the leader car's speed are not used in the specification. They are rather used to allow a positional estimator to approximate the distance at runtime more precisely.

5. COMPUTING THE ESTIMATORS

We now consider the question of how to compute positional estimators. If the estimator specification is monotone with respect to its preference relation, the estimator will be optimal among all positional estimators and unique. The uniqueness property is helpful for using the estimator in the scope of the synthesis process—iterating through all possible optimal estimators during synthesis is costly, and non-uniqueness would indicate a potential problem with the estimator specification. Positional estimators in turn have the advantage that they are bounded in size and can be easily computed, as we show in the following.

The starting point for computing optimal positional estimators is to consider the possible evolutions of the environment. As stated in Section 3, the transitions that the environment can perform are specified in the relation $\rho_e \subseteq (2^{\mathsf{AP}_{obs} \cup \mathsf{AP}_{hid}})^2$. Likewise, the (safety) specification for the estimator is stored in the relation $\rho_s \subseteq (2^{\mathsf{AP}_{obs} \cup \mathsf{AP}_{hid} \cup \mathsf{AP}_{est}})^2$.

Starting from the initial valuation x_0, we first determine the set of configurations of AP_{obs}, AP_{hid}, and AP_{est} that are reachable by computing the set

$$R = \mu X.(\{x_0\} \cup \{x' \subseteq \mathsf{AP}_{obs} \cup \mathsf{AP}_{hid} \cup \mathsf{AP}_{est} \mid \exists x \in X. (x \setminus \mathsf{AP}_{est}, x' \setminus \mathsf{AP}_{est}) \in \rho_e, (x, x') \in \rho_s\}),$$

where μ denotes the least fixpoint operation. This operation saturates the set X with reachable states of the environment and any estimator that satisfies its specification ρ_s. When evaluating μ step-wise, new states x' are added such that there already is a state x in X from which x' can be reached along a transition that satisfies ρ_e and ρ_s. As ρ_e is not concerned with estimation variables, they are removed from x and x' before checking if the transition is in ρ_e.

The final set R then represents the configurations of the environment and an estimator for ρ_e and ρ_s that are poten-

tially reachable. The fact that the values of some propositions are hidden has not been taken into account so far.

A positional estimator has to offer the same next estimate for every pair of configuration and next input that it cannot distinguish (i.e., which only differ in their hidden variables). Thus, we can compute the set of allowed transitions in a positional estimator by determining the set

$$\begin{aligned}\rho_u = &\{(x, x') \in (2^{\mathsf{AP}_{obs} \cup \mathsf{AP}_{est}})^2 \mid \forall y, y' \subseteq 2^{\mathsf{AP}_{hid}} : \\ &((x \cup y) \in R \wedge ((x \setminus \mathsf{AP}_{est} \cup y), \\ &(x' \setminus \mathsf{AP}_{est} \cup y'))) \in \rho_e \rightarrow ((x \cup y), (x' \cup y')) \in \rho_s\}.\end{aligned}$$

The expression defining ρ_u consists of two parts: it first quantifies over the possible transitions (x, x') in a positional estimator. All transitions that are part of *any* valid positional estimator are contained in the set ρ_u. To test if a transition is valid, the expression further quantifies over all hidden variable valuations (y, y') along such a transition. If $x \cup y$ is a reachable state in any estimator, and a transition from $(x \cup y)$ to $(x' \cup y')$ is allowed by ρ_e, then the definition requires the new estimate to be correct w.r.t. ρ_s.

A positional estimator that only performs transitions from ρ_u is guaranteed to be correct by definition, as ρ_u only contains transitions that *any* estimator is allowed to take regardless of any previous transitions. Note that the definition of ρ_u also ensures that any estimator that only performs state transitions in ρ_u is history-free, as ρ_u quantifies over states that are reachable in *any* estimator. Thus, an estimator that only uses transitions in ρ_u cannot use its state space to track information other than the last input and the last estimate.

LEMMA 1. *Let ρ_u be an estimator transition relation defined as above. Then, the transition relation*

$$\begin{aligned}\hat{\rho}_u = \{(x, x') \in \rho_u : x'|_{\mathsf{AP}_{est}} = \min\{x''|_{\mathsf{AP}_{est}} : (x, x'') \in \rho_u, \\ x' \setminus \mathsf{AP}_{est} = x'' \setminus \mathsf{AP}_{est}\}\}\end{aligned}$$

represents the transitions of a unique optimal positional estimator.

PROOF. Assume the converse, namely that $\hat{\rho}_u$ is not uniquely optimal. As we can describe a positional estimator by a transition relation in its (fixed) state space, this means that there exists a transition relation $\hat{\rho}'_u$ that offers a different approximation for some combination $(x, \bar{x}) \in 2^{\mathsf{AP}_{obs} \cup \mathsf{AP}_{est}} \times 2^{\mathsf{AP}_{obs}}$ that is not worse (with respect to $\leq_E$) than the (unique) approximation that $\hat{\rho}_u$ offers.

Let $y \subseteq \mathsf{AP}_{est}$ be the estimate offered by $\hat{\rho}_u$, i.e., for which we have $(x, \bar{x} \cup y) \in \hat{\rho}_u$, and $y' \subseteq \mathsf{AP}_{est}$ be the estimate that $\hat{\rho}'_u$ offers, i.e., for which we have $(x, \bar{x} \cup y') \in \hat{\rho}'_u$. If $\hat{\rho}_u$ is not uniquely optimal, then the inequality $y \leq_E y'$ does not hold.

As y is guaranteed to be the unique least estimate available in ρ_u (by the definition of $\hat{\rho}_u$ and the requirement that $\leq_E$ induces a lattice), there exists some (valid) estimate that is not in ρ_u. However, as ρ_u is defined to contain all history-free estimates that are valid from the reachable configurations in R if the environment uses only transitions in ρ_e, we have derived a contradiction. □

By definition, if ρ_u does not offer a next estimate for some state s and some next observable proposition valuation x, receiving x in state s witnesses the violation of ρ_u by the environment.

Let us conclude the section by giving another motivation for why positional estimators must be history-free, as specified in Definition 2. This requirement of a positional estimator is crucially needed for the proof of Lemma 1. Intuitively, the non-satisfaction of the requirement would allow the estimator to strategically worsen some approximations in order to decrease the number of cases in which certain estimates are reachable. This choice by the estimator limits the number of incoming edges into the state that represents the estimate, and thus a tighter estimate could be given in the next step. Consequently, optimal estimators would no longer be unique. From a practical perspective, uniqueness is however a favorable property. In particular, whenever in the later synthesis step for the controller, the specification is found to be unrealizable, uniqueness of the estimator means that unrealizability is not due to having picked the wrong estimator. So there is no need to iterate over multiple possible estimators and to try synthesizing a controller for each of them.

6. ESTIMATOR-BASED GENERALIZED REACTIVITY(1) SYNTHESIS

In order to synthesize a system from a generalized reactivity(1) specification φ that works correctly when using input from an estimator (as depicted in Figure 1), we can encode the estimator's structure into φ. A synthesized system for the modified specification is then forced to produce the estimator's execution along with the system's own choices, and thus can work *stand-alone* without the estimator finite-state machine. In this way, the synthesis procedure also knows what outputs of the estimator it has to expect under particular observations.

The shape of the estimators defined according to the construction from the previous section is well-suited to be used in the context of generalized reactivity(1) synthesis. Recall that GR(1) specifications are always structured as given in equation (1). Let $(V)_A$ be a shorthand for the LTL formula $\bigwedge_{x \in V \cap A} x \wedge \bigwedge_{x \in A, x \notin V} \neg x$ for some sets V and A. Incorporating the estimator with the transition relation $\hat{\rho}_u$ can be done by adding the property

$$\mathsf{G}\left(\bigvee_{(x,x') \in \hat{\rho}_u} (x)_{\mathsf{AP}_{obs} \cup \mathsf{AP}_{est}} \wedge \mathsf{X}(x')_{\mathsf{AP}_{obs} \cup \mathsf{AP}_{est}}\right)$$

as a conjunct to φ^g_s. Likewise, for x_0 being the initial (known) valuation of all variables in AP, we add $(x_0)_{\mathsf{AP}_{inp}}$ to φ^a_i and $(x_0)_{\mathsf{AP}_{est}}$ to φ^g_i in order to let the estimator start off from the correct variable valuations. As we require the synthesized controller to simulate the estimator, we also add all elements of AP_{est} to AP_{act}, so that the controller can choose the estimates.

As described in Proposition 1, in order to allow the synthesis problem to be treated as one over complete information, the specification of the system to be constructed needs to be altered in order to remove all references to AP_{hid}. The conjuncts introduced by encoding the estimator into φ do not use variables from AP_{hid}, so only the references already present in the original specification need to be changed. To ensure soundness when doing so, it follows from the description in Proposition 1 that the assumptions may only be weakened, while the guarantees may only be strengthened. We have seen an example for

the latter in Section 4, where the constraint ($distance' \geq 5$) $\vee$ ($speedFollower = 0 \wedge speedFollower' = 0$) was replaced by constraint ($minDistance' \geq 5$) $\vee$ ($speedFollower = 0 \wedge speedFollower' = 0$). As the estimator specification ensures that *minDistance* is always smaller than or equal to the actual (non-observable) distance, this strenthening is valid. If ($distance' \geq 5 \vee \psi$) was a constraint in the assumptions for some ψ, a suitable weaking would be ($maxDistance' \geq 5 \vee \psi$).

Since the system to be synthesized only needs to work correctly in environments that behave according to ρ_e, we have to modify φ further to account for this fact: in case φ does not contain constraints over the evolution of the environment already, constraints that are implied by ρ_e can be added to allow the synthesis engine to ignore inadmissible observable inputs. In the case in which φ already has some assumptions that implement ρ_e, they have to be replaced by constraints that do not refer to variables in AP_{hid}.

In both cases, the new assumptions can be mined from ρ_u (or equivalently from $\hat{\rho}_u$). That is, if, for some combination of the current configuration and next observable input, there does not exist a suitable estimate in ρ_u, then ρ_e is violated as we showed in the previous section. Thus, we can add the following constraint to φ_s^a:

$$\mathsf{G}\left(\bigvee_{(x,x')\in\rho_u}(x)_{\mathsf{AP}_{obs}\cup\mathsf{AP}_{est}} \wedge \mathsf{X}\,(x')_{\mathsf{AP}_{obs}}\right).$$

Finally, let us reconsider the motivation for using generalized reactivity(1) synthesis to obtain correct-by-construction controllers and compare the complexity of our estimator-based synthesis approach to standard generalized reactivity(1) synthesis under incomplete information.

Firstly, generalized reactivity(1) specifications allow a simple BDD-based synthesis algorithm that has shown its good scalability in practice many times (see, e.g., [2]). The parts of the specification changed according to the procedure from this section can be built directly as BDDs, thus obliviating the need to enumerate all elements of $\hat{\rho}_u$ explicitly. A requirement for this step is however that $\hat{\rho}_u$ is represented symbolically. Such a symbolic encoding is indeed possible as all operations to compute $\hat{\rho}_u$ outlined in the previous section can be implemented with BDDs using only standard operations, provided that ρ_s and ρ_e are also represented as BDDs. If we assume that both elements are parsed from formal specifications, this requirement is easy to fulfill.

For the comparison of time complexities, observe that after following the steps described in this section, we have a standard GR(1) synthesis problem under complete information, which is solvable in time exponential in the number of atomic propositions. Also, all operations for estimator computation in the previous section can be performed in exponential time. As the estimator size is constant, we obtain a time complexity that is exponential in $|\mathsf{AP}_{inp} \cup \mathsf{AP}_{est} \cup \mathsf{AP}_{act} \cup \mathsf{AP}_{hid}|$. As (standard) generalized reactivity(1) synthesis under incomplete information has a doubly-exponential time complexity (in $|\mathsf{AP}_{inp} \cup \mathsf{AP}_{act} \cup \mathsf{AP}_{hid}|$), we save one exponent at the expense of the added estimation variables. So unless AP_{est} is very large, we obtain a significantly reduced complexity. Note that we could require the estimator to encode the *belief space* of the system precisely, leading to an exponential number of variables in AP_{est} and a doubly-exponential overall complexity again.

The above complexity bounds are independent of the specification size unless it is super-exponential in the number of propositions. Note that GR(1) specifications can easily be distilled to at most exponential size.

7. STATE SPACE REDUCTION IN ESTIMATOR-BASED SYNTHESIS

The state space of a positional estimator, namely $2^{\mathsf{AP}_{obs}\cup\mathsf{AP}_{est}}$, can be large. While in practice, it is not uncommon that only a fraction of the states is reachable from the initial state of the estimator, symbolic reasoning techniques (such as using BDDs) cannot benefit from this fact immediately. Since such techniques are the driving factor of the scalability of modern synthesis algorithms, and thus need to be catered for, the question arises whether, in the scope of estimator computation, we can perform some specialized steps that allow to exploit the sparseness of the reachable states in the estimator.

Here, we propose to use the relationships between observable and estimate variables in order to reduce the number of elements in $\mathsf{AP}_{obs} \cup \mathsf{AP}_{est}$, which compresses the symbolic representation of the state space so that it can be exploited by symbolic reasoning techniques.

Many synthesis problems exhibit such relationships. For instance, in our running example, we know that the difference between *distance* and *observedDistance* is at most 2 which in turn implies that the differences between the estimates *minDistance* and *observedDistance* are also at most 2 in every step of the estimator's evolution. We can thus replace the variable *minDistance* by an integer variable *minDistanceDelta* and replace all occurrences of *minDistance* in the specification by *observedDistance* + *minDistanceDelta* − 2. Instead of a domain of $\{0, \ldots, 85\}$ for *minDistance*, we only need a domain of $\{0, \ldots, 5\}$ for *minDistanceDelta*. Hence, encoding *minDistanceDelta* into BDDs takes four fewer bits than encoding *minDistance*. We can also compress *maxDistance* in the same way.

For our experiments in the next section, we did not perform state space compression other than replacing *minDistance* and *maxDistance* by *minDistanceDelta* and *maxDistanceDelta*. However, it is easy to perform a reachability check of the states in the estimator in order to discover relationships between the variables that are not obvious from the specification, and in fact our implementation can uncover these, so that they can be used to speed up the synthesis step (after re-computing the estimator for the new compressed version of its specification).

8. EXPERIMENTS ON THE RUNNING EXAMPLE

We have implemented the techniques from Section 5 as a plug-in for the synthesis tool `slugs` [7] and use the same tool for performing the later synthesis step, as described in Section 6. All computations reported on in the following were single-threaded and performed on an AMD Opteron 2.8GHz computer, running Linux (x64) with 32 GB of RAM.

Starting from an initial distance of 15 between the cars, both cars having an initial speed of 3, and performing state space compression on the distance between the cars as described in the previous section, computing the estimator for the scenario described in Section 4 takes 21.38 minutes.

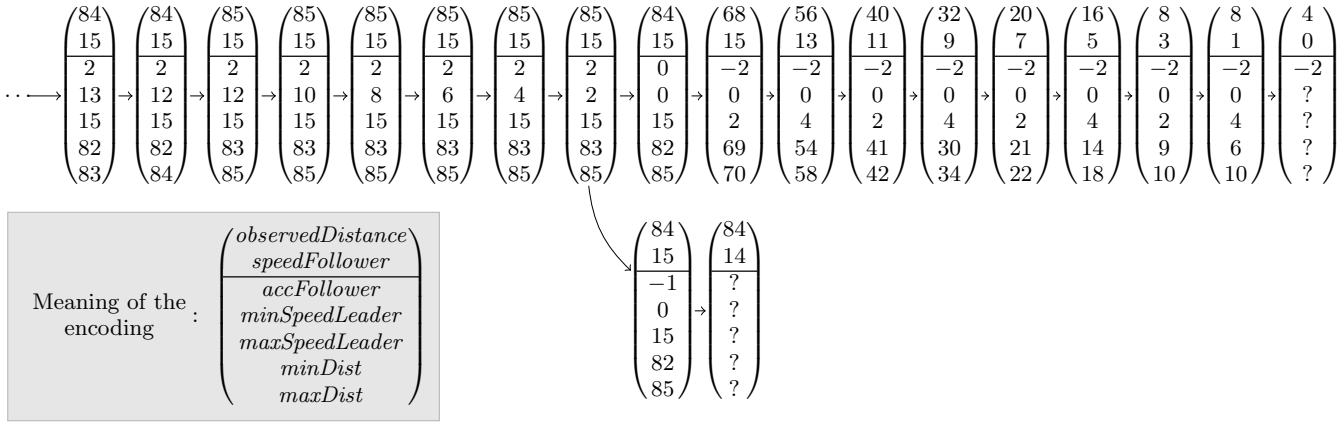

Figure 2: Two suffixes of example traces in a counter-strategy that witnesses the unrealizability of the specification described in Section 4. The traces start in the same way and branch at some point due to a different acceleration chosen by the controller. Question marks indicate that there is no possible valuation to the next output variables of the controller that would not violate the specification.

Checking the realizability of the controller specification using the computed estimator takes 28.04 minutes.

It turns out that the specification is unrealizable, meaning that the environment can drive the system into violating its specification. Two traces of a counter-strategy for the environment, which we depict in Figure 2, explain this fact. In the figure, input by the environment is shown above the column vector bars, whereas output by the controller is shown below the bars. Only the acceleration can actively be chosen by the controller, as the other four output variables represent estimates, which are governed and fixed by $\hat{\rho}_u$.

Initially, the environment chooses to gradually raise the observed distance from 15 to 64—the corresponding first 20 rounds of the execution are not shown in the figure for brevity. In the trace, the system has chosen to accelerate in order to try to satisfy the requirement that $(\mathit{maxDistance}' < 85 \vee (\mathit{speedFollower}' \geq 15))$ shall always hold. Note that once a follower car speed of 15 has been reached, further acceleration chosen by the controller has no effect.

As soon as the distance of 85 has been reached, which abstracts from all possible distances greater than or equal to 85, the estimator's knowledge about the speed of the leader car degrades gradually. When $\mathit{minSpeedLeader} = 2$ and $\mathit{maxSpeedLeader} = 15$ has been reached (i.e., the estimator's knowledge is very coarse), the environment chooses an observed value of 84. We consider two ways for the system to react. In the first case, the controller continues to maintain a follower car speed of 15 for one more time step. In this case, the top trace shows that there is not enough time any more for the follower car to still be able to come to a stand-still before the minimum distance drops below 5 in case it turns out that the leader car has a very low speed at that point. This behavior violates the requirement that the follower should have already come to a stand-still before the distance between the cars drops below 5. In case the controller chooses to only decelerate slightly, the environment may actually keep the observed distance of 84 for the next step, which may mean that the real distance is still 85 and the measurement of 84 was only off by one. At this point, the constraint $(\mathit{maxDistance}' < 85 \vee (\mathit{speedFollower}' \geq 15))$ is violated. So in both cases, the synthesized system can only either violate the specification or produce incorrect estimates.

Changing the guarantee $(\mathit{maxDistance}' < 85 \vee (\mathit{speedFollower}' \geq 15))$ to $(\mathit{maxDistance}' < 85 \vee (\mathit{speedFollower}' \geq 14))$ makes the specification realizable. The computation time needed the check realizability was 27.88 minutes in this case.

As a modification, we would like the follower car to stay as close to the leading car as possible while the leading car is running at a nominal speed of 8, which we assume to be the standard speed on the street. In order to encode this constraint into a generalized reactivity(1) specification, we add an output proposition called *inCruiseMode*. The controller is required to always eventually enter cruise mode, in which it has to maintain a distance of at most b for some $b \in \mathbb{N}$. The cruise mode may only be left when it is clear that the leader does not drive with a speed of 8 (i.e., if the estimator computes that $\mathit{minSpeedLeader} > 8$ or $\mathit{maxSpeedLeader} < 8$).

As the cruise mode cannot be enforced when the leader gets out of sight (having a distance ≥ 85), the follower does not need to eventually enter cruise mode if the leader is out of sight infinitely often.

By performing a binary search on the value of b, we find out that all values of b greater than or equal to 73 can be enforced. The computation time for the search (using the same estimator as before and performing 6 realizability checks in total) was 459.86 minutes.

Note that we picked the maximum distance value of 85 on purpose, as it yields the interesting behavior in Figure 2. If we set the maximum distance to 127, which fully utilizes the propositions that we allocated for encoding the observed distance and the estimates on the real distance between the cars, then modifying the specification to only require a follower speed that is greater than or equal to 14 when the leader car gets out of sight is not necessary—the original specification is already realizable (28.36 minutes of time for computing the estimator and 16.13 minutes for checking realizability). The smallest possible value of b that yields a realizable specification when adding cruise mode is still 73 in this case (computation time for the binary search: 629.02 minutes, 7 realizability checks in total).

We could not compare our approach to using a standard *belief-set construction* in GR(1) synthesis instead of estimators for coping with the incomplete information. The belief space has $2^{86 \cdot 16}$ states, which cannot be dealt with by any current synthesis approach.

9. CONCLUSION & DISCUSSION

We presented a notion of *estimator* which—in certain cases—decouples *incomplete-information* reactive synthesis into two steps: (1) the estimator construction step for establishing knowledge about the unobserved variables that is salient for the synthesis of controllers to satisfy given temporal logic specifications and (2) the synthesis step which relies on the knowledge generated by the estimator and boils down to conventional, complete-information synthesis.

Estimators help to abstract out from full belief-set constructions commonly used for reasoning about partialness of information. While sound-and-precise estimators may provide tighter results, positionality of our estimators provides computational tractability. While generalized(1) synthesis under incomplete information has a doubly-exponential time complexity, using estimators reduces complexity to singly-exponential time. This tractability comes at the expense of potential loss of precision. Hence, such estimators provide a trade-off between complexity and precision. Moreover, they are relatively straightforward to compute. The "positionality" restriction does not only allow to integrate them into generalized reactivity(1) synthesis but also helps with establishing a notion of a "best" estimator.

Our estimator definition is general enough to also include optimal non-positional estimators. Finding compact representations of such estimators remains as an open question. Furthermore, while we discussed estimator synthesis in a purely discrete setting, the estimator construction technique will generalize to include variables that are not necessarily restricted to be discrete valued. For example, scenarios with real-valued properties and updates that can be expressed as linear functions could be analyzed using *and-inverter graphs over linear functions* (LinAIGs) [5] as symbolic reasoning engine. In that case, the state space compression technique we discussed can still be applied effectively.

While the algorithms we proposed obey relatively desirable theoretical complexity bounds, scalability remains as a limitation as evidenced by the "long" computation times in the case studies. We emphasize though that the current implementation has not been optimized to handle integer-valued variables and expect such optimizations to significantly reduce the practical computation times.

Acknowledgements

The first author was supported by the Institutional Strategy of the University of Bremen, funded by the German Excellence Initiative. The second author was partially supported by AFOSR grant # FA9550-12-1-0302, ONR grant # N000141310778, and NSF CNS award # 1446479.

The authors thank Bernd Finkbeiner for discussions during the early phases of this work.

10. REFERENCES

[1] A. Balluchi, L. Benvenuti, M. D. Di Benedetto, and A. L. Sangiovanni-Vincentelli. Design of observers for hybrid systems. In *HSCC*, pages 76–89. 2002.

[2] R. Bloem, S. J. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. Specify, compile, run: Hardware from PSL. *Electr. Notes Theor. Comput. Sci.*, 190(4):3–16, 2007.

[3] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive(1) designs. *J. Comput. Syst. Sci.*, 78(3):911–938, 2012.

[4] P. E. Caines, R. Greiner, and S. Wang. Classical and logic-based dynamic observers for finite automata. *IMA Journal of Mathematical Control and Information*, 8(1):45–80, 1991.

[5] W. Damm, S. Disch, H. Hungar, S. Jacobs, J. Pang, F. Pigorsch, C. Scholl, U. Waldmann, and B. Wirtz. Exact state set representations in the verification of linear hybrid systems with large discrete state space. In *ATVA*, pages 425–440, 2007.

[6] D. Delvecchio, R. M. Murray, and E. Klavins. Discrete state estimators for systems on a lattice. *Automatica*, 42(2):271–285, 2006.

[7] R. Ehlers, V. Raman, and C. Finucane. Slugs GR(1) synthesizer, 2013–2015. Available at https://github.com/LTLMoP/slugs.

[8] G. Kalyon, T. L. Gall, H. Marchand, and T. Massart. Global state estimates for distributed systems. In *Formal Techniques for Distributed Systems*, pages 198–212, 2011.

[9] H. Kress-Gazit, T. Wongpiromsarn, and U. Topcu. Correct, reactive robot control from abstraction and temporal logic specifications. *IEEE Robotics and Automation Magazine*, 18(3):65–74, 2011.

[10] O. Kupferman and M. Vardi. Synthesis with incomplete informatio. In *2nd International Conference on Temporal Logic*, pages 91–106, Manchester, July 1997.

[11] D. G. Luenberger. *Optimization by vector space methods*. John Wiley & Sons, 1969.

[12] M. Oishi, I. Hwang, and C. Tomlin. Immediate observability of discrete event systems with application to user-interface design. In *CDC*, pages 2665–2672, 2003.

[13] C. M. Ozveren and A. S. Willsky. Observability of discrete event dynamic systems. *IEEE Transactions on Automatic Control*, 35(7):797–806, 1990.

[14] J. H. Reif. The complexity of two-player games of incomplete information. *Journal of Computer and System Sciences*, 29(2):274 – 301, 1984.

[15] R. D. Smallwood and E. J. Sondik. The optimal control of partially observable markov processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.

[16] S. Sohail and F. Somenzi. Safety first: a two-stage algorithm for the synthesis of reactive systems. *STTT*, 15(5-6):433–454, 2013.

[17] D. C. Tarraf, A. Megretski, and M. A. Dahleh. Finite approximations of switched homogeneous systems for controller synthesis. *IEEE Trans. Automat. Contr.*, 56(5):1140–1145, 2011.

[18] Y. Velner and A. Rabinovich. Church synthesis problem for noisy input. In *FOSSACS*, pages 275–289, 2011.

[19] A. Walker and L. Ryzhyk. Predicate abstraction for reactive synthesis. In *FMCAD*, 2014.

APPENDIX

We provide some additional information here that may be of interest to a few readers. In particular, we shortly discuss why GR(1) synthesis has an exponential time complexity, and provide a proof for doubly-exponential time complexity of the problem in case of incomplete information.

A Note on the Complexity of GR(1) Synthesis

In the introduction in the main part of the paper, we state that GR(1) synthesis has a singly-exponential time complexity, which is in contrast to other paper in which it is stated that the time complexity is "polynomial in the state space of the design."

The "state space of the design" in this statement is however exponential in the number of atomic propositions; hence we get an overall exponential complexity.[1]

Complexity of GR(1) Synthesis under Incomplete Information

Recall that one of the motivations for computing estimators is that unlike for full linear temporal logic (LTL), the synthesis problem for GR(1) specifications has a higher complexity under incomplete information than under full information. Let us formally prove this now. We reduce the acceptance problem of an alternating exponential-space Turing machine to the GR(1) synthesis problem under incomplete information. As AEXPSPACE=2EXPTIME[2], the claim that doubly-exponential time is needed for GR(1) synthesis under incomplete information then follows. The proof is essentially an adaptation of the extended computation tree logic reactive synthesis 2EXPTIME-hardness proof given by Vardi and Stockmeyer[3], which in turn builds on ideas by Fischer and Ladner[4].

THEOREM 1. *Let $f : \mathbb{N} \to \mathbb{N}$ be a function such that $f(k) > k$ for all $k \in \mathbb{N}$, and $M = (Q, \Sigma, \Gamma, \delta, q_0, g)$, be a $2^{f(k)}$-space bounded alternating Turing machine, where Q is the set of states, Σ is the input alphabet, Γ is the tape alphabet (which is a superset of Σ), $\delta : Q \times \Gamma \to (Q \times \Gamma \times \{L, U, R\})^2$ is the transition function of M, $q_0 \in Q$ is its initial state, and $g : Q \to \{accept, reject, existential, universal\}$ is a function that denotes a partitioning of the states into accepting, rejecting, existentially branching, and universally branching states.*

We can reduce the acceptance of a word $w \in \Sigma^k$ by M to the GR(1) synthesis problem under incomplete information with $O(\log_2 |\Gamma| + \log_2 |Q| + f(k))$ many input propositions, hidden propositions and output propositions and a specification of length polynomial in $f(k) + |\Gamma| + |Q|$.

PROOF. Without loss of generality, we assume that from all rejecting and accepting states, M only has transitions that let M's head stay in place and not change its state or current tape cell content after reaching the state. M is easily modified to have this property (without changing the size of Q or the set of words accepted by M). Additionally, we assume that Σ has an empty-tape symbol called $\perp$ and that M's tape head never moves left of tape position 0 or right of tape position $2^{f(k)} - 1$. A $2^{f(k)}$-space bounded alternating Turing machine M can always be modified to have this property without a super-polynomial blow-up in the size of the Turing machine.

In the following, we use the symbol _ to denote a placeholder for a value that is not of importance. The symbol $\oplus$ is used to represent mutual exclusion in LTL. An expression $\psi_1 \oplus \psi_2$ can be replaced by $\psi_1 \wedge \neg\psi_2 \vee \neg\psi_1 \wedge \psi_2$.

Specification Construction.

We construct the GR(1) specification

$$\varphi = (\varphi_i^a \wedge \varphi_s^a \wedge \varphi_l^a) \rightarrow_s (\varphi_i^g \wedge \varphi_s^g \wedge \varphi_l^g). \tag{2}$$

over the visible input propositions AP_I, the hidden input propositions AP_H, and the output propositions AP_O that forces the controller to be constructed in synthesis to simulate an accepting run tree of the alternating Turing machine.

We first of all set $\mathsf{AP}_I = \{u\}$ to be the set of observable input variables. Furthermore, we set:

$$\mathsf{AP}_H = \{match_1^S, match_2^S, match_1^Z, match_2^Z, match_1^\Gamma, match_2^\Gamma, t_1, t_2, \ldots, t_{f(k)}, m_1, \ldots, m_n, v_1, \ldots, v_{\lceil \log_2(|Q|)\rceil}\},$$

where $n = \lceil \log_2(|\Gamma|)\rceil$. The proposition u is used as external input for deciding which of the two possible transitions to take from universally branching states in the Turing machine. The meaning of the *match* variables is explained later. The variables $t_1, t_2, \ldots, t_{f(k)}$ encode one position along a $2^{f(k)}$-long tape. The variables $m_1, \ldots, m_n$ are used to store some (possible) tape cell content at the position stated by the t variables. The v variables store some specific state.

For the output of the controller, we choose

$$\mathsf{AP}_O = \{r, x_1, x_2, \ldots, x_{f(k)}, y_1, \ldots, y_n, z_1, z_2, \ldots, z_{f(k)}, s_1, \ldots, s_{\lceil \log_2(|Q|)\rceil}\}$$

as the proposition set. Here, r is used as the signal for some "initialization phase." The propositions $x_1, x_2, \ldots, x_{f(k)}$ represent the current counter for the tape position. The propositions $y_1, \ldots, y_n$ encode the tape content at the respective position. The propositions $z_1, z_2, \ldots, z_{f(k)}$ single out one particular tape cell as the current tape head position. Finally, the propositions $s_1, \ldots, s_{\lceil \log_2(|Q|)\rceil}$ are used to encode the current state.

Using some arbitrary encoding, a set of variables $\{a_i\}_{i \in I}$ for some index set I, and some value v that can be encoded into the a variables, we use the notation $(v)_a$ to represent an LTL formula that is free of temporal operators and asserts that the variables $a_1, a_2, \ldots$ encode the value v.

We use the following specification parts:

$$\begin{aligned}
\varphi_i^a &= \neg match_1^S \wedge \neg match_2^S \wedge \neg match_1^Z \wedge \neg match_2^Z \\
&\quad \wedge \neg match_1^\Gamma \wedge \neg match_2^\Gamma \\
\varphi_s^a &= \varphi_{s,1}^a \wedge \varphi_{s,2}^a \wedge \varphi_{s,3}^a \\
\varphi_l^a &= \mathbf{true} \\
\varphi_i^g &= r \wedge (0)_x \\
\varphi_s^g &= \varphi_{s,1}^g \wedge \varphi_{s,2}^g \wedge \varphi_{s,3}^g \wedge \varphi_{s,4}^g
\end{aligned}$$

[1] For a full proof, see Section 12.3.1 of R. Ehlers: *Symmetric and Efficient Synthesis*, Phd Thesis, Saarland University

[2] See Ashok K. Chandra, Dexter Kozen, Larry J. Stockmeyer: *Alternation.* J. ACM 28(1): 114-133 (1981)

[3] See Moshe Y. Vardi, Larry J. Stockmeyer: *Improved Upper and Lower Bounds for Modal Logics of Programs: Preliminary Report.* STOC 1985: 240-251

[4] See Michael J. Fischer, Richard E. Ladner: *Propositional Dynamic Logic of Regular Programs. J. Comput. Syst. Sci. 18(2): 194-211 (1979)*

$$\varphi_l^g = \mathsf{GF}\left(\bigvee_{q\in Q, g(q)=accept} (q)_s\right)$$

Here, $\varphi_{s,1}^g$, $\varphi_{s,2}^g$, $\varphi_{s,3}^g$, $\varphi_{s,4}^g$, $\varphi_{s,1}^a$, $\varphi_{s,2}^a$, and $\varphi_{s,3}^a$ are placeholders that we define below. The formula φ_l^g encodes that the controller must always eventually output the encoding of an accepting state of the Turing machine. Initially, $r = \mathbf{true}$ encodes that the machine's execution has just started.

Safety guarantee part $\varphi_{s,1}^g$ encodes that $x_1, x_2, \ldots, x_{f(k)}$ represents a cyclic counter and that r is set until the counter cycled through once.

$$\begin{aligned}\varphi_{s,1}^g = {} & \mathsf{G}(\mathsf{X}(x_1) \oplus x_1) \\ & \wedge \bigwedge_{j\in\{1,\ldots,f(k)-1\}} \mathsf{G}(\mathsf{X}(x_{j+1}) \leftrightarrow (x_{j+1} \oplus (x_j \wedge \neg\mathsf{X}(x_j)))) \\ & \wedge \mathsf{G}(\mathsf{X}r \leftrightarrow (r \wedge \neg\mathsf{X}(0)_x))\end{aligned}$$

The guarantees $\varphi_{s,1}^g$ ensure that we can use the x-variables for encoding addresses to locations on the $2^{f(k)}$ cell long tape. Initially, this tape shall have w written on it, which is encoded into $\varphi_{s,2}^g$.

$$\begin{aligned}\varphi_{s,2}^g = {} & \bigwedge_{j\in\{0,\ldots,k-1\}} \mathsf{G}(r \wedge (j)_x \rightarrow (w_j)_y) \\ & \wedge \mathsf{G}\left(r \wedge \left(\bigwedge_{i\in\{0,\ldots,k-1\}} \neg(i)_x\right) \rightarrow (\bot)_y\right) \\ & \wedge \mathsf{G}(r \rightarrow (q_0)_s) \\ & \wedge \mathsf{G}(r \rightarrow (0)_z)\end{aligned}$$

Additionally, $\varphi_{s,2}^g$ encodes that initially, the Turing machine shall start in state q_0 and with its head on tape cell 0.

Part $\varphi_{s,3}^g$ now states that the current state and the position of the tape head must only change when the counter x overflows. This allows us to assume that both variables stay constant while the complete tape content of the Turing machine is printed for one step of its computation.

$$\begin{aligned}\varphi_{s,3}^g = {} & \mathsf{G}(\neg\mathsf{X}(0)_x \rightarrow \bigwedge_{i\in 1,\ldots,\lceil\log_2(|Q|)\rceil} s_i \leftrightarrow \mathsf{X}\, s_i) \\ & \wedge \mathsf{G}(\neg\mathsf{X}(0)_x \rightarrow \bigwedge_{i\in 1,\ldots,f(k)} z_i \leftrightarrow \mathsf{X}\, z_i)\end{aligned}$$

So far, we did not state that the actual computation of the Turing machine has to be simulated correctly. It is quite tricky to encode this into a generalized reactivity(1) specification, as properties such as "at the same position of the tape after the next tape position counter reset, the tape cell content is the same as it is now" cannot be encoded into the LTL fragment supported by GR(1) synthesis. We can however make use of incomplete information to encode such a property in an indirect way.

First of all, we require the environment to always keep its values for $t_1, t_2, \ldots, t_{f(k)}, m_1, \ldots, m_n, v_1, \ldots, v_{\lceil\log_2(|Q|)\rceil}$ fixed after their initial values have been chosen:

$$\begin{aligned}\varphi_{s,1}^a = {} & \bigwedge_{i\in\{1,\ldots,f(k)\}} \mathsf{G}(t_i \leftrightarrow \mathsf{X}t_i) \\ & \wedge \bigwedge_{i\in\{1,\ldots,n\}} \mathsf{G}(m_i \leftrightarrow \mathsf{X}m_i) \\ & \wedge \bigwedge_{i\in 1,\ldots,\lceil\log_2(|Q|)\rceil} \mathsf{G}(v_i \leftrightarrow \mathsf{X}\, v_i)\end{aligned}$$

Since no constraints on the initial values of these variables exist, they can be arbitrary.

Then, we require the environment to track with the $match^\Gamma$ variables whether at the tape position encoded by the t variables, the tape cell content matched the one encoded into the m variables. Variable $match_1^\Gamma$ always gives this information for the previous tape content, while the $match_2^\Gamma$ variables do so for the current tape content. In the same manner, the $match^S$ variables track whether the state encoded by the s variables matched the one encoded by the v variables, and the $match^Z$ variables track whether the Turing machine tape head position was the same as the tape position encoded into the hidden t variables.

$$\begin{aligned}\varphi_{s,2}^a = {} & \mathsf{G}(\mathsf{X}(0)_x \rightarrow (\mathsf{X}match_1^S \leftrightarrow match_2^S)) \\ & \wedge \mathsf{G}(\mathsf{X}(0)_x \rightarrow (\mathsf{X}match_1^Z \leftrightarrow match_2^Z)) \\ & \wedge \mathsf{G}(\mathsf{X}(0)_x \rightarrow (\mathsf{X}match_1^\Gamma \leftrightarrow match_2^\Gamma)) \\ & \wedge \mathsf{G}(\neg\mathsf{X}(0)_x \rightarrow (\mathsf{X}match_1^S \leftrightarrow match_1^S)) \\ & \wedge \mathsf{G}(\neg\mathsf{X}(0)_x \rightarrow (\mathsf{X}match_1^Z \leftrightarrow match_1^Z)) \\ & \wedge \mathsf{G}(\neg\mathsf{X}(0)_x \rightarrow (\mathsf{X}match_1^\Gamma \leftrightarrow match_1^\Gamma)) \\ & \wedge \mathsf{G}(((\neg match_2^S \vee \mathsf{X}(0)_x) \wedge \neg(s = v)) \leftrightarrow \mathsf{X}\neg match_2^S) \\ & \wedge \mathsf{G}(((\neg match_2^Z \vee \mathsf{X}(0)_x) \wedge \neg(t = z)) \leftrightarrow \mathsf{X}\neg match_2^Z) \\ & \wedge \mathsf{G}(((\neg match_2^\Gamma \vee \mathsf{X}(0)_x) \wedge \neg(t = x \wedge m = y)) \\ & \qquad \leftrightarrow \mathsf{X}\neg match_2^\Gamma)\end{aligned}$$

We used the abbreviations $t = x$, $t = z$, $m = y$, and $s = v$ to simplify the equation. They mean that the $t/t/m/s$ and $x/z/y/v$ variables have the same values and can be expanded to $\bigwedge_{i\in\{0,\ldots,f(k)\}}(t_i \leftrightarrow x_i)$ / $\bigwedge_{i\in\{0,\ldots,f(k)\}}(t_i \leftrightarrow z_i)$ / $\bigwedge_{i\in\{1,\ldots,n\}}(m_i \leftrightarrow y_i)$ / $\bigwedge_{i\in\{1,\ldots,\lceil\log_2(|Q|)\rceil\}}(s_i \leftrightarrow v_i)$, respectively.

We can now enforce that the system simulates the run of the Turing machine correctly and that rejecting states are never reached. The safety guarantees $\varphi_{s,4}^g$ are devoted to ensuring this. To shorten the description of $\varphi_{s,4}^g$, we use the following sets:

$$\begin{aligned}A_0 = {} & \{(q, char, q', char', d) \in (Q \times \Gamma)^2 \times \{L, U, R\} \mid \\ & \qquad (q', char', d, \_, \_, \_) = \delta(q, char), g(q) = existential\} \\ & \cup \{(q, char, q', char', d) \in (Q \times \Gamma)^2 \times \{L, U, R\} \mid \\ & \qquad (\_, \_, \_, q', char', d) = \delta(q, char), g(q) = existential\} \\ & \cup \{(q, char, q', char', d) \in (Q \times \Gamma)^2 \times \{L, U, R\} \mid \\ & \qquad (q', char', d, \_, \_, \_) = \delta(q, char), g(q) = universal\} \\ A_1 = {} & \{(q, char, q', char', d) \in (Q \times \Gamma)^2 \times \{L, U, R\} \mid \\ & \qquad (q', char', d, \_, \_, \_) = \delta(q, char), g(q) = existential\} \\ & \cup \{(q, char, q', char', d) \in (Q \times \Gamma)^2 \times \{L, U, R\} \mid \\ & \qquad (\_, \_, \_, q', char', d) = \delta(q, char), g(q) = existential\} \\ & \cup \{(q, char, q', char', d) \in (Q \times \Gamma)^2 \times \{L, U, R\} \mid \\ & \qquad (\_, \_, \_, q', char', d) = \delta(q, char), g(q) = universal\}\end{aligned}$$

We now set:

$$\varphi_{s,4}^g = \mathsf{G}(match_2^S \wedge match_2^Z \wedge match_2^\Gamma \wedge (t = x) \wedge \neg u \rightarrow$$

$$\bigvee_{(q,char,q',char',d)\in A_0} ((q)_v \wedge (char)_m \wedge (q')_s \wedge (char')_y \wedge \psi_d))$$
$$\wedge \mathsf{G}(match_2^S \wedge match_2^Z \wedge match_2^\Gamma \wedge (t = x) \wedge u \rightarrow$$
$$\bigvee_{(q,char,q',char',d)\in A_1} ((q)_v \wedge (char)_m \wedge (q')_s \wedge (char')_y \wedge \psi_d))$$
$$\wedge \mathsf{G}(\neg match_2^Z \wedge match_2^\Gamma \wedge (t = x) \rightarrow m = y)$$

In this property, the sub-formula ψ_d for $d \in \{L, U, R\}$ encodes $z = t + 1$ if $d = R$, $z = t - 1$ if $d = L$, or $z = t$ if $d = U$. All of these can be represented in LTL:

$$\psi_R = z_1 \oplus t_1 \wedge \bigwedge_{i\in\{1,\ldots,f(k)-1\}} (z_{i+1} \leftrightarrow (t_{i+1} \oplus \bigwedge_{j\le i} \neg z_j))$$
$$\psi_U = \bigwedge_{i\in\{1,\ldots,f(k)\}} (z_i \leftrightarrow t_i)$$
$$\psi_L = t_1 \oplus z_1 \wedge \bigwedge_{i\in\{1,\ldots,f(k)-1\}} (t_{i+1} \leftrightarrow (z_{i+1} \oplus \bigwedge_{j\le i} \neg t_j))$$

In order to ensure that the environment has to keep the value of u constant while one step of the Turing machine is executed, we finally set:

$$\varphi^a_{s,3} = \mathsf{G}(\neg(f(k))_x \rightarrow (u \leftrightarrow \mathsf{X}u))$$

Note that AP_H, AP_I, AP_O, and φ are of size polynomial in $f(k)$, so if $f(k)$ is in turn polynomial in k, so is the overall specification.

Correctness.

Let us now show that φ is realizable if and only if M accepts w.

$\Rightarrow$: First of all, consider the case that M accepts w. Then, there exists an execution tree for M and w that branches at universal states and that is accepting along every branch of the tree. The specification can then be realized by simulating this tree and sequentially outputting the tape contents along a branch of the tree. The specification states that the value of the input proposition u dictates which direction to take whenever the tree branches universally. During every state transition, the system outputs the complete tape content while the x counter cycles through the tape cell numbers. While a tape is printed, the state output is always the same - it can change whenever the tape cell counter is reset.

If the tree is valid, then regardless of the values of the v, t, and m variables, the specification φ is satisfied, as these values select some tape position, tape content, and state for which the system must operate correctly. Since the system performs a full, complete, correct simulation of the Turing machine, the values of these hidden propositions thus do not matter.

All in all, this behavior satisfies the guarantees if the environment satisfies its assumptions.

$\Leftarrow$: Now assume that a controller is given that satisfies the specification. The specification makes sure that the x counter always cycles through the counter values for the tape positions. Only during the first cycle, r is set, and while r is set, the initial tape content is produced. At the same time, the state put out is the initial one and the tape head cell counter z points to cell 0. Since a state put out by the system needs to stay the same until the x counter flips over, the state encoded into s stays the same until the tape has been printed in its entirety.

Now assume that the counter has just cycled through and thus the state put out by the system may have changed. The system has to assume that the environment may have set the hidden variables such that the previous state and the previous tape cell content at the tape head was watched. Thus, the safety guarantee $\varphi^g_{s,4}$ requires the system to output the next tape position, the new tape cell content at the previous position, and the new state correctly. For universal states, the environment variable u dictates which of the possible Turing machine transitions shall be taken.

Since the system however also has to assume that it may be possible that any other tape position is watched by the hidden variables, it has to copy the content of the other, unmodified tape cells to the next round due to $\varphi^g_{s,4}$ and $\varphi^a_{s,2}$. This means that the system has to simulate the first step of the Turing machine's execution. Note that in the step after that, the contents of the hidden variables are still unknown to the controller. This means that again, a correct step in the execution of the Turing machine has to be simulated afterwards. By induction, it follows that all steps need to be simulated correctly, up to the point that the encoding of an accepting or rejecting state is printed by the controller. By the fact that accepting and rejecting states are absorbing and due to φ^g_l, rejecting states must never be visited. Runs of the controller along which no accepting Turing machine state is ever reached are also rejecting by φ^g_l.

So the controller has to output correct computations of the Turing machine until an accepting state of the Turing machine is reached. Since the machine must furthermore obey the environment's choice from universal states, the realizability of a specification means that there exists a controller that outputs an accepting execution tree for the alternating Turing machine along its possible executions, which witnesses the acceptance of w by M. □

When applying Theorem 1 to AEXPSPACE problems, we have that $f(k)$ is a polynomial function, so the AEXPSPACE(=2EXPTIME)-hardness of GR(1) synthesis under incomplete information follows.