Electronic Notes in Theoretical Computer Science 83 (2004)
URL: http://www.elsevier.nl/locate/entcs/volume83.html 14 pages

Generic Transforms on Incomplete Specifications of Asynchronous Interfaces

Radu Negulescu [1]

Dept. of Electrical and Computer Eng., McGill University
Montreal, Quebec, Canada H3A 2A7

Abstract

In [15], processes represent incomplete specifications that guarantee proper behavior only under assumed constraints. Behaviors are represented as abstract executions. Here we define a corresponding notion of morphism, called process abstractions, as maps that preserve a composition operator on processes. We show that all process abstractions can be obtained from binary relations on execution sets, and we point out a ternary symmetry for process abstractions and the main composition operators. We rework and generalize correctness-preserving properties of commonly used process maps and we study new properties and maps of interest for verification.

1 Introduction

In [14,15], we define processes that embed an assumption and a guarantee in terms of sets of executions, akin to [10] or [8] but without relying on the structure of executions. Executions are abstract, that is, make no reference to events, states, ports, or other structural or operational information. Therefore, unlike the assume-guarantee rule used in model checking (see for instance [13]), our processes and their algebraic properties do not rely on causal or temporal relationships between the assumption and the guarantee; we use the term *assumption-guarantee* to designate such less-coupled association of an assumption and a guarantee. A benefit of our approach is increased generality, which has enabled new applications in the area of asynchronous and mixed-timing circuits, such as those reported in [2,16].

In this paper we formalize maps on assumption-guarantee processes and we study the properties of such maps. We define our maps, call them *process abstractions*, by the property that they preserve a composition operator. Such operators include the product (parallel composition) and the meet and join of a refinement partial order. The contributions in this paper include:

[1] Email: radu@macs.ece.mcgill.ca

©2004 Published by Elsevier Science B. V.

- Characterization of process abstractions for assumption-guarantee processes as being induced by four binary relations between execution sets.
- Characterization of process abstractions that allow pessimistic or optimistic verifications on images of assumption-guarantee processes.
- Characterization of assumption-guarantee processes that allow pessimistic or optimistic verifications on images through an arbitrary given process abstraction.
- A *rotation* process abstraction that relates the main composition operators and the process abstractions that preserve the respective operators.
- Rework of Galois connection properties of process maps and generalization to assumption-guarantee processes and abstract executions.

We depart from previous treatments of process transforms in several respects. Generic studies of abstract interpretations [7], transition system abstractions [12,17], and model reductions [6,11], which unify several types of maps, only express guarantees in a process specification. In particular, their process domains do not have a trace-theory-style reflection or mirroring operator that swaps guarantees and assumptions. Embedded assumptions can be expressed by means of divergences in [10] or failures in [8]; however, no attempts are made in [10], or [8] to unify and generalize properties of their frequently used process maps, such as hiding (deletion, projection) and derivative (after-operator). Our executions are abstract and can be instantiated arbitrarily. Criteria that authorize verification on images of processes can be found in [11], and also include the "exact approximation" of [6] and the under- and over-approximations in [9]; our counterparts Theorems 4.1 and 4.3. Galois connection properties of abstractions have been given in [12] and [17]; our counterpart is Theorem 3.2. However, prior to our study, these criteria and properties had been given only in operational settings (transition systems or predicate transformers) and for guarantee-only interfaces, while abstract executions have been introduced by our work on process spaces.

We are not aware of previous attempts in the literature to establish a ternary symmetry of process domains as we do with rotation, or the entailed relations between parallel composition and choice operators (meet and join) and among the maps that preserve the respective operators. Finally, abstract executions and the related generalizations are an original contribution of our work on process spaces.

2 Preliminaries

2.1 Process Spaces

Process spaces are a generic theory of concurrency, parameterized by the execution domain. Executions can be sequences of events, timed words, functions of time, etc., but a priori they are a primitive notion. Here we briefly overview

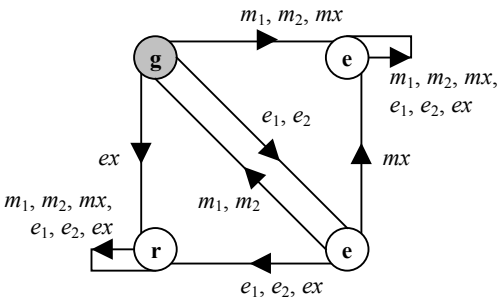
the formalism; more details and examples can be found in [14,15].

Let $\mathcal{E}$ be an arbitrary set, whose elements are called *executions*. A *process* over $\mathcal{E}$ is a pair (X, Y) of subsets of $\mathcal{E}$ such that $X \cup Y = \mathcal{E}$. The *process space* of $\mathcal{E}$, denoted by $\mathcal{S}_\mathcal{E}$, is the set of all processes over $\mathcal{E}$.

A process (X, Y) represents a contract for the interface between a device and its environment. The goal of this contract is to allow only executions from $X \cap Y$, called *goals*, to occur. The device guarantees that only executions from X (*accessible*) may occur, which means that the device should avoid executions from $\overline{X}$ (*escapes*). Also, the device assumes that only executions from Y (*accessible*) may occur, which means that the environment should avoid executions from $\overline{Y}$ (*rejects*). We use the following notation: $\mathbf{as}\, p =$ the set of accessible executions of p, $\mathbf{at}\, p =$ acceptable executions, $\mathbf{g}\, p =$ goals, $\mathbf{r}\, p$ = rejects, $\mathbf{e}\, p =$ escapes.

Example 1 Consider an "etiquette machine" that has polite conversations with its environment, intended to contain only phrases "How do you do" and "How are you". A conversation can be arbitrarily long, but finite. The available vocabulary also includes the undesirable phrase "XXX".

The behavior of the machine is represented by the process in Figure 1, whose executions consist of finite strings of greetings between the machine and the environment. The greetings are represented by symbolic events. The initial state is shaded. A string of greetings is a reject, goal, or escape for this process if it is spelled by a path starting at the initial state and ending at a state marked $\mathbf{r}$, $\mathbf{g}$, or $\mathbf{e}$, respectively.



Greeting	Event
Environment says "How do you do"	e_1
Environment says "How are you"	e_2
Environment says "XXX"	ex
Machine says "How do you do"	m_1
Machine says "How are you"	m_2
Machine says "XXX"	mx

Fig. 1. Process of the etiquette machine.

For instance, execution $ex\ mx$ is a reject because the environment violates the contract by illegal input ex; execution $m_1\ ex\ m_2$ is an escape, because the machine is not allowed to speak out of turn with illegal output m_1, even though the environment issues ex afterwards; the empty string is a goal, because neither the environment nor the machine have done anything wrong; and, execution e_2 is an escape, because it would be impolite for the machine not to respond to the greeting. □

The main process space operators are as follows. [2]

[2] The definitions here are consistent to [14] but use different execution sets (for simplicity).

- *Product* ($\times$), defined by $\mathbf{g}\,(p \times q) = \mathbf{g}\,p \cap \mathbf{g}\,q$ and $\mathbf{as}\,(p \times q) = \mathbf{as}\,p \cap \mathbf{as}\,q$, models the parallel composition of two devices.
- *Refinement* ($\sqsubseteq$), defined by $p \sqsubseteq q \;\Leftrightarrow\; \mathbf{as}\,p \supseteq \mathbf{as}\,q \wedge \mathbf{at}\,p \subseteq \mathbf{at}\,q$, models a relative notion of correctness: q can substitute 'specification' p.
- *Robustness* ($\mathcal{R}_{\mathcal{E}}$), defined by $p \in \mathcal{R}_{\mathcal{E}} \Leftrightarrow \mathbf{r}\,p = \emptyset$, models an absolute notion of correctness: the device works without the environment avoiding executions.

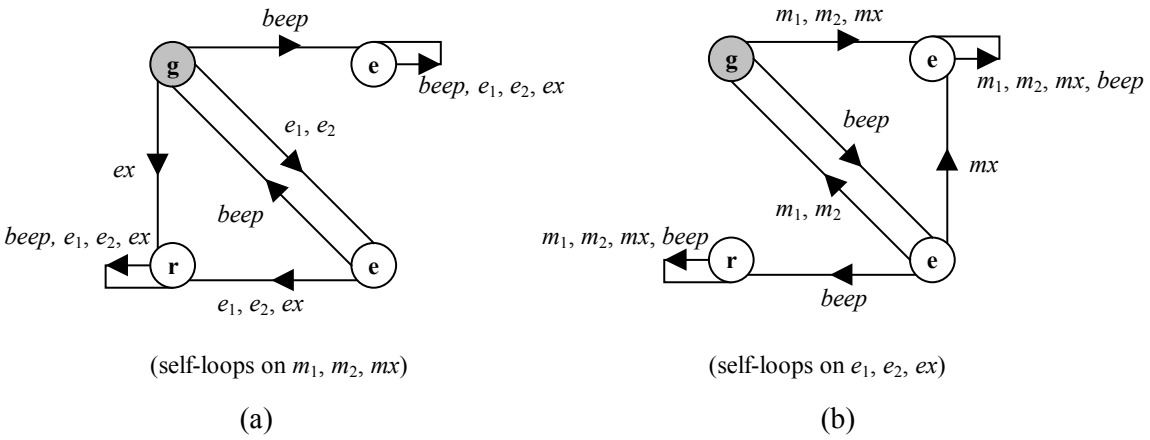


Fig. 2. Example for product: (a) Listener; (b) Speaker.

Example 2 Consider a realization of the etiquette machine from Example 1 as a system of two components, call them Listener and Speaker. Listener emits a "beep" each time it hears a greeting from the environment, provided that the greeting is not out of turn and not an "XXX". Speaker delivers a polite greeting after each "beep".

Listener and Speaker can be represented by processes over the execution set $\mathcal{E} = \{e_1, e_2, ex, m_1, m_2, mx, beep\}^*$, shown in Figure 2. By convention, we omit from the figures actions that are ignored by the respective processes, and we assume that the omitted actions produce self-loops at every state. For instance, e_1, e_2, ex are ignored by Speaker and are omitted from Figure 2 (b).

Execution e_2 *beep* m_1 is a goal for both Listener and Speaker, and it is a goal for the product process as well. Execution e_1 e_2 is a reject for Listener and a goal for Speaker; accordingly, this execution is a reject for the product process. Execution e_1 *beep beep* is an escape for Listener and a reject for Speaker, and, therefore, this execution is an escape for the product process.

The process in Figure 1 considered over the execution set $\{e_1, e_2, ex, m_1, m_2, mx, beep\}^*$ is refined by the product of Listener and Speaker, assuming the ignored *beep* produces self-loops at every state in Figure 1.

We perform such verifications by a tool, called FIREMAPS [14], that implements several process operations. The verification algorithms use state reachability analysis with binary decision diagrams to decide inclusion and

Any two non-complementary sets $\mathbf{as}\,p$, $\mathbf{at}\,p$, $\mathbf{v}\,p$, $\mathbf{g}\,p$, $\mathbf{r}\,p$, and $\mathbf{e}\,p$ uniquely define a process.

compute intersection of regular languages, which represent execution sets. □

Product is associative, commutative, and idempotent. Refinement is a partial order, and product is monotone with respect to refinement.

Refinement induces a complete lattice, in which we use $\sqcap$ to denote the *meet* and $\sqcup$ to denote the *join*. Process spaces are not Boolean algebras, but they are ternary algebras [5].

2.2 Binary Relations and Galois Connections

In the following, we use notation from [3] or introduce our own. A *poset* is a pair $\langle P, \preceq \rangle$ consisting of a set P and a partial order relation $\preceq$ on P. A *Galois connection* between posets $\langle A, \preceq_A \rangle$ and $\langle B, \preceq_B \rangle$ is a pair (α, β) of maps $\alpha : A \to B$ and $\beta : B \to A$ such that, for every $u \in A$ and $v \in B$, $u \preceq_A \beta(v) \Leftrightarrow v \preceq_B \alpha(u)$.[3]

A *relation* over two sets $\mathcal{E}_1$ and $\mathcal{E}_2$ is a subset of $\mathcal{E}_1 \times \mathcal{E}_2$. (The sign $\times$ is used both for Cartesian product and the process product.) The *converse* of a relation $\rho \subseteq \mathcal{E}_1 \times \mathcal{E}_2$ is a relation $\rho^\smile \subseteq \mathcal{E}_2 \times \mathcal{E}_1$ such that, for every $u \in \mathcal{E}_1$ and $v \in \mathcal{E}_2$, $u \rho v \Leftrightarrow v \rho^\smile u$. The *image* of set $X \subseteq \mathcal{E}_1$ through relation $\rho \subseteq \mathcal{E}_1 \times \mathcal{E}_2$ is the set $\mathrm{Im}_\rho(X) = \{v \in \mathcal{E}_2 \mid \exists\, u \in X : u \rho v\}$. The powerset of set $\mathcal{E}$ is denoted by $\wp(\mathcal{E})$. For any function $f : \wp(\mathcal{E}_1) \to \wp(\mathcal{E}_2)$, $\forall\, X \subseteq \mathcal{E}_1 : \overline{f}(X) = \overline{f(X)} = \mathcal{E}_2 \backslash f(X)$ (pointwise complementation).

The following lemma [3] characterizes a class of maps between execution sets that are of interest for defining our process abstractions.

Lemma 1 *For any map $f : \wp(\mathcal{E}_1) \to \wp(\mathcal{E}_2)$, the following are equivalent.*
(a) *There exists a map $f^\smile : \wp(\mathcal{E}_2) \to \wp(\mathcal{E}_1)$ such that $\forall\, X \subseteq \mathcal{E}_1, Y \subseteq \mathcal{E}_2 : X \cap f^\smile(Y) = \emptyset \Leftrightarrow Y \cap f(X) = \emptyset$;*
(b) *For every set $\mathcal{M}$ of subsets of $\mathcal{E}_1$, $f(\bigcup_{X \in \mathcal{M}} X) = \bigcup_{X \in \mathcal{M}} f(X)$;*
(c) *There exists a relation $\rho \subseteq \mathcal{E}_1 \times \mathcal{E}_2$ such that $f = \mathrm{Im}_\rho$.*

A function f between power sets is a *union-preserving map* (*UPM* for short) if it satisfies the properties in Lemma 1. Noting that two sets A and B are disjoint if and only if $A \subseteq \overline{B}$, Part (a) of Lemma 1 essentially says that the maps $\overline{f}$ and $\overline{f^\smile}$ form a Galois connection between $\langle \wp(\mathcal{E}_1), \subseteq \rangle$ and $\langle \wp(\mathcal{E}_2), \subseteq \rangle$. (Recall the contravariant definition of Galois connections, in which we follow [3]; here, a larger X yields larger $f(X)$ and $f^\smile(X)$ and smaller $\overline{f(X)}$ and $\overline{f^\smile(X)}$.) Part (b) characterizes such maps as those maps that preserve union. Part (c) provides a representation for such maps, showing they can be constructed from binary relations. Also notice that any function satisfying either of the properties in Lemma 1 is strict ($f(\emptyset) = \emptyset$), and that relation ρ in Part (c) is unique for any such function.

Example 3 All language homomorphisms are UPMs. For any finite sets Σ_1

[3] Galois connections are sometimes defined in a covariant manner. Following [3], we use without loss of generality the contravariant form of these definitions.

and Σ_2, a *language homomorphism* is a map $h : \Sigma_1 \to \Sigma_2^*$ extended to Σ_1^* and $\wp(\Sigma_1^*)$ by the following laws:

$$h(\varepsilon) = \varepsilon \ ,$$
$$\forall\, u_1, u_2 \in \Sigma_1^* :\ h(u_1 u_2) = h(u_1) h(u_2) \ ,$$
$$\forall\, L \subseteq \Sigma_1^* :\ h(L) = \textstyle\bigcup_{u \in L} h(u) \ .$$

Consider, for instance, *deletion* from [8]. For finite sets Γ and Σ such that $\Gamma \subseteq \Sigma$, $\mathbf{del}(\Gamma)(\cdot) : \Sigma \to \Sigma^*$ satisfies $\forall\, a \in \Sigma \setminus \Gamma : \mathbf{del}(\Gamma)(a) = a$ and $\forall\, a \in \Gamma : \mathbf{del}(\Gamma)(a) = \varepsilon$. The resulting language homomorphism $\mathbf{del}(\Gamma)(\cdot) : \wp(\Sigma^*) \to \wp(\Sigma^*)$ deletes from the argument words all occurrences of symbols from Γ.

Note, however, that not all UPMs are language homomorphisms; for instance, prefix-closure is a UPM but it is neither a language homomorphism nor the converse of a language homomorphism (Example 8.4 in [14]). □

For any UPM f between $\wp(\mathcal{E}_1)$ and $\wp(\mathcal{E}_2)$, the *underlying relation* of f is the (unique) relation $\rho_f \subseteq \mathcal{E}_1 \times \mathcal{E}_2$ that satisfies $f = \mathrm{Im}_{\rho_f}$, and the *converse* of f is the (unique) map $f^\smile : \wp(\mathcal{E}_2) \to \wp(\mathcal{E}_1)$ for which $(\overline{f}, \overline{f^\smile})$ is a Galois connection.

3 Symmetries and Process Abstractions

Process spaces admit a duality and ternary symmetry based on the following operations:

- *Reflection* $(-)$, defined by $\mathbf{as}\,(-p) = \mathbf{at}\,p$ and $\mathbf{at}\,(-p) = \mathbf{as}\,p$;
- *Rotation* $(/)$, defined by $\mathbf{as}\,/p = \overline{\mathbf{as}\,p} \cup \overline{\mathbf{at}\,p}$ and $\mathbf{at}\,/p = \mathbf{as}\,p$.

Proposition 2 *For any processes p and q,*

(a) $-\,-\,p = p$ **(c)** $-(p \sqcap q) = -p \ \sqcup -q$

(b) $///\,p = p$ **(d)** $/\,(p \times q) = /\,p \ \sqcap /\,q$

Proposition 2 (a) and (b) show that $-$ and $/$ are bijective, since they are roots of identity and their inverses are $-$ and $//$, respectively. To verify Proposition 2 (b), notice that $/$ maps the rejects of p into the goals of $/p$, goals into escapes, and escapes into rejects.

Proposition 2 (c) and (d) reveal that the main process space operators (product, meet, and join) induce isomorphic semilattice structures over a process space; we refer to these and other similar operators as compositions. Formally, binary operator * in a process space is a *composition* if there exists a bijection H such that $H(p * q) = H(p) \sqcap H(q)$ for all processes p and q.

Process abstractions are defined generically, independently of the execution domain and also independently of the composition operator used.

Definition 1 *For any compositions $*$ and $\odot$ in process spaces $\mathcal{S}_{\mathcal{E}_1}$ and $\mathcal{S}_{\mathcal{E}_2}$, map F between $\mathcal{S}_{\mathcal{E}_1}$ and $\mathcal{S}_{\mathcal{E}_2}$ is a $*\odot$* process abstraction *($*\odot$-PA for short) if*

$$\forall \mathcal{B} \subseteq \mathcal{S}_{\mathcal{E}_1} :\ F(*_{p \in \mathcal{B}}\, p) \ = \ \odot_{p \in \mathcal{B}} F(p) \ .$$

Example 4 Revisiting Example 2, the product of Listener and Speaker is not refined by the etiquette machine; a counter-example execution is *beep*, revealing that the etiquette machine does not control action *beep*. Now, let us delete *beep* from Listener×Speaker using the deletion UPM of Example 3. Letting $F(p) = (\mathbf{del}(\{beep\})(\mathbf{as}\, p),\ \overline{\mathbf{del}(\{beep\})(\mathbf{at}\, p)})$, we have that F(Listener×Speaker) is exactly the etiquette machine process of Figure 1, considered over execution set $\{e_1, e_2, ex, m_1, m_2, mx\}^*$. □

In the following, we study the algebraic properties of ⊓⊓-PAs only (read *meet-meet process abstractions*). Nevertheless, because the process space operators are isomorphic with respect to rotation and reflection (see Proposition 2), the formalization and properties we develop for meet also apply to product and the other process space compositions, as in the example given in Subsection 5.4.

Theorem 3.1 (characterization) *For any ⊓⊓-PA F between $\mathcal{S}_{\mathcal{E}_1}$ and $\mathcal{S}_{\mathcal{E}_2}$, there exist UPMs f, g, h, and i from $\wp(\mathcal{E}_1)$ to $\wp(\mathcal{E}_2)$ such that:*

$$F(p) = (f(\mathbf{as}\, p) \cup h(\mathbf{r}\, p),\ \overline{g(\mathbf{as}\, p) \cup i(\mathbf{r}\, p)})\ ,$$

for all p from $\mathcal{S}_{\mathcal{E}_1}$.

UPMs f, g, h, and i are related to F as follows. For any $X \subseteq \mathcal{E}$, we have:

$$f(X) = \mathbf{as}\, F(X, \mathcal{E}), \quad \mathbf{as}\, F(X, X) \backslash f(X) \subseteq g(X) \subseteq \mathbf{as}\, F(X, X)\ ,$$

$$h(X) = \mathbf{r}\, F(X, \mathcal{E})\ , \quad \mathbf{r}\, F(X, X) \backslash h(X) \subseteq i(X) \subseteq \mathbf{r}\, F(X, X)\ .$$

UPMs g and i might not be unique for a given F, but they can be instantiated in Theorem 3.1 to their upper bounds $\mathbf{as}\, F(X, X)$ and $\mathbf{r}\, F(X, X)$. By Lemma 1, UPMs f, g, h, and i are induced by their underlying relations. Thus, any ⊓⊓-PA can be constructed from binary relations on execution sets.

Example 5 Map F in Example 4, which implements deletion of action *beep*, can be represented in Theorem 3.1 by $f = \mathbf{del}(\{beep\})$, $g = \emptyset$, $h = \emptyset$, $i = \mathbf{del}(\{beep\}))^{\sqcap\sqcap}$. This is because the accessible and reject sets of $F(p)$ can be obtained by applying the deletion UPM on the accessible and reject sets of p, with no additional executions.

Alternately, F can be represented by $g = \mathbf{del}(\{beep\})$ and the same f, h, and i: since for all p we have $\mathbf{r}\, p \subseteq \mathbf{as}\, p$, we also have $f(\mathbf{as}\, p) \cup g(\mathbf{r}\, p) = \mathbf{del}(\{beep\})(\mathbf{as}\, p) \cup \mathbf{del}(\{beep\})(\mathbf{r}\, p) = \mathbf{del}(\{beep\})(\mathbf{as}\, p) = f(\mathbf{as}\, p)$. □

Example 6 Product with a constant is a ⊓⊓-PA. Let c be a process and Π_c a map on processes such that for any process p, $\Pi_c(p) = p \times c$. The image of p through Π_c is uniquely determined by:

$$\mathbf{as}\, \Pi_c(p) = \mathbf{as}\, c \cap \mathbf{as}\, p, \text{ and}$$
$$\mathbf{r}\, \Pi_c(p) = (\mathbf{as}\, c \cap \mathbf{r}\, p) \cup (\mathbf{r}\, c \cap \mathbf{as}\, c)\ .$$

Therefore, map Π_c can be represented as: $(\mathbf{i}d_{\mathbf{as}\, c}, \emptyset, \mathbf{i}d_{\mathbf{r}\, c}, \mathbf{i}d_{\mathbf{as}\, c})^{\sqcap\sqcap}$. □

By definition, a ⊓⊓-PA preserves meet; it follows that a ⊓⊓-PA also preserves refinement: $p \sqsubseteq q \ \Rightarrow\ F(p) \sqsubseteq F(q)$.

In the following, we focus on a simpler type of $\sqcap\sqcap$-PAs that includes most maps of interest for verification.

Definition 2 *An $\sqcap\sqcap$-PA F is* simple *(is a $\sqcap\sqcap$-SPA for short) if there exists a UPM f so that $F(p) = (f(\mathbf{as}\, p),\ \overline{f(\mathbf{r}\, p)})$. For such F and f, we write $f^{\sqcap\sqcap} = F$ and the* converse *of F is the $\sqcap\sqcap$-SPA $F^{\smile} = ((F_{\sqcap\sqcap})^{\smile})^{\sqcap\sqcap}$.*

Theorem 3.2 (reciprocity) *For any $\sqcap\sqcap$-SPA F between $\mathcal{S}_{\mathcal{E}_1}$ and $\mathcal{S}_{\mathcal{E}_2}$ and processes p over $\mathcal{E}_1$ and q over $\mathcal{E}_2$,*

$$p \times F^{\smile}(q) \in \mathcal{R}_{\mathcal{E}_1} \ \Leftrightarrow\ F(p) \times q \in \mathcal{R}_{\mathcal{E}_2}\ .$$

Reflection relates robustness and refinement verifications by the following property [14, Theorem 2.12]: $p \sqsubseteq q \ \Leftrightarrow\ -p \times q \in \mathcal{R}$, which states essentially that we can close a system by the reflection of the specification. Hence, Theorem 3.2 reveals a Galois connection $(-F, -F^{\smile})$ with respect to the converse of refinement: $p \sqsupseteq -F^{\smile}(q) \ \Leftrightarrow\ q \sqsupseteq -F(p)$.

A consequence of Theorem 3.2 is that, for specification processes of a special form, certain reductions can be performed on implementations without affecting the result of verification. Specifically, if the reflection of the specification is invariant to a process abstraction and its converse, then the process abstraction and its converse can be applied on the implementation without affecting the verdict, but possibly reducing the computational costs.

Example 7 It follows from Theorem 3.2 that, in a verification of refinement, we can hide internal actions of the implementation without affecting the result.

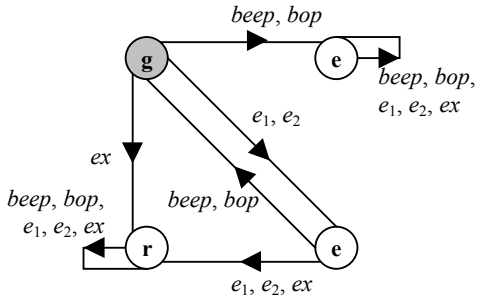


Fig. 3. Modified Speaker for Example 7.

For example, consider a specification consisting of an Etiquette2 process as in Figure 1 extended with self-loops on *beep* and *bop* at every state, and an implementation consisting of the Listener2 process in Figure 3 and a Speaker2 process as in Figure 2 (b) extended with self-loops on *bop* at every state.

Let $F = \mathbf{del}(\{beep, bop\})^{\sqcap\sqcap}$ where $\mathbf{del}(\{beep, bop\})$ is deletion of *beep* and *bop* (Example 3). We have that $F^{\smile}(F(-\text{Etiquette2})) = -\text{Etiquette2}$, and:

$$\begin{aligned}
&\text{Etiquette2} \sqsubseteq \text{Listener2} \times \text{Speaker2}\\
\Leftrightarrow\ &-\text{Etiquette2} \times \text{Listener2} \times \text{Etiquette2 is robust}\\
\Leftrightarrow\ &F(F^{\smile}(-\text{Etiquette2})) \times \text{Listener2} \times \text{Speaker2 is robust}\\
\Leftrightarrow\ &-\text{Etiquette2} \times F(F^{\smile}(\text{Listener2} \times \text{Speaker2})) \text{ is robust}\\
\Leftrightarrow\ &\text{Etiquette2} \sqsubseteq F(F^{\smile}(\text{Listener2} \times \text{Speaker2}))
\end{aligned}$$

Observe how the possibility of internal deadlock comes through deletion of internal actions. Listener2 × Speaker2 does not refine Etiquette2 because *bop* leads to deadlock: execution $m_1\,bop$ is accessible for Listener2 × Speaker2 but not for Etiquette2. We also find that $F^\smile F$ (Listener2 × Speaker2) does not refine Etiquette2, because execution m_1 is accessible for $F^\smile F$ (Listener2 × Speaker2). Moreover, these counterexamples to refinement are related: $m_1\,bop$ maps to m_1 through deletion and converse deletion. □

4 Verifications on Images

The basic verification problems we address, robustness of product and refinement by a product, are computationally intractable (PSPACE-hard [14]). To reduce computational costs, we would like to verify images of processes through a ⊓⊓-SPA instead of the original processes, which may be more complex. In this section, we establish relationships between robustness of $\times_{p\in\mathcal{B}} p$ and robustness of $\times_{p\in\mathcal{B}} F(p)$: for which processes or process abstractions does one of these robustness conditions imply the other? The following lemma gives two such relationships that actually hold for all processes and ⊓⊓-SPAs.

Lemma 3 *For any ⊓⊓-SPA F between $\mathcal{S}_{\mathcal{E}_1}$ and $\mathcal{S}_{\mathcal{E}_2}$, for any processes p and q over $\mathcal{E}_1$, and for any set $\mathcal{B} \subseteq \mathcal{S}_{\mathcal{E}_1}$ of processes, we have*

(a) $\times_{p\in\mathcal{B}} F(p) \sqsubseteq F(\times_{p\in\mathcal{B}} p)$ *(half-compositionality w.r.t. product)*

(b) $p \in \mathcal{R}_{\mathcal{E}_1} \;\Rightarrow\; F(p) \in \mathcal{R}_{\mathcal{E}_2}$ *(preservation of robustness)*

Certain elementary properties of an underlying relation entail several additional properties for the induced UPMs and ⊓⊓-SPAs. Let $\rho \subseteq \mathcal{E}_1 \times \mathcal{E}_2$ be a relation. We say that ρ is *injective* if, for all $u_1, u_2 \in \mathcal{E}_1$ and $v \in \mathcal{E}_2$ such that $u_1 \rho v$ and $u_2 \rho v$, we have $u_1 = u_2$; ρ is *co-surjective* if $\mathrm{Im}_{\rho^\smile}(\mathcal{E}_2) = \mathcal{E}_1$.

Now we give criteria for performing "optimistic" verifications (of robustness of product) on image processes. In such verifications, correct systems never fail, although flawed systems may pass. The criteria are as follows: Theorem 4.1 characterizes all processes that are suitable for such verifications for a given ⊓⊓-SPA, and Theorem 4.2 characterizes all ⊓⊓-SPAs for which *all* processes are suitable. The characterizations are fairly wide: processes that are refined through $F^\smile F$, or maps induced by injective relations, respectively.

Definition 3 *Let F be a ⊓⊓-PA between process spaces $\mathcal{S}_{\mathcal{E}_1}$ and $\mathcal{S}_{\mathcal{E}_2}$. Process p over $\mathcal{E}_1$ is* F-optimistic *if $F^\smile(F(p)) \sqsupseteq p$. F is* optimistic *if all processes from $\mathcal{S}_{\mathcal{E}_1}$ are F-optimistic.*

Theorem 4.1 *For any ⊓⊓-SPA F between process spaces $\mathcal{S}_{\mathcal{E}_1}$ and $\mathcal{S}_{\mathcal{E}_2}$, and any set $\mathcal{B}$ of F-optimistic processes from $\mathcal{S}_{\mathcal{E}_1}$,*

$$\times_{p\in\mathcal{B}} p \in \mathcal{R}_{\mathcal{E}_1} \;\Rightarrow\; \times_{p\in\mathcal{B}} F(p) \in \mathcal{R}_{\mathcal{E}_2}\ .$$

Theorem 4.2 *For any ⊓⊓-SPA $F = f^{\sqcap\sqcap}$ between process spaces $\mathcal{S}_{\mathcal{E}_1}$ and $\mathcal{S}_{\mathcal{E}_2}$, the following properties are equivalent:*

(a) *relation ρ_f is injective;*
(b) *every process p over $\mathcal{E}_1$ is F-optimistic.*

Now we give criteria for performing "pessimistic" verifications (of robustness of product) on image processes. In such verifications, flawed systems never pass, although correct systems may fail. Theorem 4.3 characterizes all processes that are suitable for such verifications for a given $\sqcap\sqcap$-SPA as processes that refine their images through $F^\smile F$. Theorem 4.2 characterizes all $\sqcap\sqcap$-SPAs for which *all* processes are suitable; remarkably, it suffices that the underlying be co-surjective.

Definition 4 *Let F be a $\sqcap\sqcap$-PA between process spaces $\mathcal{S}_{\mathcal{E}_1}$ and $\mathcal{S}_{\mathcal{E}_2}$. Process p over $\mathcal{E}_1$ is* F-pessimistic *if $F^\smile(F(p)) \sqsubseteq p$. F is* pessimistic *if all processes from $\mathcal{S}_{\mathcal{E}_1}$ are F-pessimistic.*

Theorem 4.3 *For any $\sqcap\sqcap$-SPA F between process spaces $\mathcal{S}_{\mathcal{E}_1}$ and $\mathcal{S}_{\mathcal{E}_2}$, and any set $\mathcal{B}$ of F-pessimistic processes from $\mathcal{S}_{\mathcal{E}_1}$,*

$$\times_{p\in\mathcal{B}} p \in \mathcal{R}_{\mathcal{E}_1} \;\Leftarrow\; \times_{p\in\mathcal{B}} F(p) \in \mathcal{R}_{\mathcal{E}_2} \;.$$

Theorem 4.4 *For any $\sqcap\sqcap$-SPA $F = f^{\sqcap\sqcap}$ between process spaces $\mathcal{S}_{\mathcal{E}_1}$ and $\mathcal{S}_{\mathcal{E}_2}$, the following properties are equivalent:*
(a) *relation ρ_f is co-surjective;*
(b) *every process p over $\mathcal{E}_1$ is F-pessimistic.*

One can combine optimistic and pessimistic approximations to achieve verifications on images that are equivalent to verifications of robustness of products of original processes. Our notion of independence of a process from a $\sqcap\sqcap$-PA is simply the conjunction of optimistic and pessimistic approximations.

5 Examples

In the following we describe several applications of process abstractions to reduce the costs of verification.

5.1 Derivatives

Occurrence of an event can be formalized as the image of a derivative map on processes. Let relation d_w on Σ^* be such that $u\ d_w\ v$ iff $u = wv$. Let $D_w = \mathrm{Im}_{d_w}$ be the induced UPM. For any process p over Σ^* and word $w \in \Sigma^*$, the *derivative* of p by w is the process $(D_w)^{\sqcap\sqcap}(p)$. Relation d_w is injective because $wv_1 = wv_2 \Rightarrow v_1 = v_2$, but it is not co-surjective because words that do not start with w do not have an image through d_w.

Derivatives of formal expressions [4] and similar operators (such as the after-operator in [10]) have been applied widely to represent occurrence of events in concurrency theory. One such application is reinitializing processes: change state after following an initial sequence of events. By Theorem 4.2, verifications of robustness of product can be performed directly on the reini-

tialized processes without false flaws. This allows to verify the steady-state behavior of a system separately from the initialization behavior. Monotonicity with respect to refinement and other properties of derivatives also follow from generic properties of $\sqcap\sqcap$-PAs in Section 3.

5.2 Projections of Finite Words

For any process p over Σ^* and set $A \subseteq \Sigma$, the *projection* of p on A is the process $p{\downarrow}A = (\mathbf{del}(\Sigma \backslash A))^{\sqcap\sqcap}(p)$, where **del** is deletion (Example 3). The underlying relation of is co-surjective (every word has an image through deletion).

By Theorem 4.2, it follows that verifications on images through projection are always pessimistic, and yield no false passes! Although the branching structure is not preserved by such projections, deadlock and other flaws come through as illustrated in Example 7.

5.3 Projections of Timed Infinite Words

Following loosely [1], a *timed word* over action set Σ is a pair (σ, τ) of an infinite trace from $(\Sigma^* l)^\omega$, where l is a symbol from outside Σ, called the "invisible action", and an increasing, unbounded sequence τ of positive reals. (The invisible action has no meaning; we use it to simplify the definition of projection for infinite words. Superscript ω indicates infinite repetition.) Let $T(\Sigma)$ be the set of timed words over Σ. For instance, $(a(bl)^\omega, \tau)$ such that $\forall\, i > 0 : \tau_i = i$ is in $T(\{a, b\})$, but $(a(bl)^\omega, \theta)$ such that $\forall\, i > 0 : \theta_i = 1 - 2^{-i}$ is not in $T(\{a, b\})$ because θ is bounded. (Sequence indices start at 1.)

We define projection of timed words recursively, as follows. For any $A \subseteq \Sigma$ and timed word $\xi = (\sigma, \tau) \in T(\Sigma)$, let $\xi{\downarrow}A$ be a timed word (σ', τ') such that:

- if $\sigma_1 \in A$ or $\sigma_1 = l$, then $\sigma'_1 = \sigma_1$, $\tau'_1 = \tau_1$, and $(\sigma'_{2,\ldots,\infty}, t'_{2,\ldots,\infty}) = (\sigma_{2,\ldots,\infty}, \tau_{2,\ldots,\infty}){\downarrow}A$, where $2, \ldots, \infty$ denotes the tail subsequence.
- if $\sigma_1 \in \Sigma \backslash A$ and $\sigma_1 \neq l$, then $(\sigma', \tau') = (\sigma_{2,\ldots,\infty}, \tau_{2,\ldots,\infty}){\downarrow}A$.

For instance, $(a(bl)^\omega, \tau){\downarrow}\{a\} = (al^\omega, \tau')$ such that $\tau'_i = \tau_{2i-1}$ for all $i \geq 1$.

Let $\langle{\downarrow}A\rangle_T = \{(\xi, \xi') \mid \xi' = \xi{\downarrow}A\} \subseteq T(\Sigma) \times T(\Sigma)$. Notice that $\langle{\downarrow}A\rangle_T$ is by construction a total function, hence it is co-surjective. Therefore, the induced process abstraction $\langle{\downarrow}A\rangle_T^{\sqcap\sqcap}$ is pessimistic, and verifications on projected timed processes yield no false passes. Monotonicity and other properties also follow immediately from generic properties of $\sqcap\sqcap$-SPAs.

5.4 Rotations

As an example of process abstraction defined with respect to a composition operator other than $\sqcap$, recall the rotation operation from Section 3.

One verifies that rotation is a $\times\sqcap$-PA; namely, it is $(\mathrm{Im}_{\mathbf{id}\mathcal{E}})^{\times\sqcap}$. Since the underlying relation is injective and co-surjective, all processes are independent from this PA, which is not surprising since rotation is bijective.

5.5 Delay-insensitivity

Delay-insensitive (DI) circuits are a class of asynchronous circuits whose correctness of operation does not depend on delays in components or wires. In [2], we have extended the formalization of delay-insensitivity of [18] so it can be applied to circuits that include DI parts and non-DI parts and to circuits that may have bidirectional lines. Also in [2] we have shown that our formalization allows reductions in the verification costs for such circuits.

We consider processes over Σ^*, where Σ is an arbitrary set of actions. Let Π be a set of pairs of actions from Σ, and let the relation n_Π be $\{ (u,u) \in \Sigma^* \times \Sigma^* \mid \forall (x,y) \in \Pi : u \text{ does not contain substring } xy \}$. Note that n_Π is injective, hence it enables optimistic approximations.

We further define the relation s_Π as the smallest subset of $\Sigma^* \times \Sigma^*$ such that, for every $u, v, w \in \Sigma^*$, we have:

- $n_\Pi \subseteq s_\Pi$, and
- $\forall (x,y) \in \Pi$: if $(vxyw, u) \in s_\Pi$, then $(vxyw, u) \in s_\Pi$.

In effect, $(v,w) \in s_\Pi$ if and only if w is obtained from v by replacing substrings xy by yx whenever $(x,y) \in \Pi$, until no more xy substrings exist in the trace. This replacement procedure is not always deterministic; for instance, $(cba, bca) \in s_{\{(c,b),(b,a)\}}$ and $(cba, cab) \in s_{\{(c,b),(b,a)\}}$. We can show that the procedure will terminate in a finite number of steps, however, so each word has an image through such a sequence of swaps. It follows that s_Π is co-surjective.

We say that a set of action pairs Π is *swappable* for process p if $n_\Pi(p) = s_\Pi(p)$. Generalizing the NS rule of [18], a DI process is defined by the property that swappability holds for all pairs of two inputs, two outputs, and of one output and one input (in this order), but not one input and one output (in this order).

Since n_Π is injective and s_Π is co-surjective, Theorems 4.2 and 4.4 allow for exact verifications on the images of such processes through n_Π. This leads to a verification flow where the $n_\Pi(p) = s_\Pi(p)$ rule is checked inexpensively component-wise, permitting to simplify the expensive system-wide verifications by taking into account only executions that have the desired interleaving of events of Π, i.e., only executions from the image of n_Π.

5.6 Semihiding

In [16], we have proposed a new concurrency operator, called semi-hiding, for analyzing digital systems that ignore passive edges on their control signals. Unlike hiding, semi-hiding does not eliminate all transitions of a certain signal, but only events on certain positions in the execution traces.

We construct semi-hiding as a process abstraction, as follows. We use the following notation:

- $\#_a t$ is the number of occurrences of a in t, for any $a \in \Sigma$ and $t \in \Sigma^*$.

- $Z_+ = \{1, 2, 3, \ldots\}$ is the set of positive integers (non-zero natural numbers).

The semi-hiding relation is the graph of the sh function defined recursively as follows. For any $F \in \Sigma \times Z_+$ and $t \in \Sigma^*$, $\mathrm{sh}_F(t)$ is:

- ε for $t = \varepsilon$;
- $\mathrm{sh}_F(u)$ for $u \in \Sigma*$, $a \in \Sigma$, $(a, \#_a t) \in F$ and $t = ua$;
- $\mathrm{sh}_F(u)a$ for $u \in \Sigma^*$, $a \in \Sigma$, $(a, \#_a t) \notin F$ and $t = ua$.

For instance, $\mathrm{sh}_{\{d\}\times\{2,4,\ldots\}}(dabddadbda) = dabdabda$ and $\mathrm{sh}_{\{d\}\times\{1,3,\ldots\}}(dabddadbda) = abdadba$. (Sequence indices start at 1.)

It suffices to note that semi-hiding is co-surjective to conclude that any verifications in the semi-hiding co-domain can be done pessimistically. Moreover, specifications that ignore occurrence of passive edges of certain signals (such as positive-edge-triggered flip-flops, self-resetting logic, etc.) can be defined as being semi-hiding-optimistic. This offers automatically verifiable criteria for compliance to several asynchronous and synchronous protocols that are based on active edges of control signals [16].

6 Conclusion

This paper proposes a novel framework for studying process abstractions. We show that the structure of executions is of no consequence for several algebraic properties of process abstractions, including significant criteria for performing optimistic and pessimistic verifications. We show that this framework includes several common maps on concurrent processes, such as derivative, projection, and swapping of events. Our study also proposes several new maps, which do not exist in other theories of concurrency but are nevertheless significant: semihiding enables verification of compliance to several clocking and handshaking protocols used in digital circuits, while rotation reveals that meet and product (alternative and parallel compositions) are isomorphic. It follows that process abstractions defined as preserving the respective operators are also related by rotation.

References

[1] R. Alur, D. L. Dill. A theory of timed automata. In *Theoretical Computer Science* 126:183-235, 1994.

[2] R. Berks, R. Negulescu. Partial-order correctness-preserving properties of delay-insensitive circuits. In *Proc. 7th Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, Salt Lake City, USA, March 2001. (Best paper finalist.)

[3] G. Birkhoff. *Lattice Theory.* American Mathematical Society, 1967.

[4] J. A. Brzozowski. Derivatives of regular expressions. *Journal of the ACM*, 11(4):481–494, 1964.

[5] J. A. Brzozowski, J. J. Lou, R. Negulescu. A characterization of finite ternary algebras. In *International Journal of Algebra and Computation*, 7(6):713-721, 1997.

[6] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. In *Proceedings of the Symposium on Principles of Programming Languages*, pages 343–354, 1992.

[7] P. Cousot and R. Cousot. Abstract interpretation frameworks. In *J. Logic and Comp.*, 2(4): 511–547, 1992.

[8] D. L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. MIT Press, 1989. (ACM distinguished dissertation.)

[9] D. L. Dill and H. Wong-Toi. Verification of real-time systems by successive over and under approximations. In *Proc. International Workshop on Computer Aided Verification*, pages 409–422, 1995.

[10] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[11] R. Kurshan and K. McMillan. Analysis of digital circuits through symbolic reduction. *IEEE Transactions on Computer-Aided Design*, 10(11):1356–1371, 1991.

[12] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. In *Formal Methods in System Design*, (6):1–35, 1995.

[13] K. L. McMillan. A compositional rule for hardware design refinement. In *Proceedings of Computer-Aided Verification (CAV)*, Lecture Notes in Computer Science 1254, pages 24-35. Springer-Verlag, 1997.

[14] R. Negulescu. *Process Spaces and Formal Verification of Asynchronous Circuits*. Ph.D. thesis, Univ. of Waterloo, Canada, 1998.
`http://www.macs.ece.mcgill.ca/~radu/Papers/RN_thesis.ps.gz`

[15] R. Negulescu. Process spaces. In *Proc. 11th Int. Conf. on Concurrency Theory*, pages 196–210, 2000.

[16] R. Negulescu and X. Kong. Semi-hiding operators and the analysis of active-edge specifications for digital circuits. In *Proc. Int. Conf. on Application of Concurrency to System Design*, Newcastle, U.K., June 2001.

[17] J. Sifakis. Property preserving homomorphisms of transition systems. In E. Clarke and D. Kozen, editors, *Proceedings of the 4th Workshop on Logics of Programs*, Pittsburgh, U.S.A., June 1983.

[18] T. Verhoeff. *A Theory of Delay-Insensitive Systems*. PhD thesis, Dept. of Math. and C.S., Eindhoven Univ. of Technology, May 1994.