Journal of Automated Reasoning (2020) 64:1445–1481
https://doi.org/10.1007/s10817-020-09568-7

Natural Projection as Partial Model Checking

Gabriele Costa[1] · Letterio Galletta[1] · Pierpaolo Degano[2] · David Basin[3] · Chiara Bodei[2]

Received: 22 June 2020 / Accepted: 26 June 2020 / Published online: 13 August 2020

Abstract

Verifying the correctness of a system as a whole requires establishing that it satisfies a global specification. When it does not, it would be helpful to determine which modules are incorrect. As a consequence, specification decomposition is a relevant problem from both a theoretical and practical point of view. Until now, specification decomposition has been independently addressed by the control theory and verification communities through *natural projection* and *partial model checking*, respectively. We prove that natural projection reduces to partial model checking and, when cast in a common setting, the two are equivalent. Apart from their foundational interest, our results build a bridge whereby the control theory community can reuse algorithms and results developed by the verification community. Furthermore, we extend the notions of natural projection and partial model checking from finite-state to symbolic transition systems and we show that the equivalence still holds. Symbolic transition systems are more expressive than traditional finite-state transition systems, as they can model large systems, whose behavior depends on the data handled, and not only on the control flow. Finally, we present an algorithm for the partial model checking of both kinds of systems that can be used as an alternative to natural projection.

Keywords Natural projection · Partial evaluation · Formal verification · Model checking

✉ Gabriele Costa
gabriele.costa@imtlucca.it

Letterio Galletta
letterio.galletta@imtlucca.it

Pierpaolo Degano
degano@di.unipi.it

David Basin
basin@inf.ethz.ch

Chiara Bodei
chiara@di.unipi.it

[1] SysMA Unit, IMT School for Advanced Studies, Lucca, Italy

[2] Department of Informatics, Università di Pisa, Pisa, Italy

[3] Department of Computer Science, ETH Zurich, Zurich, Switzerland

1 Introduction

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

Ideally one aims at freely passing from local to global specifications and vice versa. Most specification formalisms natively support specification composition and there are well-studied examples of operators for composing them, e.g., logical conjunction, set intersection, and the synchronous product of automata. Unfortunately, the same does not hold for specification decomposition: obtaining local specifications from a global one is, in general, much more difficult.

Over the past decades, many research communities have independently investigated decomposition methods, each focussing on the specification formalisms and assumptions appropriate for their application context. In particular, important results were obtained in the fields of *control theory* and *formal verification*.

In control theory, *natural projection* [40] is exploited to simplify systems built from multiple components, modeled as automata. Natural projection is often applied component-wise to solve the *controller synthesis problem*, i.e., for synthesizing local controllers from a global specification of an asynchronous *discrete-event system* [11]. In this way, by interacting only with a single component of a system, local controllers guarantee that the global specification is never violated. By composing local controllers in parallel with other sub-systems, it is possible to implement distributed control systems [41,42].

The formal verification community proposed *partial model checking* [1] as a technique to mitigate the state explosion problem arising when verifying large systems composed from many parallel processes. Partial model checking tackles this problem by decomposing a specification, given as a formula of the μ-calculus [27], using a *quotienting* operator, and 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 for instance, a system built from two modules satisfies a specification if and only if one of the modules satisfies the specification after quotienting against the other [1]. The use of quotienting may reduce the problem size, resulting in smaller models and hence faster verification.

Table 1 summarizes some relevant results about the two approaches for finite-state Labeled Transitions Systems; for more details, we refer the reader to Sect. 6. Since natural projection and partial model checking apply to different formalisms, they cannot be directly compared

Table 1 Summary of existing results on natural projection and partial model checking for finite-state Labeled Transition Systems

	Natural projection	Partial MC
Spec. Lang.	FSA [24,37]	μ-calculus [1,3]
Theory	FSA [24,37]	LTS [1,3]
Complexity	EXPTIME[1] [19,39]	EXPTIME [1,3]
Tools	TCT [18], IDES3 [35], DESTool [33]	mCRL2 [23], CADP [28], MuDiv [2]

Notice that the algorithm in [39] runs in PTIME on a specific class of discrete-event systems without defining a common framework. For example, a relevant question is to compare how specifications grow under the two approaches. Although it is known that both may lead to exponential growth (see [26,39] and [3]), these results apply in one case to finite-state automata (FSAs) and in the other case to μ-calculus formulae.

Although decomposition work has been carried out in different communities, there have also been proposals for the cross-fertilization of ideas and methods [17]. For instance, methods for synthesizing controllers using partial model checking are given in [7,31]. The authors of [20] and [22] propose similar techniques, using fragments of the μ-calculus and CTL*, respectively.

One of our starting points was suggested by Ehlers et al. [17], who advocate establishing formal connections between these two approaches. 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.

We address the first remark about a formal bridge by showing 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 start by defining 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 3.2).

We also provide additional results that contribute to the detailed comparison, referred to in the second remark. In particular, we propose a new algorithm for partial model checking that operates directly on Labeled Transition Systems (LTS), rather than on the μ-calculus. We prove that our algorithm is correct with respect to the traditional quotienting rules and we show that it runs in polynomial time, like the algorithms based on natural projection.

A preliminary version of the above results have been previously presented in [13], and are systematized and formally proved here.[1] In this paper we additionally lift these results to *symbolic Labeled Transition Systems* (s-LTS), a slight generalization of symbolic FSAs [15], which themselves substantially generalize traditional FSAs. Roughly speaking, the transitions of an s-LTS carry predicates rather than letters, as LTS do, and can thus handle rich, non-finite alphabets. In particular, the alphabet of an s-LTS is the carrier of an effective boolean algebra, thereby maintaining the operational flavor of transition systems. In the next section, we give an example of a concurrent program running on a GPU that shows the added expressive power of specifications rendered by s-LTSs.

Our lifting of results proceeds in several steps. First we define the notion of symbolic traces composed by transitions with predicates as labels, and we show their relationship to the more standard traces labeled by the elements of a given finite alphabet. More significantly we define symbolic synchronous composition of s-LTSs, which is crucial for composing these richer system specifications. We then introduce novel symbolic versions of partial model checking and of natural projection. Also, for the symbolic case, we prove a theorem (Theorem 5.2) that

[1] In particular, Sects. 3 and 4 previously appeared in [13], while Sects. 2 and 5, and the entire “Technical Appendix” are new.

extends the statement of Theorem 3.2 to the s-LTSs, i.e., that establishes the correspondence between partial model checking and natural projection for s-LTSs. Finally, we define a new algorithm for symbolic partial model checking directly on s-LTSs, and we prove it correct with respect to the symbolic quotienting operator. As expected, our algorithm's time complexity is exponential. This is due to the need to check the satisfiability of the predicates labeling the symbolic transitions.

We have implemented our algorithm for partial model checking on Labeled Transition Systems in the tool available online [14]. Along with the tool, we developed several case studies illustrating its application to the synthesis of both submodules and local controllers. The implementation of the algorithm for s-LTS is still under development.

Structure of the paper We start by presenting a motivating example in Sect. 2. Section 3 presents our unified theoretical framework for natural projection and partial model checking as well as its formal properties. In Sect. 4 we present the quotienting algorithm, discuss its properties, and apply it to our running example. We extend our framework to the symbolic transition systems in Sect. 5. Section 5.4 presents our novel symbolic quotienting algorithm. In Sect. 6 we briefly survey the related literature and in Sect. 7 we draw conclusions. The "Technical Appendix" contains all the formal proofs together with the correctness and the complexity of our algorithms. Finally, all the additional material about (i) implementation of the algorithms, (ii) tool usage and (iii) replication of the experiments is available at https://github.com/gabriele-costa/pests.

2 A Running Example: A GPU Kernel

In this section we introduce a simple yet realistic example that we use as running throughout the paper. The example illustrates an instance of a system made of two concurrent components, and its global specification consisting of two properties intuitively presented below. We will show how the decomposition of the global specification is done by partially evaluating it against one of the components. Then, we model check the obtained local specification against the other component, so verifying the original global specification. The first of the two properties is expressed through an LTS and discussed in Sect. 4. For the second we take advantage of the richer expressive power of s-LTS to reason about both data and control. In Sect. 5 we show how this enables a fine-grained analysis of the system behavior.

We consider a concurrent program (called *kernel*) running on a Graphical Processing Unit (GPU). The program implements a producer-consumer schema relying on a circular queue. The program is written in OpenCL,[2] a C-like language for programming GPUs. A sequential application P embodies an OpenCL kernel and uses it to accelerate some computations. In practice, P compiles the kernel at run time, loads it on the GPU memory, and launches its execution, which is carried on by a group of threads running concurrently on the different GPU cores. During the execution, each thread is bound to an identifier, called *local id*, and threads share a portion of the GPU memory, called *local memory*. A group of threads can synchronize through a *barrier*. Intuitively, a barrier is an operator that blocks the execution of each thread at a particular point. When all the threads reach the same barrier, their execution is resumed.

Consider the OpenCL kernel of Fig. 1 that implements a simple producer-consumer schema. Briefly, one instance of the kernel function `manager` is executed on each core of a GPU. Here, for simplicity, we assume that only two cores exist. A `manager` kernel

[2] https://www.khronos.org/opencl/.

```
 1  constant int SIZE = 8;
 2
 3  int consume(local int *L, local int *buffer) {
 4    local char *head = (local char *)L;
 5    int val = buffer[*head];      // dequeue value
 6    *head++;                      // increment head pointer
 7    if(*head == SIZE) {           // buffer end reached
 8      *head = 0;
 9    }
10    return val;
11  }
12
13  void produce(local int *L, local int *buffer, int val) {
14    local char *tail = ((local char *)L)+1;
15    buffer[*tail] = val;          // enqueue value
16    *tail++;                      // increment tail pointer
17    if(*tail == SIZE) {           // buffer end reached
18      *tail = 0;
19    }
20  }
21
22  kernel void manager(local int *L, local int *buffer, local int *N) {
23    int val = 0;
24    int sending = *N, receiving = *N;
25    while(sending > 0 || receiving > 0) {
26      barrier();                              // synchronization
27      if(get_local_id(0) && receiving > 0) { // consumer thread
28        val = consume(L, buffer);
29        receiving--;
30      }
31      if(!get_local_id(0) && sending > 0) {  // producer thread
32        produce(L, buffer, val+1);
33        sending--;
34      }
35    }
36  }
```

Fig. 1 A fragment of OpenCL

iteratively invokes one of two functions, `produce` and `consume`, depending on the thread identifier (either 0 or 1) returned by `get_local_id(0)`. Hence, the `manager` kernel forces each thread to assume one of the two roles, either a producer or a consumer. The two functions use the local memory to share a vector, called `buffer`, which implements a circular queue. The queue has eight slots: a new item (i.e., a four-byte integer) is inserted (by the producer) in position `L[1]` and removed (by the consumer) from position `L[0]`. In practice, the first two bytes of `L` contain the *head* and *tail* pointers of the circular queue. Thus, they are incremented after each enqueue/dequeue operation and set to `0` when they exceed the buffer limit. The two threads iterate until both the producer and the consumer processed exactly `*N` items.

The code of Fig. 1 suffers from several typical flaws. The first flaw concerns the buffer's consistency. Provided that the buffer's size is at least 8, the two threads cannot cause a buffer overflow. Nevertheless, there is no guarantee that enqueue (line 15) and dequeue (line 5) always occur in the right order. In fact, since the two threads run in parallel with no priority constraints, two unsafe configurations may be reached: (i) the consumer attempts to extract an element from the empty buffer and (ii) the producer attempts to insert an element into a full buffer.

The second potential flaw is a data race. Data races occur when two threads simultaneously access the same, shared memory location and at least one of them modifies the data. When both the threads access the same memory in write mode, it is called a *write-write* data race. Otherwise we have a *write-read* data race. The two threads of Fig. 1 handle three pointers to the shared memory space, i.e., `L`, `buffer`, and `N` (line 22). These variables are identified by the `local` modifier. No data races can occur on `N` as it is never modified. A write-read data race on `buffer` happens when the producer and the consumer access the same location. Notice that this happens under conditions similar to those discussed for the buffer consistency, e.g., enqueue and dequeue are not executed in the right order. The case of `L` is more subtle. Both `produce()` and `consume()` modify the four bytes of the variable `L` (of type `int`). However, the two functions operate on different bytes, i.e., `L[0]` and `L[1]`. The single byte granularity is achieved through a cast to type `char *` (lines 4 and 14). Hence, no data race actually affects `L`.

Verifying the correctness of GPU kernels, in general, and producer-consumer schemas, in particular, are active research fields. Static analysis techniques such as [9] and [36] aim at validating a kernel against some specific property, such as absence of data races. The tools based on these techniques support developers by identifying potentially dangerous code. Still, the developer must manually confirm these alerts since the static analysis commonly considers an over-approximation of the program's actual behavior. For instance, GPUVerify [9], a prominent static verification tool, reports a possible write-read data race on `L` when applied to the kernel of Fig. 1 (see the "Technical Appendix"). As we will see in Sect. 5, we avoid this false positive through our symbolic algorithm.

Systems are usually composed of several modules, in our example the consumer and the producer. Verifying that the system as a whole complies with a specification requires checking that it satisfies a *global* specification. If the check fails, often there is no indication of which module is not compliant, and thus one must rethink the entire implementation. Instead, through decomposition, one can specialize the specification to operate on the single modules, thereby possibly enhancing the verification of the whole system. In addition, given a global specification and a system missing some components, one can just synthesize the specifications for the missing parts. For instance, as we will show in Example 3, the program in Fig. 1 suffers from a buffer inconsistency flaw. Given a model of the producer, in Sect. 4.2 we decompose a buffer consistency specification into a partial one that the consumer must obey to avoid this misbehavior.

3 A General Framework

In this section we cast both natural projection and partial model checking in the common framework of Labeled Transition Systems.

3.1 Language Semantics Versus State Semantics

Natural projection is commonly defined over (sets of) *words* [40]. Words are finite sequences of actions, i.e., symbols labeling the transitions between the 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 distinguished state, like a final or marking state. We let $\mathcal{L}$ denote the function that maps each FSA to the corresponding *language semantics*. Given a system *Sys* and a specification *Spec*, both FSAs, then *Sys* is said to satisfy *Spec* whenever $\mathcal{L}(Sys) \subseteq \mathcal{L}(Spec)$.

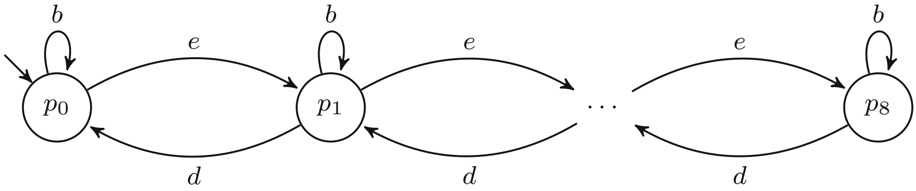


Fig. 2 The specification P of the consistency of a buffer with 8 positions, namely $P(8)$

Rather than an FSA, here we use a labeled transition system (LTS) to specify a system Sys. An LTS is similar to an FSA, but with a weaker notion of acceptance, where all states are final. We specify our running example below as an LTS.

Example 1 (Running example) Consider again the OpenCL program from Sect. 2 where the buffer positions are fixed to 8. Figure 2 depicts a transition system that encodes the specification P for the buffer's consistency, where the symbols e and d represent the (generic) enqueue and dequeue operations, respectively. Intuitively, the threads cannot perform e actions when the buffer is full (state p_8) and d actions when the buffer is empty (state p_0). Barrier synchronizations do not affect the specification's state. We indicate these actions with self-loops labeled with b. Only the three operations mentioned above are relevant for the specification P. Thus, we do not introduce further action labels. □

For partial model checking, the specification $Spec$ is defined by a formula of the μ-calculus. The standard interpretation of the formulas is given by a *state semantics*, i.e., a function that, given an LTS (for a system) Sys and a formula Φ, returns the set of states of Sys that satisfy Φ. A set of evaluation rules formalizes whether a state satisfies a formula or not. Given an LTS Sys and a μ-calculus formula Φ, we say that Sys satisfies Φ whenever its initial state does.

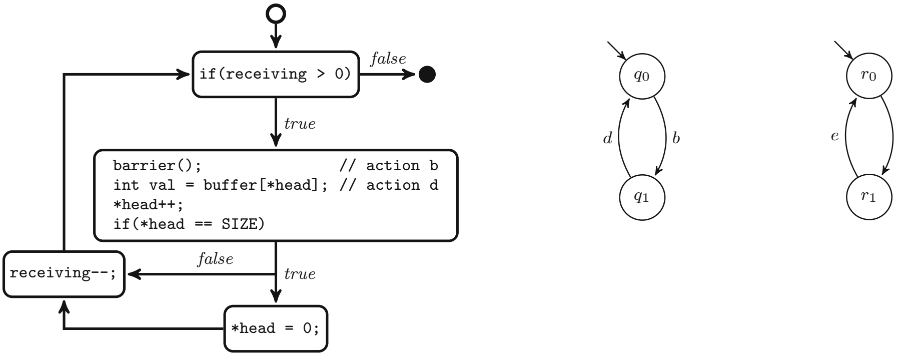
The language semantics of temporal logics is strictly less expressive than the state-based one [21]. A similar fact holds for FSAs and regular expressions [6]. Below we use a semantics from which both the state-based and the language semantics can be obtained.

3.2 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. In particular, we restrict ourselves here to deterministic transition systems.

Definition 3.1 A (deterministic) 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 $t = s \overset{a}{\rightarrow} s'$ to denote a *transition*, whenever $\rightarrow_A (a, s) = s'$, and we call s the *source* state, a the *action* label, and s' the *destination* state.

A *trace* $\sigma \in \mathcal{T}$ of an LTS A is either a single state s or a finite 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 simply as the sequence $\sigma = s_0 \boldsymbol{a_1} s_1 \boldsymbol{a_2} s_2 \ldots \boldsymbol{a_n} s_n$,



```
if(receiving > 0)
```

```
barrier();                  // action b
int val = buffer[*head];   // action d
*head++;
if(*head == SIZE)
```

```
receiving--;
```

```
*head = 0;
```

Fig. 3 From left to right: CFG of the consumer, and LTSs for the consumer (A) and producer (B)

alternating elements of S_A and Σ_A (written 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 $[\![A, \iota_A]\!]$, i.e., for those traces starting from the initial state ι_A. □

Example 2 Consider again our running example. Figure 3 depicts the LTSs A and B that model the behavior of the consumer and producer, respectively. On the left-hand side we show the control flow graph (CFG) of the consumer thread where we use a light grey font for the irrelevant instructions. Intuitively, the CFG consists of a loop iterating the execution of the central block. For this reason, the LTS A alternates actions b (for *barrier*) and d (for *dequeue*). The CFG of the producer is similar: the only difference is that it increments the *tail* pointer, rather than the *head* pointer. Hence, B is symmetric: it performs e (for *enqueue*) in place of d. The traces starting from the initial states of A and B are, respectively,

$$[\![A]\!] = \{q_0, q_0\boldsymbol{b}q_1, q_0\boldsymbol{b}q_1\boldsymbol{d}q_0, q_0\boldsymbol{b}q_1\boldsymbol{d}q_0\boldsymbol{b}q_1, \ldots\}$$
$$[\![B]\!] = \{r_0, r_0\boldsymbol{b}r_1, r_0\boldsymbol{b}r_1\boldsymbol{e}r_0, r_0\boldsymbol{b}r_1\boldsymbol{e}r_0\boldsymbol{b}r_1, \ldots\}$$

□

Typically, a system, or *plant* in control theory, consists of multiple interacting components running in parallel. Intuitively, when two LTSs are put in parallel, each proceeds asynchronously, except on those actions they share, upon which they synchronize. We render this behavior by means of the synchronous product [4]. In particular, we rephrase the definition given in [40].

Definition 3.2 Given two LTSs A and B such that $\Sigma_A \cap \Sigma_B = \Gamma$, the *synchronous product* of A and B is $A \parallel B = (S_A \times S_B, \Sigma_A \cup \Sigma_B, \rightarrow_{A\parallel B}, \langle \iota_A, \iota_B \rangle)$, where $\rightarrow_{A\parallel B}$ is as follows:

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

□

Example 3 Consider again the LTSs A and B from Fig. 3. Their synchronous product $A \parallel B$ (with $\Gamma = \{b\}$) is depicted in Fig. 4. We use bold edges to denote synchronous transitions. Intuitively, $A \parallel B$ does not satisfy $P(n)$, for any $n > 0$. In fact $\langle q_0, r_0 \rangle \xrightarrow{b} \langle q_1, r_1 \rangle \xrightarrow{d} \langle q_0, r_1 \rangle$ but $bd \notin \mathcal{L}(P(n))$. □

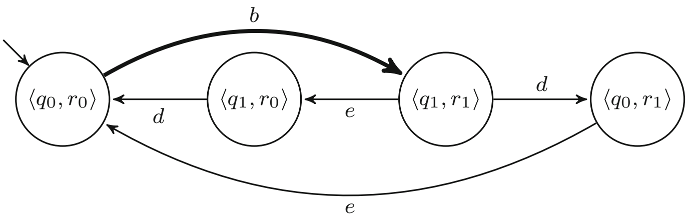

b
$\langle q_0, r_0 \rangle$
$\langle q_1, r_0 \rangle$
$\langle q_1, r_1 \rangle$
$\langle q_0, r_1 \rangle$
d
e
d
e

Fig. 4 Synchronous product $A \parallel B$, where bold transitions denote synchronous moves

Next, we generalize the notion of natural projection on languages. Intuitively, natural projection can be seen as the inverse operation with respect to the synchronous product of two LTSs. Indeed, through natural projection one recovers the LTS of one of the components of the parallel composition.

Given a computation of $A \parallel B$, natural projection extracts the relevant trace of one of the two LTSs, including the synchronized transitions (see the second case below). Note that, unlike other definitions, e.g., in [40], our traces are sequences of transitions including both states and actions. We also define the inverse projection in the expected way.

Definition 3.3 Given LTSs A and B with $\Gamma = \Sigma_A \cap \Sigma_B$, the *natural projection on A* of a trace σ of $A \parallel B$, in symbols $P_A(\sigma)$, is defined as follows:

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

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

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 $P_A^{-1}(\mathcal{T}) = \bigcup_{\sigma \in \mathcal{T}} P_A^{-1}(\sigma)$. □

Example 4 Consider the following two traces $\sigma_1 = \langle q_0, r_0 \rangle \boldsymbol{b} \langle q_1, r_1 \rangle \boldsymbol{d} \langle q_0, r_1 \rangle \boldsymbol{e} \langle q_0, r_0 \rangle$ and $\sigma_2 = \langle q_0, r_0 \rangle \boldsymbol{b} \langle q_1, r_1 \rangle \boldsymbol{e} \langle q_1, r_0 \rangle \boldsymbol{d} \langle q_0, r_0 \rangle$. We have that the projections $P_A(\sigma_1) = P_A(\sigma_2) = q_0 \boldsymbol{b} q_1 \boldsymbol{d} q_0 \in [\![A]\!]$ and $\sigma_1, \sigma_2 \in P_B^{-1}(q_0 \boldsymbol{b} q_1 \boldsymbol{d} q_0)$. □

Two classical properties [40] concerning the interplay between the synchronous product and the natural projection hold. Their proofs are trivial.

Fact 3.1 $P_A([\![A \parallel B]\!]) \subseteq [\![A]\!]$ *and* $[\![A \parallel B]\!] = P_B^{-1}([\![A]\!]) \cap P_A^{-1}([\![B]\!])$.

3.3 Equational μ-Calculus and Partial Model Checking

Below, we recall the variant of the 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 of 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 possibility $\langle \cdot \rangle$. Equations also have fix-point operators (minimum μ and maximum ν) over variables x, and can be organized in equation systems.

Definition 3.4 (*Syntax of the μ-calculus*) Given a set of variables $x \in X$ and an alphabet of actions $a \in \Sigma$, *assertions* $\phi, \phi' \in \mathcal{A}$ are given by the syntax:

$$\phi ::= \mathit{ff} \mid \mathit{tt} \mid x \mid \phi \wedge \phi' \mid \phi \vee \phi' \mid [a]\phi \mid \langle a \rangle \phi.$$

An *equation* is of the form $x =_\pi \phi$, 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 a single equation. Thus Φ is given by

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

A *top assertion* $\Phi \downarrow x$ amounts to 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 $\|\phi\|_\rho$ 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 $\|\langle a \rangle \phi\|_\rho$ (respectively, the necessity modality $\|[a]\phi\|_\rho$) 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 [10,27].

Definition 3.5 (*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.

$$\begin{array}{lll}
\|\mathit{ff}\|_\rho = \emptyset & \|\mathit{tt}\|_\rho = S_A & \|x\|_\rho = \rho(x) \\
\|\phi \wedge \phi'\|_\rho \ = \ \|\phi\|_\rho \cap \|\phi'\|_\rho & \multicolumn{2}{r}{\|[a]\phi\|_\rho \ = \ \{s \in S_A \mid \forall s'.s \xrightarrow{a}_A s' \ \Rightarrow \ s' \in \|\phi\|_\rho\}} \\
\|\phi \vee \phi'\|_\rho \ = \ \|\phi\|_\rho \cup \|\phi'\|_\rho & \multicolumn{2}{r}{\|\langle a \rangle \phi\|_\rho \ = \ \{s \in S_A \mid \exists s'.s \xrightarrow{a}_A s' \ \wedge \ s' \in \|\phi\|_\rho\}}
\end{array}$$

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 its fixed point. We now define the semantics of equation systems by:

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

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

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

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

Example 5 Consider $\Phi \downarrow x$ where $\Phi = \left\{x =_\mu [e]y \wedge \langle d\rangle tt;\ y =_\nu \langle e\rangle x \vee \langle b\rangle x\right\}$.

This system consists of two equations. Intuitively, the first equation says that after every e transition a state satisfying the second equation for y is reached ($[e]y$) and that, from the current state, there must exist at least one d transition ($\langle d\rangle tt$) The second equation states that there must exist either a e transition or a b transition. In both cases, the reached state must satisfy the x equation.

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

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

Since $U^2 = U^1$, we have obtained the fixed point U^*. Finally, we can compute $\langle\!\langle \Phi \downarrow x\rangle\!\rangle$, which amounts to $[\![A \parallel B, \langle q_1, r_0\rangle]\!]$. □

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

Definition 3.7 An LTS A satisfies a top assertion $\Phi \downarrow x$, in symbols $A \models_s \Phi \downarrow x$, if and only if $\iota_A \in \|\Phi \downarrow x\|$. 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 the state semantics ($\models_s$) with the one based on the trace semantics ($\models_\sigma$); its proof is immediate by Definition 3.6.

Fact 3.2 *$A \models_s \Phi \downarrow x$ 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 $B \models_s \Phi \downarrow x /\!\!/_B A$. 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 $x_{s_i} =_\pi \phi /\!\!/_{\Sigma_B} s_i$ 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 A's transitions, $\Phi \downarrow x /\!\!/_B A$ embeds the behavior of A.

Definition 3.8 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.

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

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

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

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

$$\begin{array}{lll}
(\langle a\rangle\phi) /\!\!/_{\Sigma_B} s = \bigvee\limits_{s \xrightarrow{a} s'} \phi /\!\!/_{\Sigma_B} s' & ([a]\phi) /\!\!/_{\Sigma_B} s = \bigwedge\limits_{s \xrightarrow{a} s'} \phi /\!\!/_{\Sigma_B} s' & \textit{if } a \in \Sigma_A \setminus \Gamma \\
(\langle b\rangle\phi) /\!\!/_{\Sigma_B} s = \langle b\rangle(\phi /\!\!/_{\Sigma_B} s) & ([b]\phi) /\!\!/_{\Sigma_B} s = [b](\phi /\!\!/_{\Sigma_B} s) & \textit{if } b \in \Sigma_B \setminus \Gamma \\
(\langle \gamma\rangle\phi) /\!\!/_{\Sigma_B} s = \bigvee\limits_{s \xrightarrow{\gamma} s'} \langle\gamma\rangle(\phi /\!\!/_{\Sigma_B} s') & ([\gamma]\phi) /\!\!/_{\Sigma_B} s = \bigwedge\limits_{s \xrightarrow{\gamma} s'} [\gamma](\phi /\!\!/_{\Sigma_B} s') & \textit{if } \gamma \in \Gamma.
\end{array}$$

□

Example 6 Consider the top assertion $\Phi \downarrow x$ of Example 5 and the LTSs A and B of Example 2. Quotienting $\Phi \downarrow x$ against A, we obtain $\Phi /\!\!/_{\Sigma_B} A \downarrow x_{q_0}$, where

$$\Phi /\!\!/_{\Sigma_A} B = \begin{cases} x_{q_0} =_\mu [e]y_{q_0} \wedge ff \\ x_{q_1} =_\mu [e]y_{q_1} \wedge tt \\ y_{q_0} =_\nu \langle e\rangle x_{q_0} \vee ff \\ y_{q_1} =_\nu \langle e\rangle x_{q_1} \vee \langle b\rangle x_{q_0} \end{cases} = \begin{cases} x_{q_0} =_\mu ff \\ x_{q_1} =_\mu [e]y_{q_1} \\ y_{q_0} =_\nu \langle e\rangle x_{q_0} \\ y_{q_1} =_\nu \langle e\rangle x_{q_1} \vee \langle b\rangle x_{q_0} \end{cases} = \left\{ x_{q_0} =_\mu ff \right\}.$$

The leftmost equations are obtained by applying the rules of Definition 3.8. Then we simplify on the right-hand sides of the first three equations, i.e., those of x_{q_0}, x_{q_1} and y_{q_0}. In particular, we apply the standard boolean transformations $\psi \wedge ff \equiv ff$, $\psi \wedge tt \equiv \psi$, and $\psi \vee ff \equiv \psi$. Finally we reduce the number of equations by removing those unreachable from the top variable x_{q_0}. For a detailed description of our simplification strategies, see [3]. Therefore $\langle\!\langle \Phi \downarrow x /\!\!/_{\Sigma_B} A \rangle\!\rangle = \emptyset$. This was expected since, as shown in Example 5, $\langle q_0, r_0\rangle \notin \| \Phi \downarrow x \|$.

□

3.4 Unifying the Logical and the Operational Approaches

Here we prove the equivalence between natural projection and partial model checking (Theorem 3.2), establishing the correspondence between quotienting and natural projection.

Theorem 3.1 *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 3.2 *For all A, B, x and Φ on $A \parallel B$,*

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

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 running example to address three problems: verification, submodule construction, and controller synthesis.

Table 2 The quotienting algorithm

```
Begin proc quotient
   input P = ($S_P$, $\Sigma_P$, $\rightarrow_P$, $i_P$)
   input A = ($S_A$, $\Sigma_A$, $\rightarrow_A$, $i_A$)
   input $\Sigma_B$

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

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 starting from two LTSs, i.e., a specification P and an agent A. Moreover, it takes as an argument the alphabet of actions Σ_B of the new transition system B. 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 using the quotienting rules from Sect. 3. Also, note that the relation $\rightarrow$ is restricted to the states of S (denoted $\rightarrow_S$).

The second procedure, called `unify` (in Table 3) translates a non-deterministic transition system back to an LTS. By using closures over λ, `unify` groups transition system states. This process is similar to the standard subset construction [24], 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 (`∧-move`). The 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 `∧-move` (line 7). If the partition is nonempty, 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. More precisely, assuming that Γ, $\Sigma_A \setminus \Gamma$, and $\Sigma_B \setminus \Gamma$ have m elements, and that P and A have n states, the complexity is $O(n^6 m^2)$ (see Appendix A.4 for more details). We avoid an exponential blow-up in our algorithm (in contrast to Table 1) since we only consider deterministic transition systems. Note that a determinization step for non-deterministic transition systems is exponential in the worst case.

Table 3 The unification algorithm

```
Begin proc unify
  input B = $(S_B,\ \Sigma_B,\ \rightarrow_B,\ i_B)$

  $I$ := $\lambda$−close($\{i_B\}$)
  $R$, $S$ := $\{I\}$
  $\rightarrow$ := $\emptyset$
  while $S \neq \emptyset$ do
    $Q$ := pick&remove($S$)
    for each $a \in \Sigma_B \setminus \{\lambda\}$
      $M$ := $\wedge$−move($Q$, $a$)
      if $M \neq \emptyset$ then
        $\rightarrow$ := $\rightarrow \cup \{(Q, a, M)\}$
        if $M \notin R$ then $S$ := $S \cup \{M\}$; $R$ := $R \cup \{M\}$ end if
      end if
    end for
  end while
  output ($R$, $\Sigma_B \setminus \{\lambda\}$, $\rightarrow$, $I$)
end proc
```

```
Begin proc $\wedge$−move
  input $Q$
  input $a$

  $M$ := $\lambda$−close( $\overset{q\in Q}{\bigcap}\{q' | q \xrightarrow{a}_B q'\}$)
  output $M$
end proc
```

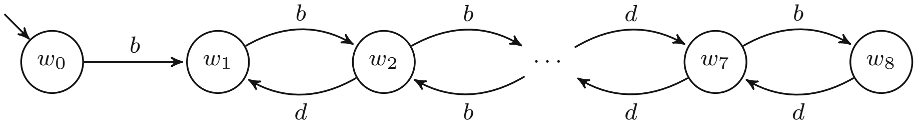


Fig. 5 Graphical representation of the consumer A'

4.2 Application to Our Running Example

Recall from Example 3 that $A \parallel B$ does not satisfy the buffer consistency property P. Informally the reason is that the barrier does not prevent the consumer A from accessing the buffer before the producer B. However, the barrier does ensure that iterations of the producer and the consumer are always paired. This implies that only the first position of the buffer is actually used.

We apply our quotienting algorithm to find an A' such that $A' \parallel B \models P$. That is, we solve an instance of the submodule construction problem for B and P. The resulting LTS is given in Fig. 5. Intuitively, A' behaves as follows. Initially, it synchronizes (action b) twice to ensure that B enqueues at least one item. Then, it either (i) synchronizes again and moves to the next state or (ii) dequeues an item (action d) and goes back one state. The reason is that each state w_i denotes a configuration under which the buffer contains i or $i-1$ items. As a result, there cannot be a state w_9 and also the state w_0 can be reached only once at the start. Finally, note that a similar construction also applies to the controller synthesis and verification problems. For the former it suffices to constrain the alphabet of A' to only contain synchronization actions, while for the latter we check that the submodule A' accepts the empty string.

5 Quotienting Symbolic Finite-State Systems

In this section, we extend our results to symbolic Finite-State Transition Systems (s-LTSs). This rather expressive formalism is a variant of symbolic Finite State Automata [16] where all states are final. The novelty with respect to a standard LTS (or to an FSA) is that the alphabet is the carrier of an effective boolean algebra and that transitions are enabled by predicates on the possibly infinitely many elements of the algebra. This model allows a convenient representation of large systems, the behavior of which also depends on the data handled, and not only on the control flow as it is the case with a standard LTS.

For example, consider again the OpenCL kernel of Fig. 1 and the kinds of flaws mentioned in Sect. 2. Buffer consistency has been addressed using the model of standard LTSs, because consistency only depends on the actions (enqueue and dequeue) performed. However, when representing data races we cannot abstract away from the affected memory location, the action performed (read/write), and the data involved. We model them using to s-LTSs. In Fig. 6, we show on the left the control-flow graph of our consumer. Since we are interested in the actions on `L` we highlight them. In the upper part on the right there is the s-LTS for the consumer. Accordingly, we show only the portion with read/write actions that are parametric with respect to the memory address `L` and its offset. In the bottom part, we display the s-LTS of the consumer.

In this example, one can encode our producer/consumer in a standard LTS, because the operations and data are finite. The price to pay is an exponential growth of the number of resulting labels and, consequently, of the transitions. Clearly, such encodings cannot be done, when data are taken from an infinite domain like the natural numbers or strings from a given alphabet. In these cases there always exists a standard LTS that accepts a language that however is *isomoprphic* to the given s-LTS (see [16]).

We start by recalling some known notions about s-LTSs, adapting them to our case as needed and illustrating them on our running example. Then, we present our contributions: a symbolic version of (i) the synchronous product operator; (ii) partial model checking and natural projection; and (iii) a quotienting algorithm.

5.1 Symbolic Labeled Transition Systems

We start by recalling the definition of an effective boolean algebra and algebraic operators over them that are the building blocks for symbolic LTSs.

Definition 5.1 [15] An *effective boolean algebra* (EBA) is a tuple $\mathcal{A} = \langle \mathfrak{D}, \Psi, \{\!|\cdot|\!\} \rangle$ where:

- $\mathfrak{D}$ is a non-empty, recursively enumerable set (called the alphabet or universe of $\mathcal{A}$);
- Ψ is a recursively enumerable set of predicates closed under the connectives $\wedge$, $\vee$, and $\neg$ such that $\bot, \top \in \Psi$; and
- $\{\!|\cdot|\!\} : \Phi \rightarrow 2^{\mathfrak{D}}$ is the denotation function such that $\{\!|\bot|\!\} = \emptyset$, $\{\!|\top|\!\} = \mathfrak{D}$, $\{\!|\varphi \wedge \psi|\!\} = \{\!|\varphi|\!\} \cap \{\!|\psi|\!\}$, $\{\!|\varphi \vee \psi|\!\} = \{\!|\varphi|\!\} \cup \{\!|\psi|\!\}$, and $\{\!|\neg\varphi|\!\} = \mathfrak{D} \setminus \{\!|\varphi|\!\}$ (for any $\varphi, \psi \in \Psi$). □

Given a predicate φ of an EBA $\mathcal{A}$, we say that φ is *satisfiable*, in symbols $\mathbf{sat}_{\mathcal{A}}(\varphi)$, when $\{\!|\varphi|\!\} \neq \emptyset$.

EBAs can be composed using several operators (see [15,38] for details). We recall those that are relevant for the definitions given below. Let $\mathcal{A}_1 = \langle \mathfrak{D}_1, \Psi_1, \{\!|\cdot|\!\}_1 \rangle$ and $\mathcal{A}_2 = \langle \mathfrak{D}_2, \Psi_2, \{\!|\cdot|\!\}_2 \rangle$ be EBAs.

(union) $\mathcal{A}_1 \oplus \mathcal{A}_2$ is the EBA $\langle \mathfrak{D}_\oplus, \Psi_\oplus, \{\!|\cdot|\!\}_\oplus \rangle$ such that

- $\mathfrak{D}_\oplus = (\mathfrak{D}_1 \times \{1\}) \cup (\mathfrak{D}_2 \times \{2\})$;
- $\Psi_\oplus = \Psi_1 \times \Psi_2$; and
- $\{\!|\langle \varphi_1, \varphi_2 \rangle|\!\}_\oplus = (\{\!|\varphi_1|\!\}_1 \times \{1\}) \cup (\{\!|\varphi_2|\!\}_2 \times \{2\})$.

(product) $\mathcal{A}_1 \otimes \mathcal{A}_2$ is the EBA $\langle \mathfrak{D}_\otimes, \Psi_\otimes, \{\!|\cdot|\!\}_\otimes \rangle$ such that

- $\mathfrak{D}_\otimes = \mathfrak{D}_1 \times \mathfrak{D}_2$;
- $\Psi_\otimes = \Psi_1 \times \Psi_2$; and
- $\{\!|\langle \varphi_1, \varphi_2 \rangle|\!\}_\otimes = \{\!|\varphi_1|\!\}_1 \times \{\!|\varphi_2|\!\}_2$.

(restriction) $\mathcal{A}_1 \upharpoonright V$ (with $V \in 2^{\mathfrak{D}_1}$) is the EBA $\langle \mathfrak{D}, \Psi, \{\!|\cdot|\!\} \rangle$ such that

- $\mathfrak{D} = \mathfrak{D}_1 \cap V$;
- $\Psi = \Psi_1$; and
- $\{\!|\varphi|\!\} = \{\!|\varphi|\!\}_1 \cap V$.

For brevity, we may write $\mathcal{A} \upharpoonright \varphi$ for $\mathcal{A} \upharpoonright \{\!|\varphi|\!\}$.

Example 7 The EBA $\mathcal{B}$ encoding the write/read actions of our running example is defined as follows.

- $\mathfrak{D} = \{r, w\} \times \mathit{Id} \times \mathbb{N}$, where *Id* stands for the set of variable identifiers of a program.

– Ψ includes equality and inequality (on both $\{r, w\}$ and Id) and ordering relationships between natural numbers.

We use the variables α and β to range over $\{r, w\}$ and X and Y for generic elements of Id, the bytes of which are identified by their position (variable n). Also, we write $\alpha(X, n) : \varphi$ to denote the predicates of Ψ and we use straightforward abbreviations such as $w(L, 0)$ for $\alpha(X, n) : \alpha = w \wedge X = L \wedge n = 0$. □

We now state the definition of an s-LTS, we introduce its symbolic traces and we show the mapping from the symbolic to the concrete traces. The definition of s-LTS is based on that of s-FA [16].

Definition 5.2 (*s-LTS*) A *symbolic LTS* (s-LTS) is a tuple $M = (Q, \mathcal{A}, \Delta, \imath)$, where Q is a finite set of states (with $\imath$ the initial state), $\mathcal{A} = \langle \mathfrak{D}, \Psi, \{\!|\cdot|\!\} \rangle$ is an EBA, and $\Delta \subseteq Q \times \Psi \times Q$ is the transition relation such that $(s, \varphi, s') \in \Delta$ only if $\mathbf{sat}_{\mathcal{A}}(\varphi)$.

An s-LTS is *deterministic* when for all (q, φ, q') and (q, φ', q''), $\varphi \wedge \varphi'$ is unsatisfiable. Given an s-LTS there always exists an equivalent, deterministic one. Thus, in the following we only consider deterministic s-LTSs.

Analogously to Definition 3.1, the traces of an s-LTS belong to $\mathcal{T}$ and have the form $\sigma = s_0 d_1 s_1 d_2 \ldots d_n s_n$, where for each $i \in [1, n]$ there exists $(s_{i-1}, \varphi, s_i) \in \Delta$ such that $d_i \in \{\!|\varphi|\!\}$. In contrast, a *symbolic trace* of the s-LTS M is a sequence $\eta = s_0 \varphi_1 s_1 \varphi_2 \ldots \varphi_n s_n$, where for each $i \in [1, n]$ there exists $(s_{i-1}, \varphi_i, s_i) \in \Delta$. We use $[\![M, s]\!]$ to denote the set of traces of M such that $s_0 = s$ and $tr(M, s)$ to denote the set of symbolic traces such that $s_0 = s$ (also we omit s when $s = \imath$).

Finally, a symbolic trace $\eta = s_0 \varphi_1 s_1 \varphi_2 \ldots \varphi_n s_n$ can be instantiated to the set of concrete traces $s2c(\eta) = \{s_0 d_1 s_1 d_2 \ldots d_n s_n \mid \forall i \in [1, n].\, d_i \in \{\!|\varphi_i|\!\}\}$. □

We next describe the symbolic model of our running example.

Example 8 Consider again the data race flaw for the OpenCL code discussed in Sect. 2. We use the EBA $\mathcal{B}$ of Example 7 to model the kernel accesses to the shared memory. The predicate $\alpha(X, n) : \varphi$ specifies the kernel accesses actions α (read or write) on the nth byte of variable X. Here φ is a constraint on the values that α, X, and n can assume. Figure 6 on the left shows the CFG of the consumer and an s-LTS modeling it, in the right, upper part. Below, we also show the s-LTS for the producer. Recall that the variable `head` points to `L[0]`, while `tail` (see Fig. 1) refers to `L[1]`. □

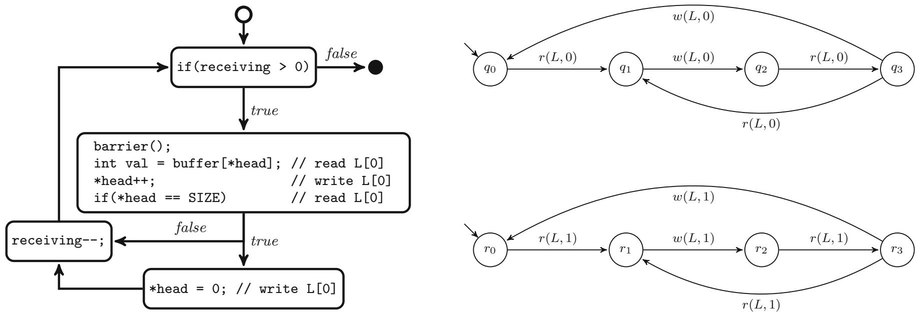
5.2 Parallel Composition of s-LTSs

Before proposing a new notion of parallel composition for s-LTSs, it is convenient to introduce an auxiliary operation on EBAs.

Definition 5.3 Given two EBAs, $\mathcal{A}_1 = \langle \mathfrak{D}_1, \Psi_1, \{\!|\cdot|\!\}_1 \rangle$ and $\mathcal{A}_2 = \langle \mathfrak{D}_2, \Psi_2, \{\!|\cdot|\!\}_2 \rangle$ and two predicates $\psi_1 \in \Psi_1$ and $\psi_2 \in \Psi_2$ (called *synchronization predicates*), we define the *parallel product* of $\mathcal{A}_1$ and $\mathcal{A}_2$ over ψ_1 and ψ_2 (in symbols $\mathcal{A}_1 \circledast_{\psi_1, \psi_2} \mathcal{A}_2$) as

$$\mathcal{A}_1 \circledast_{\psi_1, \psi_2} \mathcal{A}_2 = \mathcal{A}_1 \upharpoonright (\neg\psi_1) \oplus \mathcal{A}_2 \upharpoonright (\neg\psi_2) \oplus (\mathcal{A}_1 \upharpoonright \psi_1 \otimes \mathcal{A}_2 \upharpoonright \psi_2).$$

A predicate of $\mathcal{A}_1 \circledast_{\psi_1, \psi_2} \mathcal{A}_2$ has the form $\psi = ((\psi_{\mathcal{A}_1}, \psi_{\mathcal{A}_2}), (\psi'_{\mathcal{A}_1}, \psi'_{\mathcal{A}_2}))$, for some $\psi_{\mathcal{A}_1}, \psi'_{\mathcal{A}_1} \in \Psi_1$ and $\psi_{\mathcal{A}_2}, \psi'_{\mathcal{A}_2} \in \Psi_2$. We write $\psi_{|1}, \psi_{|2}, \psi_{|3}, \psi_{|4}$ to denote $\psi_{\mathcal{A}_1}, \psi_{\mathcal{A}_2}, \psi'_{\mathcal{A}_1}, \psi'_{\mathcal{A}_2}$, respectively. Similarly, the elements in the alphabet of $\mathcal{A}_1 \circledast_{\psi_1, \psi_2} \mathcal{A}_2$



```
if(receiving > 0)
barrier();
int val = buffer[*head]; // read L[0]
*head++;                 // write L[0]
if(*head == SIZE)        // read L[0]
receiving--;
*head = 0; // write L[0]
```

Fig. 6 From left to right: CFG of the consumer, and s-LTSs for the consumer (top) and for the producer (bottom)

have the form $((((d_1, 1), (d_2, 2)), 1), ((d_1', d_2'), 2))$, which we abbreviate to $((d_1, 1), (d_2, 2), ((d_1', d_2'), 3))$ or even, when clear from the context, to (d_1, d_2, d_1', d_2'). □

The definition of the parallel product of two s-LTSs follows. While this operation on LTSs requires a common sub-alphabet Γ, its symbolic counterpart synchronizes two s-LTSs on those actions that satisfy two distinguished, synchronization predicates. Intuitively, these predicates define the conditions under which a synchronous transition occurs. Note that we need two predicates as the involved s-LTSs can be defined on two different EBAs.

Definition 5.4 (*Parallel composition*) Given two s-LTS $M_1 = (Q_1, \mathcal{A}_1, \Delta_1, \imath_1,)$ and $M_2 = (Q_2, \mathcal{A}_2, \Delta_2, \imath_2)$ and two synchronization predicates $\psi_1 \in \Psi_1$ and $\psi_2 \in \Psi_2$, the *parallel composition* of M_1 and M_2 over ψ_1 and ψ_2 (in symbols $M_1 \parallel_{\psi_1,\psi_2} M_2$) is

$$M_1 \parallel_{\psi_1,\psi_2} M_2 = (Q_1 \times Q_2, \mathcal{A}_1 \circledast_{\psi_1,\psi_2} \mathcal{A}_2, \Delta^*, \langle \imath_1, \imath_2 \rangle),$$

where

$$\Delta^* = \bigcup_{\substack{(p_1,\varphi_1,p_1')\in\Delta_1 \\ (p_2,\varphi_2,p_2')\in\Delta_2}} \begin{cases} \{(\langle p_1, p_2\rangle, \langle \bot_1, \bot_2, \langle \varphi_1 \wedge \psi_1, \varphi_2 \wedge \psi_2 \rangle\rangle, \langle p_1', p_2'\rangle)\} \\ \{(\langle p_1, p_2\rangle, \langle \varphi_1 \wedge \neg\psi_1, \bot_2, \langle \bot_1, \bot_2 \rangle\rangle, \langle p_1', p_2\rangle)\} \\ \{(\langle p_1, p_2\rangle, \langle \bot_1, \varphi_2 \wedge \neg\psi_2, \langle \bot_1, \bot_2 \rangle\rangle, \langle p_1, p_2'\rangle)\} \end{cases}$$

and $\bot_1$ ($\bot_2$) is the false predicate of $\mathcal{A}_1$ ($\mathcal{A}_2$, respectively). □

We now apply this definition to our running example.

Example 9 The parallel composition of the two s-LTS of Fig. 6 over the synchronization predicates $\psi_1 = \alpha(X, n) : \alpha = w \wedge X = L$ and $\psi_2 = \alpha(X, n) : X = L$ is depicted in Fig. 7. For readability, we omit the transition labels and we instead discuss them here. By the definition of product, a transition's predicate can only belong to three groups: $\langle \varphi_1 \wedge \neg\psi_1, \bot, \bot\rangle$, $\langle \bot, \varphi_2 \wedge \neg\psi_2, \bot\rangle$, or $\langle \bot, \bot, \langle \varphi_1 \wedge \psi_1, \varphi_2 \wedge \psi_2 \rangle\rangle$, where φ_1 and φ_2 are predicates of the consumer and producer, respectively. Note that the predicates of the second type are not satisfiable since $\neg\psi_2$ requires that $X \neq L$ while all the φ_2 constrain $X = L$. Thus, the second group of transitions is empty. A similar observation applies to the predicates of the first group. Indeed, since $X = L$ the only assignment that satisfies $\neg\psi_1$ is for $\alpha = r$. Therefore, all these transitions are labeled with $\langle r(L, 0), \bot, \bot\rangle$. We use a thin arrow to denote them. As in Example 3 we use bold arrows to denote synchronous transitions. However, here we need to distinguish them according to their predicates. Analogous to the argument for the first group of transitions, here we have that the first component of a synchronization predicate

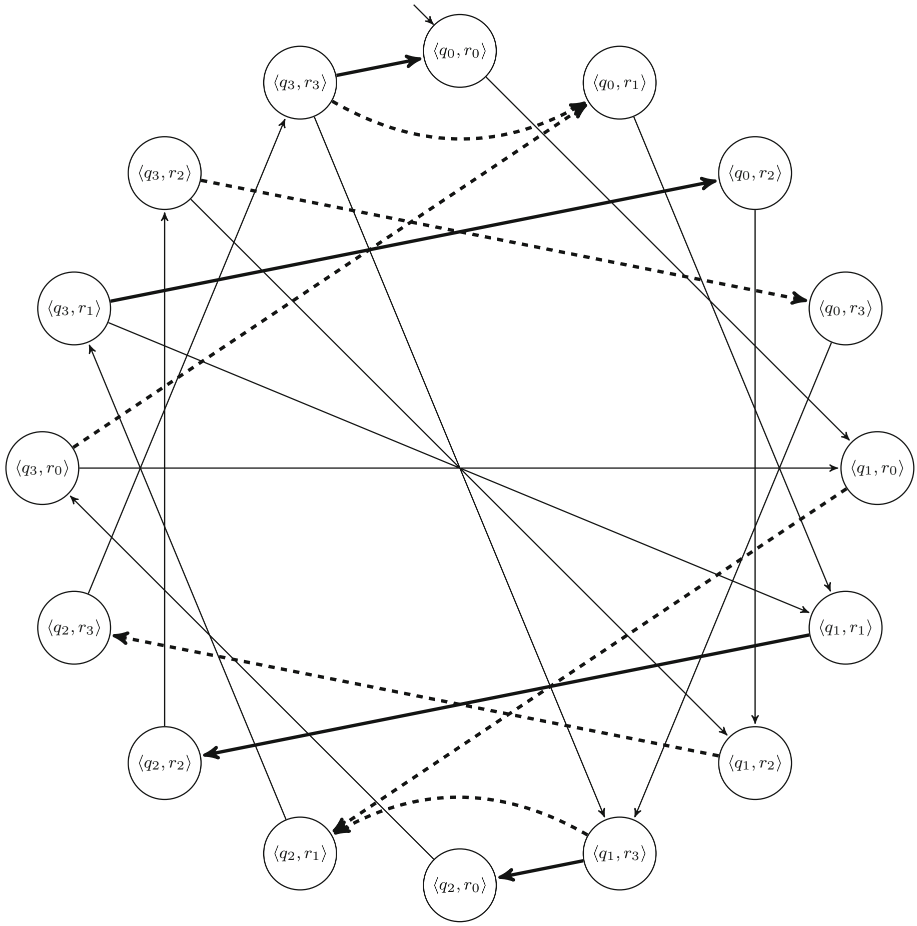

$\langle q_0, r_0\rangle$
$\langle q_0, r_1\rangle$
$\langle q_0, r_2\rangle$
$\langle q_0, r_3\rangle$
$\langle q_1, r_0\rangle$
$\langle q_1, r_1\rangle$
$\langle q_1, r_2\rangle$
$\langle q_1, r_3\rangle$
$\langle q_2, r_0\rangle$
$\langle q_2, r_1\rangle$
$\langle q_2, r_2\rangle$
$\langle q_2, r_3\rangle$
$\langle q_3, r_0\rangle$
$\langle q_3, r_1\rangle$
$\langle q_3, r_2\rangle$
$\langle q_3, r_3\rangle$

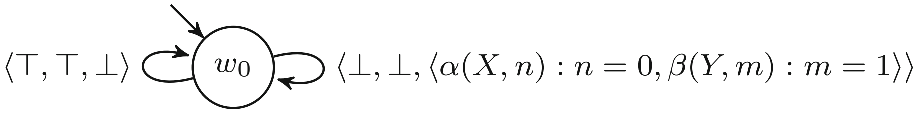
Fig. 7 The parallel composition of the producer and the consumer of Fig. 6

must be $w(L, 0)$. Thus, there are only two types of synchronous transitions depending on the second component of the synchronization predicate (either $w(L, 1)$ or $r(L, 1)$). We use dashed lines for the transitions labeled with predicate $\langle \bot, \bot, \langle w(L, 0), r(L, 1)\rangle\rangle$ and solid lines for $\langle \bot, \bot, \langle w(L, 0), w(L, 1)\rangle\rangle$.

The following small technical example illustrates a policy that ensures memory access segmentation and, thus, avoids data races.

Example 10 Consider the s-LTS W depicted in Fig. 8 that represents a policy specification to prevent data races. Briefly, W accepts any asynchronous operation carried out by each thread individually (left loop). Instead, synchronous operations are only permitted in one case (right loop), i.e., when different bytes are accessed by the two threads.

As a final remark, note that the product of Example 9 complies to this policy. Intuitively, the reason is that, for all the transition's predicates of the product, there exists at least one satisfiable predicate among the policy's transitions.



$\langle \top, \top, \bot \rangle$
w_0
$\langle \bot, \bot, \langle \alpha(X, n) : n = 0, \beta(Y, m) : m = 1 \rangle \rangle$

Fig. 8 The s-LTS W specifying the data race policy

5.3 Symbolic Natural Projection and Symbolic Quotienting

We now extend the results of Sect. 4 to the symbolic case. First we lift the natural projection to the traces of an s-LTS M. Afterwards, we define the quotient of M with respect to a pair of synchronization predicates, and give an algorithm for computing it. Finally, we state the relationships between the symbolic versions of natural projection and quotienting. In the following, we overload some names and symbols.

Definition 5.5 (*Natural projection*) Given two s-LTS $M_1 = (Q_1, \mathcal{A}_1, \Delta_1, \imath_1)$ and $M_2 = (Q_2, \mathcal{A}_2, \Delta_2, \imath_2)$ and two synchronization predicates $\psi_1 \in \Psi_1$ and $\psi_2 \in \Psi_2$, the *natural projection on* M_1 of a trace σ of $M_1 \parallel_{\psi_1,\psi_2} M_2$, in symbols $P_{M_1}(\sigma)$, is defined as follows:

$$\begin{aligned}
&P_{M_1}(\langle p_1, p_2 \rangle) &&= p_1\\
&P_{M_1}((\langle p_1, p_2 \rangle, (d_1, 1), \langle p_1', p_2 \rangle) \cdot \sigma) &&= (p_1, d_1, p_1') \cdot P_{M_1}(\sigma)\\
&P_{M_1}((\langle p_1, p_2 \rangle, (d_2, 2), \langle p_1, p_2' \rangle) \cdot \sigma) &&= P_{M_1}(\sigma)\\
&P_{M_1}((\langle p_1, p_2 \rangle, ((d_1, d_2), 3), \langle p_1', p_2' \rangle) \cdot \sigma) &&= (p_1, d_1, p_1') \cdot P_{M_1}(\sigma)
\end{aligned}$$

The natural projection on the second component M_2 is analogously defined.
Also, we extend the natural projection to sets of traces in the usual way. □

Definition 5.6 (*Symbolic natural projection*) Given two s-LTS $M_1 = (Q_1, \mathcal{A}_1, \Delta_1, \imath_1)$ and $M_2 = (Q_2, \mathcal{A}_2, \Delta_2, \imath_2)$ and two synchronization predicates $\psi_1 \in \Psi_1$ and $\psi_2 \in \Psi_2$, the *symbolic natural projection on* M_1 of a symbolic trace η of $M_1 \parallel_{\psi_1,\psi_2} M_2$, in symbols $\varPi_{M_1}(\eta)$, is defined as follows:

$$\begin{aligned}
&\varPi_{M_1}(\langle p_1, p_2 \rangle) &&= p_1 &&\\
&\varPi_{M_1}((\langle p_1, p_2 \rangle, \psi, \langle p_1', p_2 \rangle) \cdot \eta) &&= (p_1, \varphi_1, p_1') \cdot \varPi_{M_1}(\eta) &&\text{if } \psi = \langle \varphi_1, \bot_2, \langle \bot_1, \bot_2 \rangle \rangle\\
&\varPi_{M_1}((\langle p_1, p_2 \rangle, \psi, \langle p_1, p_2' \rangle) \cdot \eta) &&= \varPi_{M_1}(\eta) &&\text{if } \psi = \langle \bot_1, \varphi_2, \langle \bot_1, \bot_2 \rangle \rangle\\
&\varPi_{M_1}((\langle p_1, p_2 \rangle, \psi, \langle p_1', p_2' \rangle) \cdot \eta) &&= (p_1, \varphi_1, p_1') \cdot \varPi_{M_1}(\eta) &&\text{if } \psi = \langle \bot_1, \bot_2, \langle \varphi_1, \varphi_2 \rangle \rangle
\end{aligned}$$

The symbolic natural projection on the second component M_2 is analogously defined and we extend this definition to sets of traces in the usual way.

The *inverse projection* of a trace σ over an s-LTS $M_1 \parallel_{\psi_1,\psi_2} M_2$, in symbols $\varPi_{M_1}^{-1}(\sigma)$, is defined as $\varPi_{M_1}^{-1}(\sigma) = \{\sigma' \mid \varPi_{M_1}(\sigma') = \sigma\}$, and is lifted to sets as usual. □

The following lemma shows that the natural projection of concrete traces coincides with the "concretization" via the function $s2c$ of the symbolic traces obtained via the symbolic natural projection.

Lemma 5.1 *For every s-LTSs* $M_1 = (Q_1, \mathcal{A}_1, \Delta_1, \imath_1)$ *and* $M_2 = (Q_2, \mathcal{A}_2, \Delta_2, \imath_2)$ *and synchronization predicates* $\psi_1 \in \Psi_1$ *and* $\psi_2 \in \Psi_2$ *the following holds*

$$P_{M_i}([\![M_1 \parallel_{\psi_1,\psi_2} M_2]\!]) = s2c(\varPi_{M_i}(tr(M_1 \parallel_{\psi_1,\psi_2} M_2))) \quad (\textit{with } i \in \{1, 2\})$$

We now lift the definition of quotienting a $\mu-$equations' system Φ for s-LTSs. The symbolic quotienting operator is $\Phi /\!\!/_{\psi_1,\psi_2} M$, where ψ_1 and ψ_2 are the synchronization predicates for M and for the s-LTS to be synthetized, respectively. The schema is the same of Definition 3.8 except for the cases that handle modalities. Since we are dealing with a product of EBAs, the alphabet symbols are as in Definition 5.3. Moreover, the transitions of M are now labeled by a predicate ψ. Hence, an action d_1 in the scope of a modality is a synchronization only if it satisfies ψ_1. Instead, if it satisfies $\neg\psi_1$, it denotes an asynchronous transition. This results in checking the satisfiability of $(\psi \wedge \psi_1)(d_1)$ and $(\psi \wedge \neg\psi_1)(d_1)$, respectively.

Definition 5.7 Given a top assertion $\Phi \downarrow x$ over the EBA $\mathcal{A}_1 \circledast_{\psi_1,\psi_2} \mathcal{A}_2$, we define its quotienting on a s-LTS $M = \langle Q, \mathcal{A}_1, \Delta, \imath\rangle$, in symbols $\Phi \downarrow x /\!\!/_{\psi_1,\psi_2} M$, as follows.

$$\Phi \downarrow x /\!\!/_{\psi_1,\psi_2} M = (\Phi /\!\!/_{\psi_1,\psi_2} M) \downarrow x_\imath, \text{ where}$$

$$\epsilon /\!\!/_{\psi_1,\psi_2} M = \epsilon \qquad (x =_\pi \varphi; \Phi) /\!\!/_{\psi_1,\psi_2} M = \begin{cases} x_{s_1} =_\pi \varphi /\!\!/_{\psi_1,\psi_2} s_1 \\ \vdots \\ x_{s_n} =_\pi \varphi /\!\!/_{\psi_1,\psi_2} s_n \end{cases} ;\ \Phi /\!\!/_{\psi_1,\psi_2} M \quad (\forall\, s_i \in Q)$$

$$x /\!\!/_{\psi_1,\psi_2} s = x_s \qquad tt /\!\!/_{\psi_1,\psi_2} s = tt \qquad f\!f /\!\!/_{\psi_1,\psi_2} s = f\!f$$

$$\varphi \vee \varphi' /\!\!/_{\psi_1,\psi_2} s = \varphi /\!\!/_{\psi_1,\psi_2} s \vee \varphi' /\!\!/_{\psi_1,\psi_2} s \qquad \varphi \wedge \varphi' /\!\!/_{\psi_1,\psi_2} s = \varphi /\!\!/_{\psi_1,\psi_2} s \wedge \varphi' /\!\!/_{\psi_1,\psi_2} s$$

$$(\langle(d_1,1)\rangle\varphi) /\!\!/_{\psi_1,\psi_2} s = \bigvee_{\substack{(s,\psi,s')\in\Delta \\ (\psi\wedge\neg\psi_1)(d_1)}} \varphi /\!\!/_{\psi_1,\psi_2} s' \qquad ([(d_1,1)]\varphi) /\!\!/_{\psi_1,\psi_2} s = \bigwedge_{\substack{(s,\psi,s')\in\Delta \\ (\psi\wedge\neg\psi_1)(d_1)}} \varphi /\!\!/_{\psi_1,\psi_2} s'$$

$$(\langle(d_2,2)\rangle\varphi) /\!\!/_{\psi_1,\psi_2} s = \begin{cases} \langle d_2\rangle(\varphi /\!\!/_{\psi_1,\psi_2} s) & \text{if } \neg\psi_2(d_2) \\ f\!f & \text{otherwise} \end{cases}$$

$$([(d_2,2)]\varphi) /\!\!/_{\psi_1,\psi_2} s = \begin{cases} [d_2](\varphi /\!\!/_{\psi_1,\psi_2} s) & \text{if } \neg\psi_2(d_2) \\ tt & \text{otherwise} \end{cases}$$

$$(\langle((d_1,d_2),3)\rangle\varphi) /\!\!/_{\psi_1,\psi_2} s = \begin{cases} \displaystyle\bigvee_{\substack{(s,\psi,s')\in\Delta \\ (\psi\wedge\psi_1)(d_1)}} \langle d_2\rangle(\varphi /\!\!/_{\psi_1,\psi_2} s') & \text{if } \psi_2(d_2) \\ f\!f & \text{otherwise} \end{cases}$$

$$([((d_1,d_2),3)]\varphi) /\!\!/_{\psi_1,\psi_2} s = \begin{cases} \displaystyle\bigwedge_{\substack{(s,\psi,s')\in\Delta \\ (\psi\wedge\psi_1)(d_1)}} [d_2](\varphi /\!\!/_{\psi_1,\psi_2} s') & \text{if } \psi_2(d_2) \\ tt & \text{otherwise} \end{cases}$$

□

We next establish the correspondence between symbolic quotienting and symbolic natural projection. To this end, we must redefine the $\mu-$calculus state semantics of Definition 3.5 (and therefore the trace semantics of Definition 3.6) which applies to LTSs, rather than s-LTSs. The new definition is straightforward since, given an s-LTS $M = (Q, \mathcal{A}, \Delta, \imath)$, it only requires introducing the following notation.

$$s \xrightarrow{a}_M s' \iff \exists(s, \varphi, s') \in \Delta \text{ s.t. } \varphi(a)$$

Theorem 5.1 *For all $M_1 = (Q_1, \mathcal{A}_1, \Delta_1, \imath_1)$, $M_2 = (Q_2, \mathcal{A}_2, \Delta_2, \imath_2)$, x, and Φ on the EBA $\mathcal{A}_1 \circledast_{\psi_1,\psi_2} \mathcal{A}_2$, we have that*

$$\langle\!\langle \Phi \downarrow x /\!\!/_{\psi_1,\psi_2} M_1 \rangle\!\rangle = P_{M_2}(\langle\!\langle \Phi \downarrow x \rangle\!\rangle).$$

As was the case for standard LTS, the synchronous product of two s-LTSs satisfies a global equation system if and only if its components satisfy their quotients, i.e., their local assertions. Note that Lemma 5.1 lifts this result also to symbolic natural projection.

Theorem 5.2 *For all* $M_1 = (Q_1, \mathcal{A}_1, \Delta_1, \imath_1)$, $M_2 = (Q_2, \mathcal{A}_2, \Delta_2, \imath_2)$, x, *and* Φ *on the EBA* $\mathcal{A}_1 \circledast_{\psi_1,\psi_2} \mathcal{A}_2$, *we have that*

$$M_1 \parallel_{\psi_1,\psi_2} M_2 \models_\varsigma \Phi \downarrow x \quad (\varsigma \in \{s, \sigma\})$$

if and only if any of the following equivalent statements holds:

1. $M_1 \models_\varsigma \Phi \downarrow x /\!\!/_{\psi_1,\psi_2} M_2$ *2.* $M_2 \models_\varsigma \Phi \downarrow x /\!\!/_{\psi_1,\psi_2} M_1$
3. $M_1 \models_\sigma P_{M_1}(\langle\!\langle \Phi \downarrow x \rangle\!\rangle)$ *4.* $M_2 \models_\sigma P_{M_2}(\langle\!\langle \Phi \downarrow x \rangle\!\rangle)$.

5.4 Quotienting Algorithm

Before introducing the symbolic quotienting algorithm, we recall the definition of *Minterms*. Intuitively, *Minterms* are building blocks for translating an s-LTS into an LTS that accepts an isomorphic language. Based on the predicates appearing on transitions, *Minterms* partition the EBA domain into a finite number of satisfiability regions. It is immediate then to define an isomorphism between these regions and a finite alphabet. Note however that the transitions of the resulting LTS are exponentially many with respect to those of the original s-LTS. The details of our translation are given inside the correctness proof in the "Technical Appendix".

Definition 5.8 [16] Let $M = \langle Q, \mathcal{A}, \imath, \Delta \rangle$ be an s-LTS, and let F denote the set of predicates labeling the transitions of M. The Minterms of M is the set

$$\mathit{Minterms}(M) = \bigcup\nolimits_{I \subseteq F} \{\varphi_I = \bigwedge_{\varphi \in I} \varphi \wedge \bigwedge_{\bar{\varphi} \in F \setminus I} \neg\bar{\varphi} \mid \mathbf{sat}_{\mathcal{A}}(\varphi_I)\}.$$

□

Since our symbolic quotienting algorithm manipulates an s-LTS P encoding a specification over a parallel product $M \parallel_{\psi_1,\psi_2} N$, the predicates on the transitions of P are four-tuples (see Definition 5.4). Therefore the same holds for *Minterms*(P).

The symbolic quotienting algorithm is given in Table 4. It has the same structure of the algorithm of Table 2, thus we focus here on explaining the relationship between them.

As for the LTS case, our algorithm consists of two main procedures and an auxiliary one. The first, called `quotient` (Table 4), builds a non-deterministic s-LTS whose states are pairs, given a specification P, an agent M, and a pair of synchronization predicates ψ_1 and ψ_2. The labels record whether they derive from a transition of M ($\psi_M \wedge \psi_{P|_1} \wedge \neg\psi_1$), of P ($\psi_{P|_2} \wedge \neg\psi_2$), or whether they denote a synchronization with P ($\psi_{P|_4} \wedge \psi_2$), provided that $\mathbf{sat}_{\mathcal{A}}(\psi_M \wedge \psi_{P|_3} \wedge \psi_1)$. The second procedure is `unify`, which differs from the analogous one in Table 3 because *Minterms* are used in place of plain action labels. The same holds for the auxiliary $\wedge$`-move`, where the states in the intersection must be reachable through a transition (labeled with φ') that is compatible with the *Minterm* predicate φ, in symbols $\mathbf{sat}_{\mathcal{B}}(\varphi \wedge \varphi')$.

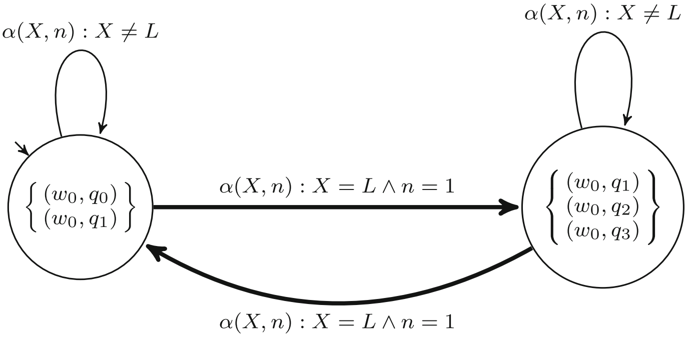
Also the symbolic quotienting algorithm is correct with respect to the previous quotienting operator (see the "Technical Appendix"). As expected, it runs in EXPTIME, because of the satisfiability requirements and because the number of *Minterms* grows exponentially with the transitions of the s-LTS. Of course, one can beforehand transform an s-LTS in an LTS by using *Minterms* and apply the quotienting algorithm of Sect. 4. The overall process still requires EXPTIME. However, the partial specification obtained in this way will be in the form of an LTS, thus lacking the expressive power of the corresponding s-LTS obtained through symbolic quotienting.

Table 4 The symbolic quotienting algorithm for s-LTS

```
Begin proc quotient
  input P = $(Q_P, \mathcal{A} \circledast_{\psi_1,\psi_2} \mathcal{B}, \Delta_P, \imath_P)$
  input M = $(Q_M, \mathcal{A}, \Delta_M, \imath_M)$

  $\bar{Q} := (Q_P \times Q_M) \setminus \bigcup_{(r,\varphi,r')\in\Delta_M} \{(s,r) \mid \mathbf{sat}_{\mathcal{A}}(\varphi \wedge \neg \bigvee_{(s,\psi,s')\in\Delta_P} \psi_{|1})\}$
  $i := (\imath_P, \imath_M)$
  $\bar{\Delta}_\lambda := \bigcup_{\substack{(q_P,\psi_P,q'_P)\in\Delta_P \\ (q_M,\psi_M,q'_M)\in\Delta_M}} \{((q_P,q_M), \psi_M \wedge \psi_{P|1} \wedge \neg\psi_1, (q'_P,q'_M))\}$
  $\bar{\Delta}_B := \bigcup_{\substack{(q_P,\psi_P,q'_P)\in\Delta_P \\ q_M\in Q_M}} \{((q_P,q_M), \psi_{P|2} \wedge \neg\psi_2, (q'_P,q_M))\}$
  $\bar{\Delta}^* := \bigcup_{\substack{(q_P,\psi_P,q'_P)\in\Delta_P \\ (q_M,\psi_M,q'_M)\in\Delta_M}} \begin{cases} \{((q_P,q_M), \psi_{P|4} \wedge \psi_2, (q'_P,q'_M))\} & \text{if } \mathbf{sat}_{\mathcal{A}}(\psi_M \wedge \psi_{P|3} \wedge \psi_1) \\ \emptyset & \text{otherwise} \end{cases}$
  N := unify($\imath$, $\bar{\Delta}_\lambda$, $\bar{\Delta}_B \cup \bar{\Delta}^*$)
  output N
End proc
```

```
Begin proc unify
  input $\imath$
  input $\bar{\Delta}_\lambda$
  input $\widehat{\Delta}$

  $I$ := close($\{i\}$, $\bar{\Delta}_\lambda$)
  $Q_N$, $S$ := $\{I\}$
  $\Delta_N$ := $\emptyset$
  while $S \neq \emptyset$ do
    $Q$ := pick&remove($S$)
    for each $\varphi$ s.t. $\exists\psi \in \mathit{Minterms}(P) : \varphi = \psi_{|i}$ with $i \in \{2,4\}$
      $M$ := $\wedge$-move($Q$, $\widehat{\Delta}$, $\varphi$)
      if $M \neq \emptyset$ then
        $\Delta_N$ := $\Delta_N \cup \{(Q,\varphi,M)\}$
        if $M \notin R$ then $S$ := $S \cup \{M\}$; $Q_N$ := $Q_N \cup \{M\}$ end if
      end if
    end for
  end while
  output $\langle \mathcal{B}, Q_N, \Delta_N, \imath \rangle$
end proc
```

```
Begin proc $\wedge$-move
  input $Q$
  input $\widehat{\Delta}$
  input $\varphi$

  $M$ := close($\bigcap^{q\in Q} \left\{ q' \;\middle|\; \begin{array}{l} (q,\varphi',q') \in \widehat{\Delta} \\ \text{and } \mathbf{sat}_{\mathcal{B}}(\varphi \wedge \varphi') \end{array} \right\}$, $\bar{\Delta}_\lambda$)
  output $M$
end proc
```



$\alpha(X,n) : X \neq L$
$\alpha(X,n) : X \neq L$
$\left\{ \begin{array}{c} (w_0, q_0) \\ (w_0, q_1) \end{array} \right\}$
$\alpha(X,n) : X = L \wedge n = 1$
$\left\{ \begin{array}{c} (w_0, q_1) \\ (w_0, q_2) \\ (w_0, q_3) \end{array} \right\}$
$\alpha(X,n) : X = L \wedge n = 1$

Fig. 9 The s-LTS corresponding to $W/\!\!/_{\psi_1,\psi_2} M_1$. Bold edges denote transitions in $\bar{\Delta}^*$

Example 11 We apply the algorithm of Table 4 to compute the quotient $W/\!\!/_{\psi_1,\psi_2} M_1$, where W is the specification of Example 10 depicted in Fig. 8, M_1 is the s-LTS of the consumer of Example 8, $\psi_1 = \alpha(X, n) : X = L \wedge \alpha = w$ and $\psi_2 = \beta(Y, m) : Y = L$.

First notice that (for some q and q') each transition in $\bar{\Delta}_\lambda$ has the form $(q, \psi_M \wedge \psi_{P|_1} \wedge \neg\psi_1, q'\})$. However, $\psi_M \wedge \neg\psi_1$ is satisfiable only if the sub-formula $\alpha(X, n) : X = L \wedge X \neq L$ is satisfiable, which is trivially false. For this reason $\bar{\Delta}_\lambda = \emptyset$.

Since $\bar{\Delta}_\lambda = \emptyset$, the set of transitions of the resulting s-LTS is given by $\bar{\Delta}_B \cup \bar{\Delta}^*$. Figure 9 shows this, where we use different edge thickness to distinguish between the transitions of $\bar{\Delta}_B$ and $\bar{\Delta}^*$.

6 Related Work

Natural projection is mostly used by the community working on control theory and discrete-event systems. In the 1980s, the seminal works by Wonham et al. (e.g., [41,42]) exploited natural projection-based algorithms for synthesizing both local and global controllers. Other authors continued this line of research and proposed extensions and refinements of these methods, see e.g., [18,19,30,39].

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 [7] 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 [32] use partial model checking to generate a control process for a partially unspecified system in order to guarantee compliance with respect to a μ-calculus formula. The generated controller takes the form of an edit automaton [8]. A quotienting-based approach was also proposed for real-time [29] and hybrid [12] systems. These paradigms aim to accurately model the behavior of, e.g., cyber-physical systems.

Some researchers have proposed techniques based on the verification of temporal logics for addressing the controller synthesis problem. Arnold et al. [5] were among the first to control a deterministic plant with a μ-calculus specification. Also Ziller and Schneider [43] and Riedweg and Pinchinat [34] reduce the problem of synthesizing a controller to checking the satisfiability of a formula in (a variant of) the μ-calculus. A similar approach was presented by Jiang and Kumar [25] and Gromyko et al. [22]. Similarly to [43] and [34], [25] present an approach that reduces the problem of synthesizing a controller to that of checking a CTL* formula's satisfiability. In contrast, [22] proposes a method based on symbolic model checking to synthesize controllers. Their approach applies to a fragment of CTL.

7 Conclusion

Our work provides results that build a bridge between supervisory control theory and formal verification. In particular, we have formally established the relationship between partial model checking and natural projection by reducing natural projection to partial model checking and proving their equivalence under common assumptions. Besides using plain Labeled Transition System for expressing system specifications, we also considered symbolic Labeled Transitions System, whose transitions carry predicates on elements from possibly infinite boolean algebras, instead of letters. Dealing with this richer model required us to introduce new notions, including a new symbolic synchronous product and new symbolic versions of partial model checking and natural projection.

Aside from establishing novel and particularly relevant connections, our work also opens new directions for investigation. Since (symbolic) natural projection is related to language theory in general, there could be other application fields where (symbolic) partial model checking can be used as an alternative. The original formulation of partial model checking applies to the μ-calculus, while our quotienting algorithm works on (symbolic) Labeled Transitions Systems. To the best of our knowledge, no quotienting algorithms exist for formalisms with a different expressive power, such as LTL or CTL, let alone symbolic variants of them.

We are also developing PESTS, a working prototype to handle both LTSs and s-LTSs. The source code and the documentation of our tool are available at https://github.com/gabriele-costa/pests, along with the experiments mentioned below. The performance of PESTS was experimentally assessed in [13] and the results are on the website under the heading "TACAS Experiments". The experiments consisted in solving instances of increasing size of CSP and SCP for LTSs modeling an Unmanned Aerial Vehicles delivery system. Furthermore, we applied PESTS to a more realistic case study concerning the verification of the LTSs modeling a Flexible manufacturing system,[3] available under the heading "Flexible manufacturing system".

Acknowledgements Open access funding provided by Scuola IMT Alti Studi Lucca within the CRUI-CARE Agreement. This work was partially supported by SNSF funded project IZK0Z2 168370 "Enforceable Security Policies in Fog Computing", by EU Horizon 2020 project No. 830892 "SPARTA", by MIUR project PRIN 2017FTXR7S "IT MATTERS" (Methods and Tools for Trustworthy Smart Systems) and by "PRA 2018 66 DECLware: Declarative methodologies for designing and deploying applications" of the Università di Pisa.

[3] Based on http://www.rt.techfak.fau.de/FGdes/index.html.

A Technical Appendix

A.1 GPUVerify

Below we show an excerpt of the output generated by GPUVerify [9], when executed on our example kernel `producer-consumer.c` in Fig. 1. The first line invokes the tool. The second line reports the false positive, i.e., that a write-read race is detected on the first byte of `L`. The rest of the output compares the instructions of the two components that cause the data race.

```
$ gpuverify --local_size=2 --global_size=2
 producer-consumer.c

  [...]
producer-consumer.c: error: possible write-read
 race on L[0] (byte 1):

Read by work item 0 in work group 0, producer-
consumer.c:15:3:
  buffer[*tail] = val;         // enqueue value
invoked from producer-consumer.c:32:7:
  produce(L, buffer, val+1);

Write by work item 1 in work group 0, producer-
consumer.c:8:5:
  *head = 0;
invoked from producer-consumer.c:28:13:
  val = consume(L, buffer);
  [...]
```

A.2 Technical Proofs

Here we prove the lemmata and theorems of the paper. We also introduce some relevant definitions on S-LTSs.

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 ρ_i over the elements of an index set I.

Definition A.1 Given a synchronous product $A \parallel B$, we define $\nabla_B(\cdot) : (X \to 2^{S_A \times S_B}) \to (X_{S_A} \to 2^{S_B})$ as

$$\nabla_B(\rho) = \bigoplus_{x \in Dom(\rho)} \bigoplus_{s_A \in S_A} [x_{s_A} \mapsto U^x_B(s_A)], \text{ where } U^x_B(s_A) = \{s_B \mid \langle s_A, s_B \rangle \in \rho(x)\}.$$

□

The following lemma intuitively states that 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 $\|\phi /\!\!/_{\Sigma_B} s_A\|_{\nabla_B(\rho)} = \{s_B \mid \langle s_A, s_B \rangle \in \|\phi\|_\rho\}$.

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

Proof By induction over the structure of ϕ.

- Cases tt and $f\!f$. Trivial.
- Case x. By the definition of $\nabla_B(\rho)$.
- Cases $\phi \wedge \phi'$, $\phi \vee \phi'$. By the induction hypothesis.
- Case $\langle a \rangle \phi$. By Definition 3.5, $\langle s_A, s_B \rangle \in \|\langle a \rangle \phi\|_\rho$ if and only if $\exists s'_A, s'_B$ such that $\langle s_A, s_B \rangle \xrightarrow{a}_{A \| B} \langle s'_A, s'_B \rangle \wedge \langle s'_A, s'_B \rangle \in \|\phi\|_\rho$. By the induction hypothesis, this is equivalent to

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

  Then we consider three exhaustive cases.

  - $a \in \Sigma_A \backslash \Gamma$. Here $s'_B = s_B$ and (1) is satisfied if and only if $s_B \in \|\bigvee_{s_A \xrightarrow{a}_A s'} \phi /\!\!/_{\Sigma_B} s'\|_{\nabla_B(\rho)}$. We conclude by applying Definition 3.8.
  - $a \in \Sigma_B \backslash \Gamma$. In this case $s'_A = s_A$ and, by Definition 3.5, (1) is equivalent to $s_B \in \|\langle a \rangle (\phi /\!\!/_{\Sigma_B} s_A)\|_{\nabla_B(\rho)}$. Again, we close the case by applying Definition 3.8.
  - $a \in \Gamma$. We combine the reasoning of the two previous cases to conclude that $s_B \in \|\bigvee_{s_A \xrightarrow{a}_A s'} \langle a \rangle (\phi /\!\!/_B s')\|_{\nabla_{\Sigma_B}(\rho)}$.

- Case $[a]\phi$. Symmetric to the previous one.

□

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

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

Proof We proceed by induction on the structure of Φ.

- Base case: $\Phi = \epsilon$. Trivial.
- Induction step: $\Phi = x =_\pi \phi; \Phi'$. By definition, $\|\Phi\|_\rho = [x \mapsto U^*] \circ \|\Phi'\|_{\rho \circ [x \mapsto U^*]}$ where U^* is the fixed point computed according to Definition 3.5. Thus, we have that $\nabla_B(\|\Phi\|_\rho) = \nabla_B([x \mapsto U^*] \circ \|\Phi'\|_{\rho \circ [x \mapsto U^*]}) = \nabla_B([x \mapsto U^*]) \circ \nabla_B(\|\Phi'\|_{\rho \circ [x \mapsto U^*]})$. By the induction hypothesis, this reduces to

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

  By Definition A.1, $\nabla_B([x \mapsto U^*]) = \bigoplus_{s \in S_A} [x_s \mapsto U^*_{B,s}]$ where $U^*_{B,s} = \{s' \mid \langle s, s' \rangle \in U^*\}$. By replacing U^* with its definition we obtain

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

  which we rewrite to

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

  By Lemma A.1, this is equivalent to

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

  where $\nabla_B(R(U)) = \nabla_B([x \mapsto U] \circ \|\Phi'\|_{\rho \circ [x \mapsto U]})$. By induction hypothesis and by Definition A.1, we have

$$\bigoplus_{s \in S_A} [x_s \mapsto U_{B,s}] \circ \nabla_B(\|\Phi'\|_{\rho \circ [x \mapsto U]})$$

$$= \bigoplus_{s\in S_A}[x_s \mapsto U_{B,s}] \circ \|\Phi' /\!\!/_{\Sigma_B} A\|_{\nabla_B(\rho) \circ \bigoplus_{s\in S_A}[x_s \mapsto U_{B,s}]}$$

As a consequence, we rewrite (3) to[4]

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

which, after repeatedly applying Definition 3.5 to each element $s \in S_A$ turns out to be $\|\Phi /\!\!/_{\Sigma_B} A\|_{\nabla_B(\rho)}$.

□

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

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

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

Theorem 3.1 *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 3.6, it suffices to establish

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

which holds if and only if

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

We conclude by Corollary A.1. □

Theorem 3.2 *For all A, B, x and Φ on $A \parallel B$,*

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

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 Andersen95partialmodel (with the additional use of Theorem 3.1). The other equivalences follow immediately by Theorem 3.1 (and by the commutativity of $\parallel$). □

We now lift the needed definition and the results above to the symbolic case.

Lemma 5.1 *For every s-LTSs $M_1 = (Q_1, \mathcal{A}_1, \Delta_1, \iota_1)$ and $M_2 = (Q_2, \mathcal{A}_2, \Delta_2, \iota_2)$ and synchronization predicates $\psi_1 \in \Psi_1$ and $\psi_2 \in \Psi_2$ the following holds*

$$P_{M_i}([\![M_1 \parallel_{\psi_1,\psi_2} M_2]\!]) = s2c(\Pi_{M_i}(tr(M_1 \parallel_{\psi_1,\psi_2} M_2))) \quad (\text{with } i \in \{1, 2\})$$

Proof From now on we assume $i = 1$ as the case for $i = 2$ is symmetric. We start by observing that, by definition, for every s-LTS M holds that $s2c(tr(M)) = [\![M]\!]$. Thus we rewrite the proof statement as

$$P_{M_1}(s2c(tr(M_1 \parallel_{\psi_1,\psi_2} M_2))) = s2c(\Pi_{M_1}(tr(M_1 \parallel_{\psi_1,\psi_2} M_2)))$$

Then we prove by induction that $\forall \eta . s2c(\Pi_{M_1}(\eta)) = P_{M_1}(s2c(\eta))$.

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

1. Case $\eta = \langle p_1, p_2\rangle$. Trivial.
2. Case $\eta = (\langle p_1, p_2\rangle, \langle \varphi_1, \bot_2, \langle \bot_1, \bot_2\rangle\rangle, \langle p_1', p_2\rangle) \cdot \eta'$. In this case $s2c(\Pi_{M_1}(\eta)) = s2c((p_1, \varphi_1, p_1') \cdot \Pi_{M_1}(\eta')) = \{(p_1, d_1, p_1') \cdot \sigma \mid d_1 \in \{\!\!\{\varphi_1\}\!\!\} \wedge \sigma \in s2c(\Pi_{M_1}(\eta'))\}$. By the induction hypothesis this is equal to $\{(p_1, d_1, p_1') \cdot \sigma \mid d_1 \in \{\!\!\{\varphi_1\}\!\!\} \wedge \sigma \in P_{M_1}(s2c(\eta'))\} = P_{M_1}(s2c(\eta))$.
3. Case $\eta = (\langle p_1, p_2\rangle, \langle \bot_1, \varphi_2, \langle \bot_1, \bot_2\rangle\rangle, \langle p_1, p_2'\rangle) \cdot \eta'$. In this case it suffices to apply the induction hypothesis on η'.
4. Case $\eta = (\langle p_1, p_2\rangle, \langle \bot_1, \bot_2, \langle \varphi_1, \varphi_2\rangle\rangle, \langle p_1', p_2'\rangle) \cdot \eta'$. This case is analogous to case 2.

□

The following definition extends Definition A.1.

Definition A.2 For all $M_1 = \langle Q_1, \mathcal{A}_1, \Delta_1, \imath_1\rangle$, $M_2 = \langle Q_2, \mathcal{A}_2, \Delta_2, \imath_2\rangle$, x, and Φ on the EBA $\mathcal{A}_1 \circledast_{\psi_1,\psi_2} \mathcal{A}_2$, we define $\nabla_{M_2}(\cdot) : (X \to 2^{Q_1 \times Q_2}) \to (X_1 \to 2^{Q_2})$

$$\nabla_{M_2}(\rho) = \bigoplus_{x \in Dom(\rho)} \bigoplus_{s_1 \in Q_1} [x_{s_1} \mapsto U^x_{M_2}(s_1)], \text{ where } U^x_{M_2}(s_1) = \{s_2 \mid \langle s_1, s_2\rangle \in \rho(x)\}.$$

□

Now we extend to the symbolic case the auxiliary lemmata A.1 and A.2.

Lemma A.3 *For all $M_1 = \langle Q_1, \mathcal{A}_1, \Delta_1, \imath_1\rangle$, $M_2 = \langle Q_2, \mathcal{A}_2, \Delta_2, \imath_2\rangle$, ρ, and ϕ on the EBA $\mathcal{A}_1 \circledast_{\psi_1,\psi_2} \mathcal{A}_2$, we have that*

$$\langle s_1, s_2\rangle \in \|\phi\|_\rho \Longleftrightarrow s_2 \in \|\phi \mathbin{/\!\!/}_{\psi_1,\psi_2} s_1\|_{\nabla_{M_2}(\rho)}.$$

Proof We proceed by induction over ϕ. The only interesting cases are those for the two modalities, that is, $\langle d\rangle\phi'$ and $[d]\phi'$. Each modality only admits three sub-cases depending on whether $d = (d_1, 1)$, $d = (d_2, 2)$ or $d = (\langle d_1, d_2\rangle, 3)$. We show the first case as the other case is symmetric.

- $\langle (d_1, 1)\rangle\phi'$. In this case $\langle s_1, s_2\rangle \in \|\langle (d_1, 1)\rangle\phi'\|_\rho$ if and only if there exists $(s_1, \varphi, s_1') \in \Delta_1$ such that $(\varphi \wedge \neg\psi_1)(d_1, 1)$ and $\langle s_1', s_2\rangle \in \|\phi'\|_\rho$. We rewrite it in the following, equivalent form.

$$\langle s_1', s_2\rangle \in \bigcup_{\substack{(s_1, \varphi, s_1') \in \Delta_1 \\ (\varphi \wedge \neg\psi_1)(d_1, 1)}} \|\phi'\|_\rho$$

  Then we apply the induction hypothesis to ϕ' and obtain

$$s_2 \in \bigcup_{\substack{(s_1, \varphi, s_1') \in \Delta_1 \\ (\varphi \wedge \neg\psi_1)(d_1, 1)}} \|\phi' \mathbin{/\!\!/}_{\psi_1,\psi_2} s_1'\|_{\nabla_{M_2}(\rho)} = \| \bigvee_{\substack{(s_1, \varphi, s_1') \in \Delta_1 \\ (\varphi \wedge \neg\psi_1)(d_1, 1)}} \phi' \mathbin{/\!\!/}_{\psi_1,\psi_2} s_1'\|_{\nabla_{M_2}(\rho)}$$

  That is, by Definition 5.7, $\|\phi \mathbin{/\!\!/}_{\psi_1,\psi_2} s_1\|_{\nabla_{M_2}(\rho)}$.
- $\langle (d_2, 2)\rangle\phi'$. By definition $\langle s_1, s_2\rangle \in \|\langle (d_2, 2)\rangle\phi'\|_\rho$ if and only if there exists $(s_2, \varphi, s_2') \in \Delta_2$ such that $(\varphi \wedge \neg\psi_2, 2)(d_2)$ holds and $\langle s_1, s_2'\rangle \in \|\phi'\|_\rho$. By induction hypothesis this is equivalent to $\exists (s_2, \varphi, s_2') \in \Delta_2$ s.t. $(\varphi \wedge \neg\psi_2)(d_2, 2)$ holds and $s_2' \in \|\phi' \mathbin{/\!\!/}_{\psi_1,\psi_2} s_1\|_{\nabla_{M_2}(\rho)}$. Since ψ_2 and d_2 are given, this formula admits two alternative reductions corresponding to Definition 5.7. If $\neg\psi_2(d_2, 2)$ holds, it is equivalent to the claim $s_2 \in \|\langle d_2\rangle\phi' \mathbin{/\!\!/}_{\psi_1,\psi_2} s_1\|_{\nabla_{M_2}(\rho)}$. Otherwise it reduces to $s_2 \in \|\mathit{ff}\|_{\nabla_{M_2}(\rho)} = \emptyset$.

- $\langle(\langle d_1, d_2\rangle, 3)\rangle\phi'$. By definition $\langle s_1, s_2\rangle \in \|\langle(\langle d_1, d_2\rangle, 3)\rangle\phi'\|_\rho$ if and only if there exist $(s_1, \varphi_1, s_1') \in \Delta_1$ and $(s_2, \varphi_2, s_2') \in \Delta_2$ such that both $(\varphi_1 \wedge \neg\psi_1)(d_1)$ and $(\varphi_2 \wedge \neg\psi_2)(d_2)$ hold and $\langle s_1', s_2'\rangle \in \|\phi'\|_\rho$. When $\psi_2(d_2, 2)$ does not hold, this reduces to the false statement (since a synchronization transition labeled with d_2 cannot occur). Otherwise, we apply the induction hypothesis to obtain

$$s_2 \in \bigcup_{\substack{(s_1, \varphi_1, s_1') \in \Delta_1 \\ (\varphi_1 \wedge \neg\psi_1)(d_1, 1)}} \|\langle d_2\rangle\phi' /\!\!/_{\psi_1,\psi_2} s_1'\|_{\nabla_{M_2}(\rho)}$$
$$= \| \bigvee_{\substack{(s_1, \varphi, s_1') \in \Delta_1 \\ (\varphi \wedge \neg\psi_1)(d_1, 1)}} \langle d_2\rangle\phi' /\!\!/_{\psi_1,\psi_2} s_1'\|_{\nabla_{M_2}(\rho)}.$$

□

Lemma A.4 *For all $M_1 = \langle Q_1, \mathcal{A}_1, \Delta_1, \imath_1\rangle$, $M_2 = \langle Q_2, \mathcal{A}_2, \Delta_2, \imath_2\rangle$, $\rho : X \to 2^{Q_1\times Q_2}$, and Φ on the EBA $\mathcal{A}_1 \circledast_{\psi_1,\psi_2} \mathcal{A}_2$, we have that*

$$\nabla_{M_2}(\|\Phi\|_\rho) = \|\Phi /\!\!/_{\psi_1,\psi_2} M_1\|_{\nabla_{M_2}(\rho)}.$$

Proof We proceed by induction on the structure of Φ.

- Base case: $\Phi = \epsilon$. Trivial.
- Induction step: $\Phi = x =_\pi \phi; \Phi'$. By definition, $\|\Phi\|_\rho = [x \mapsto U^*] \circ \|\Phi'\|_{\rho\circ[x\mapsto U^*]}$ where U^* is the fixed point computed according to Definition 3.5. Thus, we have that $\nabla_{M_2}(\|\Phi\|_\rho) = \nabla_{M_2}([x \mapsto U^*] \circ \|\Phi'\|_{\rho\circ[x\mapsto U^*]}) = \nabla_{M_2}([x \mapsto U^*]) \circ \nabla_{M_2}(\|\Phi'\|_{\rho\circ[x\mapsto U^*]})$. By the induction hypothesis, this reduces to

$$\nabla_{M_2}([x \mapsto U^*]) \circ \|\Phi' /\!\!/_{\psi_1,\psi_2} M_1\|_{\nabla_{M_2}(\rho)\circ\nabla_{M_2}([x\mapsto U^*])}. \tag{3}$$

By Definition A.2, $\nabla_{M_2}([x \mapsto U^*]) = \bigoplus_{s\in Q_1}[x_s \mapsto U^*_{M_2,s}]$ where $U^*_{M_2,s} = \{s' \mid \langle s, s'\rangle \in U^*\}$. By replacing U^* with its definition we obtain

$$U^*_{M_2,s} = \{s' \mid \langle s, s'\rangle \in \pi U.(\|\phi\|_{\rho\circ R(U)})\},$$

which we rewrite to

$$U^*_{M_2,s} = \pi U.\{s' \mid \langle s, s'\rangle \in \|\phi\|_{\rho\circ R(U)}\}.$$

By Lemma A.3, this is equivalent to

$$U^*_{M_2,s} = \pi U_{M_2,s}.(\|\phi /\!\!/_{\psi_1,\psi_2} s\|_{\nabla_{M_2}(\rho)\circ\nabla_{M_2}(R(U))}),$$

where $\nabla_{M_2}(R(U)) = \nabla_{M_2}([x \mapsto U] \circ \|\Phi'\|_{\rho\circ[x\mapsto U]})$. By induction hypothesis and by Definition A.2, we have

$$\bigoplus_{s\in Q_1}[x_s \mapsto U_{M_2,s}] \circ \nabla_{M_2}(\|\Phi'\|_{\rho\circ[x\mapsto U]}) =$$
$$\bigoplus_{s\in Q_1}[x_s \mapsto U_{M_2,s}] \circ \|\Phi' /\!\!/_{\psi_1,\psi_2} M_1\|_{\nabla_{M_2}(\rho)\circ\bigoplus_{s\in Q_1}[x_s\mapsto U_{M_2,s}]}.$$

As a consequence, we rewrite (3) to

$$\bigoplus_{s\in Q_1}[x_s \mapsto U^*_{M_2,s}] \circ \|\Phi' /\!\!/_{\psi_1,\psi_2} M_1\|_{\nabla_{M_2}(\rho)\circ\bigoplus_{s\in Q_1}[x_s\mapsto U^*_{M_2,s}]},$$

which, after repeatedly applying Definition 3.5 to each element $s \in Q_1$ reduces to $\|\Phi /\!\!/_{\psi_1,\psi_2} M_1\|_{\nabla_{M_2}(\rho)}$. □

As expected, the lemmata above directly imply the following corollary.

Corollary A.2 *For all* $M_1 = \langle Q_1, \mathcal{A}_1, \Delta_1, \imath_1\rangle$, $M_2 = \langle Q_2, \mathcal{A}_2, \Delta_2, \imath_2\rangle$, $\rho : X \to 2^{Q_1 \times Q_2}$, x *and* Φ *on the EBA* $\mathcal{A}_1 \circledast_{\psi_1,\psi_2} \mathcal{A}_2$*, we have that*

$$\langle s_1, s_2\rangle \in \|\Phi\|_\rho(x) \iff s_2 \in \|\Phi /\!\!/_{\psi_1,\psi_2} M_1\|_{\nabla_{M_2}(\rho)}(x_1).$$

Theorem 5.1 *For all* $M_1 = (Q_1, \mathcal{A}_1, \Delta_1, \imath_1)$, $M_2 = (Q_2, \mathcal{A}_2, \Delta_2, \imath_2)$, x, *and* Φ *on the EBA* $\mathcal{A}_1 \circledast_{\psi_1,\psi_2} \mathcal{A}_2$*, we have that*

$$\langle\!\langle \Phi \downarrow x /\!\!/_{\psi_1,\psi_2} M_1 \rangle\!\rangle = P_{M_2}(\langle\!\langle \Phi \downarrow x \rangle\!\rangle).$$

Proof Follows from Corollary A.2. □

Theorem 5.2 *For all* $M_1 = (Q_1, \mathcal{A}_1, \Delta_1, \imath_1)$, $M_2 = (Q_2, \mathcal{A}_2, \Delta_2, \imath_2)$, x, *and* Φ *on the EBA* $\mathcal{A}_1 \circledast_{\psi_1,\psi_2} \mathcal{A}_2$*, we have that*

$$M_1 \parallel_{\psi_1,\psi_2} M_2 \models_\varsigma \Phi \downarrow x \quad (\varsigma \in \{s, \sigma\})$$

if and only if any of the following equivalent statements holds:

1. $M_1 \models_\varsigma \Phi \downarrow x /\!\!/_{\psi_1,\psi_2} M_2$ 2. $M_2 \models_\varsigma \Phi \downarrow x /\!\!/_{\psi_1,\psi_2} M_1$
3. $M_1 \models_\sigma P_{M_1}(\langle\!\langle \Phi \downarrow x \rangle\!\rangle)$ 4. $M_2 \models_\sigma P_{M_2}(\langle\!\langle \Phi \downarrow x \rangle\!\rangle)$.

Proof The equivalence between $M_1 \parallel_{\psi_1,\psi_2} M_2 \models_\varsigma \Phi \downarrow x$ and items 1, 2 immediately follows from Corollary A.2. Then, the equivalence with items 3 and 4 follows by applying Theorem 5.1 to items 1 and 2. □

A.3 Correctness

Below we prove the correctness of our quotienting algorithms. For the LTS quotienting algorithm described in Sect. 4.1, 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. Then, for the s-LTS quotienting algorithm of Sect. 5.4 we show its correctness by proving that it preserves the isomorphism between an s-LTS and its translation into an LTS.
Correctness for LTS **Encoding,** Given an LTS $A = \langle S, \Sigma, \to, s_i\rangle$, 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 $\mathcal{A}(s) = \bigwedge_{s \not\xrightarrow{b}} [b]f\!f \wedge \bigwedge_{s \xrightarrow{a} s'} [a]x^{s'}$.

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 evaluate $\Phi_P \downarrow x^{i_P} /\!\!/_{\Sigma_B} A$. The resulting equations system is

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

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

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

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.

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

  All the occurrences of y in the first equation are replaced with the assertion associated to y. Note 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

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

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

Correctness for s-LTS The proof consists of two steps. We start by defining a translation procedure from an s-LTS M to an isomorphic LTS A_M. Our translation is based on the standard *Minterms* construction algorithm (see [16] for a detailed description). Then, we show that the symbolic quotienting algorithm applied to the s-LTSs P and M returns an s-LTS N such that its translation A_N is isomorphic to the output of the quotienting algorithm of Sect. 4.1 when applied to the translations A_P and A_M.

We recall the standard definition of *Minterms*. Let $M = \langle \mathcal{A}, Q, \imath, \Delta \rangle$ be an s-LTS, and let F denote the set of predicates labeling the transitions of M. The Minterms of M is the set

$$\mathit{Minterms}(M) = \bigcup_{I \subseteq F} \{\varphi_I = \bigwedge_{\varphi \in I} \varphi \wedge \bigwedge_{\bar{\varphi} \in F \setminus I} \neg \bar{\varphi} \mid \mathbf{sat}_{\mathcal{A}}(\varphi_I)\}.$$

Note that since every s-LTS M has a finite number of transitions, the set of Minterms is finite as well. In particular, for an s-LTS M, the size of $\mathit{Minterms}(M)$ is, in the worst case, $2^{|\Delta|}$ where $|\Delta|$ is the number of transitions of M. Minterms are used to construct a deterministic LTS being isomorphic to an s-LTS M. The construction is based on a generic labeling function $f : \mathit{Minterms}(M) \to \Sigma$ where Σ is a set of action labels (see Definition 3.1).

Our LTS translation is slightly different. In particular, we construct a labeling function that can be applied to both the s-LTSs of a product so that synchronous transitions are mapped to the same symbol. To this end we apply the Minterms construction to the policy s-LTS P. Recalling that the EBA of P is $\mathcal{A} \circledast_{\psi_1,\psi_2} \mathcal{B}$, the definition of $\mathit{Minterms}(P)$ is specialized to

$$\bigcup_{I \subseteq F} \{\varphi_I = \bigwedge_{\langle \varphi_1, \varphi_2, \langle \varphi_3, \varphi_4 \rangle \rangle \in I} \langle \varphi_1, \varphi_2, \langle \varphi_3, \varphi_4 \rangle \rangle \wedge \bigwedge_{\langle \bar{\varphi}_1, \bar{\varphi}_2, \langle \bar{\varphi}_3, \bar{\varphi}_4 \rangle \rangle \in F \setminus I} \neg \langle \bar{\varphi}_1, \bar{\varphi}_2, \langle \bar{\varphi}_3, \bar{\varphi}_4 \rangle \rangle \mid \mathbf{sat}_{\mathcal{A} \circledast_{\psi_1,\psi_2} \mathcal{B}}(\varphi_I)\},$$

which, by definition of $\circledast$, reduces to

$$\bigcup_{I \subseteq F} \{\varphi_I = \langle \bigwedge_{\substack{\varphi_1 \in I_{|_1} \\ \bar{\varphi}_1 \in (F \setminus I)_{|_1}}} \varphi_1 \wedge \neg \bar{\varphi}_1, \bigwedge_{\substack{\varphi_2 \in I_{|_2} \\ \bar{\varphi}_2 \in (F \setminus I)_{|_2}}} \varphi_2 \wedge \neg \bar{\varphi}_2, \langle \bigwedge_{\substack{\varphi_3 \in I_{|_3} \\ \bar{\varphi}_3 \in (F \setminus I)_{|_3}}} \varphi_3 \wedge \neg \bar{\varphi}_3, \bigwedge_{\substack{\varphi_4 \in I_{|_4} \\ \bar{\varphi}_4 \in (F \setminus I)_{|_4}}} \varphi_4 \wedge \neg \bar{\varphi}_4 \rangle \rangle \mid \mathbf{sat}_{\mathcal{A} \circledast_{\psi_1,\psi_2} \mathcal{B}}(\varphi_I)\},$$

where $I_{|_i}$ is a short hand for $\{\varphi_{|_i} \mid \varphi \in I\}$.

Note that, by the definition of $\mathcal{A} \circledast_{\psi_1,\psi_2} \mathcal{B}$, all the predicates $\varphi \in F$ belong to three distinguished groups, i.e., $\langle \varphi_{|_1}, \bot, \bot \rangle$, $\langle \bot, \varphi_{|_2}, \bot \rangle$ or $\langle \bot, \bot, \langle \varphi_{|_3}, \varphi_{|_4} \rangle \rangle$. Thus, there cannot exist any $\varphi_I \in \mathit{Minterms}(P)$ such that I contains two or more predicates belonging to different groups (since the $\bot$ elements would cause such predicate to be unsatisfiable). As a consequence, $\mathit{Minterms}(P)$ preserves these three groups and, possibly, introduces an element for $\varphi_\emptyset$.

Starting from $\mathit{Minterms}(P)$, we define two labeling functions $f_{\mathcal{A}}^{\psi_1} : \mathit{Minterms}(P) \to \Sigma_A \cup \Gamma \cup \{\omega_A\}$ and $f_{\mathcal{B}}^{\psi_2} : \mathit{Minterms}(P) \to \Sigma_B \cup \Gamma \cup \{\omega_B\}$ such that $\Sigma_A, \Sigma_B, \Gamma$ are pairwise disjoint, $\omega_A \neq \omega_B$ and $\omega_A, \omega_B \notin \Sigma_A \cup \Sigma_B \cup \Gamma$. In addition, we require that whenever $\psi \in \mathit{Minterms}(P)$, $f_{\mathcal{A}}^{\psi_1}$ and $f_{\mathcal{B}}^{\psi_2}$ are the smallest functions that satisfy the following conditions:

1. $\mathbf{sat}_{\mathcal{A}}(\psi_{|_3} \wedge \psi_1)$ and $\mathbf{sat}_{\mathcal{B}}(\psi_{|_4} \wedge \psi_2)$ imply $f_{\mathcal{A}}^{\psi_1}(\psi) = f_{\mathcal{B}}^{\psi_2}(\psi) \in \Gamma$
2. if $\psi = \psi_\emptyset$ then (i) $\mathbf{sat}_{\mathcal{A}}(\psi_{|_1} \vee \psi_{|_3})$ implies $f_{\mathcal{A}}^{\psi_1}(\psi) = \omega_A$ and (ii) $\mathbf{sat}_{\mathcal{B}}(\psi_{|_2} \vee \psi_{|_4})$ implies $f_{\mathcal{B}}^{\psi_2}(\psi) = \omega_B$
3. $\mathbf{sat}_{\mathcal{A}}(\psi_{|_1} \wedge \neg \psi_1)$ implies $f_{\mathcal{A}}^{\psi_1}(\psi) \in \Sigma_A$ and $\mathbf{sat}_{\mathcal{B}}(\psi_{|_2} \wedge \neg \psi_2)$ implies $f_{\mathcal{B}}^{\psi_2}(\psi) \in \Sigma_B$

Given an s-LTS $M = \langle \mathcal{A}, Q, \imath, \Delta \rangle$ the LTS isomorphic to M (w.r.t. $f_{\mathcal{A}}^{\psi_1}$) is $A_M = \langle Q, \Sigma_A \cup \Gamma \cup \{\omega_A\}, \to, \imath \rangle$ where

$$\to = \begin{array}{ll} \{p \xrightarrow{a} q \mid \exists (p, \varphi, q) \in \Delta, \psi \in \mathit{Minterms}(P) \text{ s.t. } f_{\mathcal{A}}^{\psi_1}(\psi) = a \in \Sigma_A \text{ and } \mathbf{sat}_{\mathcal{A}}(\varphi \wedge \psi_{|_1} \wedge \neg \psi_1)\} & \cup \\ \{p \xrightarrow{\gamma} q \mid \exists (p, \varphi, q) \in \Delta, \psi \in \mathit{Minterms}(P) \text{ s.t. } f_{\mathcal{A}}^{\psi_1}(\psi) = \gamma \in \Gamma \text{ and } \mathbf{sat}_{\mathcal{A}}(\varphi \wedge \psi_{|_3} \wedge \psi_1)\} & \cup \\ \{p \xrightarrow{\omega_A} q \mid \exists (p, \varphi, q) \in \Delta, \psi \in \mathit{Minterms}(P) \text{ s.t. } f_{\mathcal{A}}^{\psi_1}(\psi) = \omega_A \text{ and } \mathbf{sat}_{\mathcal{A}}(\varphi \wedge (\psi_{|_1} \vee \psi_{|_3}))\} & . \end{array}$$

The LTS isomorphic to P is obtained by applying the labeling function $f = f_{\mathcal{A}}^{\psi_1} \cup f_{\mathcal{B}}^{\psi_2}$. Note that f is defined. In fact, the domain of $f_{\mathcal{A}}^{\psi_1}$ are the formulas of type $\langle \varphi_{|_1}, \bot, \bot \rangle$

or $\langle \bot, \bot, \langle \varphi_{|3}, \varphi_{|4} \rangle \rangle$, whereas the domain of $f_{\mathcal{B}}^{\psi_2}$ are the formulas of type $\langle \bot, \varphi_{|2}, \bot \rangle$ or $\langle \bot, \bot, \langle \varphi_{|3}, \varphi_{|4} \rangle \rangle$ (see above). Moreover, the two functions map the formulas belonging to the intersection of their domains to the same values. The only exception is for $\varphi_{\emptyset}$. Indeed $f_{\mathcal{A}}^{\psi_1}(\varphi_{\emptyset}) = \omega_A \neq \omega_B = f_{\mathcal{B}}^{\psi_2}(\varphi_{\emptyset})$. Nevertheless, we can simply ignore this case as, by construction, $\forall s. s \overset{\omega_A}{\not\rightarrow}$ and $s \overset{\omega_B}{\not\rightarrow}$, that is, none of the transitions of P can occur when $\varphi_{\emptyset}$ is true. Thus the LTS isomorphic to $P = \langle \mathcal{A} \circledast_{\psi_1,\psi_2} \mathcal{B}, \mathcal{Q}, \iota, \Delta$ is $A_P = \langle Q, \Sigma_A \cup \Sigma_B \cup \Gamma, \rightarrow, \iota \rangle$ where

$$\rightarrow = \{ p \xrightarrow{f(\psi)} q \mid \exists (p, \varphi, q) \in \Delta, \psi \in \mathit{Minterms}(P) \text{ s.t. } \mathbf{sat}_{\mathcal{A}}(\varphi \wedge \psi) \}$$

To finish, we must show that, given isomorphic inputs, the two algorithms generate isomorphic outputs. To do that we prove that each step of the algorithms preserves the isomorphism. The first step is to show the mapping between the three transition functions of the symbolic quotienting algorithm, i.e., $\bar{\Delta}_\lambda$, $\bar{\Delta}_B$ and $\bar{\Delta}^*$, and the three cases of quotienting algorithm, see Table 2 line 3.

1. $((q_P, q_M), \psi_M \wedge \psi_{P|_1} \wedge \neg\psi_1, (q'_P, q'_M)) \in \bar{\Delta}_\lambda$ implies $((q_P, q_M), \lambda, (q'_P, q'_M)) \in \rightarrow$.
Since $\mathbf{sat}_{\mathcal{A}}(\psi_M \wedge \psi_{P|_1} \wedge \neg\psi_1)$ there must exist at least one $\widehat{\psi} \in \mathit{Minterms}(P)$ such that (i) both $\mathbf{sat}_{\mathcal{A}}(\widehat{\psi}_{|_1} \wedge \psi_{P|_1} \wedge \neg\psi_1)$ and $\mathbf{sat}_{\mathcal{A}}(\psi_M \wedge \widehat{\psi}_{|_1})$, (ii) $f_{\mathcal{A}}^{\psi_1}(\widehat{\psi}) = a \in \Sigma_A$. By definition, $q_P \xrightarrow{a} q'_P$ and $q_M \xrightarrow{a} q'_M$ are transitions of A_P and A_M, which implies $((q_P, q_M), \lambda, (q'_P, q'_M)) \in \rightarrow$.
2. $((q_P, q_M), \psi_{P|_2} \wedge \neg\psi_2, (q'_P, q_M)) \in \bar{\Delta}_B$ implies $((q_P, q_M), a, (q'_P, q_M)) \in \rightarrow$ (with $a \in \Sigma_B$).
We know that $\mathbf{sat}_{\mathcal{B}}(\psi_{P|_2} \wedge \neg\psi_2)$. Hence, there exists some $\widehat{\psi} \in \mathit{Minterms}(P)$ such that $\mathbf{sat}_{\mathcal{B}}(\widehat{\psi}_{|_2} \wedge \psi_{P|_2} \wedge \neg\psi_2)$. By definition, $f_{\mathcal{B}}^{\psi_2}(\widehat{\psi}) = a \in \Sigma_B$ which implies $q_P \xrightarrow{a} q'_P$ and, thus, $((q_P, q_M), a, (q'_P, q_M)) \in \rightarrow$.
3. $((q_P, q_M), \psi_{P|_4} \wedge \psi_2, (q'_P, q'_M)) \in \bar{\Delta}^*$ implies $((q_P, q_M), a, (q'_P, q'_M)) \in \rightarrow$ (with $a \in \Gamma$).
Since $((q_P, q_M), \psi_{P|_4} \wedge \psi_2, (q'_P, q'_M)) \in \bar{\Delta}^*$ we have that there exists $(q_M, \psi_M, q'_M) \in \Delta_M$ such that $\mathbf{sat}_{\mathcal{A}}(\psi_M \wedge \psi_{P|_3} \wedge \psi_1)$ holds (Table 4, line 5). Thus there is a $\widehat{\psi} \in \mathit{Minterms}(P)$ such that $f_{\mathcal{A}}^{\psi_1}(\widehat{\psi}) = a \in \Gamma$ and $\mathbf{sat}_{\mathcal{A}}(\widehat{\psi}_{|_3} \wedge \psi_M \wedge \psi_{P|_3} \wedge \psi_1)$ and, consequently, $q_M \xrightarrow{a} q'_M$. Moreover, since $a \in \Gamma$ we have that $f_{\mathcal{B}}^{\psi_2}(\widehat{\psi}) = a$ which implies $\mathbf{sat}_{\mathcal{B}}(\widehat{\psi}_{|4} \wedge \psi_{P|_4} \wedge \psi_2)$. From $\mathbf{sat}_{\mathcal{A}}(\widehat{\psi}_{|_3} \wedge \psi_M \wedge \psi_{P|_3} \wedge \psi_1)$ we infer $\mathbf{sat}_{\mathcal{A}}(\widehat{\psi}_{|_3} \wedge \psi_{P|_3} \wedge \psi_1)$ which, together with $\mathbf{sat}_{\mathcal{B}}(\widehat{\psi}_{|4} \wedge \psi_{P|_4} \wedge \psi_2)$, implies that $q_P \xrightarrow{a} q'_P$.

Now we show that the procedures `unify` preserves the isomorphism between the transitions. This requires proving the same property for the procedures $\wedge$`-move` and `close`. The latter is a trivial consequence of the isomorphism between the transitions in $\bar{\Delta}_\lambda$ and the λ transitions proved above. A similar argument applies to $\wedge$`-move`. Indeed we only need to show that (q, φ', q') and $\mathbf{sat}_{\mathcal{B}}(\varphi \wedge \varphi') \in \bar{\Delta}_B \cup \bar{\Delta}^*$ if and only if $q \xrightarrow{a}_B q'$ where $f_{\mathcal{B}}^{\psi_2}(\varphi) = a$. Again, this follows from the transition isomorphism.

To conclude, we observe that there is a plain correspondence between the steps of the two `unify` procedures with the exception of line 6. However, the transition isomorphism provides us with the required correspondence. This holds because $\mathit{Minterms}(P)$ is tripartite in a way that the elements of each partition are mapped to Γ, $\Sigma_B \backslash \Gamma$, and $\Sigma_A \backslash \Gamma$. Thus, the restriction to their second and fourth components limits this mapping to Σ_B (note that ω_B cannot occur on the transitions of B as $\varphi_{\emptyset}$ is never satisfied when in conjunction with any other predicate).

A.4 Complexity

We estimate the worst case complexity of the `quotient` algorithm of Sect. 4. For simplicity, we assume that $|\Gamma| = |\Sigma_A \backslash \Gamma| = |\Sigma_B \backslash \Gamma| = m$ and $|S_A| = |S_P| = n$. The first part, i.e., the generation of the non-deterministic transition system, requires at most $| \rightarrow_P | \cdot | \rightarrow_A | \leq n^4 m^2$ steps (since both P and A have at most $n^2 m$ transitions). The resulting 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 non deterministic transition system B. Similarly to the ε-closures of an NFA, they can be computed in advance (see [24]). 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, ∧`-move` computes the sets of the reachable states with a transition labeled by $a \neq \lambda$, starting from one of the closures (which has size at most n^2). Since B is built from P and A, both deterministic, for each symbol a and pair of states there is at most one transition labeled with a. Thus, having n^2 states and $2m$ symbols in Σ_B, there are no more than $2n^4 m$ Σ_B-transitions. Thus, in $2n^4 m$ we obtain the set on which we compute the λ-closure. Recall that we already computed them, so we just need to select the required one.

To conclude, we observe that ∧`-move` is iterated at most $n^2 m$ times. Indeed, if $q, q' \in \lambda$-`close`$(\{\hat{q}\})$ such that $q \neq q'$ and $q \xrightarrow{a} q'$ (for some $a \in \Sigma_B$) then $q \notin \lambda$-`close`(q'). Therefore, the number of λ-closures stored in S, and thus the algorithm iterations cannot exceed $n^2 \cdot 2m$. Hence, the overall complexity is $O(2n^4 m \cdot n^2 \cdot 2m) = O(n^6 m^2)$.

We already discussed in Sect. 5 the complexity of the symbolic quotienting algorithm.

References

1. Andersen, H.R.: Partial model checking (extended abstract). In: Proceedings of Tenth Annual IEEE Symposium on Logic in Computer Science, pp. 398–407. IEEE Computer Society Press (1995)
2. Andersen, H.R., Lind-Nielsen, J.: MuDiv: A tool for partial model checking. Demo presentation at CONCUR (1996)
3. Andersen, H.R., Lind-Nielsen, J.: Partial model checking of modal equations: a survey. Int. J. Softw. Tools Technol. Transf. **2**(3), 242–259 (1999). https://doi.org/10.1007/s100090050032
4. Arnold, A., Nivat, M.: Comportements de processus. In: Les Mathématiques de l'Informatique, pp. 35–68. Colloque AFCET (1982)
5. Arnold, A., Vincent, A., Walukiewicz, I.: Games for synthesis of controllers with partial observation. Theor. Comput. Sci. **1**(303), 7–34 (2003)
6. Baeten, J.C.M., Luttik, B., Muller, T., Van Tilburg, P.: Expressiveness modulo bisimilarity of regular expressions with parallel composition. Math. Struct. Comput. Sci. **26**, 933–968 (2016)
7. Basu, S., Kumar, R.: Quotient-based approach to control of nondeterministic discrete-event systems with -calculus specification (2006). http://home.eng.iastate.edu/~rkumar/PUBS/acc06-muctrl.pdf
8. Bauer, L., Ligatti, J., Walker, D.: More enforceable security policies. In: Foundations of Computer Security. Copenhagen, Denmark (2002). http://www.ece.cmu.edu/~lbauer/papers/editauto-fcs02.pdf
9. Betts, A., Chong, N., Donaldson, A., Qadeer, S., Thomson, P.: GPUVerify: A Verifier for GPU Kernels. In: Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications, OOPSLA '12, pp. 113–132. ACM, New York, NY, USA (2012). https://doi.org/10.1145/2384616.2384625
10. Bradfield, J., Stirling, C.: Handbook of Modal Logic, Chapter Modal Mu-Calculi, vol. 3. Elsevier, Amsterdam (2006)
11. Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems. Kluwer, Dordrecht (1999)
12. Cassez, F., Laroussinie, F.: Model-checking for hybrid systems by quotienting and constraints solving. In: Emerson, E.A., Sistla, A.P. (eds.) Computer Aided Verification, pp. 373–388. Springer, Berlin (2000)
13. Costa, G., Basin, D., Bodei, C., Degano, P., Galletta, L.: From natural projection to partial model checking and back. In: Beyer, D., Huisman, M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, pp. 344–361. Springer, Cham (2018)

14. Costa, G., Basin, D., Bodei, C., Degano, P., Galletta, L.: Pests: partial evaluator of simple transition systems. GitHub: https://github.com/gabriele-costa/pests. https://doi.org/10.6084/m9.figshare.5918707.v1(2018)
15. D'Antoni, L., Veanes, M.: Monadic second-order logic on finite sequences. In: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, pp. 232–245. ACM, New York, NY, USA (2017). https://doi.org/10.1145/3009837.3009844
16. D'Antoni, L., Veanes, M.: The power of symbolic automata and transducers. In: 29th International Conference on Computer Aided Verification (CAV'17). Springer (2017)
17. Ehlers, R., Lafortune, S., Tripakis, S., Vardi, M.: Bridging the gap between supervisory control and reactive synthesis: case of full observation and centralized control. IFAC Proc. Vol. **47**(2), 222–227 (2014)
18. Feng, L., Wonham, W.M.: TCT: A computation tool for supervisory control synthesis. In: Proceedings of 2006 8th International Workshop on Discrete Event Systems, pp. 388–389 (2006). https://doi.org/10.1109/WODES.2006.382399
19. Feng, L., Wonham, W.M.: On the computation of natural observers in discrete-event systems. Discrete Event Dyn. Syst. **20**(1), 63–102 (2010). https://doi.org/10.1007/s10626-008-0054-3
20. Feuillade, G., Pinchinat, S.: Modal specifications for the control theory of discrete event systems. Discrete Event Dyn. Syst. **17**(2), 211–232 (2007). https://doi.org/10.1007/s10626-006-0008-6
21. Giacobazzi, R., Ranzato, F.: States vs. traces in model checking by abstract interpretation. In: Proceedings of The 9th International Static Analysis Symposium, SAS'02, Lecture Notes in Computer Science, vol. 2477, pp. 461–476. Springer (2002)
22. Gromyko, A., Pistore, M., Traverso, P.: A tool for controller synthesis via symbolic model checking. In: 8th International Workshop on Discrete Event Systems, pp. 475–476 (2006). https://doi.org/10.1109/WODES.2006.382523
23. Groote, J.F., Mousavi, M.R.: Modeling and Analysis of Communicating Systems. MIT Press, Cambridge (2014)
24. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 3rd edn. Addison-Wesley Longman Publishing Co., Inc, Boston (2006)
25. Jiang, S., Kumar, R.: Supervisory control of discrete event systems with ctl* temporal logic specifications. SIAM J. Control Optim. **44**(6), 2079–2103 (2006)
26. Jirásková, G., Masopust, T.: On a structural property in the state complexity of projected regular languages. Theoret. Comput. Sci. **449**, 93–105 (2012). https://doi.org/10.1016/j.tcs.2012.04.009
27. Kozen, D.: Results on the propositional mu-calculus. Theor. Comput. Sci. **27**, 333–354 (1983)
28. Lang, F., Mateescu, R.: Partial Model Checking Using Networks of Labelled Transition Systems and Boolean Equation Systems. Lecture Notes in Computer Science, vol. 7214, pp. 141–156. Springer, New York (2012)
29. Laroussinie, F., Larsen, K.G.: CMC: A Tool for Compositional Model-Checking of Real-Time Systems, pp. 439–456. Springer, Boston (1998). https://doi.org/10.1007/978-0-387-35394-4_27
30. Lin, F., Wonham, W.: Decentralized supervisory control of discrete-event systems. Inf. Sci. **44**(3), 199–224 (1988). https://doi.org/10.1016/0020-0255(88)90002-3
31. Martinelli, F., Matteucci, I.: Synthesis of local controller programs for enforcing global security properties. In: 3rd International Conference on Availability, Reliability and Security (ARES), pp. 1120–1127 (2008). https://doi.org/10.1109/ARES.2008.196
32. Martinelli, F., Matteucci, I.: A framework for automatic generation of security controller. Softw. Test. Verif. Reliab. **22**(8), 563–582 (2012). https://doi.org/10.1002/stvr.441
33. Moor, T., Schmidt, K., Perk, S.: libFAUDES—an open source C++ library for discrete event systems. In: 9th International Workshop on Discrete Event Systems, pp. 125–130 (2008). https://doi.org/10.1109/WODES.2008.4605933
34. Riedweg, S., Pinchinat, S.: Quantified mu-calculus for control synthesis. In: Mathematical Foundations of Computer Science 2003, 28th International Symposium, MFCS 2003 Proceedings, Lecture Notes in Computer Science, vol. 2747, pp. 642–651. Springer (2003)
35. Rudie, K., Grigorov, L.: Integrated Discrete-Event Systems (IDES). https://qshare.queensu.ca/Users01/rudie/www/software.html (2017). Department of Electrical and Computer Engineering, Queen'sUniversity in Kingston, ON, Canada
36. Sharma, R., Bauer, M., Aiken, A.: Verification of producer-consumer synchronization in GPU programs. In: Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15–17, 2015, pp. 88–98. ACM (2015). https://doi.org/10.1145/2737924.2737962
37. Su, R., Wonham, W.M.: Global and local consistencies in distributed fault diagnosis for discrete-event systems. IEEE Trans. Autom. Control **50**(12), 1923–1935 (2005). https://doi.org/10.1109/TAC.2005.860291

38. Veanes, M.: Applications of Symbolic Finite Automata. In: CIAA'13, *LNCS*, vol. 7982, pp. 16–23. Springer (2013). https://www.microsoft.com/en-us/research/publication/applications-of-symbolic-finite-automata/
39. Wong, K.C.: On the complexity of projections of discrete-event systems. In: Proceedings of IEEE Workshop on Discrete Event Systems, pp. 201–208 (1998)
40. Wonham, W.M.: Supervisory control of discrete-event systems. http://www.control.toronto.edu/DES (2017). Department of Electrical and Computer Engineering, University of Toronto, ON, Canada
41. Wonham, W.M., Ramadge, P.J.: On the supremal controllable sublanguage of a given language. In: Proceedings of the 23rd IEEE Conference on Decision and Control, pp. 1073–1080 (1984). https://doi.org/10.1109/CDC.1984.272178
42. Wonham, W.M., Ramadge, P.J.: Modular supervisory control of discrete-event systems. Math. Control Signals Syst. **1**(1), 13–30 (1988). https://doi.org/10.1007/BF02551233
43. Ziller, R., Schneider, K.: Combining supervisor synthesis and model checking. ACM Trans. Embed. Comput. Syst. **4**(2), 331–362 (2005)