The Quotient in Preorder Theories

Inigo Incer
Leonardo Mangeruca
Tiziano Villa
Alberto L. Sangiovanni-Vincentelli

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2020-179

http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-179.html

September 1, 2020

The Quotient in Preorder Theories

Íñigo X. Íncer Romeo
University of California, Berkeley, USA
inigo@eecs.berkeley.edu

Leonardo Mangeruca
Raytheon Technologies Research Center, Rome, Italy
leonardo.mangeruca@rtx.com

Tiziano Villa
Università di Verona, Italy
tiziano.villa@univr.it

Alberto Sangiovanni-Vincentelli
University of California, Berkeley, USA
alberto@eecs.berkeley.edu

Seeking the largest solution to an expression of the form $Ax \leq B$ is a common task in several domains of engineering and computer science. This largest solution is commonly called quotient. Across domains, the meanings of the binary operation and the preorder are quite different, yet the syntax for computing the largest solution is remarkably similar. This paper is about finding a common framework to reason about quotients. We only assume we operate on a preorder endowed with an abstract monotonic multiplication and an involution. We provide a condition, called admissibility, which guarantees the existence of the quotient, and which yields its closed form. We call preordered heaps those structures satisfying the admissibility condition. We show that many existing theories in computer science are preordered heaps, and we are thus able to derive a quotient for them, subsuming existing solutions when available in the literature. We introduce the concept of sieved heaps to deal with structures which are given over multiple domains of definition. We show that sieved heaps also have well-defined quotients.

1 Introduction

The identification of missing objects is a common task in engineering. Suppose an engineer wishes to implement a design with a mathematical description B, and will use a component with a description A to implement this design. In order to find out what needs to be added to A in order to implement B, the engineer seeks a component x in an expression of the form $A \bullet x = B$, where $\bullet$ is an operator yielding the composite of two design elements. Many compositional theories include the notion of a preorder, usually called refinement. The statement $A \leq C$ usually reads "A refines C" or "A is more specific than C." In this setting, the problem is recast as finding an x such that $A \bullet x \leq B$. It is often assumed that the composition operation is monotonic with respect to this preorder. Therefore, if x is a solution, so is any y satisfying $y \leq x$. This focuses our attention on finding the largest x that satisfies the expression. The literature often calls this largest solution *quotient*.

1.1 Background

The logic synthesis community has been a pioneer in defining and solving special cases of the quotient problem for combinational and sequential logic circuit design ([24, 12]) under names like circuit rectification or engineering change or component replacement. In combinational synthesis, much work has been reported to support algebraic and Boolean division: given dividend f and divisor g, find the quotient q and remainder r such $f = q \cdot g + r$ (for $\cdot, +$ standard Boolean operators AND and OR, respectively), as key operation to restructure multi-level Boolean networks [17]. The quotient problem for combinational circuits was formulated as a general replacement problem in [9]: given the combinational circuits A and

C whose synchronous composition produces the circuit specification B, what are the legal replacements of C that are consistent with the input-output relation of B? The valid replacements for C were defined as the combinational circuits x such that $A \circ x \subseteq B$, and the largest solution for x was characterized by the closed formula $x = \left(A \circ B^{\perp}\right)^{\perp}$, where $(\cdot)^{\perp}$ is a unary operator that complements the input-output relation of the circuit to which it is applied (switching the inputs and outputs), while a hiding operation gets rid of the internal signals.

In sequential optimization, the typical question addressed was, given a finite-state machine (FSM) A, find an FSM x such that their synchronous composition produces an FSM behaviorally equivalent to a specification FSM B, i.e., solve over FSMs the equation $A \circ x = B$, where $\circ$ is synchronous composition and equality is FSM input-output equivalence. Various topologies were solved, starting with serial composition where the unknown was either the head or tail machine, to more complex interconnections with feedback. As a matter of fact, sometimes both A and x were known, but the goal was to change them into FSMs yielding better logical implementations, while preserving their composition, with the objective to optimize a sequential circuit by computing and exploiting the flexibility due to its modular structure and its environment (see [17, 38, 21]). An alternative formulation of FSM network synthesis was provided by encoding the problem in the logic WS1S (Weak Second-Order Logic of 1 Successor), which enables to characterize the set of permissible behaviors at a node of a given network of FSMs by WS1S formulas [1], corresponding to regular languages and so to effective operations on finite state automata. [1]

Another stream of contributions has been motivated by component-based design of parallel systems with an interleaving semantics (denoted in our exposition by the composition operator $\diamond$). The problem is stated by Merlin and Bochmann [31] as follows: “Given a complete specification of a given module and the specifications of some submodules, the method described below provides the specification of an additional submodule that, together with the other submodules, will provide a system that satisfies the specification of the given module.” The problem was reduced to solving equations or inequalities over process languages, which are usually prefix-closed regular languages represented by labeled transition systems. A closed-form solution of the inequality $A \diamond x \subseteq B$ over prefix-closed regular languages, written as $proj_x(A \diamond B) - proj_x(A \diamond \overline{B})$ (where $proj_x$ is a projection over the alphabet of x), was given in [31, 19].[2] This approach to solve the equation $A \diamond x = B$ has been further extended to obtain restricted solutions that satisfy properties such as safety and liveness, or are restricted to be FSM languages, which need to be input-progressive and avoid divergence (see [19, 7, 40]). The quotient problem has been investigated also for delay-insensitive processes to model asynchronous sequential circuits, see [13, 30, 32]. Equations of the form $A \diamond x \leq B$ were defined, and their largest closed-form solutions were written as $x = (A \diamond B^{\sim})^{\sim}$, where $(\cdot)^{\sim}$ is a suitable unary operation.

An important application from discrete control theory is the model matching problem: design a controller whose composition with a plant matches a given specification (see [2, 16]). Another significant application of the quotient computation has been the protocol design problem, and in particular, the protocol conversion problem (see [27, 18, 35, 33, 25, 20, 41, 11]). Protocol converter synthesis has been studied also over a variant of Input/Output Automata (IOA, [29]), called Interface Automata (IA, [15, 14]), yielding a similar quotient equation $A \diamond_{IA} x \subseteq B$ and closed-form solution $\left(A \diamond_{IA} B^{\perp}\right)^{\perp}$, where $\diamond_{IA}$ is an appropriate interleaving composition defined for interface automata, and $(\cdot)^{\perp}$ is again a unary operation [6].

Some research focused on modal specifications represented by automata whose transitions are typed

[1] A detailed survey of previous work in this area can be found in [23, 40].

[2] For a discussion about the maximality of this solution and for more references, we refer to [40], Sec. 5.2.1.

with *may* and *must* modalities, as in [28, 36], with a solution of the quotient problem for nondeterministic automata provided in [3]. It is outside the scope of this paper to address the quotient problem for real-time and hybrid systems (see [10, 8] for verification and control in such settings).

As seen above, the quotient problem was studied by different research communities working on various application domains and formalisms. Often similar formulations and solutions were reached albeit obfuscated by the different notations and objectives of the synthesis process. This motivated a concentrated effort to distill the core of the problem, modeling it as solving equations over languages of the form $A \parallel x \preceq B$, where A and B are known components and x is unknown, $\parallel$ is a composition operator, and $\preceq$ is a conformance relation (see [39] and the monograph [40] for full accounts). The notion of language was chosen as the most basic formalism to specify the components of the equation, and language containment $\subseteq$ was selected as conformance relation. Two basic composition operators were defined each encapsulating a family of variants: synchronous composition ($\bullet$) modeling the classical step-lock coordination, and interleaving composition ($\diamond$) modeling asynchrony by which components may progress at different rates (there are subtle issues in comparing the two types, as mentioned in [26, 42]). Therefore two language equations were defined: $A \bullet x \subseteq B$ and $A \diamond x \subseteq B$, where the details of the operations to convert alphabets according to the interconnection topologies are hidden in the formula. It turned out that the largest solutions have the same structure, respectively, $\overline{A \bullet \overline{B}}$ and $\overline{A \diamond \overline{B}}$. This led to investigate the algebraic properties required by the composition operators to deliver the previous largest closed-form solutions to unify the two formulas [39]. This effort assumed that the underlying objects were sets, and that their operations were given in terms of set operations. This work, thus, could not account for quotient computations in more complex theories, like interface automata.

As a parallel development, in recent years we have seen the growth of a rigorous theory of system design based on the algebra of contracts (see the monograph [5]). In this theory, a strategic role is played by assume-guarantee (AG) contracts, in which the *missing component problem* arises: when the given components are not capable of discharging the obligations of the requirements, define a quotient operation that computes the contract for a component, so that by its addition to the original set the resulting system fulfills the requirements. The quotient of AG contracts was completely characterized very recently by a closed-form solution proved in [37]. Once again, the syntax of the quotient has the form $\left(A \parallel B^{-1}\right)^{-1}$ for contracts A and B and standard contract operations.

In summary, even though the concrete models of the components, composition operators, conformance relations and inversion functions vary significantly across chosen models and application domains, the quotient formulas have similar syntax across theories.

1.2 Motivation and contributions

The motivation of this paper is to *propose the underlying mathematical structure common to all these instances of quotient computation to be able to derive directly the solution formula for any equation satisfying the properties of this common structure.*

We show that we can compute the quotient by only assuming the axioms of a *preorder*, enriched with a binary operation of *source multiplication* and a unary *involution* operation. In particular we introduce the new algebraic notion of *preordered heaps* characterized by a condition, called *admissibility*, which guarantees the existence of the solution and yields a closed form for it. Then we show that a number of theories in computer science meet this condition, e.g., Boolean lattices, AG contracts, and interface automata; so for all of them we are able to (re-)derive axiomatically the formulas that compute their related quotients. We also introduce the concept of *sieved heaps* to deal with structures defined over multiple domains, and we show that the equations $A \bullet x \leq B$ admit a solution also over sieved heaps, generalizing the known solutions of equations on languages over multiple alphabets with respect to synchronous and interleaving composition, well studied in the literature.

1.3 Organization

The paper is structured as follows. Sec. 2 develops the basic mathematical machinery of preordered heaps, whereas Sec. 3 shows that various theories are preordered heaps. Sec. 4 introduces sieved heaps, whereas Sec. 5 applies them to equations over languages with multiple alphabets. Sec. 6 concludes. Proofs omitted in the main text can be found in Appendix A.

2 Preordered heaps

In this section we introduce an algebraic structure for which the existence of quotients is guaranteed. We show in Section 3 that many theories in computer science are instances of this concept. First we introduce the notation we will use:

- Let P be a set and let $\mu \colon P \times P \to P$ be a binary operation on P. For any element $a \in P$, we let $\mu_a \colon P \to P$ be the function $\mu_a = \mu \circ (a \times \mathrm{id})$, where id is the identity operator and $(a \times \mathrm{id}) \colon P \to P^2$ is the unary function $(a \times \mathrm{id}) \colon b \mapsto (a,b)$. Similarly, we let $\mu^a = \mu \circ (\mathrm{id} \times a)$. If we call μ multiplication, μ_a is left multiplication by a, and μ^a is right multiplication by a.
- For any set P, we let the mapping flip: $P \times P \to P \times P$ be $\mathrm{flip}(a,b) = (b,a)$ $(a,b \in P)$.
- Consider a set P and a binary relation $\leq$ on P. Then $\leq$ is a preorder if it is reflexive and transitive; i.e., for all a,b and c in P, we have $a \leq a$ (reflexivity) and if $a \leq b$ and $b \leq c$ then $a \leq c$ (transitivity). If a preorder is antisymmetric, ($a \leq b$ and $b \leq a$ implies $a = b$), then it is a partial order.
- Let $(P, \leq)$ be a preorder and let $a,b \in P$. If $a \leq b$ and $b \leq a$, we write $a \simeq b$.
- Let $F \colon P \to P$. We say that F is monotonic or order-preserving if $a \leq b \Rightarrow Fa \leq Fb$ for all $a,b \in P$. Similarly, we say that F is antitone or order-reversing if $a \leq b \Rightarrow Fb \leq Fa$ for all $a,b \in P$.
- Suppose that $L,R \colon P \to P$ are two monotonic maps on P. We say that (L,R) form an adjoint pair, or that L is the left adjoint of R (R is respectively the right adjoint of L), or that the pair (L,R) forms a Galois connection when for all $b,c \in P$, we have $Lb \leq c$ if and only if $b \leq Rc$.
- Let $F,G \colon P \to P$ be functions on a preorder P. We say that $F \leq G$ when $Fa \leq Ga$ for all $a \in P$.

2.1 The concept of preordered heap

As we discussed in the introduction, many times in engineering and computer science one encounters expressions of the form $A \bullet x \leq B$, and one wishes to solve for the largest x that satisfies the expression. The symbols have different specific meanings in the various domains, yet in all applications we know, the syntax for computing the quotient always has the form $\overline{A \bullet \overline{B}}$, where $\overline{(\cdot)}$ is an involution (i.e., a unary operator which is its own inverse). To give meaning to the inequality, at a minimum we need a preorder and a binary operation; to give meaning to the quotient expression, we need to assume the existence of an involution. In all compositional theories, the refinement order has the connotation of specificity: if $a \leq b$ then a is a refinement of b. The binary operation is usually interpreted as composition. The product $a \bullet b$ is understood as the design obtained when operating both a and b in a topology given by the mathematical description of each component. The unary operation is sometimes understood as giving an external view on an object. If a component has mathematical description a, then $\overline{a}$ gives the view that the environment has of the design element. In Boolean algebras, this unary operation is negation. In interface theories, it's usually an operation which switches inputs and output behaviors.

We thus introduce an algebraic structure consisting of a preorder, a binary operation which is monotonic in both arguments, and an involution which is antitone. We have called the binary operation *source multiplication* for reasons having to do with category theory: we will show that this operation serves as the left functor of an adjunction. Therefore, its application to an object of the preorder yields the *source* of one of the two arrows in the adjunction. Why not simply call it multiplication? Because source multiplication together with the involution generate another binary operation. This second operation we call *target multiplication* because its application to an object yields the *target* of one of the arrows in the adjunction. The unary operation will simply be called *involution*.

The algebraic structure will be called *preordered heap*. The inspiration came from engineering design. In some design methodologies, design elements at the same level of abstraction are not comparable in the refinement order. Indeed, a refinement of a design element usually yields a design element in a more concrete layer. But we are placing all components under the same mathematical structure. This suggested the name *heap*. We add the adjective *preorder* simply to differentiate the concept from existing algebraic heaps. We are ready for the definition:

Definition 2.1. *A* preordered heap *is a structure $(P, \leq, \mu, \gamma)$, where $(P, \leq)$ is a preorder; $\mu\colon P \times P \to P$ is a binary operation on P, monotonic in both arguments, called* source multiplication*; and $\gamma\colon P \to P$ is an antitone operation on P called* involution. *These operations satisfy the following axioms:*

- *A1: $\gamma^2 = id$.*
- *A2a (left admissibility): $\mu_a \circ \gamma \circ \mu^a \circ \gamma \leq id \qquad (a \in P)$.*
- *A2b (right admissibility): $\mu^a \circ \gamma \circ \mu_a \circ \gamma \leq id \qquad (a \in P)$.*

Note 2.1. *In Definition 2.1, we did not assume commutativity in μ. If μ is commutative, we have $\mu = \mu \circ flip$, so $\mu_a = \mu \circ (a \times id) = \mu \circ flip \circ (a \times id) = \mu \circ (id \times a) = \mu^a$. It follows that for a commutative preordered heap, axioms A2a and A2b become*

$$(\mu_a \circ \gamma)^2 \leq id. \tag{1}$$

We have discussed all elements in the definition of a preordered heap, except for the admissibility conditions. What are they? Consider left admissibility: $\mu_a \circ \gamma \circ \mu^a \circ \gamma \leq$ id. Let $b \in P$ and set $B = (\gamma \circ \mu^a \circ \gamma)(b)$. Left admissibility means that B satisfies the expression $\mu(a,x) \leq b$. Similarly, set $C = (\gamma \circ \mu_a \circ \gamma)(b)$. Right admissibility means that C satisfies $\mu(x,a) \leq b$. When μ is commutative, we of course have $B = C$. We will soon show a surprising fact: the axioms of a preordered heap are sufficient to guarantee that B and C are in fact the largest solutions to both expressions, i.e., B and C are the quotients for left and right source multiplication, respectively. We show this immediately after introducing an important binary operation called target multiplication, but first we consider an example.

Example. Consider a Boolean lattice B. The lattice is clearly a preorder. Take the involution to be the negation operator. This is an antitone operator and satisfies A1: $\neg\neg b = b$ for all $b \in B$. Take source multiplication to be the meet of the lattice (i.e., logical AND). This operation is monotonic in the preorder. Since this source multiplication is commutative, the admissibility conditions reduce to checking (1). For $a, b \in B$, we have $(\mu_a \circ \gamma)^2 b = a \wedge \neg(a \wedge \neg b) = a \wedge (\neg a \vee b) = a \wedge b \leq b$. Thus, the Boolean lattice satisfies the admissibility conditions, making it a preordered heap.

2.2 Target multiplication

For the rest of this section, let $(P,\leq,\mu,\gamma)$ be a preordered heap. We define the *target multiplication* $\tau\colon P\times P\to P$ as $\tau=\gamma\circ\mu\circ(\gamma\times\gamma)$. Since $\gamma^2=\mathrm{id}$ (axiom A1), we can also write $\mu=\gamma\circ\tau\circ(\gamma\times\gamma)$, i.e., the diagram

$$\begin{tikzcd} P\times P \arrow[r, "\mu"] \arrow[d, leftrightarrow, "\gamma\times\gamma"'] & P \arrow[d, leftrightarrow, "\gamma"] \\ P\times P \arrow[r, "\tau"] & P \end{tikzcd}$$

commutes.

We could have defined a preordered heap in terms of target multiplication instead of source multiplication. The two operations are closely linked. In fact, we will see in the next section that these operations form an adjoint pair.

Example. We showed that Boolean lattices are preordered heaps. For B a Boolean lattice and $a,b\in B$, we have $\tau(a,b)=\gamma\circ\mu(\gamma a,\gamma b)=\neg(\neg a\wedge\neg b)=a\vee b$. This suggests that the relation between source and target multiplications is a generalization of De Morgan's identities for Boolean algebras.

We will use the following identities: for $a\in P$,

$$\begin{aligned} \mu_a&=\gamma\circ\tau\circ(\gamma\times\gamma)\circ(a\times\mathrm{id})=\gamma\circ\tau\circ(\gamma a\times\mathrm{id})\circ\gamma=\gamma\circ\tau_{\gamma a}\circ\gamma \qquad \text{and}\\ \mu^a&=\gamma\circ\tau\circ(\gamma\times\gamma)\circ(\mathrm{id}\times a)=\gamma\circ\tau\circ(\mathrm{id}\times\gamma a)\circ\gamma=\gamma\circ\tau^{\gamma a}\circ\gamma. \end{aligned} \tag{2}$$

2.3 Solving inequalities in preordered heaps

For $a,b\in P$, we are interested in the conditions under which we can find the largest $x\in P$ such that $\mu(a,x)\leq b$. The following theorem says that source multiplication in a preordered heap is "invertible."

Theorem 2.2. *Let $(P,\leq,\mu,\gamma)$ be a preordered heap and let τ be its target multiplication. Then for $a\in P$, $(\mu_a,\tau^{\gamma a})$ and $(\mu^a,\tau_{\gamma a})$ are adjoint pairs.*

Proof. Let $b,c\in P$ with $b\leq\tau^{\gamma a}(c)$. We have $\mu_a(b)\leq(\mu_a\circ\tau^{\gamma a})(c)=(\mu_a\circ\gamma\circ\mu^a\circ\gamma)(c)\leq c$, by left admissibility (by A2a).

Conversely, assume that $\mu_a(b)\leq c$. Then

$$\begin{aligned} \mu_a\circ\gamma^2(b)&\leq c && \text{(by A1)}\\ \gamma\circ(\mu_a\circ\gamma)(\gamma b)&\geq\gamma(c)\\ (\mu^a\circ\gamma)\circ(\mu_a\circ\gamma)(\gamma b)&\geq(\mu^a\circ\gamma)(c)\\ (\gamma b)&\geq(\mu^a\circ\gamma)(c) && \text{(by A2b)}\\ b&\leq(\gamma\circ\mu^a\circ\gamma)(c)=\tau^{\gamma a}(c). && \text{(by A1)} \end{aligned}$$

The adjointness of $(\mu^a,\tau_{\gamma a})$ follows from a similar reasoning. □

The fact that $(\mu_a,\tau^{\gamma a})$ is an adjoint pair means that left source multiplication by a is "inverted" by right target multiplication by γa, i.e.,

$$\mu(a,x)\leq b \quad \text{if and only if} \quad x\leq\tau(b,\gamma a).$$

In other words, the largest solution of $\mu(a,x)\leq b$ is $x=\tau(b,\gamma a)$. Using the familiar multiplicative notation for source multiplication, and $(\cdot)/a=\tau^{\gamma a}$ for "right division by a," we have shown that the largest solution of $ax\leq b$ is $x=b/a$. Calling $a\backslash(\cdot)=\tau_{\gamma a}$ "left division by a," we have shown that the largest solution of $xa\leq b$ is $x=a\backslash b$. These two divisions are related as follows:

Corollary 2.3 (Isolating the unknown)**.** *Let P be a preordered heap and $a,x,y\in P$. Then $y\leq a/x$ if and only if $x\leq y\backslash a$.*

Proof. By two applications of Theorem 2.2, we obtain $y \leq a/x = \tau^{\gamma x}(a) \Leftrightarrow \mu(x,y) \leq a \Leftrightarrow x \leq \tau_{\gamma y}(a) = y \backslash a$. □

Theorem 2.2 is our main result. It shows that preordered heaps have sufficient structure for the computation of quotients. When we prove that a structure is a preordered heap, this theorem immediately yields the existence of an adjoint for multiplication, and its closed form.

In general, to show that a theory is a preordered heap, we must identify its involution and source multiplication. Then we have to verify the admissibility conditions. How difficult is that? Our original problem was identifying the largest x satisfying $\mu(a,x) \leq b$ for some notion of multiplication μ, involution γ, and preorder $\leq$. As we discussed, left admissibility requires that $\tau^{\gamma a}b$ satisfies the inequality $\mu(a,x) \leq b$, and right admissibility requires that $\tau_{\gamma a}b$ satisfies $\mu(x,a) \leq b$. What the theorem tells us is that they are *the largest solutions* to $\mu(a,x) \leq b$ and $\mu(x,a) \leq b$, respectively. In other words, the theorem saves us the effort of making an argument for the optimality of the solutions.

Theorem 2.2 also suggests the following observation. For a given $a \in P$, we have adjoint pairs $(\mu_a, \tau^{\gamma a})$ and $(\mu^a, \tau_{\gamma a})$. As we noticed, this means we can find the largest x such that $\mu(a,x) \leq b$ or $\mu(x,a) \leq b$. But it also means that we can find *the smallest* x such that $b \leq \tau(a,x)$ or $b \leq \tau(x,a)$. This is because, $\mu_{\gamma a}$ is the left adjoint of τ^a, and $\mu^{\gamma a}$ is the left adjoint of τ_a. For all examples we will discuss, source multiplication plays the role of the usual composition operation of the theory. But preordered heaps make it clear that μ and τ are closely related operations. In fact, preordered heaps generalize De Morgan's identities (see section 2.2). Thus, while inequalities of the form $\mu(a,x) \leq b$ are more common in the literature, preordered heaps indicate that we can also solve inequalities of the form $b \leq \tau(a,x)$. As we will see, for some theories there is clear understanding of how target multiplication can be used, but for others its use is unknown.

Example. In the case of a Boolean lattice B, what is the quotient? We showed in previous examples that B is a preordered heap, and we identified its target multiplication. For $a, b \in B$, we can write an expression of the form $\mu(a,x) \leq b$. By Theorem 2.2, we know the largest x that satisfies this expression is $\tau^{\gamma a}b = \tau(b, \neg a) = b \vee \neg a$, i.e., the quotient is the implication $a \rightarrow b$.

2.4 Preordered heaps with identity

In the definition of a preordered heap, we did not assume that source multiplication has an identity. Here we consider briefly what happens when it does. Multiplicative identities are common, and in fact, there exists a multiplicative identity in all compositional theories we know.

Suppose P is a preordered heap and $e \in P$ is a left identity for source multiplication, i.e., $\mu_e \simeq \text{id}$. By Theorem 2.2, $(\text{id}, \tau^{\gamma e})$ is an adjoint pair. The right adjoint of id is id. Since adjoints are unique up to isomorphism, $\tau^{\gamma e} \simeq \text{id}$. This means that γe is a right identity element for τ. Moreover, in view of (2), $\tau_{\gamma e} \simeq \text{id}$. By Theorem 2.2, (μ^e, id) is an adjoint pair. By the same reasoning just followed, we must have $\mu^e \simeq \text{id}$. We record this result:

Corollary 2.4. *Let $(P, \leq, \mu, \gamma)$ be a preordered heap. If $e \in P$ is a left (or right) identity for source multiplication, it is a double-sided identity for source multiplication, and γe is a double-sided identity for target multiplication. Analogously, if $e \in P$ is a left (or right) identity for target multiplication, it is a double-sided identity for target multiplication, and γe is a double-sided identity for source multiplication.*

Example. Let B be a Boolean lattice. The top element of the lattice, usually denoted 1, is an identity for source multiplication: $1 \wedge a = a$ for all $a \in B$. The previous corollary tells us that $\neg 1 = 0$ is a double sided identity for target multiplication, which we identified to be disjunction.

3 Additional instances of preordered heaps

As described in Section 2, as soon as we verify that a theory is a preordered heap, we know how to compute quotients for that theory. Here we show that assume-guarantee (AG) contracts and interface automata are preordered heaps. In both cases, we first define the algebraic aspects of the theory, and then we proceed to show that it is a preordered heap, which involves verifying the axioms of Definition 2.1. After we do this, we invoke Theorem 2.2 to express its quotient in closed-form. The literature for both theories is large, and we only discuss them algebraically. To learn about their uses and the design methodologies based on them, we suggest [5] and [15].

3.1 AG contracts

Assume-guarantee contracts are an algebra and a methodology to support compositional system design and analysis. Fix once and for all a set B whose elements we call behaviors. Subsets of B are referred to as behavioral properties or trace properties. An AG contract is a pair of properties $C = (A, G)$ satisfying $A \cup G = B$. Contracts are used as specifications: a component adheres to contract C if it meets the guarantees G when instantiated in an environment that satisfies the assumptions A. The specific form of these properties is not our concern now; we are only interested in the algebraic definitions. The algebra of assume-guarantee contracts was introduced by R. Negulescu [32] (there called *process spaces*) to deal with assume-guarantee reasoning for concurrent programs. The algebra was reintroduced, together with a methodology for system design, by Benveniste et al. [4] to apply assume-guarantee reasoning to the design and analysis of any engineered system. Now we describe the operations of this algebra.

For $C' = (A', G')$ another contract, the partial order of AG contracts, called *refinement*, is given by $C \leq C'$ when $G \subseteq G'$ and $A \supseteq A'$. The involution of AG contracts, called reciprocal, is given by $\gamma C = (G, A)$. This operation is clearly antitone and meets axiom A1. Source multiplication is contract composition: $\mu(C, C') = (A \cap A' \cup \neg(G \cap G'), G \cap G')$. This operation yields the tightest contract obeyed by the composition of two design elements, each obeying contracts C and C', respectively. Composition is monotonic in the refinement order of AG contracts. We need to verify the admissibility conditions. Since source multiplication for AG contracts is commutative, we verify (1):

$$\begin{aligned}
(\mu_C \circ \gamma)^2 C' &= (\mu_C \circ \gamma) \circ (\mu_C)(G', A') = \mu_C(G \cap A', A \cap G' \cup \neg(G \cap A')) \\
&= (A \cap G \cap A' \cup \neg G \cup \neg(A \cap G' \cup \neg A'), G \cap (A \cap G' \cup \neg A')) \\
&= (A \cap A' \cup \neg G \cup \neg A \cap A' \cup \neg G' \cap A', G \cap (A \cap G' \cup \neg A')) \\
&= (A' \cup \neg G, G \cap (A \cap G' \cup \neg A')) \leq (A', G') = C',
\end{aligned}$$

where in the last step we used the fact that $\neg A' \subseteq G'$, which follows from $A' \cup G' = B$. We conclude that AG contracts satisfy the admissibility conditions, and thus have preordered heap structure.

What is target multiplication for AG contracts? From its definition, we have $\tau(C, C') = \gamma \circ \mu \circ (\gamma C, \gamma C') = \gamma \circ \mu\left((G, A), (G', A')\right) = (A \cap A', G \cap G' \cup \neg(A \cap A'))$. This is an operation on contracts called *merging*. One of the main objectives of the theory of assume-guarantee contracts is to deal with *multiple viewpoints*, i.e., a multiplicity of design concerns, each having a contract representing the specification for that concern (e.g., functionality, timing, etc.). In [34], it is argued that the operation of merging is used to bring multiple viewpoint specifications into a single contract object.

Since AG contracts are preordered heaps, we get their quotient formulas from Theorem 2.2. The adjoint of $\mu_{C'}$ is $\tau^{\gamma C'} = \gamma \circ \mu^{C'} \circ \gamma$. Applying this to C yields $\tau^{\gamma C'}(C) = \gamma \circ \mu^{C'}(G, A) = (A \cap G', G \cap A' \cup \neg(A \cap G'))$. This closed-form expression for the quotient of AG contracts was first reported in [37].

Also by Theorem 2.2, the left adjoint of merging by a fixed contract C' is the operation $\mu(C,\gamma C') = \mu((A,G),(G',A')) = (A \cap G' \cup \neg(G \cap A'), G \cap A')$. This operation was recently introduced under the name of *separation* in [34].

3.2 Interface automata

We show that Interface Automata as introduced in [15] have preordered heap structure. To achieve this result, we first provide the relevant definitions for interface automata. All definitions match those of [15], except for our definition of alternating simulation for interface automata.

An interface automaton $P = \langle V_P, V_P^{\text{init}}, \mathscr{A}_P^I, \mathscr{A}_P^O, \mathscr{A}_P^H, \mathscr{T}_P \rangle$ consists of the following elements:

- V_P is a set of states.
- $V_P^{\text{init}} \subseteq V_P$ is a set of initial states. Following [15], we require that V_P^{init} contains at most one state.
- $\mathscr{A}_P^I, \mathscr{A}_P^O$, and $\mathscr{A}_P^H$ are mutually disjoint sets of input, output, and internal actions. We denote by $\mathscr{A}_P = \mathscr{A}_P^I \cup \mathscr{A}_P^O \cup \mathscr{A}_P^H$ the set of all actions.
- $\mathscr{T}_P \subseteq V_P \times \mathscr{A}_P \times V_P$ is a set of steps.

Following [15], if $a \in \mathscr{A}_P^I$ (resp. $a \in \mathscr{A}_P^O$, $a \in \mathscr{A}_P^H$), then (v,a,v') is called an input (resp. output, internal) step. We denote by $\mathscr{T}_P^I$ (resp. $\mathscr{T}_P^O$, $\mathscr{T}_P^H$) the set of input (resp. output, internal) steps. An action $a \in \mathscr{A}_P$ is enabled at a state $v \in V_P$ if there is a step $(v,a,v') \in \mathscr{T}_P$ for some $v' \in V_P$. We indicate by $\mathscr{A}_P^I(v), \mathscr{A}_P^O(v), \mathscr{A}_P^H(v)$ the subsets of input, output, and internal actions that are enabled at the state v, and we let $\mathscr{A}_P(v) = \mathscr{A}_P^I(v) \cup \mathscr{A}_P^O(v) \cup \mathscr{A}_P^H(v)$.

Definition 3.1. *If P and Q are interface automata, let $\mathsf{shared}(P,Q) = (\mathscr{A}_P^I \cap \mathscr{A}_Q^O) \cup (\mathscr{A}_P^O \cap \mathscr{A}_Q^I)$. The product $P \otimes Q$ is the interface automaton with the following constituents: $V_{P\otimes Q} = V_P \times V_Q$, $V_{P\otimes Q}^{init} = V_P^{init} \times V_Q^{init}$, $\mathscr{A}_{P\otimes Q}^I = (\mathscr{A}_P^I \cup \mathscr{A}_Q^I) - \mathsf{shared}(P,Q)$, $\mathscr{A}_{P\otimes Q}^O = (\mathscr{A}_P^O \cup \mathscr{A}_Q^O) - \mathsf{shared}(P,Q)$, $\mathscr{A}_{P\otimes Q}^H = \mathscr{A}_P^H \cup \mathscr{A}_Q^H \cup \mathsf{shared}(P,Q) - (\mathscr{A}_{P\otimes Q}^I \cup \mathscr{A}_{P\otimes Q}^O)$, and*

$$\begin{aligned}
\mathscr{T}_{P\otimes Q} = &\left\{((v,u),a,(v',u)) \;\middle|\; (v,a,v') \in \mathscr{T}_P \wedge a \in \mathscr{A}_P - \mathscr{A}_Q \wedge u \in V_Q\right\} \\
\cup &\left\{((v,u),a,(v,u')) \;\middle|\; (u,a,u') \in \mathscr{T}_Q \wedge a \in \mathscr{A}_Q - \mathscr{A}_P \wedge v \in V_P\right\} \\
\cup &\left\{((v,u),a,(v',u')) \;\middle|\; (v,a,v') \in \mathscr{T}_P \wedge (u,a,u') \in \mathscr{T}_Q \wedge a \in \mathscr{A}_P \cap \mathscr{A}_Q\right\}.
\end{aligned}$$

We call illegal those states of the product in which one of the interface automata can take a step through a shared action, but the other can't. These states are removed from the product in the definition of composition of interface automata. Given two composable interface automata P and Q, the set $\mathsf{Illegal}(P,Q) \subseteq V_P \times V_Q$ of illegal states of $P \otimes Q$ is given by

$$\mathsf{Illegal}(P,Q) = \left\{ (v,u) \in V_P \times V_Q \;\middle|\; \exists a \in \mathsf{shared}(P,Q). \begin{pmatrix} a \in \mathscr{A}_P^O(v) \wedge a \notin \mathscr{A}_Q^I(u) \\ \vee \\ a \in \mathscr{A}_Q^O(u) \wedge a \notin \mathscr{A}_P^I(v) \end{pmatrix} \right\}.$$

An environment for an interface automaton R is an interface automaton E such that E is composable with R, E is nonempty, $\mathscr{A}_E^I = \mathscr{A}_R^O$, and $\mathsf{Illegal}(R,E) = \emptyset$. A legal environment for the pair (P,Q) is an environment for $P \otimes Q$ such that no state in $\mathsf{Illegal}(P,Q) \times V_E$ is reachable in $(P \otimes Q) \otimes E$. We say that a pair $(v,u) \in V_P \times V_Q$ of states is compatible if there is an environment E for $P \otimes Q$ such that no state in $\mathsf{Illegal}(P,Q) \times V_E$ is reachable in $(P \otimes Q) \otimes E$ from the state $\{(v,u)\} \times V_E^{\text{init}}$. Two interface automata P and Q are compatible if the initial state $(v,u) \in V_P^{init} \times V_Q^{init}$ is compatible. We write $\mathsf{Cmp}(P,Q)$ for the set of compatible states of $P \otimes Q$. With these notions, we can define parallel composition for interface automata.

Given two compatible interface automata P and Q, the composition $P \parallel Q$ is an interface automaton with the same action sets as $P \otimes Q$. The states are $V_{P\parallel Q} = \mathsf{Cmp}(P,Q)$; the initial states are $V^{\text{init}}_{P\parallel Q} = V^{\text{init}}_{P\otimes Q} \cap \mathsf{Cmp}(P,Q)$; and the steps are $\mathscr{T}_{P\parallel Q} = \mathscr{T}_{P\otimes Q} \cap (\mathsf{Cmp}(P,Q) \times \mathscr{A}_{P\parallel Q} \times \mathsf{Cmp}(P,Q))$.

Let $v \in V_P$, the set $\mathsf{IntReach}_P(v)$ is the smallest set $U \subseteq V_P$ such that $v \in U$ and if $u \in U$ and $(u,a,u') \in \mathscr{T}^H_P$, then $u' \in U$. Moreover, we let

$$\mathsf{ExtEn}^O_P(v) = \bigcup_{u \in \mathsf{IntReach}(v)} \mathscr{A}^O_P(u) \quad \text{and} \quad \mathsf{ExtEn}^I_P(v) = \bigcup_{u \in \mathsf{IntReach}(v)} \mathscr{A}^I_P(u)$$

be the sets of externally enabled output and input actions, respectively, at v. And for all externally enabled input and output actions $a \in \mathsf{ExtEn}^I_P(v) \cup \mathsf{ExtEn}^O_P(v)$, we let

$$\mathsf{ExtDest}_P(v,a) = \left\{ u' \;\middle|\; \exists (u,a,u') \in \mathscr{T}_P.\ u \in \mathsf{IntReach}_P(v) \right\}.$$

With these notions, we can define an alternating simulation between interface automata.

Definition 3.2. *Consider two interface automata P and Q. A binary relation $\preceq \subseteq V_Q \times V_P$ is an alternating simulation from Q to P if for all states $u \in V_Q$ and $v \in V_P$ such that $u \preceq v$, the following conditions hold:*
(a) $\mathsf{ExtEn}^I_P(v) \subseteq \mathsf{ExtEn}^I_Q(u)$, $\qquad \mathsf{ExtEn}^O_Q(u) \subseteq \mathsf{ExtEn}^O_P(v)$.
(b) For all actions $a \in \mathsf{ExtEn}^O_Q(u)$ and all states $u' \in \mathsf{ExtDest}_Q(u,a)$, there is a state $v' \in \mathsf{ExtDest}_P(v,a)$ such that $u' \preceq v'$ and for all actions $a \in \mathsf{ExtEn}^I_P(v)$ and all states $v' \in \mathsf{ExtDest}_P(v,a)$, there is a state $u' \in \mathsf{ExtDest}_Q(u,a)$ such that $u' \preceq v'$.

Now we use the notion of alternating simulation to establish a preorder for interface automata: the interface automaton Q refines the interface automaton P, written $Q \preceq P$, if $\mathscr{A}^I_P \subseteq \mathscr{A}^I_Q$, $\mathscr{A}^O_P \supseteq \mathscr{A}^O_Q$, and there is an alternating simulation $\preceq$ from Q to P, a state $v \in V^{\text{init}}_P$, and a state $u \in V^{\text{init}}_Q$ such that $u \preceq v$.

Let $P = \langle V_P, V^{\text{init}}_P, A^I_P, A^O_P, A^H_P, T_P \rangle$ be an interface automaton. The mirror of P, denoted $P^\top$, is given by $P^\top = \langle V_P, V^{\text{init}}_P, A^O_P, A^I_P, A^H_P, T_P \rangle$. The mirror operation is clearly an involution, i.e., $\left(P^\top\right)^\top = P$. Let the source multiplication μ be the parallel composition of interface automata, γ be the mirror operation, and let the preorder be refinement. We state the main claim of this section:

Proposition 3.3. *A theory of interface automata is a preordered heap.*

Since interface automata have preordered heap structure, for given interface automata P and Q, Theorem 2.2 enables us to find largest solutions R for equations of the form $\mu(Q,R) \leq P$. The quotient for interface automata was first reported in [6]. Now that we know interface automata have preordered heap structure, we can ask: what is target multiplication for interface automata? The operation is given by $\tau(P,Q) = \left(P^\top \parallel Q^\top\right)^\top$. We propose to call this operation *merging* in analogy to the case of AG contracts. Similarly, by Theorem 2.2, merging by fixed Q, τ_Q, has a left adjoint given by $\mu^{\gamma Q}(P) = P \parallel Q^\top$. For the same reason, we propose to call this binary operation *separation*. In AG contracts, merging and separation are used to handle multiple viewpoints in a design. To the best of our understanding, the notion of handling multiple design viewpoints has not been discussed for interface automata. Maintaining the analogy to AG contracts, we suspect that merging and separation here defined provide interface automata the ability to handle these multiple viewpoints. Exploring this idea is material for future work.

4 Sieved heaps

Some theories in computer science require manipulating objects which are not defined over the same domain. For example, consider a language L_1 defined over an alphabet Σ_1. Let Σ_2 be another alphabet for which L_2 is a language. The powerset of a set is a Boolean lattice, so we have two preordered heaps $P_{\Sigma_1} = 2^{\Sigma_1^*}$ and $P_{\Sigma_2} = 2^{\Sigma_2^*}$ whose source multiplications and involutions are intersection and negation ($*$ is the Kleene star—we will define operations carefully in the section on languages). With the theory of preordered heaps, we know how to solve inequalities for P_{Σ_1} and for P_{Σ_2}. Suppose we define an operation that allows us to compose $L_1 \in P_{\Sigma_1}$ with $L_2 \in P_{\Sigma_2}$. How do we solve inequalities involving L_1 and L_2 then? These languages belong to different preordered heaps. It is natural to define such an operation by mapping L_1 and L_2 to a common preordered heap, which by definition, has its own notion of source multiplication. We need a notion of mapping between preordered heaps:

Definition 4.1. *Let $(P, \leq, \mu, \gamma)$ and $(P', \leq', \mu', \gamma')$ be two preordered heaps. A preordered heap homomorphism $f\colon P \to P'$ is an order-preserving map which commutes with the source multiplications and involutions, i.e.,* $$\begin{tikzcd} P \times P \arrow[r, "f \times f"] \arrow[d, "\mu"'] & P' \times P' \arrow[d, "\mu'"] \\ P \arrow[r, "f"] & P' \end{tikzcd}$$ *and* $$\begin{tikzcd} P \arrow[r, "f"] \arrow[d, "\gamma"'] & P' \arrow[d, "\gamma'"] \\ P \arrow[r, "f"] & P' \end{tikzcd}$$ *commute.*

Preordered heaps P_{Σ_1} and P_{Σ_2} are indexed by alphabets. The common preordered heap where L_1 and L_2 can be mapped is determined by Σ_1 and Σ_2. As we will see in the next section, one option is to say that they generate the alphabet $\Sigma_c = \Sigma_1 \cup \Sigma_2$, and we can define maps $\iota_1\colon P_{\Sigma_1} \to P_{\Sigma_c}$ and $\iota_2\colon P_{\Sigma_2} \to P_{\Sigma_c}$ that embed languages over Σ_1 and Σ_2 to those defined under Σ_c. This observation tells us that we can use a structure S in order to index preordered heaps; this structure must have a binary operation defined in it. This operation will fulfill the role of identifying the alphabets where two languages can meet. Call this structure S, and let $\cdot$ be its binary operation. If we have two languages defined over the same alphabet, we should not need to move to another alphabet to compute the source multiplication of the two languages; thus, the binary operation of S should be idempotent. We will also require the operation to be commutative since it makes no difference whether we go to the language generated by Σ_1 and Σ_2 or to that generated by Σ_2 and Σ_1. A similar reasoning leads us to require associativity. Thus, S is endowed with an associative, commutative, idempotent binary operation, which means it is a semilattice. We make the choice to interpret it as an upper semi-lattice because we have the intuition that the languages generated by two smaller languages should be larger than any of the two, but this interpretation does not impose any algebraic limitations: an upper semilattice can be turned into a lower semilattice simply by flipping it upside-down.

We introduce the notion of a sieved, preordered heap (sieved heap, for short) that allows us to move objects between different domains of definition or different levels of abstraction. A sieved heap is a collection of preordered heaps indexed by an upper semilattice S together with mappings between the preordered heaps. We call these mappings concretizations. An upper semilattice can be interpreted as a partial order: for $a, b \in S$, we say that $a \leq ab$. Thus, the shortest definition for a sieved heap is that it is a functor from the preorder category S to **PreHeap**, the preordered heap category, whose objects are preordered heaps and whose arrows are preordered heap homomorphisms. We will give a longer definition. But first, why the adjective sieved? A sieved heap consists of a collection of preordered heaps and maps between them. We interpret these preordered heaps as structures containing varying amounts of detail about an object. This varying granularity motivated the name. This is the definition of this composite structure:

Definition 4.2. *Let S be a semilattice. Let $\{(P_x, \leq_x, \mu_x, \gamma_x)\}_{x \in S}$ be a collection of preordered heaps such that for every $x, y, z \in S$ we have a unique preordered heap homomorphism $\iota\colon P_x \to P_{xy}$ referred to as a concretization and making* $$\begin{tikzcd} P_{xy} \arrow[dr, "\iota'"] & \\ P_x \arrow[u, "\iota"] \arrow[r, "\iota''"] & P_{xyz} \end{tikzcd}$$ *commute. We require the concretization $\iota\colon P_x \to P_x$ to be the identity. Let $P = \oplus_{x \in S} P_x$, where $\oplus$ stands for disjoint union. We call $(P, \leq, \mu, \gamma)$ an S-sieved heap, where*

$\mu\colon P\times P\to P$ is an operation called source multiplication, and $\gamma\colon P\to P$ is called involution. Let $a\in P_x$ and $b\in P_y$, and let $\iota_x\colon P_x\to P_{xy}$ and $\iota_y\colon P_y\to P_{xy}$ be concretizations. These operations are given by

$$\mu(a,b)=\mu_{xy}\left(\iota_x(a),\iota_y(b)\right)\quad \textit{and}\quad \gamma(a)=\gamma_x(a).$$

Moreover, we say that $a\leq b$ if and only if there exists $z\in S$ and concretizations $\iota\colon P_x\to P_z$ and $\iota'\colon P_y\to P_z$ such that $\iota(a)\leq_z\iota'(b)$, where $\leq_z$ is the preorder of P_z.

Target multiplication τ for P is defined in a similar way: $\tau(a,b)=\tau_{xy}\left(\iota_x(a),\iota_y(b)\right)$, where τ_{xy} is the target multiplication of the preordered heap P_{xy}.

4.1 Sieved heaps are preordered heaps

Now we show that a sieved heap is itself a preordered heap. To do this, we must show that the relation $\leq$ over sieved heaps is a preorder, that source multiplication defined for a sieved heap is monotonic, that its involution is antitone, and that it meets the admissibility conditions. The following statements show that sieved heaps have these properties.

Lemma 4.3. *The relation $\leq$ on an S-sieved heap P is a preorder.*

Proof. Reflexivity. Let $a\in P_x$. Let ι be the concretization $\iota\colon P_x\to P_x$. Then $\iota a\leq_x\iota a$ because $\leq_x$ is a preorder in P_x; this means that $a\leq a$ in P.

Transitivity. Let $b\in P_y$ and $c\in P_z$ and suppose that $a\leq b$ and $b\leq c$. Then there exist $v,w\in S$ such that $\iota_x a\leq_v\iota_y b$ and $\iota'_y b\leq_w\iota_z c$, where the diagram

$$\begin{tikzcd} P_v \arrow[r, "\iota_v"] & P_{vw} & P_w \arrow[l, "\iota_w"'] \\ P_x \arrow[u, "\iota_x"] & P_y \arrow[ul, "\iota_y"] \arrow[ur, "\iota'_y"'] & P_z \arrow[u, "\iota_z"'] \end{tikzcd}$$

shows the relevant concretization maps (these diagrams commute per Definition 4.2). We obtain immediately $\iota_v\circ\iota_x a\leq_{vw}\iota_v\circ\iota_y b$ and $\iota_w\circ\iota'_y b\leq_{vw}\iota_w\circ\iota_z c$. From the diagram, $\iota_v\circ\iota_y=\iota_w\circ\iota'_y$, which means that $\iota_v\circ\iota_x a\leq_{vw}\iota_w\circ\iota_z c$, which means that $a\leq c$. □

Lemma 4.4. *Source multiplication on P is monotonic in both arguments.*

Proof. Let $a,b,c\in P$ with $a\leq c$. Suppose that $a\in P_x$, $b\in P_y$, and $c\in P_z$. Since $a\leq c$, there exist $u\in S$ such that $\iota_x a\leq_u\iota_z c$ for concretizations $\iota_x\colon P_x\to P_u$ and $\iota_z\colon P_z\to P_u$. Note that this means there exist $u',u''\in S$ such that $u=xu'$ and $u=zu''$. But this implies that $uy=xyu'$ and $uy=yzu''$. Thus, there exist concretizations $\iota_{xy}\colon P_{xy}\to P_{uy}$ and $\iota_{yz}\colon P_{yz}\to P_{uy}$, and

$$\begin{tikzcd} & P_y \arrow[dl, "\iota'_y"'] \arrow[d, "\iota_y"] \arrow[dr, "\iota''_y"] & \\ P_{xy} \arrow[r, "\iota_{xy}"'] & P_{uy} & P_{yz} \arrow[l, "\iota_{yz}"] \\ P_x \arrow[u, "\iota'_x"] \arrow[r, "\iota_x"] & P_u \arrow[u, "\iota_u"'] & P_z \arrow[l, "\iota_z"'] \arrow[u, "\iota'_z"'] \end{tikzcd} \tag{3}$$

commutes. Since $a\leq c$, we have

$$\mu_{uy}\left(\iota_u\circ\iota_x a,\iota_y b\right)\leq_{uy}\mu_{uy}\left(\iota_u\circ\iota_z c,\iota_y b\right). \tag{4}$$

By the commutativity of the diagram, $\iota_y=\iota_{xy}\circ\iota'_y=\iota_{yz}\circ\iota''_y$ and $\iota_u\circ\iota_x=\iota_{xy}\circ\iota'_x$ and $\iota_u\circ\iota_z=\iota_{yz}\circ\iota'_z$. Using these identities, we can rewrite (4) as

$$\mu_{uy}\left(\iota_{xy}\circ\iota'_x a,\iota_{xy}\circ\iota'_y b\right)\leq_{uy}\mu_{uy}\left(\iota_{yz}\circ\iota'_z c,\iota_{yz}\circ\iota''_y b\right),\quad\text{which implies that}$$

$$\iota_{xy}\circ\mu_{xy}\left(\iota'_x a,\iota'_y b\right)\leq_{uy}\iota_{yz}\circ\mu_{yz}\left(\iota'_z c,\iota''_y b\right)\ \text{and thus}\ \iota_{xy}\circ\mu\left(a,b\right)\leq_{uy}\iota_{yz}\circ\mu\left(c,b\right).$$

This shows that $\mu\left(a,b\right)\leq\mu\left(c,b\right)$. Monotonicity in the second argument is proved in the same way. □

Theorem 4.5. *An S-sieved heap P is a preordered heap.*

Proof. By lemma 4.3, we know that $(P,\leq)$ is a preorder. By lemma 4.4, we know that source multiplication for P is monotonic. From the definition of involution γ for P, it is immediate that this operation is antitone and that $\gamma^2 = \text{id}$. We must show the admissibility conditions. Let $a \in P_x$ and $b \in P_y$. Using the notation of (3), we have $\mu(a, \gamma \circ \mu(\gamma b, a)) = \mu(a, \gamma \circ \mu_{xy}(\iota_y' \circ \gamma b, \iota_x' a)) = \mu_{xy}(\iota_x' a, \gamma \circ \mu_{xy}(\gamma \circ \iota_y' b, \iota_x' a)) \leq \iota_y' b$, where we used the left admissibility of the preordered heap P_{xy}. But this means that $\mu(a, \gamma \circ \mu(\gamma b, a)) \leq b$. We conclude that P meets the left admissibility condition. Applying the same procedure tells us that P also has right admissibility. Thus, P is a preordered heap. □

Now that we know that sieved heaps are preordered heaps, we can compute quotients in these structures. We will now consider the solution of inequalities over languages as an application of sieved heaps.

5 Sieved heaps and language inequalities

Language inequalities arise as the formalization of the problem of synthesizing an unknown component in hardware and software systems. In this section, we provide preliminaries on languages and discuss their properties and operations. A fuller treatment of language properties can be found in [42, 40]. Our objective is to show that commonly studied language structures are sieved heaps, which allows us to axiomatically find their quotients per the results of Section 4.

5.1 Operations on languages

An alphabet is a finite set of symbols. The set of all finite strings over a fixed alphabet X is denoted by $X^\star$. $X^\star$ includes the empty string ε. A subset $L \subseteq X^\star$ is called a **language** over alphabet X. [22] is a standard reference on this subject.

A **substitution** f is a mapping of an alphabet Σ to subsets of $\Delta^\star$ for some alphabet Δ. The substitution f is extended to strings by setting $f(\varepsilon) = \{\varepsilon\}$ and $f(xa) = f(x)f(a)$. The following are well-studied language operations.

- Given a language L over alphabet X and an alphabet V, consider the substitution $l \colon X \to 2^{(X\times V)^\star}$ defined as $l(x) = \{(x,v) \mid v \in V\}$. Then the language $L_{\uparrow V} = \cup_{\alpha \in L} l(\alpha)$ over alphabet $X \times V$ is the **lifting** of language L to alphabet V.
- Given a language L over alphabet X and an alphabet V, consider the mapping $e \colon X \to 2^{(X\cup V)^\star}$ defined as $e(x) = \{\alpha x \beta \mid \alpha, \beta \in (V - X)^\star\}$. Then the language $L_{\Uparrow V} = \cup_{\alpha \in L} e(\alpha)$ over alphabet $X \cup V$ is the **expansion** of language L to alphabet V, i.e., words in $L_{\Uparrow V}$ are obtained from those in L by inserting anywhere in them words from $(V - X)^\star$. Notice that e is not a substitution and that $e(\varepsilon) = \{\alpha \mid \alpha \in V^\star\}$.

The following proposition states that language liftings and expansions meet the properties of concretization maps of a sieved heap. These results will be used in the next section dealing with inequalities over languages.

Proposition 5.1. *Liftings and expansions are order-preserving and commute with intersection and complementation.*

5.2 Composition of languages and inequalities involving languages

Consider two systems A and B with associated languages $L(A)$ and $L(B)$. The systems communicate with each other by a channel U and with the environment by channels I and O. The following two well-studied operators describe the external behavior of the composition of $L(A)$ and $L(B)$.

Definition 5.2. *Given the disjoint alphabets I,U,O, a language L_1 over $I \times U$, and a language L_2 over $U \times O$, the* **synchronous composition** *of languages L_1 and L_2 is the language $(L_1)_{\uparrow O} \cap (L_2)_{\uparrow I}$, denoted by $L_1 \bullet L_2$, defined over $I \times U \times O$.*

Definition 5.3. *Given the disjoint alphabets I,U,O, a language L_1 over $I \cup U$, and a language L_2 over $U \cup O$, the* **parallel composition** *of languages L_1 and L_2 is the language $(L_1)_{\Uparrow O} \cap (L_2)_{\Uparrow I}$, denoted by $L_1 \diamond L_2$, defined over $I \cup U \cup O$.*

Example. Let $L_1 = \{a, aa\}$ be a language of the alphabet $\Sigma_1 = \{a,b\}$, and $\Sigma_2 = \{c,d\}$ be another alphabet for which $L_2 = \{c\}$ is a language. Then $L_1 \bullet L_2 = \{(a,c)\}$ and $L_1 \diamond L_2 = \{ac, ca, caa, aca, aac\}$.

Synchronous composition abstracts the parallel execution of modules in lock step, assuming a global clock and instant communication by a broadcasting mechanism, modeling the product semantics common in the hardware community. In asynchronous composition modules execute independently at different speeds assuming clocks which progress at arbitrary rates relative to one another, modeling the interleaving semantics common in the software community. A comparison can be found in [26]. Now we show that we can interpret the above products as the source multiplication of a sieved heap. For each product, we first need to identify a suitable indexing semilattice. Then we need to build the appropriate preordered heaps and their maps.

5.2.1 Synchronous equations

Semilattice. Suppose we have a disjoint family $F = \{\Sigma_i\}_{1 \le i \le n}$ of alphabets for some positive integer n, and let $S = 2^F$. Then S is a semilattice under the operation of set union, i.e., if $x, y \in S$, we have $xy = x \cup y$.
Preordered heaps. For any $x \in S$, let $|x|$ be the cardinality of x. There exist natural numbers $k_1, \ldots, k_{|x|}$ such that $x = \{\Sigma_{k_j}\}_{1 \le j \le |x|} \subseteq F$ and $1 \le k_i < k_j \le n$ for $i < j$. We map each x to a preordered heap as follows. We define the alphabet over x as $\alpha(x) = \Sigma_{k_1} \times \cdots \times \Sigma_{k_{|x|}}$, and we set $P_x = 2^{\alpha(x)^*}$. Source multiplication μ_x for P_x is intersection, and involution γ_x is complementation. $(P_x, \le_x, \mu_x, \gamma_x)$ is a Boolean lattice, thus a preordered heap, as shown in Section 2.
Concretizations. For $x, y \in S$, P_{xy} is clearly a preordered heap because $xy \in S$. We also define the preordered heap $P_{x,y} = 2^{\Sigma^*_{x,y}}$ for $\Sigma_{x,y} = \alpha(x) \times \alpha(y - x)$ with source multiplication equal to set intersection and involution equal to complementation. Note that the only difference between P_{xy} and $P_{x,y}$ is the order in which the alphabets Σ_i appear in each: P_{xy} contains all sets of finite strings over the alphabet $\alpha(xy)$, and $P_{x,y}$ contains all sets of finite strings over the alphabet $\alpha(x) \times \alpha(y - x)$. Thus, P_{xy} and $P_{x,y}$ are isomorphic as sets. Let $\beta : P_{x,y} \to P_{xy}$ be this isomorphism, which is easily seen to be a preordered heap isomorphism.

This allows us to define the concretization ι_x as follows:
$$\begin{tikzcd} & P_{xy} \\ P_x \arrow[ur, dashed, "\iota_x"] \arrow[r, "(\cdot)\uparrow_{\alpha(y-x)}"] & P_{x,y} \arrow[u, "\beta"'] \end{tikzcd}.$$

From Proposition 5.1, we know that $(\cdot) \uparrow_{\alpha(y-x)}$ is a preordered heap map. Thus, we have an S-sieved heap $\{(P_x, \le_x, \mu_x, \gamma_x)\}_{x \in S}$. Since sieved heaps are preordered heaps (Theorem 4.5), for $A \in P_x$ and $B \in P_y$, an equation of the form $A \bullet z \le B$ has the largest solution $Z \in P_{xy}$ with

$$Z = \neg \left(\neg \beta' \left(B \uparrow_{\alpha(x-y)} \right) \cap \beta'' \left(A \uparrow_{\alpha(y-x)} \right) \right),$$

where $\beta'\colon P_{y,x} \to P_{xy}$ and $\beta''\colon P_{x,y} \to P_{xy}$ are extensions of the alphabet permutations to languages, as described above.
Example. Let I, U, and O be disjoint alphabets. Then S consists of all subsets of $\{I, O, U\}$. Let $i = \{I\}$, $u = \{U\}$, and $o = \{O\}$. The preordered heap P_{iu} consists of all languages over the alphabet $I \times U$. P_{uo} consists of all languages over $U \times O$. If $L_1 \in P_{iu}$, the concretization $\iota\colon P_{iu} \to p_{iuo}$ maps L_1 to a language over $I \times U \times O$. Observe that the order in which each alphabet appears is important and set from the beginning; this eliminates any potential ambiguities with the ordering of the alphabets (e.g., is it the alphabet $I \times U$ or $U \times I$?). By definition, this concretization map is $(\cdot) \uparrow_O$. In the same way, the concretization $\iota'\colon P_{uo} \to p_{iuo}$ is $\beta \circ (\cdot) \uparrow_I$, where $\beta : P_{uo,i} \to P_{iuo}$ permutes the symbols of the language so that they appear in the order (a, b, c) with $a \in I$, $b \in U$, and $c \in O$. Thus, source multiplication is $\mu(L_1, L_2) = L_1 \uparrow_O \cap \beta\,(L_2 \uparrow_I)$, which is the synchronous product.

5.2.2 Asynchronous equations

Now we form a semilattice S whose elements are abstract sets and whose operation is set union. Let $x \in S$, and define $P_x = 2^{x^*}$. For $y \in S$, the concretization $P_x \xrightarrow{\iota} P_{xy}$ is $\iota = (\cdot) \Uparrow_{y-x}$. Proposition 5.1 shows that ι is a preordered heap map. Thus, we have a sieved heap $\{(P_x, \leq_x, \mu_x, \gamma_x)\}_{x \in S}$.

Since sieved heaps are preordered heaps (Theorem 4.5), we are in a position to solve language equations under asynchronous composition. Let $x, y \in S$, $A \in P_x$ and $B \in P_y$. The largest solution to the equation $A \diamond z \leq B$ yields $Z \in P_{xy}$ with $Z = \neg\,(\neg B \Uparrow_{x-y} \cap A \Uparrow_{y-x})$.
Example. As before, let I, U, and O be disjoint alphabets, and let $I, U, O \in S$, where S is a semilattice with the operation of set union. The preordered heap P_{IU} consists of all languages over $I \cup U$. Similarly, the preordered heap P_{UO} consists of all languages over $U \cup O$. The embedding $\iota\colon P_{IU} \to P_{IUO}$ is simply $(\cdot) \Uparrow_O$, and the embedding $\iota'\colon P_{UO} \to P_{IUO}$ is $(\cdot) \Uparrow_I$. Thus, for $L_1 \in P_{IU}$ and $L_2 \in P_{UO}$, source multiplication is $\mu(L_1, L_2) = L_1 \Uparrow_O \cap L_2 \Uparrow_I$, which is the asynchronous product.

6 Conclusions

The comparison of the closed form computation of quotients ranging from language equations to AG contracts suggested a new algebraic structure, called *preordered heap*, endowed with the axioms of preorders, together with a monotonic multiplication and an involution. We showed that an admissibility condition allows to solve equations over preordered heaps, and we gave the closed form of the solution. We showed that various theories qualify as preordered heaps and therefore admit such explicit solution. In particular, we showed that the conditions for being preordered heaps hold for Boolean lattices, assume-guarantee contracts, and for interface automata: in all cases we were able to derive axiomatically the quotients, which had been previously obtained by specific analysis of each theory. Finally we defined equations over sieved heaps to handle components defined over multiple alphabets, and rederived as special cases the solution of language equations known in the literature.

Acknowledgements

We are grateful for the comments of our anonymous reviewers. This work was supported in part by NSF Contract CPS Medium 1739816; MIUR, Project “Italian Outstanding Departments, 2018-2022”; INDAM, GNCS 2020, “Strategic Reasoning and Automated Synthesis of Multi-Agent Systems”; University of Verona, Cooperint 2019, Program for Visiting Researchers.

References

[1] A. Aziz, F. Balarin, R.K. Brayton & A. L. Sangiovanni-Vincentelli (2000): *Sequential synthesis using S1S*. *IEEE Transactions on Computer-Aided Design* 19(10), pp. 1149–1162, doi:10.1109/43.875301.

[2] G. Barrett & S. Lafortune (1998): *Bisimulation, the Supervisory Control Problem and Strong Model Matching for Finite State Machines*. *Discrete Event Dynamic Systems: Theory & Applications* 8(4), pp. 377–429, doi:10.1023/A:1008301317459.

[3] Nikola Beneš, Benoît Delahaye, Uli Fahrenberg, Jan Křetínský & Axel Legay (2013): *Hennessy-Milner Logic with Greatest Fixed Points as a Complete Behavioural Specification Theory*. In Pedro R. D'Argenio & Hernán Melgratti, editors: *CONCUR 2013 – Concurrency Theory*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 76–90, doi:10.1007/978-3-642-40184-8_7.

[4] A. Benveniste, B. Caillaud, A. Ferrari, L. Mangeruca, R. Passerone & C. Sofronis (2007): *Multiple Viewpoint Contract-Based Specification and Design*. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf & Willem-Paul de Roever, editors: *Formal Methods for Components and Objects*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 200–225, doi:10.1007/978-3-540-92188-2_9.

[5] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. L. Sangiovanni-Vincentelli, W. Damm, T. A. Henzinger & K. G. Larsen (2018): *Contracts for System Design*. *Foundations and Trends®in Electronic Design Automation* 12(2-3), pp. 124–400, doi:10.1561/1000000053.

[6] P. Bhaduri & S. Ramesh (2008): *Interface synthesis and protocol conversion*. *Formal Asp. Comput.* 20(2), pp. 205–224, doi:10.1007/s00165-007-0045-4.

[7] G. Bochmann (2013): *Using logic to solve the submodule construction problem*. *Discrete Event Dynamic Systems* 23(1), pp. 27–59, doi:10.1007/s10626-011-0127-6.

[8] Patricia Bouyer, Franck Cassez & François Laroussinie (2011): *Timed Modal Logics for Real-Time Systems - Specification, Verification and Control*. *Journal of Logic, Language and Information* 20(2), pp. 169–203, doi:10.1007/s10849-010-9127-4.

[9] J.R. Burch, D. Dill, E. Wolf & G. DeMicheli (1993): *Modelling hierarchical combinational circuits*. In: *The Proceedings of the International Conference on Computer-Aided Design*, pp. 612–617, doi:10.1109/ICCAD.1993.580149.

[10] Franck Cassez & François Laroussinie (2000): *Model-Checking for Hybrid Systems by Quotienting and Constraints Solving*. In E. Allen Emerson & Aravinda Prasad Sistla, editors: *Computer Aided Verification*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 373–388, doi:10.1007/10722167_29.

[11] G. Castagnetti, M. Piccolo, T. Villa, N. Yevtushenko, R. K. Brayton & A. Mishchenko (2015): *Automated Synthesis of Protocol Converters with BALM-II*. In D. Bianculli, R. Calinescu & B. Rumpe, editors: *Software Engineering and Formal Methods. SEFM 2015 Collocated Workshops: ATSE, HOFM, MoKMaSD, and VERY*SCART. York, UK, September 7-8, 2015*, pp. 281–296, doi:10.1007/978-3-662-49224-6_23.

[12] E. Cerny & M. Marin (1977): *An approach to unified methodology of combinational switching circuits*. *IEEE Transactions on Computers* vol. C-26(8), pp. 745–756, doi:10.1109/TC.1977.1674912.

[13] W. Chen, J. Udding & T. Verhoeff (1989): *Networks of communicating processes and their (de-)composition*. In J.L.A. van de Snepscheut, editor: *Mathematics of Program Construction*, *Lecture Notes in Computer Science* 375, Springer Berlin Heidelberg, pp. 174–196, doi:10.1007/3-540-51305-1_10.

[14] L. De Alfaro (2003): *Game Models for Open Systems*. In N. Dershowitz, editor: *Verification: Theory and Practice*, *Lecture Notes in Computer Science* 2772, Springer Verlag, pp. 269–289, doi:10.1007/978-3-540-39910-0_12.

[15] L. De Alfaro & T. A. Henzinger (2001): *Interface Automata*. *SIGSOFT Softw. Eng. Notes* 26(5), pp. 109–120, doi:10.1145/503209.503226.

[16] M. D. Di Benedetto, A. Sangiovanni-Vincentelli & T. Villa (2001): *Model Matching for Finite State Machines*. *IEEE Transactions on Automatic Control* 46(11), pp. 1726–1743, doi:10.1109/9.964683.

[17] M. Fujita, Y. Matsunaga & M. Ciesielski (2001): *Multi-Level Logic Optimization*. In R. Brayton, S. Hassoun & T. Sasao, editors: *Logic Synthesis and Verification*, Kluwer, pp. 29–63, doi:10.1007/978-1-4615-0817-5_2.

[18] P. Green (1986): *Protocol Conversion*. *IEEE Transactions on Communications* 34(3), pp. 257–268, doi:10.1109/32.4655.

[19] E. Haghverdi & H. Ural (1999): *Submodule construction from concurrent system specifications*. *Information and Software Technology* 41(8), pp. 499–506, doi:10.1016/S0950-5849(99)00014-2.

[20] H. Hallal, R. Negulescu & A. Petrenko (2000): *Design of divergence-free protocol converters using supervisory control techniques*. In: *7th IEEE International Conference on Electronics, Circuits and Systems, ICECS 2000*, 2, pp. 705–708, doi:10.1109/ICECS.2000.912975.

[21] S. Hassoun & T. Villa (2001): *Optimization of Synchronous Circuits*. In R. Brayton, S. Hassoun & T. Sasao, editors: *Logic Synthesis and Verification*, Kluwer, pp. 225–253, doi:10.1007/978-1-4615-0817-5_2.

[22] J.E. Hopcroft, R. Motwani & J.D. Ullman (2001): *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, doi:10.1145/568438.568455.

[23] T. Kam, T. Villa, R. K. Brayton & A. L. Sangiovanni-Vincentelli (1997): *Synthesis of FSMs: Functional Optimization*. Kluwer Academic Publishers, Boston, doi:10.1007/978-1-4757-2622-0.

[24] J. Kim & M.M. Newborn (1972): *The simplification of sequential machines with input restrictions*. *IRE Transactions on Electronic Computers*, pp. 1440–1443, doi:10.1109/T-C.1972.223521.

[25] R. Kumar, S. Nelvagal & S.I. Marcus (1997): *A discrete event systems approach for protocol conversion*. *Discrete Event Dynamic Systems: Theory & Applications* 7(3), pp. 295–315, doi:10.1023/A:1008258331497.

[26] R.P. Kurshan, M. Merritt, A. Orda & S.R. Sachs (1999): *Modelling asynchrony with a synchronous model*. *Formal Methods in System Design* vol. 15(no. 3), pp. 175–199, doi:10.1007/3-540-60045-0_61.

[27] S. S. Lam (1988): *Protocol Conversion*. *IEEE Trans. Softw. Eng.* 14(3), pp. 353–362, doi:10.1109/32.4655.

[28] K.G. Larsen & L. Xinxin (1990): *Equation solving using modal transition systems*. In: *Logic in Computer Science, 1990. LICS '90, Proceedings., Fifth Annual IEEE Symposium on e*, pp. 108–117, doi:10.1109/LICS.1990.113738.

[29] N. Lynch & M. Tuttle (1989): *An introduction to Input/Output automata*. *CWI-Quarterly* 2(3), pp. 219–246, doi:10.1.1.83.7751.

[30] W.C. Mallon, J.T. Tijmen & T. Verhoeff (1999): *Analysis and Applications of the XDI Model*. In: *International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 231–242, doi:10.1109/ASYNC.1999.761537.

[31] P. Merlin & G. v. Bochmann (1983): *On the Construction of Submodule Specifications and Communication Protocols*. *ACM Transactions on Programming Languages and Systems* 5(1), pp. 1–25, doi:10.1145/357195.357196.

[32] R. Negulescu (2000): *Process spaces*. In C. Palamidessi, editor: *Proceedings of CONCUR 2000, 11th International Conference on Concurrency Theory*, *LNCS* 1877, Springer-Verlag, pp. 199–213, doi:10.1007/3-540-44618-4_16.

[33] R. Passerone, L. De Alfaro, T. A. Henzinger & A. L. Sangiovanni-Vincentelli (2002): *Convertibility verification and converter synthesis: two faces of the same coin*. In Lawrence T. Pileggi & Andreas Kuehlmann, editors: *ICCAD*, ACM, pp. 132–139. Available at `http://doi.acm.org/10.1145/774572.774592`.

[34] R. Passerone, Í. Íncer Romeo & A. L. Sangiovanni-Vincentelli (2019): *Coherent Extension, Composition, and Merging Operators in Contract Models for System Design*. *ACM Trans. Embed. Comput. Syst.* 18(5s), doi:10.1145/3358216.

[35] R. Passerone, J. A. Rowson & A. L. Sangiovanni-Vincentelli (1998): *Automatic Synthesis of Interfaces Between Incompatible Protocols*. In: *DAC*, pp. 8–13. Available at `http://doi.acm.org/10.1145/277044.277047`.

[36] J.-B. Raclet, E. Badouel, A. Benveniste, B. Caillaud, A. Legay & R. Passerone (2011): *A modal interface theory for component-based design*. Fundamenta Informaticae 108(1-2), pp. 119–149, doi:10.3233/FI-2011-416.

[37] Í. Íncer Romeo, A. L. Sangiovanni-Vincentelli, C.-W. Lin & E. Kang (2018): *Quotient for Assume-Guarantee Contracts*. In: 16th ACM-IEEE International Conference on Formal Methods and Models for System Design, MEMOCODE '18, p. 67–77, doi:10.1109/MEMCOD.2018.8556872.

[38] E. Sentovich & D. Brand (2001): *Flexibility in Logic*. In R. Brayton, S. Hassoun & T. Sasao, editors: Logic Synthesis and Verification, Kluwer, pp. 65–88, doi:10.1007/978-1-4615-0817-5_2.

[39] T. Villa, A. Petrenko, N. Yevtushenko, A. Mishchenko & R. K. Brayton (2015): *Component-Based Design by Solving Language Equations*. Proceedings of the IEEE 103(11), pp. 2152–2167, doi:10.1109/JPROC.2015.2450937.

[40] T. Villa, N. Yevtushenko, R. K. Brayton, A. Mishchenko, A. Petrenko & A. L. Sangiovanni-Vincentelli (2012): *The Unknown Component Problem: Theory and Applications*. Springer, doi:10.1007/978-0-387-68759-9.

[41] S. Watanabe, K. Seto, Y. Ishikawa, S. Komatsu & M. Fujita (2007): *Protocol Transducer Synthesis using Divide and Conquer approach*. In: Design Automation Conference, 2007. ASP-DAC '07. Asia and South Pacific, pp. 280–285, doi:10.1109/ASPDAC.2007.357999.

[42] N. Yevtushenko, T. Villa, R. K. Brayton, A. Mishchenko & A. L. Sangiovanni-Vincentelli (2004): *Composition Operators in Language Equations*. In: International Workshop on Logic and Synthesis, pp. 409–415.

A Additional proofs

A.1 Proof of Proposition 3.3

Lemma A.1. *The composition of composable interface automata is monotonic in the refinement order.*

Proof. Let P, Q and R be three interface automata such that $Q \preceq P$ and R is composable both with P and Q, we wish to show that $Q \parallel R \preceq P \parallel R$ and $R \parallel Q \preceq R \parallel P$.

From the preorder for interface automata, we need to prove that the action sets satisfy $\mathscr{A}^I_{P\parallel R} \subseteq \mathscr{A}^I_{Q\parallel R}$ and $\mathscr{A}^O_{Q\parallel R} \subseteq \mathscr{A}^O_{P\parallel R}$, and that there exists an alternating simulation from $Q \parallel R$ to $P \parallel R$. To prove the condition on the action sets, note that

$$\begin{aligned}
\mathscr{A}^I_{P\parallel R} &= \left(\mathscr{A}^I_P \cup \mathscr{A}^I_R\right) - \mathsf{shared}(P,R) \\
&= \left(\mathscr{A}^I_P \cup \mathscr{A}^I_R\right) - \left(\mathscr{A}^O_R \cap \mathscr{A}^I_P \cup \mathscr{A}^O_P \cap \mathscr{A}^I_R\right) \\
&= \left(\mathscr{A}^I_P - \mathscr{A}^O_R\right) \cup \left(\mathscr{A}^I_R - \mathscr{A}^O_P\right)
\end{aligned}$$

and

$$\begin{aligned}
\mathscr{A}^O_{P\parallel R} &= \left(\mathscr{A}^O_P \cup \mathscr{A}^O_R\right) - \mathsf{shared}(P,R) \\
&= \left(\mathscr{A}^O_P \cup \mathscr{A}^O_R\right) - \left(\mathscr{A}^O_R \cap \mathscr{A}^I_P \cup \mathscr{A}^O_P \cap \mathscr{A}^I_R\right) \\
&= \left(\mathscr{A}^O_P - \mathscr{A}^I_R\right) \cup \left(\mathscr{A}^O_R - \mathscr{A}^I_P\right).
\end{aligned}$$

Similar expressions apply to $Q \parallel R$. Since $\mathscr{A}^I_P \subseteq \mathscr{A}^I_Q$ and $\mathscr{A}^O_Q \subseteq \mathscr{A}^O_P$, we have

$$\mathscr{A}^I_{P\parallel R} \subseteq \mathscr{A}^I_{Q\parallel R} \tag{5}$$

and

$$\mathscr{A}^O_{Q\|R} \subseteq \mathscr{A}^O_{P\|R}. \tag{6}$$

We focus now on the alternating simulation. Since $Q \preceq P$, there exists an alternating simulation from Q to P. We define a relation from $Q \parallel R$ to $P \parallel R$ as follows: given $(u,r) \in V_{Q\|R}$ and $(v,r) \in V_{P\|R}$, let $(u,r) \preceq (v,r)$ if and only if $u \preceq v$. We wish to show that the relation just defined is an alternating simulation. First we prove part (a) of Definition 3.2. Recall that, due to the definition of composition, we need to remove shared actions between P and R from the corresponding externally enabled action sets of the composed automaton $P \parallel R$, and similarly for $Q \parallel R$. We have

$$\mathsf{ExtEn}^O_{P\|R}(v,r) = \bigcup_{(v',r')\in\mathsf{IntReach}(v,r)} \mathscr{A}^O_{P\|R}(v',r')$$

$$\mathsf{ExtEn}^I_{P\|R}(v,r) = \bigcup_{(v',r')\in\mathsf{IntReach}(v,r)} \mathscr{A}^I_{P\|R}(v',r')$$

From Definition 3.1 (IA product), it follows that

$$\begin{aligned}\mathscr{A}^O_{P\|R}(v',r') &= \left(\mathscr{A}^O_P(v') \cup \mathscr{A}^O_R(r')\right) - \mathsf{shared}(P,R)\\ &= \left(\mathscr{A}^O_P(v') \cup \mathscr{A}^O_R(r')\right) - \left(\mathscr{A}^O_R \cap \mathscr{A}^I_P \cup \mathscr{A}^O_P \cap \mathscr{A}^I_R\right)\\ &= \left(\mathscr{A}^O_P(v') - \mathscr{A}^I_R\right) \cup \left(\mathscr{A}^O_R(r') - \mathscr{A}^I_P\right).\end{aligned}$$

Hence,

$$\begin{aligned}\mathsf{ExtEn}^O_{P\|R}(v,r) &= \bigcup_{(v',r')\in\mathsf{IntReach}(v,r)} \left(\left(\mathscr{A}^O_P(v') - \mathscr{A}^I_R\right) \cup \left(\mathscr{A}^O_R(r') - \mathscr{A}^I_P\right)\right)\\ &= \left(\bigcup_{(v',r')\in\mathsf{IntReach}(v,r)} \left(\mathscr{A}^O_P(v') - \mathscr{A}^I_R\right)\right) \bigcup\\ &\quad \left(\bigcup_{(v',r')\in\mathsf{IntReach}(v,r)} \left(\mathscr{A}^O_R(r') - \mathscr{A}^I_P\right)\right)\\ &= \left(\left(\bigcup_{(v',r')\in\mathsf{IntReach}(v,r)} \mathscr{A}^O_P(v')\right) - \mathscr{A}^I_R\right) \bigcup\\ &\quad \left(\left(\bigcup_{(v',r')\in\mathsf{IntReach}(v,r)} \mathscr{A}^O_R(r')\right) - \mathscr{A}^I_P\right)\\ &= \left(\mathsf{ExtEn}^O_P(v) - \mathscr{A}^I_R\right) \cup \left(\mathsf{ExtEn}^O_R(r) - \mathscr{A}^I_P\right).\end{aligned}$$

Similarly,

$$\begin{aligned}\mathscr{A}^I_{P\|R}(v',r') &= \left(\mathscr{A}^I_P(v') \cup \mathscr{A}^I_R(r')\right) - \mathsf{shared}(P,R)\\ &= \left(\mathscr{A}^I_P(v') \cup \mathscr{A}^I_R(r')\right) - \left(\mathscr{A}^O_R \cap \mathscr{A}^I_P \cup \mathscr{A}^O_P \cap \mathscr{A}^I_R\right)\\ &= \left(\mathscr{A}^I_P(v') - \mathscr{A}^O_R\right) \cup \left(\mathscr{A}^I_R(r') - \mathscr{A}^O_P\right).\end{aligned}$$

Hence,

$$\begin{aligned}
\mathsf{ExtEn}^{I}_{P\|R}(v,r) &= \bigcup_{(v',r')\in\mathsf{IntReach}(v,r)} \left(\left(\mathscr{A}^{I}_{P}(v') - \mathscr{A}^{O}_{R}\right) \cup \left(\mathscr{A}^{I}_{R}(r') - \mathscr{A}^{O}_{P}\right)\right) \\
&= \left(\bigcup_{(v',r')\in\mathsf{IntReach}(v,r)} \left(\mathscr{A}^{I}_{P}(v') - \mathscr{A}^{O}_{R}\right)\right) \bigcup \\
&\quad \left(\bigcup_{(v',r')\in\mathsf{IntReach}(v,r)} \left(\mathscr{A}^{I}_{R}(r') - \mathscr{A}^{O}_{P}\right)\right) \\
&= \left(\left(\bigcup_{(v',r')\in\mathsf{IntReach}(v,r)} \mathscr{A}^{I}_{P}(v')\right) - \mathscr{A}^{O}_{R}\right) \bigcup \\
&\quad \left(\left(\bigcup_{(v',r')\in\mathsf{IntReach}(v,r)} \mathscr{A}^{I}_{R}(r')\right) - \mathscr{A}^{O}_{P}\right) \\
&= \left(\mathsf{ExtEn}^{I}_{P}(v) - \mathscr{A}^{O}_{R}\right) \cup \left(\mathsf{ExtEn}^{I}_{R}(r) - \mathscr{A}^{O}_{P}\right).
\end{aligned}$$

Applying a similar reasoning to $\mathsf{ExtEn}^{O}_{Q\|R}(u,r)$ and $\mathsf{ExtEn}^{I}_{Q\|R}(u,r)$ yields

$$\mathsf{ExtEn}^{O}_{Q\|R}(u,r) = \left(\mathsf{ExtEn}^{O}_{Q}(u) - \mathscr{A}^{I}_{R}\right) \cup \left(\mathsf{ExtEn}^{O}_{R}(r) - \mathscr{A}^{I}_{Q}\right) \quad \text{and}$$

$$\mathsf{ExtEn}^{I}_{Q\|R}(u,r) = \left(\mathsf{ExtEn}^{I}_{Q}(u) - \mathscr{A}^{O}_{R}\right) \cup \left(\mathsf{ExtEn}^{I}_{R}(r) - \mathscr{A}^{O}_{Q}\right).$$

Hence,

$$\mathsf{ExtEn}^{O}_{Q}(u) \subseteq \mathsf{ExtEn}^{O}_{P}(v) \quad \text{implies} \quad \mathsf{ExtEn}^{O}_{Q}(u) - \mathscr{A}^{I}_{R} \subseteq \mathsf{ExtEn}^{O}_{P}(v) - \mathscr{A}^{I}_{R}$$

and from (5)

$$\mathscr{A}^{I}_{P} \subseteq \mathscr{A}^{I}_{Q} \quad \text{implies} \quad \mathsf{ExtEn}^{O}_{R}(r) - \mathscr{A}^{I}_{Q} \subseteq \mathsf{ExtEn}^{O}_{R}(r) - \mathscr{A}^{I}_{P}.$$

Leading to $\mathsf{ExtEn}^{O}_{Q\|R}(u,r) \subseteq \mathsf{ExtEn}^{O}_{P\|R}(v,r)$. Similarly,

$$\mathsf{ExtEn}^{I}_{P}(v) \subseteq \mathsf{ExtEn}^{I}_{Q}(u) \quad \text{implies} \quad \mathsf{ExtEn}^{I}_{P}(v) - \mathscr{A}^{O}_{R} \subseteq \mathsf{ExtEn}^{I}_{Q}(u) - \mathscr{A}^{O}_{R}$$

and from (6)

$$\mathscr{A}^{O}_{Q} \subseteq \mathscr{A}^{O}_{P} \quad \text{implies} \quad \mathsf{ExtEn}^{I}_{R}(r) - \mathscr{A}^{O}_{P} \subseteq \mathsf{ExtEn}^{I}_{R}(r) - \mathscr{A}^{O}_{Q}.$$

Leading to $\mathsf{ExtEn}^{I}_{P\|R}(v,r) \subseteq \mathsf{ExtEn}^{I}_{Q\|R}(u,r)$.

To prove part (b) of Definition 3.2, let us define the *right alternating image* of Q in P with respect to the alternating simulation relation $\preceq$ as the interface automaton $P^{Q\preceq}$ such that

$$V_{P^{Q\preceq}} = \{v \in V_P \mid \exists u \in V_Q, u \preceq v\} \quad \text{and}$$

$$\mathscr{T}_{P^{Q\preceq}} = \left\{(v,a,v') \;\middle|\; (v,a,v') \in \mathscr{T}_P \text{ and } v,v' \in V_{P^{Q\preceq}}\right\}.$$

Note that $P^{Q\preceq}$ is a sub-automaton of P, because its sets of states, actions and transitions are contained in those of P by definition. From Definition 3.2, it follows that the alternating simulation from Q to P induces an alternating simulation from $P^{Q\preceq}$ to P defined as follows: for all $v \in V_{P^{Q\preceq}}$, let $v \preceq v$.

To show that it is an alternating simulation, first consider part (a) of Definition 3.2. By construction $\mathsf{ExtEn}^I_P(v) \subseteq \mathsf{ExtEn}^I_{P^{Q\preceq}}(v)$. To see this recall that there is a simulation from Q to P. If $u \preceq v$, then by part (b) of Definition 3.2, for all $a \in \mathsf{ExtEn}^I_P(v)$ and for all states $v' \in \mathsf{ExtDest}_P(v,a)$, there exists $u' \in \mathsf{ExtDest}_Q(u,a)$ such that $u' \preceq v'$. Hence, $v' \in V_{P^{Q\preceq}}$ and $(v,a,v') \in \mathscr{T}^I_{P^{Q\preceq}}$, implying that $a \in \mathsf{ExtEn}^I_{P^{Q\preceq}}(v)$ and $v' \in \mathsf{ExtDest}_{P^{Q\preceq}}(v,a)$. Obviously, $\mathsf{ExtEn}^O_{P^{Q\preceq}}(v) \subseteq \mathsf{ExtEn}^O_P(v)$ by definition, because $P^{Q\preceq}$ is a sub-automaton of P. Consider now part (b) of Definition 3.2. The first half of (b) is obvious, because by definition $P^{Q\preceq}$ is a sub-automaton of P, hence for all $a \in \mathsf{ExtEn}^O_{P^{Q\preceq}}(v)$, we have $\mathsf{ExtDest}_{P^{Q\preceq}}(v,a) \subseteq \mathsf{ExtDest}_P(v,a)$. The second half of (b) follows by the same reasoning above that led us to prove that $\mathsf{ExtEn}^I_P(v) \subseteq \mathsf{ExtEn}^I_{P^{Q\preceq}}(v)$.

We now define the *left alternating image* of P in Q with respect to the alternating simulation relation $\preceq$ as the interface automaton $Q^{\preceq P}$ such that

$$V_{Q^{\preceq P}} = \{u \in V_Q \mid \exists v \in V_P, u \preceq v\} \quad \text{and}$$

$$\mathscr{T}_{Q^{\preceq P}} = \left\{(u,a,u') \;\middle|\; (u,a,u') \in \mathscr{T}_Q \text{ and } u,u' \in V_{Q^{\preceq P}}\right\}.$$

Note that $Q^{\preceq P}$ is a sub-automaton of Q, because its sets of states, actions and transitions are contained in those of Q by definition. From Definition 3.2, it follows that the alternating simulation from Q to P induces an alternating simulation from Q to $Q^{\preceq P}$ defined as follows: for all $u \in V_{Q^{\preceq P}}$, let $u \preceq u$. To show this, first consider part (a) of Definition 3.2. By construction $\mathsf{ExtEn}^O_Q(u) \subseteq \mathsf{ExtEn}^O_{Q^{\preceq P}}(u)$. To see this recall that there is a simulation from Q to P. If $u \preceq v$, then by part (b) of Definition 3.2, for all $a \in \mathsf{ExtEn}^O_Q(u)$ and for all states $u' \in \mathsf{ExtDest}_Q(u,a)$, there exists $v' \in \mathsf{ExtDest}_P(v,a)$ such that $u' \preceq v'$. Hence, $u' \in V_{Q^{\preceq P}}$ and $(u,a,u') \in \mathscr{T}^O_{Q^{\preceq P}}$, implying that $a \in \mathsf{ExtEn}^O_{Q^{\preceq P}}(u)$ and $u' \in \mathsf{ExtDest}_{Q^{\preceq P}}(u,a)$. Obviously, $\mathsf{ExtEn}^I_{Q^{\preceq P}}(u) \subseteq \mathsf{ExtEn}^I_Q(u)$ by definition, because $Q^{\preceq P}$ is a sub-automaton of Q. Consider now part (b) of Definition 3.2. The second half of (b) is obvious, because by definition $Q^{\preceq P}$ is a sub-automaton of Q, hence for all $a \in \mathsf{ExtEn}^I_{Q^{\preceq P}}(u)$, we have $\mathsf{ExtDest}_{Q^{\preceq P}}(u,a) \subseteq \mathsf{ExtDest}_Q(u,a)$. The second half of (b) follows by the same reasoning above that led us to prove that $\mathsf{ExtEn}^O_Q(u) \subseteq \mathsf{ExtEn}^O_{Q^{\preceq P}}(u)$.

From the definitions of right and left alternating images, it follows directly that the alternating simulation $u \preceq v$ from Q to P is also an alternating simulation from $Q^{\preceq P}$ to $P^{Q\preceq}$, because $Q^{\preceq P}$ and $P^{Q\preceq}$ have been obtained from Q and P, respectively, by eliminating only those states (and corresponding transitions) that are not related by the simulation $\preceq$. Note that the alternating simulation from $Q^{\preceq P}$ to $P^{Q\preceq}$ is both serial and surjective by definition, i.e., for all $u \in V_{Q^{\preceq P}}$ there exists $v \in V_{P^{Q\preceq}}$ such that $u \preceq v$ and for all $v \in V_{P^{Q\preceq}}$ there exists $u \in V_{Q^{\preceq P}}$ such that $u \preceq v$, respectively.

The relation $Q \preceq Q^{\preceq P}$ induces a self-simulation from $Q \parallel R$ to $Q^{\preceq P} \parallel R$ defined by $(u,r) \preceq (u,r)$ if and only if $(u,r) \in Q^{\preceq P} \parallel R$ and $u \preceq u$. Similarly, $P^{Q\preceq} \preceq P$ induces a self-simulation from $P^{Q\preceq} \parallel R$ to $P \parallel R$ defined by $(v,r) \preceq (v,r)$ if and only if $(v,r) \in P^{Q\preceq} \parallel R$ and $v \preceq v$. Now we prove that the alternating simulation from $Q^{\preceq P} \parallel R$ to $P^{Q\preceq} \parallel R$ induced by $u \preceq v$ and defined at the beginning of the proof satisfies condition (b) of Definition 3.2. First note that since the alternating simulation from $Q^{\preceq P}$ to $P^{Q\preceq}$ is both serial and surjective, then also the alternating simulation from $Q^{\preceq P} \parallel R$ to $P^{Q\preceq} \parallel R$ is both serial and surjective. This is because $(u,r) \preceq (v,r)$ if and only if $u \preceq v$. Hence, if $(u,r) \preceq (v,r)$, then for all $a \in \mathsf{ExtEn}^O_{Q^{\preceq P}\parallel R}(u,r)$ and all states $(u',r') \in \mathsf{ExtDest}_{Q^{\preceq P}\parallel R}((u,r),a)$, there is a state $(v',r') \in \mathsf{ExtDest}_{P^{Q\preceq}\parallel R}((v,r),a)$ such that $(u',r') \preceq (v',r')$, and for all $a \in \mathsf{ExtEn}^I_{P^{Q\preceq}\parallel R}(v,r)$ and for all states $(v',r') \in \mathsf{ExtDest}_{P^{Q\preceq}\parallel R}((v,r),a)$, there is a state $(u',r') \in \mathsf{ExtDest}_{Q^{\preceq P}\parallel R}((u,r),a)$ such that $(u',r') \preceq (v',r')$. Hence, $Q^{\preceq P} \parallel R \preceq P^{Q\preceq} \parallel R$. We have shown we have the following relations:

$$Q \parallel R \leq Q^{\preceq P} \parallel R \leq P^{Q\preceq} \parallel R \leq P \parallel R,$$

which means that $Q \parallel R \preceq P \parallel R$. The proof for $R \parallel Q \preceq R \parallel P$ is similar. □

Note A.1. *To simplify notation in the expressions that follow, we set $P_Q = Q \parallel \left(Q \parallel P^\top\right)^\top$, for interface automata P and Q.*

Lemma A.2. *Given two interface automata P and Q such that Q and $P^\top$ are composable, then $\mathscr{A}^I_{P_Q} = \mathscr{A}^I_P$ and $\mathscr{A}^O_{P_Q} = \mathscr{A}^O_P$.*

Proof. Let us first compute the action sets for $Q \parallel P^\top$. Recall that the mirror operation switches input actions with output actions and vice-versa. Hence, the shared actions between Q and $P^\top$ are $(\mathscr{A}^I_Q \cap \mathscr{A}^I_P) \cup (\mathscr{A}^O_Q \cap \mathscr{A}^O_P)$. Then

$$\mathscr{A}^I_{Q\parallel P^\top} = \left(\mathscr{A}^I_Q \cup \mathscr{A}^O_P\right) - \mathsf{shared}(Q, P^\top) = \left(\mathscr{A}^I_Q - \mathscr{A}^I_P\right) \cup \left(\mathscr{A}^O_P - \mathscr{A}^O_Q\right) \quad \text{and}$$

$$\mathscr{A}^O_{Q\parallel P^\top} = \left(\mathscr{A}^O_Q \cup \mathscr{A}^I_P\right) - \mathsf{shared}(Q, P^\top) = \left(\mathscr{A}^O_Q - \mathscr{A}^O_P\right) \cup \left(\mathscr{A}^I_P - \mathscr{A}^I_Q\right).$$

Note that Q and $\left(Q \parallel P^\top\right)^\top$ are composable because by Definition 3.1 internal actions are renamed as a result of the composition and

$$\mathscr{A}^I_Q \cap \left(\left(\mathscr{A}^O_Q - \mathscr{A}^O_P\right) \cup \left(\mathscr{A}^I_P - \mathscr{A}^I_Q\right)\right) = \emptyset \quad \text{and}$$

$$\mathscr{A}^O_Q \cap \left(\left(\mathscr{A}^I_Q - \mathscr{A}^I_P\right) \cup \left(\mathscr{A}^O_P - \mathscr{A}^O_Q\right)\right) = \emptyset.$$

To compute the action sets for P_Q we can reuse the formulas above by substituting $Q \parallel P^\top$ in place of P:

$$\begin{aligned}
\mathscr{A}^I_{P_Q} &= \left(\mathscr{A}^I_Q - \mathscr{A}^I_{Q\parallel P^\top}\right) \cup \left(\mathscr{A}^O_{Q\parallel P^\top} - \mathscr{A}^O_Q\right) \\
&= \left(\mathscr{A}^I_Q - \left(\left(\mathscr{A}^I_Q - \mathscr{A}^I_P\right) \cup \left(\mathscr{A}^O_P - \mathscr{A}^O_Q\right)\right)\right) \cup \left(\left(\left(\mathscr{A}^O_Q - \mathscr{A}^O_P\right) \cup \left(\mathscr{A}^I_P - \mathscr{A}^I_Q\right)\right) - \mathscr{A}^O_Q\right) \\
&= \left(\mathscr{A}^I_Q \cap \mathscr{A}^I_P\right) \cup \left(\mathscr{A}^I_P - \mathscr{A}^I_Q\right) \\
&= \mathscr{A}^I_P.
\end{aligned}$$

We also have

$$\begin{aligned}
\mathscr{A}^O_{P_Q} &= \left(\mathscr{A}^O_Q - \mathscr{A}^O_{Q\parallel P^\top}\right) \cup \left(\mathscr{A}^I_{Q\parallel P^\top} - \mathscr{A}^I_Q\right) \\
&= \left(\mathscr{A}^O_Q - \left(\mathscr{A}^O_Q - \mathscr{A}^O_P \cup \mathscr{A}^I_P - \mathscr{A}^I_Q\right)\right) \cup \left(\left(\mathscr{A}^I_Q - \mathscr{A}^I_P \cup \mathscr{A}^O_P - \mathscr{A}^O_Q\right) - \mathscr{A}^I_Q\right) \\
&= \left(\mathscr{A}^O_Q \cap \mathscr{A}^O_P\right) \cup \left(\mathscr{A}^O_P - \mathscr{A}^O_Q\right) \\
&= \mathscr{A}^O_P.
\end{aligned}$$

□

Lemma A.3. *Given two interface automata P and Q such that Q and $P^\top$ are composable, then for all $(u,(u,v)) \in V_{P_Q}$, $\mathscr{A}^I_{P_Q}(u,(u,v)) \supseteq \mathscr{A}^I_P(v)$ and $\mathscr{A}^O_{P_Q}(u,(u,v)) \subseteq \mathscr{A}^O_P(v)$.*

Proof. In the composition of two automata we need to remove all communication actions from all input and output action sets. Communication actions are those actions that are outputs of one automaton and inputs of the other automaton. Recall moreover the output (resp. input) actions of $P^\top$ are input (resp. output) actions of P. From Lemma A.2 we have that for all $(u,v) \in V_{Q\parallel P^\top}$,

$$\mathscr{A}^I_{Q\parallel P^\top}(u,v) = \left(\mathscr{A}^I_Q(u) - \mathscr{A}^I_P\right) \cup \left(\mathscr{A}^O_P(v) - \mathscr{A}^O_Q\right) \quad \text{and}$$

$$\mathscr{A}^O_{Q\|P^\top}(u,v) = \left(\mathscr{A}^O_Q(u) - \mathscr{A}^O_P\right) \cup \left(\mathscr{A}^I_P(v) - \mathscr{A}^I_Q\right).$$

To compute the action sets for P_Q we can reuse the formulas above by substituting $Q \parallel P^\top$ in place of P.

$$\begin{aligned}
\mathscr{A}^I_{P_Q}(u,(u,v)) &= \left(\mathscr{A}^I_Q(u) - \mathscr{A}^I_{Q\|P^\top}\right) \cup \left(\mathscr{A}^O_{Q\|P^\top}(u,v) - \mathscr{A}^O_Q\right)\\
&= \left(\mathscr{A}^I_Q(u) - \left((\mathscr{A}^I_Q - \mathscr{A}^I_P) \cup (\mathscr{A}^O_P - \mathscr{A}^O_Q)\right)\right) \cup\\
&\quad \left(\left((\mathscr{A}^O_Q(u) - \mathscr{A}^O_P) \cup (\mathscr{A}^I_P(v) - \mathscr{A}^I_Q)\right) - \mathscr{A}^O_Q\right)\\
&= \left(\mathscr{A}^I_Q(u) \cap \mathscr{A}^I_P\right) \cup \left(\mathscr{A}^I_P(v) - \mathscr{A}^I_Q\right)\\
&\supseteq \left(\mathscr{A}^I_P(v) \cap \mathscr{A}^I_Q\right) \cup \left(\mathscr{A}^I_P(v) - \mathscr{A}^I_Q\right)\\
&= \mathscr{A}^I_P(v).
\end{aligned}$$

Note that $\mathscr{A}^I_Q(u) \cap \mathscr{A}^I_P \supseteq \mathscr{A}^I_P(v) \cap \mathscr{A}^I_Q$, because $\mathscr{A}^I_P \supseteq \mathscr{A}^I_P(v)$ by definition. Moreover, $\mathscr{A}^I_Q(u) \supseteq \mathscr{A}^I_P(v)$, because otherwise (u,v) would be an illegal state, i.e. there would exist a communication action from $P^\top$ to Q that would not be accepted by Q in state (u,v) (see discussion on illegal states in Section 3.2). Since $(u,v) \in V_{Q\|P^\top}$ by assumption, then (u,v) is not an illegal state, because illegal states are removed in the automaton resulting from the parallel composition of two automata.

$$\begin{aligned}
\mathscr{A}^O_{P_Q}(u,(u,v)) &= \left(\mathscr{A}^O_Q(u) - \mathscr{A}^O_{Q\|P^\top}\right) \cup \left(\mathscr{A}^I_{Q\|P^\top}(u,v) - \mathscr{A}^I_Q\right)\\
&= \left(\mathscr{A}^O_Q(u) - \left((\mathscr{A}^O_Q - \mathscr{A}^O_P) \cup (\mathscr{A}^I_P - \mathscr{A}^I_Q)\right)\right) \cup\\
&\quad \left(\left((\mathscr{A}^I_Q(u) - \mathscr{A}^I_P) \cup (\mathscr{A}^O_P(v) - \mathscr{A}^O_Q)\right) - \mathscr{A}^I_Q\right)\\
&= \left(\mathscr{A}^O_Q(u) \cap \mathscr{A}^O_P\right) \cup \left(\mathscr{A}^O_P(v) - \mathscr{A}^O_Q\right)\\
&\subseteq \left(\mathscr{A}^O_P(v) \cap \mathscr{A}^O_Q\right) \cup \left(\mathscr{A}^O_P(v) - \mathscr{A}^O_Q\right)\\
&= \mathscr{A}^O_P(v).
\end{aligned}$$ □

Note that $\mathscr{A}^O_Q(u) \cap \mathscr{A}^O_P \subseteq \mathscr{A}^O_P(v) \cap \mathscr{A}^O_Q$, because $\mathscr{A}^O_Q(u) \subseteq \mathscr{A}^O_Q$ by definition. Moreover, $\mathscr{A}^O_Q(u) \subseteq \mathscr{A}^O_P(v)$, because otherwise (u,v) would be an illegal state, i.e. there would exist a communication action from Q to $P^\top$ that would not be accepted by $P^\top$ in state (u,v) (see discussion on illegal states in Section 3.2). Since $(u,v) \in V_{Q\|P^\top}$ by assumption, then (u,v) is not an illegal state, because illegal states are removed in the automaton resulting from the parallel composition of two automata.

Lemma A.4.

(a) $(u,(u,v)) \in V_{P_Q}$ *and* $(u',(u',v')) \in \mathit{IntReach}_{P_Q}(u,(u,v))$ *imply* $v' \in \mathit{IntReach}_P(v)$.

(b) $(u,(u,v)) \in V_{P_Q}$ *and* $v' \in \mathit{IntReach}_P(v)$ *imply* $(u,(u,v')) \in \mathit{IntReach}_{P_Q}(u,(u,v))$.

Proof. (a) Since $(u',(u',v')) \in \mathsf{IntReach}_{P_Q}(u,(u,v))$, then by definition there exist internal actions a_1, ..., a_{n-1} and states $(u_1,(u_1,v_1)),\dots,(u_n,(u_n,v_n))$ such that $u_1 = u$, $v_1 = v$, $u_n = u'$, $v_n = v'$, and $((u_i,(u_i,v_i)), a_i, (u_{i+1},(u_{i+1},v_{i+1}))) \in \mathscr{T}^H_{P_Q}$. Now, $((u_i,(u_i,v_i)),a_i,(u_{i+1},(u_{i+1},v_{i+1}))) \in \mathscr{T}^H_{P_Q}$ implies either $(u_i,a_i,u_{i+1}) \in \mathscr{T}^H_Q$ and $v_i = v_{i+1}$ or $(v_i,a_i,v_{i+1}) \in \mathscr{T}^H_P$ and $u_i = u_{i+1}$, i.e. either $v_i = v_{i+1}$ or $(v_i,a_i,v_{i+1}) \in \mathscr{T}^H_P$. Hence, $v' \in \mathsf{IntReach}_P(v)$.
(b) Since $v' \in \mathsf{IntReach}_{P_Q}(v)$, then by definition there exist internal actions $a_1,\dots,a_{n-1}$ and states $v_1,\dots,v_n$ such that $v_1 = v$, $v_n = v'$, and $(v_i,a_i,v_{i+1}) \in \mathscr{T}^H_P$. Hence, $(u,(u,v)) \in V_{P_Q}$ implies that $((u,(u,v_i)), a_i, (u,(u,v_{i+1}))) \in \mathscr{T}^H_{P_Q}$, implying that $(u,(u,v')) \in \mathsf{IntReach}_{P_Q}(u,(u,v))$. □

Lemma A.5. *For all* $(u,(u,v)) \in V_{P_Q}$

$$\begin{aligned} &(a) \quad \mathsf{ExtEn}^O_{P_Q}(u,(u,v)) \subseteq \mathsf{ExtEn}^O_P(v) \\ &(b) \quad \mathsf{ExtEn}^I_{P_Q}(u,(u,v)) \supseteq \mathsf{ExtEn}^I_P(v) \end{aligned}$$

Proof. (a) Observe that

$$\begin{aligned} \mathsf{ExtEn}^O_{P_Q}(u,(u,v)) &= \bigcup_{(u',(u',v')) \in \mathsf{IntReach}_{P_Q}(u,(u,v))} \mathscr{A}^O_{P_Q}(u',(u',v')) \\ &\subseteq \bigcup_{v' \in \mathsf{IntReach}_P(v)} \mathscr{A}^O_P(v') \\ &= \mathsf{ExtEn}^O_P(v). \end{aligned}$$

This is because from Lemma A.3, we have $\mathscr{A}^O_{P_Q}((u',(u',v'))) \subseteq \mathscr{A}^O_P(v')$, and from Lemma A.4, we have that $(u',(u',v')) \in \mathsf{IntReach}_{P_Q}(u,(u,v))$ implies $v' \in \mathsf{IntReach}_P(v)$.
(b) Observe that

$$\begin{aligned} \mathsf{ExtEn}^I_{P_Q}(u,(u,v)) &= \bigcup_{(u',(u',v')) \in \mathsf{IntReach}_{P_Q}(u,(u,v))} \mathscr{A}^I_{P_Q}(u',(u',v')) \\ &\supseteq \bigcup_{v' \in \mathsf{IntReach}_P(v)} \mathscr{A}^I_P(v') \\ &= \mathsf{ExtEn}^I_P(v). \end{aligned}$$

This is because from Lemma A.3, we have $\mathscr{A}^I_{P_Q}((u',(u',v'))) \supseteq \mathscr{A}^I_P(v')$, and from Lemma A.4, we have that $v' \in \mathsf{IntReach}_P(v)$ implies $(u',(u',v')) \in \mathsf{IntReach}_{P_Q}(u,(u,v))$. □

Lemma A.6.

(a) *For all* $(u,(u,v)) \in V_{P_Q}$ *and for all* $a \in \mathsf{ExtEn}^O_{P_Q}(u,(u,v))$,

$$(u',(u',v')) \in \mathsf{ExtDest}_{P_Q}((u,(u,v)),a) \quad \textit{implies} \quad v' \in \mathsf{ExtDest}_P(v,a)$$

(b) *For all* $v \in V_P$ *and for all* $a \in \mathsf{ExtEn}^I_P(v)$,

$$v' \in \mathsf{ExtDest}_P(v,a) \textit{ implies } \exists u' \in V_Q \textit{ s.t. } (u',(u',v')) \in \mathsf{ExtDest}_{P_Q}((u,(u,v)),a).$$

Proof. (a) Note that $a \in \mathsf{ExtEn}^O_{P_Q}(u,(u,v))$ and $(u',(u',v')) \in \mathsf{ExtDest}_{P_Q}((u,(u,v)),a)$ imply that there exist internal actions $a_1,...,a_{n-1}$ and states $(u_1,(u_1,v_1)),...,(u_{n+1},(u_{n+1},v_{n+1}))$ such that $u_1 = u$, $v_1 = v$, and $((u_i,(u_i,v_i)),a_i,(u_{i+1},(u_{i+1},v_{i+1}))) \in \mathscr{T}^H_{P_Q}$ and $((u_n,(u_n,v_n)),a,(u_{n+1},(u_{n+1},v_{n+1}))) \in \mathscr{T}^O_{P_Q}$. Then, $(u_1,(u_1,v_1)),...,((u_n,(u_n,v_n)) \in \mathsf{IntReach}_{P_Q}(u,(u,v))$, $a \in \mathscr{A}^O_{P_Q}(u_n,(u_n,v_n))$ and $(u_{n+1},(u_{n+1},v_{n+1})) \in \mathsf{ExtDest}_{P_Q}((u,(u,v)),a)$. From Lemma A.4, we have $(u',(u',v')) \in \mathsf{IntReach}_{P_Q}(u,(u,v))$ implies $v' \in \mathsf{IntReach}_P(v)$. Hence, $v_1,...,v_n \in \mathsf{IntReach}_P(v)$. Since $a \in \mathscr{A}^O_{P_Q}(u_n,(u_n,v_n))$ from the reasoning above and, from Lemma A.3, $\mathscr{A}^O_{P_Q}(u_n,(u_n,v_n)) \subseteq \mathscr{A}^O_P(v_n)$, then $a \in \mathscr{A}^O_P(v_n)$, so that $v_{n+1} \notin \mathsf{IntReach}_P(v)$, implying that $v_{n+1} \in \mathsf{ExtDest}_P(v,a)$.
(b) Similarly, $a \in \mathsf{ExtEn}^I_P(v)$ and $v' \in \mathsf{ExtDest}_P(v,a)$ imply that there exist internal actions $a_1,...,a_{n-1}$ and states $v_1,...,v_{n+1}$ such that $v_1 = v$, and $(v_i,a_i,v_{i+1}) \in \mathscr{T}^H_P$ and $(v_n,a,v_{n+1}) \in \mathscr{T}^I_P$. Then, $v_1,...,v_n \in \mathsf{IntReach}_P(v)$, $a \in \mathscr{A}^I_P(v_n)$ and $v_{n+1} \in \mathsf{ExtDest}_P(v,a)$. Since $(u,(u,v)) \in V_{P_Q}$ by hypothesis, then, from Lemma A.4, we have $v' \in \mathsf{IntReach}_{P_Q}(v)$ implies $(u,(u,v')) \in \mathsf{IntReach}_{P_Q}(u,(u,v))$. Hence, $(u,(u,v_1))$,

..., $(u,(u,v_n)) \in \mathsf{IntReach}_P(u,(u,v))$. Since $a \in \mathscr{A}^I_P(v_n)$ and $\mathscr{A}^I_P(v_n) \subseteq \mathscr{A}^I_{P_Q}(u_n,(u_n,v_n))$ from Lemma A.3, then there exists $u' \in V_Q$ such that $(u',(u',v_{n+1})) \notin \mathsf{IntReach}_{P_Q}(u',(u',v))$, implying that $(u',(u',v_{n+1})) \in \mathsf{ExtDest}_{P_Q}((u,(u,v)),a)$. □

Lemma A.7. *The mirror operation is antitone.*

Proof. If $\preceq$ is an alternating simulation from Q to P, then we show that $\preceq^\top$ is an alternating simulation from $P^\top$ to $Q^\top$, where $\preceq^\top$ is defined by $v \preceq^\top u$ if and only if $u \preceq v$. Condition (a) from Definition 3.2 holds because (i) $\mathsf{ExtEn}^O_{P^\top}(v) = \mathsf{ExtEn}^I_P(v)$, (ii) $\mathsf{ExtEn}^I_{P^\top}(v) = \mathsf{ExtEn}^O_P(v)$, (iii) $\mathsf{ExtEn}^O_{Q^\top}(v) = \mathsf{ExtEn}^I_Q(v)$, and (iv) $\mathsf{ExtEn}^I_{Q^\top}(v) = \mathsf{ExtEn}^O_Q(v)$. Hence

$$\mathsf{ExtEn}^I_P(v) \subseteq \mathsf{ExtEn}^I_Q(v) \quad \text{iff} \quad \mathsf{ExtEn}^O_{P^\top}(v) \subseteq \mathsf{ExtEn}^O_{Q^\top}(v),$$

$$\mathsf{ExtEn}^O_Q(v) \subseteq \mathsf{ExtEn}^O_P(v) \quad \text{iff} \quad \mathsf{ExtEn}^I_{Q^\top}(v) \subseteq \mathsf{ExtEn}^I_{P^\top}(v).$$

Condition (b) from Definition 3.2 follows because from (i-iv) above we have that for all actions $a \in \mathsf{ExtEn}^I_{Q^\top}(u)$ and all states $u' \in \mathsf{ExtDest}_{Q^\top}(u,a)$, there is a state $v' \in \mathsf{ExtDest}_{P^\top}(v,a)$ such that $v' \preceq^\top u'$ and for all actions $a \in \mathsf{ExtEn}^O_{P^\top}(v)$ and all states $v' \in \mathsf{ExtDest}_{P^\top}(v,a)$, there is a state $u' \in \mathsf{ExtDest}_{Q^\top}(u,a)$ such that $v' \preceq^\top u'$ if and only if for all actions $a \in \mathsf{ExtEn}^O_Q(u)$ and all states $u' \in \mathsf{ExtDest}_Q(u,a)$, there is a state $v' \in \mathsf{ExtDest}_P(v,a)$ such that $u' \preceq v'$ and for all actions $a \in \mathsf{ExtEn}^I_P(v)$ and all states $v' \in \mathsf{ExtDest}_P(v,a)$, there is a state $u' \in \mathsf{ExtDest}_Q(u,a)$ such that $v' \preceq u'$. □

Proof of Proposition 3.3

Proof. We have to show that (a) the mirror operation is an involution; (b) the involution is antitone; (c) source multiplication is monotonic; and (d) source multiplication satisfies the admissibility conditions.

(a) This is immediate from the definition of mirror.

(b) This is guaranteed by Lemma A.7.

(c) Monotonicity was proved in Lemma A.1.

(d) It is sufficient to show that $P_Q \preceq P$ for arbitrary choices of P, because then we have $\mu_Q \circ \gamma \circ \mu_Q \circ \gamma \preceq \text{id}$, where $\mu_Q = (Q \parallel \cdot)$ and $\gamma = (\cdot)^\top$. By the preorder for interface automata, we need to show that (i) $\mathscr{A}^I_P \subseteq \mathscr{A}^I_{P_Q}$, (ii) $\mathscr{A}^O_{P_Q} \subseteq \mathscr{A}^O_P$, and (iii) there is an alternating simulation from P_Q to P.
(i-ii) follow directly from Lemma A.2.
For (iii), note that the states in P_Q have the form $(u,(u,v))$, where $u \in V_Q$ and $v \in V_P$. We show that the relation $(u,(u,v)) \preceq v$ is an alternating simulation from P_Q to P.
Condition (a) of Definition 3.2 holds because, from Lemma A.5, we have

$$\mathsf{ExtEn}^O_{P_Q}((u,(u,v))) \subseteq \mathsf{ExtEn}^O_P(v) \quad \text{and} \quad \mathsf{ExtEn}^I_{P_Q}((u,(u,v))) \qquad \supseteq \mathsf{ExtEn}^I_P(v). \tag{7}$$

Condition (b) of Definition 3.2 follows from Lemma A.6. Indeed, $(u_0,(u_0,v_0)) \preceq v_0$ by construction of the alternating simulation. Now, let $(u,(u,v)) \preceq v$, then for all actions

$$a \in \mathsf{ExtEn}^O_{P_Q}((u,(u,v)))$$

and for all states

$$(u',(u',v')) \in \mathsf{ExtDest}_{P_Q}((u,(u,v)),a),$$

we have $v' \in \mathsf{ExtDest}_P(v,a)$ by Lemma A.6 and $(u',(u',v')) \preceq v'$ by construction of the alternating simulation. Moreover, for all actions

$$a \in \mathsf{ExtEn}^I_P(v)$$

and for all states

$$v' \in \mathsf{ExtDest}_P(v,a),$$

there exists $u' \in V_Q$ such that $(u',(u',v')) \in \mathsf{ExtDest}_{P_Q}((u,(u,v)),a)$ by Lemma A.6 and $(u',(u',v')) \preceq v'$ as above.

□

A.2 Proof of Proposition 5.1

Proof of Proposition 5.1

Proof. Let L, L_1, L_2 be languages over alphabet X, we need to show that the following properties hold: (a) $L_1 \subseteq L_2$ implies $L_{1\,\uparrow V} \subseteq L_{2\,\uparrow V}$; (b) $L_1 \subseteq L_2$ implies $L_{1\,\Uparrow V} \subseteq L_{2\,\Uparrow V}$; (c) $(L_1 \cap L_2)_{\uparrow V} = L_{1\,\uparrow V} \cap L_{2\,\uparrow V}$; (d) $(L_1 \cap L_2)_{\Uparrow V} = L_{1\,\Uparrow V} \cap L_{2\,\Uparrow V}$; (e) $\overline{L}_{\uparrow V} = \overline{L_{\uparrow V}}$; and (f) $\overline{L}_{\Uparrow V} = \overline{L_{\Uparrow V}}$.

(a) and (b) are a consequence of the fact that $L_1 \subseteq L_2$ implies $L_1 \cup L_2 = L_2$. Observe that $L_{1\uparrow V} = \cup_{\alpha \in L_1} l(\alpha) \subseteq \cup_{\alpha \in L_1 \cup L_2} l(\alpha) = \cup_{\alpha \in L_2} l(\alpha) = L_{2\uparrow V}$ and $L_{1\Uparrow V} = \cup_{\alpha \in L_1} e(\alpha) \subseteq \cup_{\alpha \in L_1 \cup L_2} e(\alpha) = \cup_{\alpha \in L_2} e(\alpha) = L_{2\Uparrow V}$.

Parts (c) and (d) are proved by Proposition 2.2 in [42].

(e) $\alpha_1 \neq \alpha_2$ implies $l(\alpha_1) \cap l(\alpha_2) = \emptyset$, then

$$L_{\uparrow V} \cap \overline{L}_{\uparrow V} = (\cup_{\alpha \in L} l(\alpha)) \cap (\cup_{\alpha \in \overline{L}} l(\alpha)) = \emptyset.$$

Moreover, we have $L_{\uparrow V} \cup \overline{L}_{\uparrow V} = (\cup_{\alpha \in L} l(\alpha)) \cup (\cup_{\alpha \in \overline{L}} l(\alpha)) = (X \times V)^\star$. Hence, $\overline{L}_{\uparrow V} = (X \times V)^\star - L_{\uparrow V} = \overline{L_{\uparrow V}}$.

(f) In a similar way, since $\alpha_1 \neq \alpha_2$ implies $e(\alpha_1) \cap e(\alpha_2) = \emptyset$, then

$$L_{\Uparrow V} \cap \overline{L}_{\Uparrow V} = (\cup_{\alpha \in L} e(\alpha)) \cap (\cup_{\alpha \in \overline{L}} e(\alpha)) = \emptyset.$$

We also have $L_{\Uparrow V} \cup \overline{L}_{\Uparrow V} = (\cup_{\alpha \in L} e(\alpha)) \cup (\cup_{\alpha \in \overline{L}} e(\alpha)) = (X \cup V)^\star$. Hence, $\overline{L}_{\Uparrow V} = (X \cup V)^\star - L_{\Uparrow V} = \overline{L_{\Uparrow V}}$.

□