From Natural Projection to Partial Model Checking and Back (Technical proofs and experiments)

Gabriele Costa[1⋆], David Basin[2], Chiara Bodei[3], Pierpaolo Degano[3], and Letterio Galletta[3⋆]

[1] DIBRIS, Università degli Studi di Genova
gabriele.costa@unige.it
[2] Department of Computer Science, ETH Zurich
basin@inf.ethz.ch
[3] Dipartimento di Informatica, Università di Pisa
{chiara,degano,galletta}@di.unipi.it

1 Introduction

System verification requires comparing a system's behavior against a specification. When the system consists of several components, we can distinguish between *local* and *global* specifications. A local specification applies to a single component, whereas a global specification must hold for the entire system. Since these specifications are needed to reason at different levels of abstraction, both of them are often present.

Ideally, we would like to freely pass from local to global specifications and vice versa. Most specification formalisms natively support specification composition. Logical conjunction, set intersection, and the synchronous product of automata are all examples of operators for composing specifications. Unfortunately, the same does not hold for specification decomposition: obtaining local specifications from a global one is, in general, extremely complex, as illustrated below.

Example 1. Consider the classical submodule construction problem (SCP) [26]: given a system and a global specification, find a submodule whose composition with the system leads to a global behavior conformant to the global specification. For instance, imagine that we aim to control the usage of a buffer of size n, where two agents, A and B, can insert and remove items. Now assume that the A's behavior is given as "insert one item when the buffer is empty and delete one item when it is full", while that of B is unknown. If we want to prevent buffer overflow and underflow, some questions arise about B's behavior. For example, are there compatible behaviors for B? Is there a *most general* one? How could we effectively compute it? These questions require decomposing the buffer overflow/underflow specification so that it only refers to B, while exploiting the known structure of A. □

⋆ This author is now affiliated with IMT Institute for Advanced Studies Lucca.

	Natural Projection	Partial MC
Spec. Lang.	FSA [30,18]	μ-calculus [1,3]
Theory	FSA [30,18]	LTS [1,3]
Complexity	EXPTIME[4] [31,13]	EXPTIME [1,3]
Tools	TCT [12], IDES3 [29], DESTool [27]	mCRL2 [17], CADP [22], MuDiv [2]

Table 1. Comparing the results on natural projection and partial model checking.

Over the past decades, researchers have investigated methods for decomposing specifications. Interestingly, different communities have tackled this problem independently, each considering specification formalisms and assumptions appropriate for their application context. In particular, important results were obtained in two distinct fields: *control theory* and *formal verification*.

In control theory, *natural projection* [32] is used for simplifying systems built from multiple components and modeled as automata. It is often applied component-wise to synthesize local controllers from a global specification of asynchronous *discrete-event system* [9], namely the *controller synthesis problem* (CSP). Briefly, local controllers guarantee that the global specification is not violated by only interacting with a single component of a system. The local controllers can be used to implement distributed control systems [33,34] by composing them in parallel with other sub-systems.

In the formal verification community, *partial model checking* [1] was proposed as a technique for mitigating the state explosion problem when verifying large systems built from many parallel processes. Partial model checking tackles this problem by decomposing a specification, given as a formula of the μ-calculus [21], using a *quotienting* operator, thereby supporting the analysis of the individual processes independently. Quotienting carries out a partial evaluation of a specification while *preserving the model checking problem*. Thus, a system built by composing two modules satisfies a specification if and only if one of the modules satisfies the specification after quotienting against the other [1]. This may reduce the problem size, resulting in smaller models and hence faster verification.

Table 1 summarizes some relevant facts about the two approaches and we refer the reader to Section 5 for a more detailed analysis. Since natural projection and partial model checking apply to different formalisms, they cannot be directly compared without defining a common framework (see below). For instance, a relevant question is comparing how specifications grow under the two approaches. Although it is known that both may lead to exponential growth (see [31,20] and [3]), these results apply in one case to finite-state automata (FSAs) and in the other case to μ-calculus formulae.

Over the past few years, there has been a substantial cross-fertilization between the two research communities [11]. For instance, methods for synthesizing

[4] In [31] an algorithm is given that runs in PTIME for a specific class of discrete-event systems.

controllers using partial model checking are given in [6,24]. Also the authors of [14] and [16] propose similar techniques, but they use fragments of the μ-calculus and CTL*, respectively.

We follow here the suggestion of [11], who advocate formally connecting these two approaches so as to make them interchangeable. In their words:

> "Such a formal bridge should be a source of inspiration for new lines of investigation that will leverage the power of the synthesis techniques that have been developed in these two areas. [...]
> It would be worthwhile to develop case studies that would allow a detailed comparison of these two frameworks in terms of plant and specification modeling, computational complexity of synthesis, and implementation of derived supervisor/controller."

As for the first remark, we show that, under reasonable assumptions, natural projection reduces to partial model checking and, when cast in a common setting, they are equivalent. To this end, we first define a common theoretical framework for both. In particular, we slightly extend both the notion of natural projection and the semantics of the μ-calculus in terms of the satisfying traces. These extensions allow us to apply natural projection to the language denoted by a specification. In addition, we extend the main property of the quotienting operator by showing that it corresponds to the natural projection of the language denoted by the specification, and vice versa (Theorem 4).

We provide additional results that contribute to answering to the second remark. Namely we define a new algorithm for partial model checking directly on LTSs (rather than on the μ-calculus). We prove our algorithm correct with respect to the traditional quotienting rules and we show it runs in polynomial time, like the algorithms based on natural projection. We have implemented this algorithm in a tool, which is available online.[5] Along with the tool, we developed several case studies illustrating its application to the synthesis of both sub-modules and local controllers.

Structure of the paper. Section 2 presents our unified theoretical framework for natural projection and partial model checking. Section 3 contains our main theoretical results, in particular Theorem 4 on the equivalence of quotienting and natural projection. In Section 4 we introduce a novel quotienting algorithm, discuss its properties, and apply it to our running example. In Section 5 we briefly survey the related literature and in Section 6 we draw conclusions. The formal proofs together with the correctness and the complexity of our algorithm, and some experimental results are given in the appendices.

2 A General Framework

In this section we cast both natural projection and partial model checking in a common framework. We start with a running example that we will use through-

[5] `https://github.com/SCPTeam/pests`

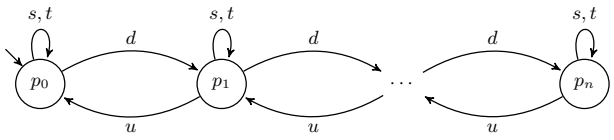

Fig. 1. An n-positions buffer specification $P(n)$.

out the paper. We consider a scenario inspired by [10] and [32], which is an instance of Example 1.

Example 2 (Running example). A drone package delivery (DPD) system relies on unmanned aerial vehicles (UAVs) to transport goods within a given area. Drones interact with docking stations where they can pick up (action u) or deposit (action d) an item. These actions are only observable to the docking station. Additional interactions are represented by the two control actions s (for synchronize) and t for (terminate). An action t takes place when UAVs are requested to leave the station, e.g., due to a maintenance operation, while s is used for the global synchronization of both the docking station and UAVs.

Figure 1 depicts a transition system that encodes the specification of an n-positions buffer $P(n)$ handled by a docking station. Intuitively, UAVs cannot perform d actions when the buffer is full (state p_n) and u actions when the buffer is empty (state p_0). Since synchronization actions s and t are immaterial, they label self-loops. □

2.1 Language semantics versus state semantics

Natural projection is commonly defined over (sets of) *words* [32]. Words are finite sequences of actions, i.e., symbols labeling each transition between two states of a finite-state automaton (FSA). The language of an FSA is the set of all words that label a sequence of transitions from an initial state to some special, e.g., final or marking, state. We call the function $\mathcal{L}$ that maps each FSA to the corresponding language *language semantics*. Given a system T and a specification S, both as FSAs, then T is said to satisfy S whenever $\mathcal{L}(T) \subseteq \mathcal{L}(S)$.

For partial model checking, the specification S is defined by a formula of the μ-calculus. In this case, the standard interpretation is given through a *state semantics*, i.e., a function that given a system T and a formula Φ returns the set of states of T that satisfy Φ. Usually, T is given as a labeled transition system (LTS). An LTS is similar to an FSA, but with a weaker notion of acceptance, where all states are final. A set of evaluation rules formalizes whether a state satisfies a formula or not. Given an LTS T and a μ-calculus formula Φ, we say that T satisfies Φ whenever its initial state does.

The language semantics of temporal logics is strictly less expressive than the state-based one [15]. A similar fact holds for FSA and regular expressions [5]. Here we use a semantics from which both the state-based and the language semantics can be obtained.

Fig. 2. From left to right, two UAVs adding to (A) and removing from (B) the buffer.

2.2 Operational model and natural projection

We now slightly generalize the existing approaches based on partial model checking and on supervisory control theory used for locally verifying global properties of discrete event systems. We then constructively prove that the two approaches are equally expressive so that techniques from one can be transferred to the other. To this end, we consider models expressed as (finite) labeled transition systems, which describe the behavior of discrete systems.

Definition 1. *A labeled transition system (LTS) is a tuple* $A = (S_A, \Sigma_A, \rightarrow_A, \imath_A)$ *where* S_A *is a finite set of states (with* $\imath_A$ *the initial state),* Σ_A *is a finite set of action labels, and* $\rightarrow_A$*:* $S_A \times \Sigma_A \rightarrow S_A$ *is the transition function. We write* $t = s \xrightarrow{a} s'$ *to denote a* transition*, whenever* $\rightarrow_A (a, s) = s'$*, and we call the state* s *the* source*, the symbol* a *the* action*, and the state* s' *the* destination*.*

A trace $\sigma \in \mathcal{T}$ *of an LTS* A *is either a single state* s *or a sequence of transitions* $t_1 \cdot t_2 \cdot \ldots$ *such that for each* t_i*, its destination is the source of* t_{i+1} *(if any). When unnecessary, we omit the source of* t_{i+1}*, and write a trace as the sequence* $\sigma = s_0\mathbf{a_1}s_1\mathbf{a_2}s_2 \ldots \mathbf{a_n}s_n$*, alternating elements of* S_A *and* Σ_A *(in boldface for readability). Finally, we denote by* $[\![A, s]\!]$ *the set of traces of* A *starting from state* s *and we write* $[\![A]\!]$ *for* $[\![A, \imath_A]\!]$*, i.e., for those traces starting from the initial state* $\imath_A$*.* □

Example 3. Consider again our running example. Figure 2 depicts the LTSs A and B. A models an UAV that deposits (d) two items in the buffer and performs a synchronization action (s). Optionally, A can also leave the docking station (t). In contrast, B repeatedly picks an item and synchronizes. Both A and B may also leave the docking station (t). Notice that the traces $[\![A]\!]$ starting from the initial state of A are $[\![A]\!] = \{q_0, q_0\mathbf{d}q_1, q_0\mathbf{t}q_3, q_0\mathbf{d}q_1\mathbf{d}q_2, q_0\mathbf{d}q_1\mathbf{d}q_2\mathbf{s}q_0, \ldots\}$. In contrast, the traces starting from the initial state of B are

$$[\![B]\!] = \{r_0, r_0\mathbf{u}r_1, r_0\mathbf{t}r_2, r_0\mathbf{u}r_1\mathbf{s}r_0, r_0\mathbf{u}r_1\mathbf{s}r_0\mathbf{u}r_1, \ldots\}$$ □

Typically, a system, or *plant* in control theory terms, consists of multiple interacting components, running in parallel. Below we rephrase the synchronous product of [32]. Intuitively, when two LTSs are put in parallel, each proceeds asynchronously, except on those actions they share, upon which they synchronize.

Definition 2. *Given two LTSs* A *and* B *such that* $\Sigma_A \cap \Sigma_B = \Gamma$*, the* synchronous product *of* A *and* B *is* $A \parallel B = (S_A \times S_B, \Sigma_A \cup \Sigma_B, \rightarrow_{A\parallel B}, \langle \imath_A, \imath_B \rangle)$*,*

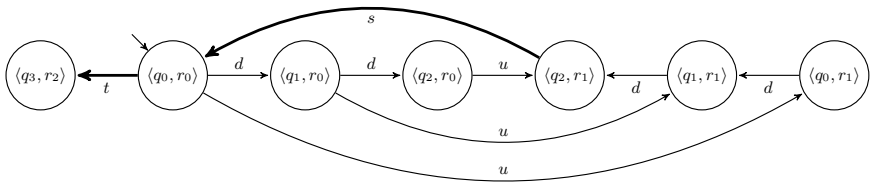

Fig. 3. The synchronous product $A \parallel B$ (bold transitions denote synchronous moves).

where $\rightarrow_{A\parallel B}$ *is as follows:*

$\langle s_A, s_B\rangle \xrightarrow{a}_{A\parallel B} \langle s'_A, s_B\rangle$ if $s_A \xrightarrow{a}_A s'_A$ and $a \in \Sigma_A \setminus \Gamma$
$\langle s_A, s_B\rangle \xrightarrow{b}_{A\parallel B} \langle s_A, s'_B\rangle$ if $s_B \xrightarrow{b}_B s'_B$ and $b \in \Sigma_B \setminus \Gamma$
$\langle s_A, s_B\rangle \xrightarrow{\gamma}_{A\parallel B} \langle s'_A, s'_B\rangle$ if $s_A \xrightarrow{\gamma}_A s'_A, s_B \xrightarrow{\gamma}_A s'_B,$ and $\gamma \in \Gamma$. □

Example 4. Consider again the LTSs A and B of Figure 2. Their synchronous product $A \parallel B$ (with $\Gamma = \{s, t\}$) is depicted in Figure 3. □

Next, we generalize the natural projection on languages, which is a kind of inverse operation of the synchronous product of two LTSs. Given a computation of $A \parallel B$, the natural projection extracts the relevant trace of one of the LTSs, including the synchronized transitions (see the second case below). Note that, unlike the definition given, e.g., in [32], our definition returns a sequence of transitions, including both states and actions. We also define the inverse projection in the expected way.

Definition 3. *For LTSs A and B with $\Gamma = \Sigma_A \cap \Sigma_B$, the* natural projection on A *of a trace σ, in symbols $P_A(\sigma)$, is defined as follows:*

$$\begin{array}{lll} P_A(\langle s_A, s_B\rangle) & = s_A & \\ P_A(\langle s_A, s_B\rangle \xrightarrow{a}_{A\parallel B} \langle s'_A, s'_B\rangle \cdot \sigma) & = s_A \xrightarrow{a}_A s'_A \cdot P_A(\sigma) & \text{if } a \in \Sigma_A \\ P_A(\langle s_A, s_B\rangle \xrightarrow{b}_{A\parallel B} \langle s_A, s'_B\rangle \cdot \sigma) & = P_A(\sigma) & \text{if } b \in \Sigma_B \setminus \Gamma. \end{array}$$

Natural projection on second component B is analogously defined. We extend the natural projection to sets of traces in the usual way: $P_A(T) = \{P_A(\sigma) \mid \sigma \in T\}$.

The inverse projection *of a trace σ over an LTS $A \parallel B$, in symbols $P_A^{-1}(\sigma)$, is defined as $P_A^{-1}(\sigma) = \{\sigma' \mid P_A(\sigma') = \sigma\}$. Its extension to sets is $P_A^{-1}(T) = \bigcup_{\sigma\in T} P_A^{-1}(\sigma)$.* □

Example 5. Consider the two traces $\sigma_1 = \langle q_0, r_0\rangle\mathbf{d}\langle q_1, r_0\rangle\mathbf{u}\langle q_1, r_1\rangle\mathbf{d}\langle q_2, r_1\rangle\mathbf{s}\langle q_0, r_0\rangle$ and $\sigma_2 = \langle q_0, r_0\rangle\mathbf{u}\langle q_0, r_1\rangle\mathbf{d}\langle q_1, r_1\rangle\mathbf{d}\langle q_2, r_1\rangle\mathbf{s}\langle q_0, r_0\rangle$. We have that the projections $P_A(\sigma_1) = P_A(\sigma_2) = q_0\mathbf{d}q_1\mathbf{d}q_2\mathbf{s}q_0 \in [\![A]\!]$, and $\sigma_1, \sigma_2 \in P_B^{-1}(q_0\mathbf{d}q_1\mathbf{d}q_2\mathbf{s}q_0)$. Also notice that all the traces of the form $(\langle q_0, r_0\rangle\mathbf{d})^*\sigma_1$ belong to $P_B^{-1}(q_0\mathbf{d}q_1\mathbf{d}q_2\mathbf{s}q_0)$. □

Two classical properties [32] concerning the interplay between the synchronous product and the natural projection hold, the proofs of which are trivial.

Fact 1. $P_A([\![A \parallel B]\!]) \subseteq [\![A]\!]$ *and* $[\![A \parallel B]\!] = P_B^{-1}([\![A]\!]) \cap P_A^{-1}([\![B]\!])$.

2.3 Equational μ-calculus and partial model checking

Below, we recall the variant of μ-calculus commonly used in partial model checking called *modal equations* [1]. A specification is given as a sequence of modal equations, and one is typically interested in the value of the top variable that is the simultaneous solution to all the equations. Equations have variables on the left-hand side and assertions on the right-hand side. Assertions are built from the boolean constants *ff* and *tt*, variables x, boolean operators $\wedge$ and $\vee$, and modalities for necessity $[\cdot]$ and for possibility $\langle\cdot\rangle$. Equations also have fixpoint operators (minimum μ and maximum ν) over variables x, and can be organized in equation systems.

Definition 4 (Syntax of the μ-calculus). *Given a set of variables $x \in X$ and an alphabet of actions $a \in \Sigma$,* assertions *$\varphi, \varphi' \in \mathcal{A}$ are given by the syntax:*

$$\varphi ::= ff \mid tt \mid x \mid \varphi \wedge \varphi' \mid \varphi \vee \varphi' \mid [a]\varphi \mid \langle a\rangle\varphi.$$

An equation *is $x =_\pi \varphi$, where $\pi \in \{\mu, \nu\}$, μ denotes a minimum fixed point equation, and ν a maximum one. An* equation system *Φ is a possibly empty sequence (ϵ) of equations, where each variable x occurs in the left-hand side of at most single equation. Thus Φ is given by*

$$\Phi ::= x =_\pi \varphi; \Phi \mid \epsilon.$$

A top assertion *$\Phi{\downarrow}x$ projects the simultaneous solution of an equation system Φ onto the top variable x.* □

We define the semantics of modal equations in terms of the traces of an LTS by extending the usual state semantics of [1] as follows. First, given an assertion φ, its state semantics $\|\varphi\|_\rho$ is given by the set of states of an LTS that satisfy φ in the context ρ, where the function ρ assigns meaning to variables. The boolean connectives are interpreted as intersection and union. The possibility modality $\|\langle a\rangle\varphi\|_\rho$ (respectively, the necessity modality $\|[a]\varphi\|_\rho$) denotes the states for which some (respectively, all) of their outgoing transitions labeled by a lead to states that satisfy φ. For more details on μ-calculus see [21,8].

Definition 5 (Semantics of the μ-calculus [1]). *Let A be an LTS, and $\rho : X \to 2^{S_A}$ be an* environment *that maps variables to sets of A's states. Given an assertion φ, the* state semantics *of φ is the mapping $\|\cdot\| : \mathcal{A} \to (X \to 2^{S_A}) \to 2^{S_A}$ inductively defined as follows.*

$$\|ff\|_\rho = \emptyset \qquad \|tt\|_\rho = S_A \qquad \|x\|_\rho = \rho(x)$$

$$\|\varphi \wedge \varphi'\|_\rho = \|\varphi\|_\rho \cap \|\varphi'\|_\rho \qquad \|[a]\varphi\|_\rho = \{s \in S_A \mid \forall s'.s \xrightarrow{a}_A s' \Rightarrow s' \in \|\varphi\|_\rho\}$$

$$\|\varphi \vee \varphi'\|_\rho = \|\varphi\|_\rho \cup \|\varphi'\|_\rho \qquad \|\langle a\rangle\varphi\|_\rho = \{s \in S_A \mid \exists s'.s \xrightarrow{a}_A s' \wedge s' \in \|\varphi\|_\rho\}$$

We extend the state semantics from assertions to equation systems. First we introduce some auxiliary notation. The empty mapping is represented by $[\,]$, $[x \mapsto U]$ *is the environment where* U *is assigned to* x, *and* $\rho \circ \rho'$ *is the mapping obtained by composing* ρ *and* ρ'. *Given a function* $f(U)$ *on the powerset of* S_A, *let* $\pi U.f(U)$ *be the corresponding fixed-point. We now define the semantics of equation systems by:*

$$\begin{array}{lll} \|\epsilon\|_\rho & = [\,] & \\ \|x =_\pi \varphi; \Phi\|_\rho & = R(U^*) & \text{where } U^* = \pi U.\|\varphi\|_{\rho \circ R(U)} \\ & & \text{and } \; R(U) = [x \mapsto U] \circ \|\Phi\|_{\rho \circ [x \mapsto U]}. \end{array}$$

Finally, for top assertions, let $\|\Phi \downarrow x\|$ *be a shorthand for* $\|\Phi\|_{[\,]}(x)$. □

Note that whenever we apply function composition $\circ$, its arguments have disjoint domains. Next, we present the trace semantics: a trace starting from a state s satisfies φ if s does.

Definition 6. *Given an LTS* A, *an environment* ρ, *and a state* $s \in S_A$, *the* trace semantics *of an assertion* φ *is a function* $\langle\!\langle \cdot \rangle\!\rangle : \mathcal{A} \to S_A \to (X \to 2^{S_A}) \to \mathcal{T}$, *that we also extend to equation systems, defined as follows.*

$$\langle\!\langle \varphi \rangle\!\rangle^s_\rho = \begin{cases} [\![A, s]\!] \text{ if } s \in \|\varphi\|_\rho \\ \emptyset \text{ otherwise} \end{cases} \qquad \langle\!\langle \Phi \rangle\!\rangle_\rho = \lambda x. \bigcup_{s \in \|\Phi\|_\rho(x)} [\![A, s]\!].$$

We write $\langle\!\langle \Phi \downarrow x \rangle\!\rangle$ *in place of* $\langle\!\langle \Phi \rangle\!\rangle_{[\,]}(x)$. □

Example 6. Consider $\Phi \downarrow x$ where $\Phi = \{x =_\mu [d]y \wedge \langle u \rangle tt; y =_\nu \langle d \rangle x \vee \langle s \rangle x\}$. We compute $\|\Phi \downarrow x\|$ with respect to $A \parallel B$. $\|\Phi \downarrow x\| = U^* = \mu U.F(U)$ where $F(U) = \|[d]y \wedge \langle u \rangle tt\|_{[x \mapsto U, y \mapsto G(U)]}$ and $G(U) = \nu U'.\|\langle d \rangle x \vee \langle s \rangle x\|_{[x \mapsto U, y \mapsto U']} = \|\langle d \rangle x \vee \langle s \rangle x\|_{[x \mapsto U]}$ (since y does not occur in the assertion). Following the Knaster-Tarski theorem, we compute $U^* = \bigcup^n F^n(\emptyset)$:

1. $G(\emptyset) = \|\langle d \rangle x \vee \langle s \rangle x\|_{[x \mapsto \emptyset]} = \emptyset$ and $U^1 = F(\emptyset) = \|[d]y \wedge \langle u \rangle tt\|_{[x \mapsto \emptyset, y \mapsto \emptyset]} = \{\langle q_2, r_0 \rangle\}$ (i.e., the only state that admits u but not d).
2. $G(\{\langle q_2, r_0 \rangle\}) = \|\langle d \rangle x \vee \langle s \rangle x\|_{[x \mapsto \{\langle q_2, r_0 \rangle\}]} = \{\langle q_1, r_0 \rangle\}$ (since $\langle q_1, r_0 \rangle \xrightarrow{d} \langle q_2, r_0 \rangle$) and $U^2 = F(\{\langle q_2, r_0 \rangle\}) = \|[d]y \wedge \langle u \rangle tt\|_{[x \mapsto \{\langle q_2, r_0 \rangle\}, y \mapsto \{\langle q_1, r_0 \rangle\}]} = \{\langle q_0, r_0 \rangle, \langle q_2, r_0 \rangle\}$.
3. $G(U^2) = \|\langle d \rangle x \vee \langle s \rangle x\|_{[x \mapsto \{\langle q_0, r_0 \rangle, \langle q_2, r_0 \rangle\}]} = \{\langle q_1, r_0 \rangle, \langle q_2, r_1 \rangle\}$ and $U^3 = F(U^2) = \|[d]y \wedge \langle u \rangle tt\|_{[x \mapsto U^2, y \mapsto G(U^2)]} = \{\langle q_0, r_0 \rangle, \langle q_2, r_0 \rangle\}$.

Since $U^2 = U^3$, we have obtained the fixed point U^*. Finally, we can compute $\langle\!\langle \Phi \downarrow x \rangle\!\rangle$, which amounts to $[\![A, \langle q_0, r_0 \rangle]\!] \cup [\![A, \langle q_2, r_0 \rangle]\!]$. □

We now define when an LTS satisfies an equation system. Recall that $[\![A]\!]$ stands for $[\![A, \imath_A]\!]$.

Definition 7. *An LTS* A *satisfies a top assertion* $\Phi \downarrow x$, *in symbols* $A \models_s \Phi \downarrow x$, *if and only if* $\imath_A \in \|\Phi \downarrow x\|$. *Moreover, let* $A \models_\sigma \Phi \downarrow x$ *if and only if* $[\![A]\!] \subseteq \langle\!\langle \Phi \downarrow x \rangle\!\rangle$. □

The following fact relates the notion of satisfiability defined in terms of state semantics ($\models_s$) with the one based on trace semantics ($\models_\sigma$); its proof is immediate by Definition 6.

Fact 2. *$A \models_s \Phi{\downarrow}x$ if and only if $A \models_\sigma \Phi{\downarrow}x$.*

As previously mentioned, partial model checking is based on the *quotienting* operation $/\!\!/$. Roughly, the idea is to specialize the specification of a composed system on a particular component. Below, we define the *quotienting* operation [1] on the LTS $A \parallel B$. Quotienting reduces $A \parallel B \models_s \Phi$ to solving $B \models_s \Phi{\downarrow}x/\!\!/_B A$. Note that each equation of the system Φ gives rise to a system of equations, one for each state s_i of A, all of the same kind, minimum or maximum (thus forming a π-*block* [3]). This is done by introducing a fresh variable x_{s_i} for each state s_i. Intuitively, the equation $x_{s_i} =_\pi \varphi/\!\!/_{\Sigma_B} s_i$ represents the requirements on B when A is in state s_i. Since the occurrence of the variables on the right-hand side depends on the A's transitions, $\Phi{\downarrow}x/\!\!/_B A$ embeds the behavior of A.

Definition 8. *Given a top assertion $\Phi \downarrow x$, we define the quotienting of the assertion on an LTS A with respect to an alphabet Σ_B as follows.*

$$\Phi{\downarrow}x/\!\!/_{\Sigma_B} A = (\Phi/\!\!/_{\Sigma_B} A){\downarrow}x_{\imath_A}, \text{ where}$$

$$\epsilon/\!\!/_{\Sigma_B} A = \epsilon \qquad (x =_\pi \varphi; \Phi)/\!\!/_{\Sigma_B} A = \begin{cases} x_{s_1} =_\pi \varphi/\!\!/_{\Sigma_B} s_1 \\ \vdots \qquad\qquad ;\ \Phi/\!\!/_{\Sigma_B} A \quad (\forall\ s_i \in S_A) \\ x_{s_n} =_\pi \varphi/\!\!/_{\Sigma_B} s_n \end{cases}$$

$$x/\!\!/_{\Sigma_B} s = x_s \qquad tt/\!\!/_{\Sigma_B} s = tt \qquad ff/\!\!/_{\Sigma_B} s = ff$$

$$\varphi \vee \varphi'/\!\!/_{\Sigma_B} s = \varphi/\!\!/_{\Sigma_B} s \vee \varphi'/\!\!/_{\Sigma_B} s \qquad \varphi \wedge \varphi'/\!\!/_{\Sigma_B} s = \varphi/\!\!/_{\Sigma_B} s \wedge \varphi'/\!\!/_{\Sigma_B} s$$

$$(\langle a\rangle\varphi)/\!\!/_{\Sigma_B} s = \bigvee_{s \xrightarrow{a} s'} \varphi/\!\!/_{\Sigma_B} s' \qquad ([a]\varphi)/\!\!/_{\Sigma_B} s = \bigwedge_{s \xrightarrow{a} s'} \varphi/\!\!/_{\Sigma_B} s' \qquad \text{if } a \in \Sigma_A \setminus \Gamma$$

$$(\langle b\rangle\varphi)/\!\!/_{\Sigma_B} s = \langle b\rangle(\varphi/\!\!/_{\Sigma_B} s) \qquad ([b]\varphi)/\!\!/_{\Sigma_B} s = [b](\varphi/\!\!/_{\Sigma_B} s) \qquad \text{if } b \in \Sigma_B \setminus \Gamma$$

$$(\langle\gamma\rangle\varphi)/\!\!/_{\Sigma_B} s = \bigvee_{s \xrightarrow{\gamma} s'} \langle\gamma\rangle(\varphi/\!\!/_{\Sigma_B} s') \qquad ([\gamma]\varphi)/\!\!/_{\Sigma_B} s = \bigwedge_{s \xrightarrow{\gamma} s'} [\gamma](\varphi/\!\!/_{\Sigma_B} s') \qquad \text{if } \gamma \in \Gamma. \quad \square$$

Example 7. Conside the top assertion $\Phi \downarrow x$ of Example 6 and the LTSs A and B of Example 3. Quotienting $\Phi \downarrow x$ against B we obtain $\Phi/\!\!/_{\Sigma_A} B \downarrow x_{r_0}$ where

$$\Phi/\!\!/_{\Sigma_A} B = \begin{cases} x_{r_0} =_\mu [d]y_{r_0} \wedge tt \\ x_{r_1} =_\mu [d]y_{r_1} \wedge ff \\ x_{r_2} =_\mu [d]y_{r_2} \wedge ff \\ y_{r_0} =_\nu \langle d\rangle x_{r_0} \vee ff \\ y_{r_1} =_\nu \langle d\rangle x_{r_1} \vee \langle s\rangle x_{r_0} \\ y_{r_2} =_\nu \langle d\rangle x_{r_2} \vee ff \end{cases} = \begin{cases} x_{r_0} =_\mu [d]y_{r_0} \\ x_{r_1} =_\mu ff \\ x_{r_2} =_\mu ff \\ y_{r_0} =_\nu \langle d\rangle x_{r_0} \\ y_{r_1} =_\nu \langle d\rangle x_{r_1} \vee \langle s\rangle x_{r_0} \\ y_{r_2} =_\nu \langle d\rangle x_{r_2} \end{cases} = \begin{cases} x_{r_0} =_\mu [d]y_{r_0} \\ y_{r_0} =_\nu \langle d\rangle x_{r_0}. \end{cases}$$

The leftmost equations are obtained by applying the rules of Definition 8. Then we perform some trivial simplifications on the right-hand sides. Finally we reduce the number of equations by removing those unreachable from the top variable x_{r_0}. For a detailed description of the simplification strategies we refer the reader to [3]. By proceeding as in Example 6, we obtain $\{q_0, q_2, q_3\}$ as the fixpoint. $\square$

3 Unifying the Logical and the Operational Approaches

In this section we prove the equivalence between natural projection and partial model checking (Theorem 4). To start, we introduce an auxiliary definition that roughly acts as a quotienting of an environment ρ. Below, we will write $\bigoplus_{i \in I} \rho_i$ for the finite composition of functions over the elements of a set I.

Definition 9. *Given a synchronous product $A \parallel B$, we define $\Delta_B(\cdot) : (X \to 2^{S_A \times S_B}) \to (X_{S_A} \to 2^{S_B})$ as*

$$\Delta_B(\rho) = \bigoplus_{x \in Dom(\rho)} \bigoplus_{s_A \in S_A} [x_{s_A} \mapsto U_B^x(s_A)], \text{ where } U_B^x(s_A) = \{s_B \mid \langle s_A, s_B \rangle \in \rho(x)\} \quad \square$$

A technical lemma follows. Intuitively, quotienting an assertion (and an environment) preserves the semantics, i.e., a state $\langle s_A, s_B \rangle$ satisfies φ if and only if s_B satisfies the quotient of φ on B. Indeed, the following statement can be rewritten as $\|\varphi /\!\!/_{\Sigma_B} s_A\|_{\Delta_B(\rho)} = \{s_B \mid \langle s_A, s_B \rangle \in \|\varphi\|_\rho\}$.

Lemma 1. *For all A, B, ρ and φ on $A \parallel B$, $\langle s_A, s_B \rangle \in \|\varphi\|_\rho \iff s_B \in \|\varphi /\!\!/_{\Sigma_B} s_A\|_{\Delta_B(\rho)}$.*

We next extend Lemma 1 to a system of equations, providing an alternative view of quotienting an assertion on a component of a synchronous product.

Lemma 2. *For all A, B, ρ and Φ on $A \parallel B$, $\Delta_B(\|\Phi\|_\rho) = \|\Phi /\!\!/_{\Sigma_B} A\|_{\Delta_B(\rho)}$.*

The following corollary is immediate (recall that x_{s_A} is the variable corresponding to the quotient of x on s_A).

Corollary 1. *For all A, B, ρ, x and Φ on $A \parallel B$,*

$$\langle s_A, s_B \rangle \in \|\Phi\|_\rho(x) \iff s_B \in \|\Phi /\!\!/_{\Sigma_B} A\|_{\Delta_B(\rho)}(x_{s_A}).$$

We next establish the correspondence between quotienting and natural projection.

Theorem 3. *For all A, B, x and Φ on $A \parallel B$, $\langle\!\langle \Phi \downarrow x /\!\!/_{\Sigma_B} A \rangle\!\rangle = P_B(\langle\!\langle \Phi \downarrow x \rangle\!\rangle)$.*

The following theorem states that the synchronous product of two LTSs satisfies a global equation system if and only if its components satisfy their quotients, i.e., their local assertions.

Theorem 4. *For all A, B, x and Φ on $A \parallel B$,*

$$A \parallel B \models_\varsigma \Phi \downarrow x \qquad (\varsigma \in \{s, \sigma\})$$

if and only if any of the following equivalent statements holds

1. $A \models_\varsigma \Phi \downarrow x /\!\!/_{\Sigma_A} B$ *2.* $B \models_\varsigma \Phi \downarrow x /\!\!/_{\Sigma_B} A$
3. $A \models_\sigma P_A(\langle\!\langle \Phi \downarrow x \rangle\!\rangle)$ *4.* $B \models_\sigma P_B(\langle\!\langle \Phi \downarrow x \rangle\!\rangle)$

4 Quotienting Finite-State Systems

In this section we present an algorithm for quotienting a finite-state system defined as an LTS. Afterwards, we prove its correctness with respect to the standard quotienting operator and we study its complexity. Finally, we apply it to our working example to address three problems, i.e., verification, submodule construction, and controller synthesis.

4.1 Quotienting algorithm

Our algorithm consists of two procedures that are applied sequentially. The first, called `quotient` (Table 2), builds a non-deterministic transition system from two LTSs, i.e., a specification P and an agent A. Moreover, it takes as an argument the alphabet of actions of the new transition system. Non-deterministic transition systems have a distinguished label λ, and serve as an intermediate representation. The states of the resulting transition system include all the pairs of states of P and A, except for those that denote a violation of P (line 1). The transition relation (line 3) is defined by applying the quotienting rules from Section 2. Also, notice that the transition relation $\rightarrow$ is restricted to the states of S (denoted $\rightarrow_S$).

The second procedure, called `unify` (given in Table 3) translates a non-deterministic transition system back to an LTS. Using closures over λ, `unify` groups states of the transition system. This process is similar to the standard subset construction algorithm [18], except that we put an $a \in \Sigma_B \setminus \Gamma$ transition between two groups Q and M only if (i) M is the intersection of the λ-closures of the states reachable from Q with an a transition and (ii) all the states of Q admit at least an a transition leading to a state of M ($\wedge$`-move`). Procedure `unify` works as follows. Starting from the λ-closure of B's initial state (line 1) it repeats a partition generation cycle (lines 4–13). Each cycle removes an element Q from the set S of the partitions to be processed. Then, for all the actions in $\Sigma_B \setminus \{\lambda\}$, a partition M is computed by $\wedge$`-move` (line 7). If the partition is not empty a new transition is added from Q to M (line 9). Also, if M is a freshly generated partition, i.e., $M \notin R$, it is added to both S and R (line 10). The procedure terminates when no new partitions are generated.

Our quotienting algorithm is correct with respect to the quotienting operator and runs in PTIME. A detailed account of the correctness and complexity of the algorithm is in Appendix B.

4.2 Prototype and application to the running example

We implemented the algorithm presented above as part of a tool suite for the partial evaluation of finite state models called the *partial evaluator of simple transition systems* (PESTS).[6] We applied the prototype to some case studies,

[6] The tools in our library work on FSA. The source code and case studies are available at `https://github.com/SCPTeam/pests`

Begin proc quotient
  input P = $<S_P\,,\ \Sigma_P\,,\ \rightarrow_P\,,\ i_P>$
  input A = $<S_A\,,\ \Sigma_A\,,\ \rightarrow_A\,,\ i_A>$
  input Σ_B

1: $S := (S_P \times S_A) \setminus \bigcup_{a \in \Sigma_A} \{(s_P, r_A) \mid s_P \stackrel{a}{\not\rightarrow}_P \wedge r_A \stackrel{a}{\rightarrow}_A\}$
2: $i := (i_P, i_A)$
3: $\rightarrow := \bigcup_{s_P} \begin{cases} \bigcup_{a \in \Sigma_A \setminus \Gamma} \{((s_P, r_A), \lambda, (s'_P, r'_A)) \mid s_P \stackrel{a}{\rightarrow}_P s'_P \wedge r_A \stackrel{a}{\rightarrow}_A r'_A\} \\ \bigcup_{a \in \Sigma_B \setminus \Gamma} \{((s_P, r_A), a, (s'_P, r_A)) \mid s_P \stackrel{a}{\rightarrow}_P s'_P\} \\ \bigcup_{a \in \Gamma} \{((s_P, r_A), a, (s'_P, r'_A)) \mid s_P \stackrel{a}{\rightarrow}_P s'_P \wedge r_A \stackrel{a}{\rightarrow}_A r'_A\} \end{cases}$
4: B := $<S\,,\ \Sigma_B\,,\ \rightarrow_S\,,\ i>$
5: output unify(B)
End proc

Table 2. The quotienting algorithm.

including a real world one based on a flexible manufacturing system.[7] For the sake of presentation we only show here the application to the running example. In particular, we leverage our algorithm to address three different problems: (*i*) reducing the verification of a parallel composition to that of a single component, (*ii*) synthesizing a submodule that respects a global specification (SCP), and (*iii*) synthesizing a controller for a given component (CSP).

Verification. Here we want to check whether $A \parallel B \not\models_s P(2)$. To do that we follow the approach of [1], i.e., we start by quotienting the specification $P(2)$ against A (see Figure 2). The result is a two-state specification P' having a single transition labeled with t. Clearly $B \not\models_s P'$ and a counterexample is the trace $\sigma = \mathbf{r_0} u \mathbf{r_1} s \mathbf{r_0} t \mathbf{r_2}$, as $\sigma \in [\![B]\!]$ while $\sigma \notin [\![P']\!]$. As a consequence $A \parallel B \not\models_s P$. Intuitively, this is because after the two d actions, A performs a single s, which is insufficient to delimit a "safety zone" for actions u by B (which might occur too late, e.g., after another d by A). Thus, P' does not allow an s that might permit A to carry out the third d move before a u action.

Fixing the example. Given to the previous reasoning argument, we cannot synthesize meaningful submodules and controllers starting from A. To fix our example, we therefore replace A with A' as depicted in Figure 4. A' resembles A but has an extra state q_4 that enables a second s transition. Intuitively, it represents the "safety zone" just described.

SCP. We now apply the quotienting algorithm to A' in the case of buffer sizes 2 and 3 to construct the submodules that comply with $P(2)$ and $P(3)$, respectively. Thus, we set $\Sigma_B = \{u, s, t\}$. In this way, the quotienting algorithm generates a

[7] *FlexFact* `http://www.rt.eei.uni-erlangen.de/FGdes/productionline.html`

```
Begin proc unify
  input B = <S_B , Σ_B , →_B , i_B>

 1: I := λ−close({i_B})
 2: R, S := {I}
 3: → := ∅
 4: while S ≠ ∅ do
 5:   Q := pick&remove(S)
 6:   for each a ∈ Σ_B \ {λ}
 7:     M := ∧−move(Q, a)
 8:     if M ≠ ∅ then
 9:       → := → ∪ {(Q, a, M)}
10:       if M ∉ R then S := S ∪ {M}; R := R ∪ {M} end if
11:     end if
12:   end for
13: end while
14: output <R, Σ_B \ {λ}, →, I>
end proc
```

```
Begin proc ∧−move
  input Q
  input a

1:  M := λ−close( ⋂_{q∈Q} {q' | q -a->_B q'} )
2:  output M
end proc
```

Table 3. The unification algorithm.

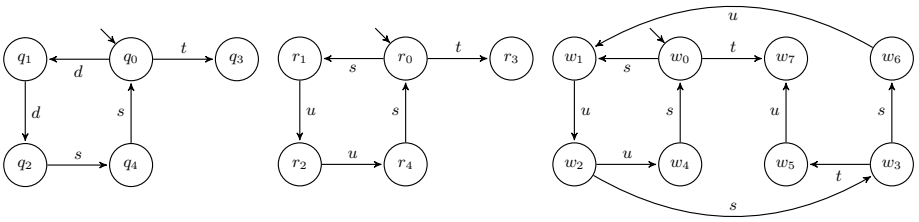

Fig. 4. Graphical representation of A', B_2 and B_3 (from left to right).

component that, not only synchronizes through actions s and t, but also performs actions u autonomously. The resulting agents B_2 and B_3 appear in Figure 4.

The agent B_2 synchronizes on the first s transition of A' to ensure that both the d actions have been performed. Afterwards, the buffer must be cleared (two u actions) before synchronizing again on s (thereby permitting A' to cycle). Synchronizing on t is also possible. In this case, no further modifications of the buffer happen.

With a buffer of size 3, the agent B_3 is more complex. Intuitively, A' can perform its two d actions only when the buffer contains at most 1 item. Thus, B_3 has two loops. The inner loop (passing through the states $w_0\, w_1\, w_2\, w_4$) is analogous to that of B_2 (where two u actions are performed in sequence) and, if completed, it empties the buffer. Moreover, the specification includes an external loop ($w_2\, w_3\, w_6\, w_1$) that removes two elements from the full buffer of size 3. As expected, the two cycles can be combined. Finally, notice that the action t

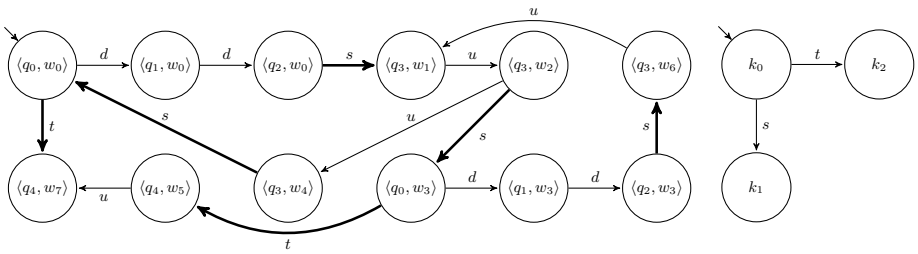

Fig. 5. Graphical representation of $A' \parallel B_3$ and C (from left to right).

can occur under two conditions: if the buffer contains no items (w_0) or exactly 1 item (w_3). In the second case, a final u action (w_5) can occur.

CSP. We consider now the problem of synthesizing a controller for $A' \parallel B_3$ (see Figure 5). In particular, we want a controller to enforce $P(2)$ on it.[8] To generate the controller, we run the quotienting algorithm with $\Sigma_B = \{s, t\}$, i.e., we force the algorithm to build a component that can only synchronize on the *controllable* actions s and t. The resulting controller is depicted on the right of Figure 5. Intuitively, the controller only admits two operations: either t or s. The first case is when A' and B_3 terminate (state $\langle q_4, w_7 \rangle$). Otherwise, only a single s action can occur. As a matter of fact, after one s action the target reaches $\langle q_3, w_1 \rangle$ completely filling in the stack with two d actions. The system can then reach both $\langle q_3, w_2 \rangle$ and $\langle q_3, w_4 \rangle$. Since an s action leads from $\langle q_3, w_2 \rangle$ to $\langle q_0, w_3 \rangle$, where the system can perform other two d transitions, it is not allowed.

5 Related Work

Natural projection is mostly used by the community working on control theory and on discrete-event systems. In the 1980s, the seminal works by Wonham et al. (e.g., [33,34]) exploited natural projection-based algorithms for synthesizing both local and global controllers. Along this line of research, other authors proposed extensions and refinements, relying on natural projection (e.g., see [12,13,31,23]).

Partial model checking has been successfully applied to the synthesis of controllers. Given an automaton representing a plant and a μ-calculus formula, Basu and Kumar [6] compute the quotient of the specification with respect to the plant. The satisfiability of the resulting formula is checked using a tableau that also returns a valid model yielding the controller. Their tableau works similarly to our quotienting algorithm, but applies to a more specific setting, as they are interested in generating controllers. In contrast, Martinelli and Matteucci [25] use partial model checking to generate a control process for a partially unspecified system in order to guarantee the compliance against a μ-calculus formula. The generated controller takes the form of an edit automaton [7].

[8] Notice that $A' \parallel B_3$ does not comply with $P(2)$ as B_3 was synthesized from $P(3)$.

Some authors proposed techniques based on the formal verification of temporal logics for addressing CSP. Arnold et al. [4] were among the first to control a deterministic plant with a μ-calculus specification. Also Ziller and Schneider [35] and Riedwge and Pinchinat [28] reduce the problem of synthesizing a controller to check the satisfiability of a formula of (variants of) the μ-calculus. A similar approach was presented by Jiang and Kumar [19] and Gromyko et al. [16]. Similarly to [35] and [28], [19] puts forward an approach that reduces the problem of synthesizing a controller to that of checking a CTL* formula's satisfiability. In contrast, [16] proposes a method based on symbolic model checking to synthesize a controller. Their approach applies to (a fragment of) CTL.

6 Conclusion

Our work goes in the same direction of [11] and provides new results that build a new bridge between supervisory control theory and formal verification. In particular, we have formally established the relationship between partial model checking and natural projection. Natural projection have been reduced to partial model checking and proved equivalent, under common assumptions. This equivalence helps to explain why some authors use partial model checking and others use natural projection to synthesize controllers. We have also developed a working prototype that has been applied to the running example of the paper as well as to a realistic case study.

Besides establishing an interesting and novel connection, our work also opens new directions for investigation. Since natural projection is related to language theory in general, there could be other application fields where partial model checking can be used as an alternative. The original formulation of the partial model checking applies to the μ-calculus, while our quotienting algorithm works on LTSs. To the best of our knowledge, no quotienting algorithms exist for formalisms with a different expressive power, such as LTL or CTL.

References

1. Henrik Reif Andersen. Partial model checking (extended abstract). In *Proceedings of Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 398–407. IEEE Computer Society Press, 1995.
2. Henrik Reif Andersen and Jørn Lind-Nielsen. MuDiv: A tool for partial model checking. Demo presentation at CONCUR, 1996.
3. Henrik Reif Andersen and Jorn Lind-Nielsen. Partial model checking of modal equations: A survey. *International Journal on Software Tools for Technology Transfer*, 2(3):242–259, 1999.
4. A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controllers with partial observation. *Theor. Comput. Sci.*, 1(303):7–34, 2003.
5. Jos C. M. Baeten, Bas Luttik, Tim Muller, and Paul Van Tilburg. Expressiveness modulo bisimilarity of regular expressions with parallel composition. *Mathematical Structures in Computer Science*, 26:933–968, 2016.
6. Samik Basu and Ratnesh Kumar. Quotient-based approach to control of nondeterministic discrete-event systems with -calculus specification, 2006. Available at `http://home.eng.iastate.edu/~rkumar/PUBS/acc06-muctrl.pdf`.
7. Lujo Bauer, Jarred Ligatti, and David Walker. More enforceable security policies. In *Foundations of Computer Security*, Copenhagen, Denmark, July 2002.
8. Julian Bradfield and Colin Stirling. *Handbook of Modal Logic, Volume 3*, chapter Modal Mu-Calculi. Elsevier Science, 2006.
9. Christos G. Cassandras and Stéphane Lafortune. *Introduction to Discrete Event Systems*. Kluwer, 1999.
10. Open Fog Consortium. Out of the Fog: Use Case Scenarios. Supply Chain, High-Scale Drone Package Delivery. `https://www.openfogconsortium.org/wp-content/uploads/OpenFog-Transportation-Drone-Delivery-Use-Case.pdf`, July 2016. Accessed on January 2017.
11. Rüdiger Ehlers, Stéphane Lafortune, Stavros Tripakis, and Moshe Vardi. Bridging the Gap between Supervisory Control and Reactive Synthesis: Case of Full Observation and Centralized Control. *IFAC Proceedings Volumes*, 47(2):222 – 227, 2014.
12. Lei Feng and W. Murray Wonham. TCT: A computation tool for supervisory control synthesis. In *Proceedings of 2006 8th International Workshop on Discrete Event Systems*, pages 388–389, 2006.
13. Lei Feng and W. Murray Wonham. On the computation of natural observers in discrete-event systems. *Discrete Event Dynamic Systems*, 20(1):63–102, 2010.
14. Guillaume Feuillade and Sophie Pinchinat. Modal specifications for the control theory of discrete event systems. *Discrete Event Dynamic Systems*, 17(2):211–232, 2007.
15. Roberto Giacobazzi and Francesco Ranzato. States vs. traces in model checking by abstract interpretation. In *Proceedings of The 9th International Static Analysis Symposium, SAS'02*, volume 2477 of *Lecture Notes in Computer Science*, pages 461–476. Springer, 2002.
16. A. Gromyko, M. Pistore, and P. Traverso. A Tool for Controller Synthesis via Symbolic Model Checking. In *8th International Workshop on Discrete Event Systems*, pages 475–476, July 2006.
17. Jan Friso Groote and Mohammad Reza Mousavi. *Modeling and Analysis of Communicating Systems*. The MIT Press, 2014.

18. John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.
19. S. Jiang and R. Kumar. Supervisory control of discrete event systems with ctl* temporal logic specifications. *SIAM J. Control and Optimization*, 44(6):2079–2103, 2006.
20. Galina Jirásková and Tomáš Masopust. On a structural property in the state complexity of projected regular languages. *Theoretical Computer Science*, 449:93–105, 2012.
21. D. Kozen. Results on the propositional mu-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983.
22. Frédéric Lang and Radu Mateescu. *Partial Model Checking Using Networks of Labelled Transition Systems and Boolean Equation Systems*, volume 7214 of *Lecture Notes in Computer Science*, pages 141–156. Springer, 2012.
23. F. Lin and W.M. Wonham. Decentralized supervisory control of discrete-event systems. *Information Sciences*, 44(3):199 – 224, 1988.
24. F. Martinelli and I. Matteucci. Synthesis of local controller programs for enforcing global security properties. In *3rd International Conference on Availability, Reliability and Security (ARES)*, pages 1120–1127, March 2008.
25. Fabio Martinelli and Ilaria Matteucci. A framework for automatic generation of security controller. *Softw. Test., Verif. Reliab.*, 22(8):563–582, 2012.
26. Philip Merlin and Gregor V. Bochmann. On the Construction of Submodule Specifications and Communication Protocols. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 5(1):1–25, January 1983.
27. T. Moor, K. Schmidt, and S. Perk. libFAUDES – An open source C++ library for discrete event systems. In *9th International Workshop on Discrete Event Systems*, pages 125–130, May 2008.
28. S. Riedweg and S. Pinchinat. Quantified mu-calculus for control synthesis. In *Mathematical Foundations of Computer Science 2003, 28th International Symposium, MFCS 2003 Proceedings*, volume 2747 of *Lecture Notes in Computer Science*, pages 642–651. Springer, 2003.
29. Karen Rudie and Lenko Grigorov. Integrated Discrete-Event Systems (IDES). Online at `https://qshare.queensu.ca/Users01/rudie/www/software.html`, (Visited on) February 2017. Department of Electrical and Computer Engineering, Queen's University in Kingston, ON, Canada.
30. Rong Su and W. Murray Wonham. Global and local consistencies in distributed fault diagnosis for discrete-event systems. *IEEE Transactions on Automatic Control*, 50(12):1923–1935, Dec 2005.
31. Kai C. Wong. On the Complexity of Projections of Discrete-Event Systems. In *Proceedings of IEEE Workshop on Discrete Event Systems*, pages 201–208, 1998.
32. W. Murray Wonham. Supervisory control of discrete-event systems. Online at `http://www.control.toronto.edu/DES`, (Visited on) February 2017. Department of Electrical and Computer Engineering, University of Toronto, ON, Canada.
33. W. Murray Wonham and P. J. Ramadge. On the supremal controllable sublanguage of a given language. In *Proceedings of the 23rd IEEE Conference on Decision and Control*, pages 1073–1080, Dec 1984.
34. W. Murray Wonham and P. J. Ramadge. Modular supervisory control of discrete-event systems. *Mathematics of Control, Signals and Systems*, 1(1):13–30, 1988.
35. R. Ziller and K. Schneider. Combining supervisor synthesis and model checking. *ACM Trans. Embedded Comput. Syst.*, 4(2):331–362, 2005.

A Technical Proofs

Lemma 1. *For all A, B, ρ and φ on $A \parallel B$, $\langle s_A, s_B\rangle \in \|\varphi\|_\rho \iff s_B \in \|\varphi /\!\!/_{\Sigma_B} s_A\|_{\Delta_B(\rho)}$.*

Proof. By induction over the structure of φ.

- Cases tt and ff. Trivial.
- Case x. By definition of $\Delta_B(\rho)$.
- Cases $\varphi \wedge \varphi'$, $\varphi \vee \varphi'$. By the induction hypothesis.
- Case $\langle a\rangle\varphi$. By the Definition 5, $\langle s_A, s_B\rangle \in \|\langle a\rangle\varphi\|_\rho$ if and only if $\exists s'_A, s'_B$ such that $\langle s_A, s_B\rangle \xrightarrow{a}_{A\parallel B} \langle s'_A, s'_B\rangle \wedge \langle s'_A, s'_B\rangle \in \|\varphi\|_\rho$. By the induction hypothesis, this is equivalent to

$$\exists s'_A, s'_B \text{ such that } \langle s_A, s_B\rangle \xrightarrow{a}_{A\parallel B} \langle s'_A, s'_B\rangle \wedge s'_B \in \|\varphi /\!\!/_{\Sigma_B} s'_A\|_{\Delta_B(\rho)} \quad (1)$$

  Then we consider three exhaustive cases.
  - $a \in \Sigma_A \setminus \Gamma$. Here $s'_B = s_B$ and (1) is satisfied if and only if $s_B \in \| \bigvee_{s_A \xrightarrow{a}_A s'} \varphi /\!\!/_{\Sigma_B} s'\|_{\Delta_B(\rho)}$. We conclude by applying Definition 8.
  - $a \in \Sigma_B \setminus \Gamma$. In this case $s'_A = s_A$ and, by Definition 5, (1) is equivalent to claiming $s_B \in \|\langle a\rangle(\varphi /\!\!/_{\Sigma_B} s_A)\|_{\Delta_B(\rho)}$. Again, we close the case by applying Definition 8.
  - $a \in \Gamma$. We combine the reasoning of the two previous cases to conclude that $s_B \in \| \bigvee_{s_A \xrightarrow{a}_A s'} \langle a\rangle(\varphi /\!\!/_B s')\|_{\Delta_{\Sigma_B}(\rho)}$.
- Case $[a]\varphi$. Symmetric to the previous one.

□

Lemma 2. *For all A, B, ρ and Φ on $A \parallel B$, $\Delta_B(\|\Phi\|_\rho) = \|\Phi /\!\!/_{\Sigma_B} A\|_{\Delta_B(\rho)}$.*

Proof. We proceed by induction on the structure of Φ.

- Base case: $\Phi = \epsilon$. Trivial.
- Inductive step: $\Phi = x =_\pi \varphi; \Phi'$. By definition $\|\Phi\|_\rho = [x \mapsto U^*] \circ \|\Phi'\|_{\rho\circ[x\mapsto U^*]}$ where U^* is the fixed point computed according to Definition 5. Thus, we have that $\Delta_B(\|\Phi\|_\rho) = \Delta_B([x \mapsto U^*] \circ \|\Phi'\|_{\rho\circ[x\mapsto U^*]}) = \Delta_B([x \mapsto U^*]) \circ \Delta_B(\|\Phi'\|_{\rho\circ[x\mapsto U^*]})$. By the induction hypothesis this reduces to

$$\Delta_B([x \mapsto U^*]) \circ \|\Phi' /\!\!/_B A\|_{\Delta_B(\rho)\circ\Delta_B([x\mapsto U^*])}. \quad (2)$$

  By Definition 9, $\Delta_B([x \mapsto U^*]) = \bigoplus_{s\in S_A} [x_s \mapsto U^*_{B,s}]$ where $U^*_{B,s} = \{s' \mid \langle s, s'\rangle \in U^*\}$. By replacing U^* with its definition we obtain that

$$U^*_{B,s} = \{s' \mid \langle s, s'\rangle \in \pi U.\|\varphi\|_{\rho\circ R(U)}\},$$

which we rewrite it to

$$U^*_{B,s} = \pi U.\{s' \mid \langle s, s' \rangle \in \|\varphi\|_{\rho \circ R(U)}\}.$$

By Lemma 1, this is equivalent to

$$U^*_{B,s} = \pi U_{B,s}.\|\varphi /\!\!/_{\Sigma_B} s\|_{\Delta_B(\rho) \circ \Delta_B(R(U))},$$

where $\Delta_B(R(U)) = \Delta_B([x \mapsto U] \circ \|\Phi'\|_{\rho \circ [x \mapsto U]})$. By Definition 9, we have $\bigoplus_{s \in S_A} [x_s \mapsto U_{B,s}] \circ \Delta_B(\|\Phi'\|_{\rho \circ [x \mapsto U]}) = \bigoplus_{s \in S_A} [x_s \mapsto U_{B,s}] \circ \|\Phi' /\!\!/_{\Sigma_B} A\|_{\Delta_B(\rho)} \circ \bigoplus_{s \in S_A} [x_s \mapsto U_{B,s}]$ (by the induction hypothesis).
As a consequence, we rewrite (2) to[9]

$$\bigoplus_{s \in S_A} [x_s \mapsto U^*_{B,s}] \circ \|\Phi' /\!\!/_{\Sigma_B} A\|_{\Delta_B(\rho)} \circ \bigoplus_{s \in S_A} [x_s \mapsto U^*_{B,s}],$$

which, after repeatedly applying Definition 5 to each element $s \in S_A$ turns out to be $\|\Phi /\!\!/_{\Sigma_B} A\|_{\Delta_B(\rho)}$.

□

Theorem 3. *For all A, B, x and Φ on $A \parallel B$, $\langle\!\langle \Phi{\downarrow}x /\!\!/_{\Sigma_B} A \rangle\!\rangle = P_B(\langle\!\langle \Phi{\downarrow}x \rangle\!\rangle)$.*

Proof. By Definition 6, the thesis becomes

$$\langle\!\langle \Phi /\!\!/_{\Sigma_B} A \rangle\!\rangle_{[\,]}(x_{\imath_A}) = P_B(\langle\!\langle \Phi \rangle\!\rangle_{[\,]}(x)),$$

which holds if and only if

$$\langle \imath_A, \imath_B \rangle \in \|\Phi\|_{[\,]}(x) \iff \imath_B \in \|\Phi /\!\!/_{\Sigma_B} A\|_{[\,]}(x_{\imath_A}).$$

The thesis follows by Corollary 1. □

Theorem 4. *For all A, B, x and Φ on $A \parallel B$,*

$$A \parallel B \models_\varsigma \Phi{\downarrow}x \qquad (\varsigma \in \{s, \sigma\})$$

if and only if any of the following equivalent statements holds

1. $A \models_\varsigma \Phi{\downarrow}x /\!\!/_{\Sigma_A} B$ *2.* $B \models_\varsigma \Phi{\downarrow}x /\!\!/_{\Sigma_B} A$
3. $A \models_\sigma P_A(\langle\!\langle \Phi{\downarrow}x \rangle\!\rangle)$ *4.* $B \models_\sigma P_B(\langle\!\langle \Phi{\downarrow}x \rangle\!\rangle)$

Proof. The equivalence of items 1 and 2 and $A \parallel B \models_\varsigma \Phi \downarrow x$ is in [1] (with the additional use of Theorem 3). The other equivalences follow immediately by Theorem 3 (and by the commutativity of $\|$). □

B Correctness and complexity

Below we discuss the correctness and computational complexity of the algorithms presented in the paper.

[9] Notice that the order of the x_s equations is immaterial as they form a π-block.

B.1 Correctness

To prove the correctness of the algorithm, we show that it is equivalent to the standard quotienting operator applied to a suitable encoding of an LTS as a specification of the equational μ-calculus.

Encoding, Given an LTS $A = \langle S, \Sigma, \rightarrow, s_i \rangle$ we build a system of equations as follows:

$$\Phi_A = \{x^{s_1} =_\mu \mathcal{A}(s_1); \ldots; x^{s_n} =_\mu \mathcal{A}(s_n)\}$$

where $S = \{s_1, \ldots, s_n\}$ and $\mathcal{A}(s) = \bigwedge_{s \not\xrightarrow{b}} [b]ff \ \wedge \ \bigwedge_{s \xrightarrow{a} s'} [a]x^{s'}$

Thus we define the top assertion of the formula derived from A as $\Phi_A \downarrow x^{s_i}$.

Quotienting. Starting from the inputs of `quotient`, we now evaluate $\Phi_P \downarrow x^{i_P} /\!\!/_{\Sigma_B} A$. The resulting equations system is

$$\Phi' = \begin{cases} x^{s_1}_{r_1} =_\mu \mathcal{A}(s_1) /\!\!/_{\Sigma_B} r_1 \\ \quad \ldots \\ x^{s_i}_{r_j} =_\mu \mathcal{A}(s_i) /\!\!/_{\Sigma_B} r_j \\ \quad \ldots \\ x^{s_n}_{r_m} =_\mu \mathcal{A}(s_n) /\!\!/_{\Sigma_B} r_m \end{cases}$$

with all the equations of the following form, where $\alpha \in \Sigma_A \setminus \Gamma$, $\beta \in \Sigma_B \setminus \Gamma$ and $\gamma \in \Gamma$.

$$x^{s_i}_{r_j} =_\mu \overbrace{\bigwedge_{\substack{s_i \not\xrightarrow{a} \\ r_j \xrightarrow{a} r'}} ff}^{(1)} \wedge \overbrace{\bigwedge_{\substack{s_i \not\xrightarrow{a} \\ r_j \not\xrightarrow{a}}} [a]ff}^{(2)} \wedge \overbrace{\bigwedge_{\substack{s_i \xrightarrow{\alpha} s' \\ r_j \xrightarrow{\alpha} r'}} x^{s'}_{r'}}^{(3)} \wedge \overbrace{\bigwedge_{s_i \xrightarrow{\beta} s'} [\beta]x^{s'}_{r_j}}^{(4)} \wedge \overbrace{\bigwedge_{\substack{s_i \xrightarrow{\gamma} s' \\ r_j \xrightarrow{\gamma} r'}} [\gamma]x^{s'}_{r'}}^{(5)}$$

Trivially, the equation system described above corresponds to the non-deterministic transition system obtained by the `quotient` algorithm. A state (s, r) results in a variable associated to an assertion that characterizes the outgoing transitions distinguishing among (1) a is required but not done, (2) a is not allowed, (3) λ moves, (4) Σ_B, and (5) Γ actions.

Correctness. To conclude we show that all the steps of the algorithms described above correspond to valid transformations, i.e., they preserve equivalence. A detailed description of the first two transformations can be found in [3].

- *Constant propagation* is applied to remove equations of the form $x =_\mu ff$. This step is carried out by the `quotient` algorithm when removing the corresponding states from the transition system.
- *Unguardedness removal* carries out the following transformation.

$$\begin{cases} x =_\mu \varphi \\ \vdots \\ y =_\mu \varphi' \end{cases} \quad \text{becomes} \quad \begin{cases} x =_\mu \varphi\{\varphi'/x\} \\ \vdots \\ y =_\mu \varphi' \end{cases}$$

All the occurrences of y in the first equation are replaced with the assertion associated to y. Notice that this transformation only applies if all the occurrences of y are unguarded, i.e., not under the scope of any modal operator, in the first equation. Also, we extend it to remove redundant recurrences, namely we transform $x =_\mu \varphi \wedge x$ in $x =_\mu \varphi$. In our algorithm, this operation corresponds to a λ-closure.

– *Variable introduction* requires more attention. It is simple to verify that the previous transformations do not preserve the structure of our equations. Indeed, unguardedness removal can introduce in the assertions more instances of the same action modality, while we require exactly one. Concretely, the assertions have the form

$$x =_\mu [a]v_1^a \wedge \ldots \wedge [a]v_k^a \wedge [b]v_1^b \cdots ,$$

where v stands for either a variable or ff. If some of the v_i^a are equal to ff, $[a](v_1^a \wedge \ldots \wedge v_k^a)$ reduces to $[a]ff$. Otherwise, we rewrite it as $x =_\mu [a](y_1 \wedge \ldots \wedge y_k) \wedge [b](\ldots)$. Thus, we replace the conjunctions of variables with $[a](y_{\{1,\ldots,k\}})$ and we introduce a new equation $y_{\{1,\ldots,k\}} =_\mu y_1 \wedge \ldots \wedge y_k$. Clearly, the number of these new variables is bounded by $2^{|S|}$. This transformation, plus unguardedness removal, corresponds to the $\wedge$`-move` operation. It is simple to notice that these transformations restore the format of our encoding, thus denoting an LTS that is the output of our algorithm.

B.2 Complexity

We estimate the worst case complexity of the `quotient` algorithm. For simplicity, we assume that $|\Gamma| = |\Sigma_A \setminus \Gamma| = |\Sigma_B \setminus \Gamma| = m$ and $|S_A| = |S_P| = n$. The first part, i.e., the generation of the non-deterministic transition system, requires at most $|\rightarrow_P| \cdot |\rightarrow_A| \leq m^2 n^4$ steps. The transition system has at most n^2 states.

Concerning `unify`, we first observe the following facts. The algorithm works on the λ-closures of the states of the transition system. Similarly to the ε-closures of an NFA, they can be computed in advance (see [18]). The cost is cubic with respect to the number of states, i.e., $O(n^6)$ in our case. The total number of closures is bounded by n^2.

At each step, $\wedge$`-move` computes the sets of the a-reachable states, starting from one of the closures (which has size at most n^2). Since there are no more than nm Γ-transitions (and at most m Σ_B-transitions), in $n^3 m$ steps we obtain the set on which we compute the λ-closure. Since we already computed them, we just need to combine the existing ones.

To conclude we observe that $\wedge$`-move` iterates at most $n^2 m$ times. In particular, new elements are added to S whenever a new λ-closure is generated. However, these closures are computed from a set of states that are a-reachable from all the states of another closure. Clearly, there can be at most $n^2 m$ distinct closures of this type, otherwise there would be more than $n^4 m^2$ edges in the transition system. The overall complexity is then $O(2n^5 m^3 + n^6 + n^4 m^2)$.

n	States	Transitions	Time (ms)
5	24	36	1
10	74	121	6
15	149	256	50
20	249	441	347
25	374	676	1205
30	524	961	3729
35	699	1296	9144
40	899	1681	21768
45	1124	2116	43750
50	1374	2601	58869
55	1649	3136	97184
60	1949	3721	197555
65	2274	4356	307549
70	2624	5041	462417
75	2999	5776	674102
80	3399	6561	1094193
85	3824	7396	1507860
90	4274	8281	2070366
95	4749	9216	2796649
100	5249	10201	5685985

n	States	Transitions	Time (ms)
5	54	54	30
10	209	209	88
15	464	464	398
20	819	819	1979
25	1274	1274	8013
30	1829	1829	23953
35	2484	2484	84634
40	3239	3239	153595
45	4094	4094	342434
50	5049	5049	194525
55	6104	6104	325718
60	7259	7259	624646
65	8514	8514	959671
70	9869	9869	1487936
75	11324	11324	2429919
80	12879	12879	3888799
85	14534	14534	5546023
90	16289	16289	7353658
95	18144	18144	9589415
100	20099	20099	12922962

Table 4. The dimension of some instances of the SCP problem (on the left) and CSP problem (on the right) and the time required by our PESTS to solve them.

C Experiments

We tested our tool on the case studies presented in Example 2 and we used our tool PESTS to solve the Submodule Construction Problem (SCP) and the Controller Synthesis Problem (CSP).

In Table 4, we report the results of these experiments, carried on an Intel Core i7-6700HQ, CPU 2.60GHz, with 16 GB RAM. The first column represents the buffer size n (and the UAV capacity is $n-1$); the second and third column represent the number of states and transitions of the automata manipulated by the algorithm (recall from the example in sub-section 4.2 that the automata of CSP have the same number of states and transitions); and the last column represents the time necessary to compute the solution.

Although PESTS is at the early stages of development the results we obtained are quite promising. We believe that a suitable engineering of the tool can improve its performance and we plan to carry out a comparison with other existing tools as part of the future work.