Draft (31 July 2001)
53 pages

A Truly Concurrent Semantics for a Simple Parallel Programming Language

Paul Gastin

LIAFA, Université Paris 7, 2 Place Jussieu, F-75251 Paris Cedex 05

Michael Mislove [1]

Department of Mathematics, Tulane University, New Orleans, LA 70118

Abstract

This paper gives the first truly concurrent denotational, i.e., compositional semantics for a simple, deterministic parallel language. By *truly concurrent* we mean the denotational model does not rely on an interleaving of concurrent actions for its definition. Thus, our semantics does not reduce parallelism to nondeterminism, as is done in the more established approaches to concurrency. We also present a natural SOS-style operational semantics for our language, and we prove a congruence theorem relating the two semantics. This implies the operational model itself is compositional. The congruence theorem also implies the denotational model is adequate with respect to the operational semantics, and we characterize the relatively mild conditions under which the denotational semantics is fully abstract with respect to the operational semantics.

Our simple language includes a (weak) sequential composition operator which takes advantage of the truly concurrent nature of the semantics, as well as a parallel composition operator which allows local events to execute asynchronously, while requiring synchronizing events to execute simultaneously. In addition, the language supports a restriction operator and includes recursion.

The denotational semantics also is novel for its treatment of recursion. The meaning of a recursive process is defined using a least fixed point on a *subdomain* that is determined by the body of the recursion, and that varies from one process to another. Nonetheless, the recursion operators in the language have continuous interpretations in the denotational model.

Our denotational model is based on a domain-theoretic generalization of Mazurkiewicz traces in which the concatenation operator, as well as the other operators from our language can be given continuous interpretations.

[1] The support of the National Science Foundation and the US Office of Naval Research is gratefully acknowledged.

1 Introduction

The standard approach to modelling parallel composition in process algebra is to take sequential composition and nondeterministic choice as primitive operations and then to use an interleaving approach to model parallel composition. This approach was laid out in the seminal work of Hennessy and Plotkin [7], where they showed how power domains could be used to provide the denotational models necessary for such an approach.

On the other hand, the literature abounds with truly concurrent models of computation, including, for example, prime event structures [17] and pomsets [13], as well as many others (cf., e.g., [2,11,12]). By *truly concurrent* we mean the model does not rely on an interleaving of concurrent actions for its definition. These truly concurrent models have been used to describe the behavior of concurrent transition systems such as Petri nets [15]. Remarkably, while all of these models are domains, none of them were used to give a *denotational* semantics for a parallel language. A denotational model – by its very nature – is *compositional*, so that the meaning of a compound process is the composition in the model of the meanings of its constituent parts.

The contribution of this paper is to present the first truly concurrent denotational semantics for a simple, deterministic parallel language. This language includes a number of interesting operators. The atomic actions from which processes are composed are assigned resources representing what they need to execute. Two actions are then *independent* if they share no resources. The *weak sequential composition* of two processes then allows an action from the second process to start as soon as its needed resources are available, even if the first process is still active. On the other hand, dependent actions still need to occur in the order specified. We think this is a very attractive feature that corresponds to the automatic parallelization of processes. We also can force processes to execute sequentially by using a blocking event between them on which all their events are dependent.

The language also includes a *parallel composition operator* which allows local events to occur asynchronously, while requiring synchronizing events to occur simultaneously. In addition, the language supports a *restriction operator* which confines processes to specified sets of resources. This provides a mechanism for controlling the resources granted to processes. Finally, the language includes process variables and recursion.

Our denotational model is based on the resource traces model of Gastin and Teodosiu [5], which is a domain-theoretic generalization of Mazurkiewicz traces [8] in which the concatenation operator has a continuous interpretation. We augment this model with continuous interpretations of the other operators from our language. We need these continuous interpretations in order to give a fixed point semantics for recursion. It should be noted that we are not using the traditional least fixed point semantics for recursion (cf. [1]). Instead, the meaning of a recursive process is defined using a least fixed point on a *subdomain* that is determined by the resources needed by the body of the recursion, and that varies from one process to another. Remarkably, the recursion operators in the language have continuous interpretations in the denotational model.

In addition to the denotational model, we also present a natural SOS-style operational semantics for our language. The fact that we do not use a least fixed point semantics for the denotation of a recursive process – but instead one that first requires us to compute an appropriate resource set for each process – is reflected in the operational model as well.

Indeed, our transition rules for processes require the addition of a *resource environment* – a mapping from the set of variables in our language to the set of resources, so that the behavior of a process varies from resource environment to resource environment. We believe this approach is novel, certainly in the world of process algebra.

We prove a congruence theorem showing that the behavior function defined by the operational semantics assigns the same unique resource trace to a process as does the denotational model. In addition to implying the operational model itself is compositional, this immediately implies the denotational semantics is adequate with respect to the operational model. Finally, we characterize the relatively mild conditions under which the denotational model is fully abstract with respect to the operational semantics.

This paper represents the beginning of a longer-range project whose goal is to develop a truly concurrent denotational semantics for a process algebra that has the expressive power of the more traditional languages – such as *CSP* [16] and *CCS* [9], and to then compare the relative strength of this approach with the more established ones. But the focus in this paper is as much on the mathematical techniques that have been developed, as on the language being modelled. Indeed, the language we study here is not as expressive as the traditional process algebras since we deliberately left out all operators involving nondeterminism in order to concentrate only on concurrency-based operators. Therefore, our simple parallel language has only a few, well-chosen operators, and these are included not because of specific applications to computing, but instead to illustrate the mathematical ideas and techniques that have evolved in this work.

The rest of the paper is organized as follows. In the next section, we provide some preliminary background on domain theory and on the resource traces model of Gastin and Teodosiu [5]. These are the main ingredients of our approach to providing semantic models for our language. The syntax of our language is the subject of the next section, and this is followed by a section in which we explore the properties of the resource mapping that assigns to each process its set of resources. The results of this section are needed for both the operational and denotational semantics. There follows a section detailing the operational semantics of our language, and then one giving the denotational model. Section 7 is devoted to our main theorem which states that the operational semantics is congruent with the denotational semantics. The last section presents our results about adequacy and full abstraction, and the paper closes with a short dsicussion on what we have achieved and on future work.

2 Preliminaries

In this section we review some basic results from domain theory, and then some results from trace theory. A standard reference for domain theory is [1], and most of the results we cite can be found there. We provide specific references for those results which can be found elsewhere. Similarly, for the theory of traces the reader is referred to [4]. Specific results on resource traces can be found in [5]. Section 2.3 contains new results.

2.1 Domain Theory

To begin, a *poset* is a partially ordered set, usually denoted P. The least element of P (if it exists) is denoted $\perp$, and a subset $D \subseteq P$ is *directed* if each finite subset $F \subseteq D$ has an upper bound in D. Note that since $F = \emptyset$ is a possibility, a directed subset must be non-empty. A *(directed) complete partial order (dcpo)* is a poset P in which each directed set has a least upper bound. If P also has a least element, then it is called a *cpo*. If P and Q are posets and $f\colon P \to Q$ is a monotone map, then f is *(Scott) continuous* if f preserves sups of directed sets: if $D \subseteq P$ is directed and $x = \sqcup D \in P$ exists, then $\sqcup f(D) \in Q$ exists and $f(\sqcup D) = \sqcup f(D)$.

If P is a dcpo, the element $k \in P$ is *compact* if, for each directed subset $D \subseteq P$, if $k \sqsubseteq \sqcup D$, then $(\exists d \in D)\ k \sqsubseteq d$. The set of compact elements of P is denoted $K(P)$, and for an element $x \in P$, $K(x) = K(P) \cap \downarrow x$, where $\downarrow x = \{y \in P \mid y \sqsubseteq x\}$. P is *algebraic* if $K(x)$ is directed and $x = \sqcup K(x)$ for each $x \in P$.

Similarly, an element p of a dcpo P is *(complete) prime* if for each subset $X \subseteq P$ which have a least upper bound, $p \leq \sqcup X$ implies $p \leq x$ for some $x \in X$. The set of prime elements below some element $x \in P$ is denoted by $Pr(x)$. P is p-algebraic if $x = \sqcup Pr(x)$ for each $x \in P$.

One of the things that makes cpos useful is the following result, which provides a uniform method for assigning meanings to recursive processes that the cpo is being used to model:

Theorem 2.1 (Tarski, Knaster, Scott) *If $f\colon P \to P$ is a continuous selfmap of a cpo, then f has a least fixed point given by* $\mathrm{fix}(f) = \sqcup_{n\geq 0} f^n(\perp)$ □

Actually, Tarski proved that the set of fixed points of a monotone selfmap of a complete lattice is a complete sublattice (whose least element is the least fixed point), and Knaster and Scott added the result that the fixed point is achieved in countably many iterations if the selfmap is continuous, under varying assumptions.

We will use Theorem 2.1, but in a somewhat unconventional way. Instead of defining the meaning of recursive constructs in our models via least fixed points, they will be defined as the least fixed point of a selfmap of a subdomain of the model determined by the resources the recursive process requires to complete its computation. Even in this setting, the fixed point operator is continuous; more details are provided when we actually define our denotational semantics.

2.2 Resource Traces

Mazurkiewicz trace theory begins with an alphabet Σ of actions that can be executed by processes, and a reflexive, symmetric *dependency relation* $D \subseteq \Sigma \times \Sigma$ which defines which actions are *dependent.* The actions $a, b \in \Sigma$ are *independent* if and only if $(a, b) \notin D$. In trace theory, independent actions can occur concurrently while dependent actions must be ordered. The concatenation operator from trace theory takes advantage of this by allowing the beginning of the second process to occur independently of the end of the first process provided these events are independent. Therefore, this concatenation is only *weakly sequential.*

Though Mazurkiewicz traces with the prefix ordering form a domain, the concatenation operator does not have a continuous interpretation on this domain. In order to overcome this problem, Diekert, Gastin and Teodosiu [3,5] generalized Mazurkiewicz trace theory by introducing trace based domains with a new approximation ordering and on which the concatenation operator can be given a Scott-continuous continuous interpretation. This opens the way for giving truly concurrent denotational semantics for parallel languages.

In this paper, we have chosen to use the *resource traces* model of Gastin and Teodosiu [5] as the basis for the denotational model for our language. In this approach, things are a bit more concrete than described above. We start with a finite alphabet Σ of actions and a finite set of *resources* $\mathcal{R}$, as well as a function $\mathrm{res}\colon \Sigma \to \mathcal{P}(\mathcal{R})$ which assigns to each action a *non-empty* set of resources it needs in order to execute. One can then define the reflexive and symmetric dependency relation D by

$$D = \{(a, b) \in \Sigma \times \Sigma \mid \mathrm{res}(a) \cap \mathrm{res}(b) \neq \emptyset\}.$$

Its complement $I = (\Sigma \times \Sigma) \setminus D$ is called the *independence relation* on Σ. The fact that each action must have some resources means there is no *auto concurrency* – instances of actions that are independent of themselves.

A *real trace* t over (Σ, res) is the isomorphism class of a labeled, directed acyclic graph $t = [V, E, \lambda]$, where V is a countable set of *events*, $E \subseteq V \times V$ is the *synchronization relation* on V, and $\lambda\colon V \to \Sigma$ is a node-labeling satisfying

- $\forall p \in V,\ {\downarrow}p = \{q \in V \mid (q, p) \in E^*\}$ is finite,
- $\forall p, q \in V,\ (\lambda(p), \lambda(q)) \in D \iff (p, q) \in E \cup E^{-1} \cup \{(p, p) \mid p \in V\}$

The trace t is finite if V is finite and the length of t is $|t| = |V|$. The sets of finite traces and of real traces over (Σ, res) are denoted by $\mathbb{M}(\Sigma, \mathrm{res})$ and by $\mathbb{R}(\Sigma, \mathrm{res})$, or simply by $\mathbb{M}$ and $\mathbb{R}$, respectively. We denote by $1 = (\emptyset, \emptyset, \emptyset)$ the empty trace.

The alphabet of a real trace t is the set $\mathrm{alph}(t) = \lambda(V)$ of letters which occur in t. We also define the *alphabet at infinity* of t as the set $\mathrm{alphinf}(t)$ of letters which occur infinitely often in t. We extend the resource mapping to real traces by defining $\mathrm{res}(t) = \mathrm{res}(\mathrm{alph}(t)) = \bigcup\{\mathrm{res}(a) \mid a \in \mathrm{alph}(t)\}$. The *resources at infinity* of t is the set $\mathrm{resinf}(t) = \mathrm{res}(\mathrm{alphinf}(t))$. A real trace is finite iff $\mathrm{alphinf}(t) = \mathrm{resinf}(t) = \emptyset$.

A partial concatenation operation is defined on real traces as follows: If $t_1 = [V_1, E_1, \lambda_1]$ and $t_2 = [V_2, E_2, \lambda_2]$ are real traces such that $\mathrm{resinf}(t_1) \cap \mathrm{res}(t_2) = \emptyset$, then the concatenation of t_1 and t_2 is the real trace $t_1 \cdot t_2 = [V, E, \lambda]$ obtained by taking the disjoint union of t_1 and t_2 and adding necessary edges from t_1 to t_2, i.e., $V = V_1 \dot\cup V_2$, $\lambda = \lambda_1 \dot\cup \lambda_2$, and $E = E_1 \dot\cup E_2 \dot\cup (V_1 \times V_2 \cap \lambda^{-1}(D))$. In this representation, the *empty trace* $1 = [\emptyset, \emptyset, \emptyset]$ is the identity.

The prefix ordering is defined on real traces by $r \leq t$ iff there exists a real trace s such that $t = r \cdot s$. In other words, a prefix of t is a downward closed subgraph of t. When $r \leq t$, then the trace s satisfying $t = r \cdot s$ is unique and is denoted by $r^{-1}t$. $(\mathbb{R}, \leq)$ is a dcpo with the empty trace as least element. The compact elements of $(\mathbb{R}, \leq)$ are exactly the finite traces. A real trace $t = [V, E, \lambda]$ is prime iff it is finite and has exactly one maximal vertex.

The monoid of finite traces $(\mathbb{M}, \cdot)$ is isomorphic to the quotient monoid $\Sigma^*/{\sim}$ of the free monoid Σ^* of finite words over Σ, modulo the least congruence generated by $\{(ab, ba) \mid (a, b) \in I\}$. Similarly, an infinite real trace $t = [V, E, \lambda]$ may be identified with the equivalence class of infinite words that are linearizations of the graph $[V, E, \lambda]$.

Just as in the case of the concatenation of words, the concatenation operation on $\mathbb{M}$ is not monotone with respect to the prefix order. It is for this reason that $\mathbb{M}$ cannot be completed into a dcpo on which concatenation is continuous, and so it is not clear how to use traces as a basis for a domain-theoretic model for the concatenation operator of trace theory.

This shortcoming was overcome by the work of Diekert, Gastin and Teodosiu [3,5]. In this paper, we will use the latter work as a basis for the denotational models for our language. The *resource trace domain* over $(\Sigma, \mathcal{R}, \mathrm{res})$ is then defined to be the family

$$\mathbb{F}(\Sigma, \mathrm{res}) = \{(r, R) \mid r \in \mathbb{R}(\Sigma, \mathrm{res}),\ R \subseteq \mathcal{R} \text{ and } \mathrm{resinf}(r) \subseteq R\}.$$

This set will be simply denoted by $\mathbb{F}$ when no ambiguities may occur. For a resource trace $x = (r, R) \in \mathbb{F}$, we call $\mathrm{Re}(x) = r$ the *real part* of x and $\mathrm{Im}(x) = R$ the *imaginary part* of x. Most resource traces are meant to describe approximations of actual processes. The real part describes what has already been observed from the process and the imaginary part is the set of resources allocated to the process for its completion. The set of resource traces $\mathbb{F}$ is thus endowed with a partial order called the *approximation order:*

$$(r, R) \sqsubseteq (s, S) \ \Leftrightarrow\ r \leq s \text{ and } R \supseteq S \cup \mathrm{res}(r^{-1}s).$$

We also endow $\mathbb{F}$ with the concatenation operation

$$(r, R) \cdot (s, S) = (r \cdot \mu_R(s), R \cup S \cup \sigma_R(s)),$$

where $\mu_R(s)$ is the largest prefix u of s satisfying $\mathrm{res}(u) \cap R = \emptyset$ and $\sigma_R(s) = \mathrm{res}(\mu_R(s)^{-1}s)$. Intuitively, the product $(r, R) \cdot (s, S)$ is the best approximation we can compute for the composition of two processes if we only know their approximations (r, R) and (s, S).

It turns out that $(\mathbb{F}, \sqsubseteq)$ is a dcpo with least element $(1, \mathcal{R})$, where 1 is the empty trace. Moreover, the concatenation operator defined above is continuous with respect to this order. In other words, $(\mathbb{F}, \sqsubseteq, \cdot)$ is a continuous algebra in the sense of domain theory. The dcpo $(\mathbb{F}, \sqsubseteq)$ is also algebraic and a resource trace $x = (r, R)$ is compact if and only if it is finite, that is, iff its real part r is finite.

The dcpo $(\mathbb{F}, \sqsubseteq)$ is also a *prime algebraic domain* in the sense of Winskel [17]. Each resource trace x is the least upper bound of the prime resource traces below x. We also know a characterization of prime traces: the resource trace $(r, R) \in \mathbb{F}$ is prime iff either $r = 1$ and $|R| = |\mathcal{R}| - 1$ or r is a prime real trace with its maximal vertex labeled with a letter $a \in \Sigma$ such that $R \cup \mathrm{res}(a) = \mathcal{R}$.

We close this section with a simple result about the resource mapping. For this, we order the set $\mathcal{P}(\mathcal{R})$ with reverse containment in order to get the cpo $(\mathcal{P}(\mathcal{R}), \supseteq)$. We are using reverse containment rather than natural containment because there is a natural continuous embedding from the lattice of resource sets $(\mathcal{P}(\mathcal{R}), \supseteq)$ into the domain of resource traces (defined by $R \mapsto (1, R)$) if we are using the reverse containment. Moreover, the mapping res can be extended to a continuous map from $(\mathbb{F}, \sqsubseteq)$ to $(\mathcal{P}(\mathcal{R}), \supseteq)$ as stated below.

Proposition 2.2 *The resource mapping* $\mathrm{res}\colon \Sigma \to \mathcal{P}(\mathcal{R})$ *extends to a continuous mapping* $\mathrm{res}\colon (\mathbb{F}, \sqsubseteq) \to (\mathcal{P}(\mathcal{R}), \supseteq)$ *defined by* $\mathrm{res}(r, R) = \mathrm{res}(r) \cup R$.

Proof. By definition, if $(r, R) \in \mathbb{F}$, then $\mathrm{res}(r) = \cup\{\mathrm{res}(a) \mid a \in \mathrm{alph}(r)\}$. If $(r, R) \sqsubseteq (s, S)$, then $r \leq s$ and $R \supseteq S \cup \mathrm{res}(r^{-1}s)$, and so

$$\mathrm{res}(r, R) = \mathrm{res}(r) \cup R \supseteq \mathrm{res}(r) \cup \mathrm{res}(r^{-1}s) \cup S = \mathrm{res}(s) \cup S = \mathrm{res}(s, S).$$

Thus $\mathrm{res}\colon (\mathbb{F}, \sqsubseteq) \to (\mathcal{P}(\mathcal{R}), \supseteq)$ is monotone.

If $X \subseteq \mathbb{F}$ is consistent (i.e., X has an upper bound), then ([?]) $\sqcup X = (\mathrm{Re}(\sqcup X), \mathrm{Im}(\sqcup X))$, where

$$\mathrm{Re}(\sqcup X) = \bigsqcup_{x \in X} \mathrm{Re}(x)$$

and

$$\mathrm{Im}(\sqcup X) = \bigcap_{x \in X} \mathrm{Im}(x).$$

Since res is monotone, we have $\sqcup \mathrm{res}(X) \supseteq \mathrm{res}(\sqcup X)$. Conversely, let $\alpha \in \sqcup \mathrm{res}(X) = \cap_{x \in X} \mathrm{res}(x)$. Either $\alpha \in \mathrm{res}(\mathrm{Re}(x))$ for some $x \in X$. In this case, we have

$$\alpha \in \mathrm{res}(\mathrm{Re}(x)) \subseteq \mathrm{res}(\mathrm{Re}(\sqcup X)) \subseteq \mathrm{res}(\sqcup X).$$

Or $\alpha \in \cap_{x \in X} \mathrm{Im}(x)$ and we can conclude since

$$\cap_{x \in X} \mathrm{Im}(x) = \mathrm{Im}(\sqcup X) \subseteq \mathrm{res}(\sqcup X).$$

□

2.3 Alphabetic Mappings

We now present some results about alphabetic mappings over real traces and over resource traces. These results are new and will be useful for the denotational semantics of our parallel composition operator (cf. Section 6).

Let $\mathrm{res} : \Sigma \to \mathcal{P}(\mathcal{R})$ and $\mathrm{res}' : \Sigma' \to \mathcal{P}(\mathcal{R})$ be two resource maps over the alphabets Σ and Σ'. The associated dependence relations over Σ and Σ' are denoted by D and D'.

Let $\varphi : \Sigma \to \Sigma' \cup \{1\}$ be an alphabetic mapping such that $\mathrm{res}'(\varphi(a)) \subseteq \mathrm{res}(a)$ for all $a \in \Sigma$. We extend φ to real traces. If $r = [V, E, \lambda] \in \mathbb{R}(\Sigma, \mathrm{res})$, then we define $\varphi(r) = [V', E', \lambda']$ by

- $V' = \{e \in V \mid \varphi \circ \lambda(e) \neq 1\}$,
- $\lambda' = \varphi \circ \lambda$ and
- $E' = E \cap \lambda'^{-1}(D') = \{(e, f) \in E \mid \lambda'(e)\ D'\ \lambda'(f)\}$.

Proposition 2.3

(i) *For all* $r \in \mathbb{R}(\Sigma, \mathrm{res})$*,* $\varphi(r)$ *is a real trace over* (Σ', res')*. Therefore* φ *is a mapping from* $\mathbb{R}(\Sigma, \mathrm{res})$ *to* $\mathbb{R}(\Sigma', \mathrm{res}')$*.*

(ii) $\varphi : (\mathbb{R}(\Sigma, \mathrm{res}), \cdot) \to (\mathbb{R}(\Sigma', \mathrm{res}'), \cdot)$ *is a morphism.*

(iii) $\varphi : (\mathbb{R}(\Sigma, \mathrm{res}), \leq) \to (\mathbb{R}(\Sigma', \mathrm{res}'), \leq)$ *is continuous.*

Proof. (i) Since E is acyclic, $E' \subseteq E$ is acyclic as well and for each vertex $e \in V'$, $\{f \in E' \mid (f, e) \in E'^*\} \subseteq \{f \in E \mid (f, e) \in E^*\}$ is finite. Now, if $(e, f) \in E'$, then by definition, $\lambda'(e)\ D'\ \lambda'(f)$. Conversely, let $e, f \in E'$ such that $\lambda'(e)\ D'\ \lambda'(f)$. Since $\mathrm{res}'(\varphi(a)) \subseteq \mathrm{res}(a)$ for all $a \in \Sigma$, we deduce that $\lambda(e)\ D\ \lambda(f)$. Hence, either $e = f$ or $(e, f) \in E$ or $(f, e) \in E$. In the last two cases, it follows $(e, f) \in E'$ or $(f, e) \in E'$.

(ii) Let $r_1 = (V_1, E_1, \lambda_1)$ and $r_2 = (V_2, E_2, \lambda_2)$ be two real traces over Σ such that $\mathrm{resinf}(r_1) \cap \mathrm{res}(r_2) = \emptyset$ [2]. The product is $r = r_1 \cdot r_2 = (V, E, \lambda)$ with $V = V_1 \dot\cup V_2$, $\lambda = \lambda_1 \dot\cup \lambda_2$ and $E = E_1 \dot\cup E_2 \dot\cup \{(e, f) \in V_1 \times V_2 \mid \lambda_1(e)\ D\ \lambda_2(f)\}$. Now, let $r_1' = \varphi(r_1) = (V_1', E_1', \lambda_1')$, $r_2' = \varphi(r_2) = (V_2', E_2', \lambda_2')$ and $r' = \varphi(r) = (V', E', \lambda')$. Our definition for extending φ to real traces then implies that $V' = V_1' \dot\cup V_2'$, $\lambda' = \lambda_1' \dot\cup \lambda_2'$ and

$$\begin{aligned} E' &= E_1' \dot\cup E_2' \dot\cup \{(e, f) \in V_1 \times V_2 \mid \lambda_1(e)\ D\ \lambda_2(f) \text{ and } \lambda_1'(e)\ D'\ \lambda_2'(f)\} \\ &= E_1' \dot\cup E_2' \dot\cup \{(e, f) \in V_1' \times V_2' \mid \lambda_1'(e)\ D'\ \lambda_2'(f)\} \end{aligned}$$

Therefore, $r' = r_1' \cdot r_2'$.

(iii) Since the order on real traces is the prefix order, from the previous point we deduce that φ is monotone. Now, let $X \subseteq \mathbb{R}(\Sigma, \mathrm{res})$ be any set such that $\sqcup X = r = (V, E, \lambda)$ exists. Then $\varphi(X) \subseteq {\downarrow}\varphi(\sqcup X)$, so $\sqcup\varphi(X)$ exists and $\sqcup\varphi(X) \leq \varphi(\sqcup X) = r' = (V', E', \lambda')$. Conversely, let $e \in V'$, let p' be the prime prefix of r' defined by $V_{p'} = \{f \in V' \mid (f, e) \in E'^*\}$ and let p be the prime prefix of r defined by $V_p = \{f \in V \mid (f, e) \in E^*\}$. Since $p \leq r$, we have $\varphi(p) \leq \varphi(r) = r'$. Now p' is a prime prefix of r' whose maximal vertex is contained in the prefix $\varphi(p)$ of r'. Therefore, p' is a prefix of $\varphi(p)$. Finally, p is a prime prefix of $r = \sqcup X$, so there exists $x \in X$ such that $p \leq x$. Hence, $p' \leq \varphi(p) \leq \varphi(x) \leq \sqcup\varphi(X)$. Since this holds for all prime prefixes p' of $r' = \varphi(\sqcup X)$, it follows that $r' = \varphi(\sqcup X) \leq \sqcup\varphi(X)$. □

Note that $\varphi : \mathbb{R}(\Sigma, \mathrm{res}) \to \mathbb{R}(\Sigma', \mathrm{res}')$ may also be defined via words. Let $\psi : \Sigma^\infty \to \Sigma'^\infty$ be defined by $\psi(a_1 a_2 \cdots) = \varphi(a_1)\varphi(a_2)\cdots$. Then, the following diagram commutes:

$$\begin{array}{ccc} \Sigma^\infty & \xrightarrow{\psi} & \Sigma'^\infty \\ {\scriptstyle []}\big\downarrow & & \big\downarrow{\scriptstyle []} \\ \mathbb{R}(\Sigma, \mathrm{res}) & \xrightarrow{\varphi} & \mathbb{R}(\Sigma', \mathrm{res}') \end{array}$$

where $[\,]$ denotes the canonical mapping from words to traces.

Indeed, $\varphi([a]) = \varphi(a) = [\psi(a)]$ for all $a \in \Sigma$ and since φ, ψ and $[]$ are morphisms we deduce that $\varphi([u]) = [\psi(u)]$ for all $u \in \Sigma^*$. Now, φ, ψ and $[]$ are also continuous and for all $u \in \Sigma^\infty$, we obtain $\varphi([u]) = \sqcup_n \varphi([u_n]) = \sqcup[\psi(u_n)] = [\psi(u)]$ where (u_n) is any increasing sequences of words which converges to u.

This implies in particular that if u and v are two equivalent words over Σ, then $\psi(u)$ and $\psi(v)$ are equivalent words over Σ'. Also, in order to compute $\varphi(r)$ for some real trace r, one may take any linearization u of r ($r = [u]$) and compute $[\psi(u)]$.

We now extend φ to a mapping over resource traces of $\mathbb{F}(\Sigma, \mathrm{res})$ simply by defining $\varphi(r, R) = (\varphi(r), R)$. Since $\mathrm{res}'(\varphi(a)) \subseteq \mathrm{res}(a)$ for all $a \in \Sigma$, we deduce that $\mathrm{resinf}'(\varphi(r)) \subseteq \mathrm{resinf}(r) \subseteq R$ and $(\varphi(r), R)$ is a resource trace over Σ'. Hence, $\varphi : \mathbb{F}(\Sigma, \mathrm{res}) \to \mathbb{F}(\Sigma', \mathrm{res}')$ is well defined.

Note that $\varphi : \mathbb{F}(\Sigma, \mathrm{res}) \to \mathbb{F}(\Sigma', \mathrm{res}')$ is not necessarily a morphism. Indeed, consider the

[2] Note that, if $\mathrm{resinf}(r_1) \cap \mathrm{res}(r_2) \neq \emptyset$ then $r_1 \cdot r_2$ would not be a real trace since it would have vertices with infinite past.

mappings

$$\begin{aligned} \mathrm{res}: a &\mapsto \{\alpha\} \\ b &\mapsto \{\alpha,\gamma\} \\ c &\mapsto \{\gamma\} \end{aligned} \qquad\qquad \begin{aligned} \varphi : a &\mapsto a \\ b &\mapsto 1 \\ c &\mapsto c \end{aligned}$$

Then, we have $\varphi((a^\omega,\{\alpha\})\cdot(bc,\emptyset)) = (a^\omega,\{\alpha,\gamma\})$ but $\varphi((a^\omega,\{\alpha\}))\cdot\varphi((bc,\emptyset)) = (ca^\omega,\{\alpha\})$. What is more surprising is that $\varphi : \mathbb{F}(\Sigma,\mathrm{res}) \to \mathbb{F}(\Sigma',\mathrm{res}')$ is not even a morphism when $\varphi : \Sigma \to \Sigma'$ is a non-erasing mapping. Indeed, consider now $\Sigma = \Sigma' = \{a, c\}$ with

$$\begin{aligned} \mathrm{res}: a &\mapsto \{\alpha,\beta\} \\ c &\mapsto \{\beta,\gamma\} \end{aligned} \qquad\qquad \begin{aligned} \mathrm{res}' : a &\mapsto \{\alpha\} \\ c &\mapsto \{\gamma\} \end{aligned}$$

Then the identity from $(\mathbb{F}(\Sigma,\mathrm{res}),\cdot)$ to $(\mathbb{F}(\Sigma',\mathrm{res}'),\cdot)$ is not a morphism.

Proposition 2.4

(i) $\varphi : (\mathbb{F}(\Sigma,\mathrm{res}),\sqsubseteq) \to (\mathbb{F}(\Sigma',\mathrm{res}'),\sqsubseteq)$ *is continuous.*

(ii) *If* $\mathrm{res}'(\varphi(a)) = \mathrm{res}(a)$ *for all* $a \in \Sigma$ *then* $\varphi : (\mathbb{F}(\Sigma,\mathrm{res}),\cdot) \to (\mathbb{F}(\Sigma',\mathrm{res}'),\cdot)$ *is a (non-erasing) morphism.*

Proof. (i) Let $(r,R),(s,S) \in \mathbb{F}(\Sigma,\mathrm{res})$ be such that $(r,R) \sqsubseteq (s,S)$. Since $\varphi : \mathbb{R}(\Sigma,\mathrm{res}) \to \mathbb{R}(\Sigma',\mathrm{res}')$ is a morphism, we have $\varphi(r) \le \varphi(s)$ and $\varphi(r)^{-1}\varphi(s) = \varphi(r^{-1}s)$. Hence, we obtain $\mathrm{res}'(\varphi(r)^{-1}\varphi(s)) \subseteq \mathrm{res}(r^{-1}s) \subseteq R$ and we deduce that $(\varphi(r),R) \sqsubseteq (\varphi(s),S)$.

Now, let X be any subset of $\mathbb{F}(\Sigma,\mathrm{res})$ such that $\sqcup X$ exists. Then $\sqcup\varphi(X)$ exists,

$$\begin{aligned} \mathrm{Re}(\sqcup\varphi(X)) &= \bigsqcup_{x\in X}\mathrm{Re}(\varphi(x)) = \bigsqcup_{x\in X}\varphi(\mathrm{Re}(x)) = \varphi(\bigsqcup_{x\in X}\mathrm{Re}(x)) \\ &= \varphi(\mathrm{Re}(\sqcup X)) = \mathrm{Re}(\varphi(\sqcup X)), \end{aligned}$$

and

$$\mathrm{Im}(\sqcup\varphi(X)) = \bigcap_{x\in X}\mathrm{Im}(\varphi(x)) = \bigcap_{x\in X}\mathrm{Im}(x) = \mathrm{Im}(\sqcup X) = \mathrm{Im}(\varphi(\sqcup X)).$$

Therefore, $\sqcup\varphi(X) = \varphi(\sqcup X)$.

(ii) For all $a \in \Sigma$, $\mathrm{res}'(\varphi(a)) = \mathrm{res}(a) \neq \emptyset$, so $\varphi(a) \neq 1$. In this case, we also have $a\,D\,b$ iff $\varphi(a)\,D'\,\varphi(b)$. Hence, if $r = (V,E,\lambda) \in \mathbb{R}(\Sigma,\mathrm{res})$ we obtain $\varphi(r) = (V,E,\varphi\circ\lambda)$.

Let $(r,R),(s,S)$ be two resource traces over (Σ,res). Using again the hypothesis we obtain $\mu_R(\varphi(s)) = \varphi(\mu_R(s))$ and $\sigma_R(\varphi(s)) = \sigma_R(s)$. Therefore,

$$\begin{aligned} (\varphi(r),R)\cdot(\varphi(s),S) &= (\varphi(r)\cdot\mu_R(\varphi(s)), R\cup S\cup\sigma_R(\varphi(s))) \\ &= (\varphi(r\cdot\mu_R(s)), R\cup S\cup\sigma_R(s)) \\ &= \varphi((r,R)\cdot(s,S)) \end{aligned}$$

□

3 The Language

In this section we introduce a simple parallel programming language which utilizes the concatenation operator from trace theory as a basic mechanism for combining processes, replacing the more traditional sequential composition operator of process algebra. The goal is to present an operational and denotational model for this language based on the resource traces model of Gastin and Teodosiu [5]. We begin once again with a finite set Σ of atomic actions, a finite set $\mathcal{R}$ of resources, and a mapping $\mathrm{res}\colon \Sigma \to \mathcal{P}(\mathcal{R})$ which assigns to each $a \in \Sigma$ a *non-empty* set of resources. We view $\mathrm{res}(a)$ as the set of resources – memory, ports, etc. – that the action a needs in order to execute. Two actions $a, b \in \Sigma$ may be executed concurrently if and only if they are independent – i.e. iff they do not share any resource.

We define the BNF-like syntax of the language $\mathcal{L}$ we study as

$$p ::= \mathrm{STOP} \mid a \mid p \circ p \mid p|_R \mid p \mathop{\|}_{C} p \mid x \mid \mathrm{rec}\, x.p$$

where a ranges over Σ, R, C range over $\mathcal{P}(\mathcal{R})$ and x ranges over a countable set V of variables. Here,

- STOP is the process capable of no actions but claiming all resources; it is full deadlock.
- $a \in \Sigma$ denotes the process which can execute the action a and then terminate normally.
- $p \circ q$ denotes the weak sequential composition of the two argument processes with the understanding that independent actions commute with one another: $a \circ b = b \circ a$ if $a, b \in I$. We call $\circ$ *weak sequential composition* because it enforces sequential composition of those actions which are dependent, while allowing those which are independent of one another to execute concurrently.
- $p|_R$ denotes the process p with all resources restricted to the subset $R \subseteq \mathcal{R}$. The intention is that only those actions a from p can execute for which $\mathrm{res}(a) \subseteq R$; all other actions are disabled.
- $p \mathop{\|}_{C} q$ denotes the parallel composition of the component processes. It supports a parallel composition with synchronization on those events that share resources in C, which we view as the set of channels across which communication occurs. Local events, that is events from one process that do not share any resource with C or with the other process, may also occur independently.

  Since our semantics does not allow choices, this process can only make progress as long as there are no actions from one component that use resources that some action from the other component also uses, except in the case of synchronization actions. If this condition is violated, the process deadlocks.
- $x \in V$ is a process variable.
- $\mathrm{rec}\, x.p$ denotes recursion of the process body p in the variable x.

Some words about choice also are in order. One of the principal impetuses for our work is the desire to understand the differences between parallel composition, choice and nondeterminism. Historically, nondeterministic choice arose as a convenient means with which to model parallel composition: the parallel composition of sequential processes is modeled by the set of possible interleavings of the actions of each component. This quickly led to the development of power domains to provide a domain-theoretic analogue to the power set of a set.

In our approach, we are interested in exploring an alternative model for parallelism. A parallel composition involves choice whenever there is a competition between conflicting events. Since we use a truly concurrent semantic domain, our events are not necessarily in conflict and we can consider a very natural and important form of cooperative parallel composition which does not require choice or nondeterminism. Each process consists of local events which occur independently from the other process and of synchronization events which are executed in matching pairs. Indeed, these synchronization events may introduce conflict when the two processes offer non matching synchronization events. Since we want to stress the difference between concurrency and choice we have decided to block conflicting events in our parallel composition. Note that this situation does not occur in a cooperative parallel composition, e.g. in a parallel sorting algorithm.

It is important to understand the algebraic interpretation of our language $\mathcal{L}$. We can (and do) take the view that the BNF-like syntax given above is one way to express the signature of a universal algebra. In our case, the algebra is single sorted, and the signature, Ω, can be written as $\Omega = \cup_n \Omega_n$, where the index n denotes the arity of the operators in the subset Ω_n: we have

- Nullary operators: $\Omega_0 = \{STOP\} \cup \Sigma \cup V$,
- Unary operators: $\Omega_1 = \{-|_R \mid R \subseteq \mathcal{R}\} \cup \{\mathrm{rec}\, x.- \mid x \in V\}$,
- Binary operators: $\Omega_2 = \{\circ\} \cup \{ \underset{C}{\|} \mid C \subseteq \mathcal{R}\}$, and
- $\Omega_n = \emptyset$ for all other n.

With this view, $\mathcal{L}$ is the *initial* Ω*-algebra*. This means that, given any other Ω-algebra A (and we shall see several), there is a unique Ω-algebra homomorphism $\phi_A \colon \mathcal{L} \to A$, i.e., a unique compositional mapping from $\mathcal{L}$ to A.

A simple example is that this allows us to define a rank function $\rho \colon \mathcal{L} \to \mathbb{N}$ using this approach. This rank function is then the basis for the many arguments by *structural induction* that we shall use in this paper. The definition of ρ is straightforward; we begin by defining the structure of an Ω-algebra on $\mathbb{N}$:

- $[\![\mathrm{STOP}]\!] = [\![a]\!] = [\![x]\!] = 1$ for all $a \in \Sigma$ and all $x \in V$;
- $[\![|_R]\!] = [\![\mathrm{rec}\, x]\!] = \mathrm{succ} \colon \mathbb{N} \to \mathbb{N}$ by $n \mapsto n+1$;
- $[\![\circ]\!] = [\![\underset{C}{\|}]\!] \colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ by $(n,m) \mapsto n+m+1$.

Then $\mathbb{N}$ becomes an Ω-algebra with these definitions, and so the fact that $\mathcal{L}$ is initial implies there is a unique function $\rho \colon \mathcal{L} \to \mathbb{N}$ which is compositional. For example, $\rho(a \circ b) = 3$, and $\rho((\mathrm{rec}\, x.a \circ x) \underset{C}{\|} (a|_R)) = 7$.

For the sake of completeness, we include the standard definition of (free) variables of a process and of the substitution of a process for a variable in another process. The reader familiar with this can proceed directly to the next section.

Definition 3.1 Let $p \in \mathcal{L}$ be a process. Then the *set of variables of* p, $\mathrm{V}(p)$, and the *set of free variables in* p, $\mathrm{F}(p)$, are defined by

- $\mathrm{V}(\mathrm{STOP}) = \mathrm{F}(\mathrm{STOP}) = \mathrm{V}(a) = \mathrm{F}(a) = \emptyset$, $(\forall a \in \Sigma)$.
- $\mathrm{V}(x) = \mathrm{F}(x) = x$, $(\forall x \in V)$.
- $\mathrm{V}(p \circ q) = \mathrm{V}(p) \cup \mathrm{V}(q)$, $\mathrm{F}(p \circ q) = \mathrm{F}(p) \cup \mathrm{F}(q)$.
- $\mathrm{V}(p|_R) = \mathrm{V}(p)$, $\mathrm{F}(p|_R) = \mathrm{F}(p)$.

- $\mathrm{V}(p \parallel_C q) = \mathrm{V}(p) \cup \mathrm{V}(q)$, $\mathrm{F}(p \parallel_C q) = \mathrm{F}(p) \cup \mathrm{F}(q)$.
- $\mathrm{V}(\mathrm{rec}\, x.p) = \mathrm{V}(p)$, $\mathrm{F}(\mathrm{rec}\, x.p) = \mathrm{F}(p) \setminus \{x\}$.

The facts that $\mathrm{V}(p)$ and $\mathrm{F}(p)$ are well-defined can be deduced by first defining Ω-algebra structures on $\mathcal{P}(V)$, the set of variables, and then defining these mappings to be the unique Ω-algebra maps from $\mathcal{L}$ to $\mathcal{P}(V)$ with the appropriate structure. For example, in the case of F, we could define

$$[\![\mathrm{STOP}]\!] = [\![a]\!] = \emptyset, [\![x]\!] = \{x\}, [\![|_R]\!] = \mathrm{Id}_{\mathcal{P}(V)}, [\![\parallel_c]\!] = [\![\circ]\!] = \cup, \text{and}$$
$$[\![\mathrm{rec}\, x]\!] : \mathcal{P}(V) \to \mathcal{P}(V) \text{ defined by } [\![\mathrm{rec}\, x]\!](W) = W \setminus \{x\},$$

and then $\mathrm{F}\colon \mathcal{L} \to \mathcal{P}(V)$ is the unique Ω-algebra map respecting this algebra structure on $\mathcal{P}(V)$.

We now are ready to define substitution. In order for the rule for recursion given below to work, we must make the assumption that the set V of variables is infinite; then substitution is most easily done using structural induction:

Definition 3.2 For process p and q and a variable $x \in V$, we now define the *substitution* $p[q/x]$ as follows:

- If $x \notin \mathrm{F}(p)$, then $p[q/x] = p$.
- $x[q/x] = q$.
- $(p_1|_R)[q/x] = (p_1[q/x])|_R$.
- $(p_1 \circ p_2)[q/x] = p_1[q/x] \circ p_2[q/x]$.
- $(p_1 \parallel_C p_2)[q/x] = p_1[q/x] \parallel_C p_2[q/x]$.
- If $y \neq x$ and $y \notin \mathrm{F}(q)$, then $(\mathrm{rec}\, y.p_1)[q/x] = \mathrm{rec}\, y.(p_1[q/x])$.
- If $y \neq x$ and $y \in \mathrm{F}(q)$, then choose $z \in V \setminus (\mathrm{V}(p_1) \cup \mathrm{F}(q))$. Then

$$(\mathrm{rec}\, y.p_1)[q/x] = (\mathrm{rec}\, z.p_1[z/y])[q/x] = \mathrm{rec}\, z.(p_1[z/y][q/x]).$$

Substitution also can be defined as an Ω-algebra homomorphism, but this is much more complicated than in the case of defining free variables. The problem is in making the selection of a fresh variable in the last clause into a compositional map – it relies on first defining a related map called *Clashset* which determines when a term has a free variable which will be captured by the intended substitution. More details about this can be found in [10].

4 The Resource Mapping

The set of resources $\mathcal{R}$ and the resource mapping $\mathrm{res}\colon \Sigma \to \mathcal{P}(\mathcal{R})$ play a crucial role in our semantic models. In this section we define the resources which may be used by a process $p \in \mathcal{L}$, and we show that the extension of $\mathrm{res}\colon \Sigma \to \mathcal{P}(\mathcal{R})$ to $\mathcal{L}$ gives a denotational model for our language. This is crucial for defining the operational semantics of weak sequential composition and of parallel composition (cf. Table 1 below), and it is of fundamental importance in how we define the meaning of recursive processes. The extension of $\mathrm{res}\colon \Sigma \to \mathcal{P}(\mathcal{R})$ to the full language $\mathcal{L}$ with variables and recursion requires us to define the resource set associated with a process with free variables; for this we use *resource environments*, mappings $\sigma : V \to \mathcal{P}(\mathcal{R})$ assigning a resource set to each variable. Any resource environment $\sigma \in \mathcal{P}(\mathcal{R})^V$ can be *locally overridden* in its value at x:

$$\sigma[x \mapsto R](y) = \begin{cases} R & \text{if } y = x, \\ \sigma(y) & \text{otherwise,} \end{cases}$$

where $R \in \mathcal{P}(\mathcal{R})$ is any resource set we wish to assign at x.

Now, we define inductively the set of resources $\mathrm{res}(p, \sigma)$ associated with a process $p \in \mathcal{L}$ in the resource environment $\sigma \in \mathcal{P}(\mathcal{R})^V$ by:

- $\mathrm{res}(\mathrm{STOP}, \sigma) = \mathcal{R}$,
- $\mathrm{res}(a, \sigma) = \mathrm{res}(a)$ for all $a \in \Sigma$,
- $\mathrm{res}(p|_R, \sigma) = \mathrm{res}(p, \sigma) \cap R$ for all $R \subseteq \mathcal{R}$,
- $\mathrm{res}(p \circ q, \sigma) = \mathrm{res}(p, \sigma) \cup \mathrm{res}(q, \sigma)$,
- $\mathrm{res}(p \underset{C}{\|} q, \sigma) = \mathrm{res}(p, \sigma) \cup \mathrm{res}(q, \sigma)$,
- $\mathrm{res}(x, \sigma) = \sigma(x)$ for all $x \in V$,
- $\mathrm{res}(\mathrm{rec}\, x.p, \sigma) = \mathrm{res}(p, \sigma[x \mapsto \emptyset])$.

For instance, we have $\mathrm{res}(\mathrm{STOP}|_R, \sigma) = R$, $\mathrm{res}(\mathrm{rec}\, x.(a \circ x \circ b), \sigma) = \mathrm{res}(a) \cup \mathrm{res}(b)$ and $\mathrm{res}((\mathrm{rec}\, x.(x \circ a)) \underset{C}{\|} (\mathrm{rec}\, y.(b \circ y))) = \mathrm{res}(a) \cup \mathrm{res}(b)$.

The definition of the resource map for a recursion may look strange since one may expect a fixed point. Actually, as shown below (Proposition 4.3), it is a fixed point. But before proving this, we establish two easy lemmas.

Lemma 4.1 *Let $p \in \mathcal{L}$ be a process and $x \in V$ a variable that does not occur free in p. Then,* $\mathrm{res}(p, \sigma)$ *does not depend on $\sigma(x)$.*

In particular, if $p \in \mathcal{L}$ is a closed *term (no free variables) then* $\mathrm{res}(p, \sigma)$ *does not depend on the resource environment σ.*

Proof. Let $\sigma, \sigma' \in \mathcal{P}(\mathcal{R})^V$ be two resource environments such that $\sigma(y) = \sigma'(y)$ for all $y \neq x$. We show by induction on p that $\mathrm{res}(p, \sigma) = \mathrm{res}(p, \sigma')$ for each process p in which x does not occur free.

The result is immediate for the basis cases STOP, $a \in \Sigma$ or $y \in V$ such that $y \neq x$. It follows by induction for restriction $p_1|_R$, weak sequential composition $p_1 \circ p_2$, and parallel composition $p_1 \underset{C}{\|} p_2$. Now, assume that $p = \mathrm{rec}\, y.p_1$ with $y \neq x$ and that x does not occur free in p_1. We have

$$\mathrm{res}(\mathrm{rec}\, y.p_1, \sigma) = \mathrm{res}(p_1, \sigma[y \mapsto \emptyset]) = \mathrm{res}(p_1, \sigma'[y \mapsto \emptyset]) = \mathrm{res}(\mathrm{rec}\, y.p_1, \sigma')$$

Finally, if $p = \mathrm{rec}\, x.p_1$ then $\sigma[x \mapsto \emptyset] = \sigma'[x \mapsto \emptyset]$ and we have

$$\mathrm{res}(\mathrm{rec}\, x.p_1, \sigma) = \mathrm{res}(p_1, \sigma[x \mapsto \emptyset]) = \mathrm{res}(p_1, \sigma'[x \mapsto \emptyset]) = \mathrm{res}(\mathrm{rec}\, x.p_1, \sigma')$$

$\square$

Lemma 4.2 *Let $p \in \mathcal{L}$ be a process and $\sigma \in \mathcal{P}(\mathcal{R})^V$ be a resource environment. We have*

$$\mathrm{res}(p,\sigma) \subseteq \mathrm{res}(p,\sigma[x \mapsto \emptyset]) \cup \sigma(x)$$

Proof. The result is immediate for $p = x$ and it follows from Lemma 4.1 if x is not free in p. In particular, it holds for the basis cases STOP, $a \in \Sigma$ or $y \in V$. It follows directly by induction for restriction $p_1|_R$, weak sequential composition $p_1 \circ p_2$, and parallel composition $p_1 \underset{C}{\|} p_2$. Finally, assume that $p = \mathrm{rec}\, y.p_1$ with $y \neq x$. Then we have

$$\begin{aligned}\mathrm{res}(p,\sigma) &= \mathrm{res}(p_1,\sigma[y \mapsto \emptyset])\\ &\subseteq \mathrm{res}(p_1,\sigma[y \mapsto \emptyset, x \mapsto \emptyset]) \cup \sigma[y \mapsto \emptyset](x)\\ &\subseteq \mathrm{res}(\mathrm{rec}\, y.p_1,\sigma[x \mapsto \emptyset]) \cup \sigma(x).\end{aligned}$$ □

Next, we show that the resource map is continuous and that our definition of recursion fulfills what one expects – that it is a fixed point for the body of the recursion. As explained in Section 2.2 we order the set $\mathcal{P}(\mathcal{R})$ with reverse containment.

Proposition 4.3 *Let $p \in \mathcal{L}$ be a process and $\sigma \in \mathcal{P}(\mathcal{R})^V$ be a resource environment. Then,*

(i) *The mapping*

$$\mathrm{res}(p,-) : (\mathcal{P}(\mathcal{R}),\supseteq)^V \to (\mathcal{P}(\mathcal{R}),\supseteq)$$

is continuous.

(ii) $R = \mathrm{res}(\mathrm{rec}\, x.p,\sigma) = \nu S.\mathrm{res}(p,\sigma[x \mapsto S])$, *i.e.* $\mathrm{res}(\mathrm{rec}\, x.p,\sigma)$ *is the greatest fixed point of the continuous map*

$$\begin{aligned}\mathrm{res}(p,\sigma[x \mapsto -]) : (\mathcal{P}(\mathcal{R}),\supseteq) &\to (\mathcal{P}(\mathcal{R}),\supseteq)\\ S &\mapsto \mathrm{res}(p,\sigma[x \mapsto S])\end{aligned}$$

In particular, $\mathrm{res}(\mathrm{rec}\, x.p,\sigma) = \mathrm{res}(p,\sigma[x \mapsto \mathrm{res}(\mathrm{rec}\, x.p,\sigma)])$.

Proof. (1) Again, we prove this by structural induction on p. The result is clear for the cases of STOP and $a \in \Sigma$, since these give constant maps. For $x \in V$, the mapping $\mathrm{res}(x,-)$ is a projection which is indeed continuous. The mapping $-\cup- : (\mathcal{P}(\mathcal{R}),\supseteq)^2 \to (\mathcal{P}(\mathcal{R}),\supseteq)$ is continuous, and since the composition of continuous maps is continuous, the result follows for $p_1 \circ p_2$ and $p_1 \underset{C}{\|} p_2$. Similarly, $-\cap- : (\mathcal{P}(\mathcal{R}),\supseteq)^2 \to (\mathcal{P}(\mathcal{R}),\supseteq)$ is continuous and we obtain the result for $p_1|_R$. Finally, the selfmap of $(\mathcal{P}(\mathcal{R}),\supseteq)^V$ defined by $\sigma \mapsto \sigma[x \mapsto \emptyset]$ is continuous since $(\mathcal{P}(\mathcal{R}),\supseteq)^V$ is given the product topology. The result follows by composition of continuous maps for recursion $\mathrm{rec}\, x.p_1$.

(2) Using Lemma 4.2 we obtain

$$\mathrm{res}(p,\sigma[x \mapsto R]) \subseteq \mathrm{res}(p,\sigma[x \mapsto \emptyset]) \cup R = R,$$

and since the mapping $\mathrm{res}(p,-)$ is monotone, we also have

$$R = \mathrm{res}(p,\sigma[x \mapsto \emptyset]) \subseteq \mathrm{res}(p,\sigma[x \mapsto R]).$$

Therefore, R is a fixed point of the mapping $\mathrm{res}(p,\sigma[x \mapsto -])$ and since it is obtained by iteration from the greatest element $\emptyset$ of $(\mathcal{P}(\mathcal{R}),\supseteq)$, it is the greatest fixed point of $\mathrm{res}(p,\sigma[x \mapsto -])$. □

We now present a lemma which will be useful when studying the operational semantics of our language. It explains how to compute the resource map of the process $p[q/x]$ which is obtained by substituting the process q for each free occurrence of the variable x in the process p.

Lemma 4.4 *Let $p, q \in \mathcal{L}$ be two processes and $\sigma \in \mathcal{P}(\mathcal{R})^V$ be a resource environment. Then*

$$\mathrm{res}(p[q/x], \sigma) = \mathrm{res}(p, \sigma[x \mapsto \mathrm{res}(q, \sigma)]).$$

Proof. We proceed by induction on p. The result is immediate for $p = x$ and it follows from Lemma 4.1 if x is not free in p. In particular, it holds for the basis cases STOP, $a \in \Sigma$ or $y \in V$. It follows directly by induction for restriction $p_1|_R$, weak sequential composition $p_1 \circ p_2$, and parallel composition $p_1 \underset{C}{\|} p_2$. Finally, assume that $p = \mathrm{rec}\, y.p_1$ with $y \neq x$. Then, we may assume that y is not free in q and $p[q/x] = \mathrm{rec}\, y.(p_1[q/x])$. We obtain

$$\begin{aligned}
\mathrm{res}(\mathrm{rec}\, y.(p_1[q/x]), \sigma) &= \mathrm{res}(p_1[q/x], \sigma[y \mapsto \emptyset]) \\
&= \mathrm{res}(p_1, \sigma[y \mapsto \emptyset, x \mapsto \mathrm{res}(q, \sigma[y \mapsto \emptyset])]) && \text{by induction} \\
&= \mathrm{res}(p_1, \sigma[x \mapsto \mathrm{res}(q, \sigma), y \mapsto \emptyset]) && \text{by Lemma 4.1} \\
&= \mathrm{res}(\mathrm{rec}\, y.p_1, \sigma[x \mapsto \mathrm{res}(q, \sigma)]).
\end{aligned}$$

□

We conclude this section by showing that we can endow the set of continuous maps $[\mathcal{P}(\mathcal{R})^V \to \mathcal{P}(\mathcal{R})]$ with a structure of a continuous Ω-algebra which yields precisely the resource map defined above. The constants STOP and a ($a \in \Sigma$) are interpreted as constant maps, the constant $x \in V$ is interpreted as a projection, restriction $|_R$ is intersection with R, the two compositions $\circ$ and $\underset{C}{\|}$ are union, and finally, recursion $\mathrm{rec}\, x$ is a greatest fixed point. More precisely, for continuous maps $f, g \colon \mathcal{P}(\mathcal{R})^V \to \mathcal{P}(\mathcal{R})$ and resource environment $\sigma \in \mathcal{P}(\mathcal{R})^V$, we define:

$$[\![\mathrm{STOP}]\!] \in [\mathcal{P}(\mathcal{R})^V \to \mathcal{P}(\mathcal{R})] \text{ defined by } [\![\mathrm{STOP}]\!](\sigma) = \mathcal{R},$$

$$[\![a]\!] \in [\mathcal{P}(\mathcal{R})^V \to \mathcal{P}(\mathcal{R})] \text{ defined by } [\![a]\!](\sigma) = \mathrm{res}(a),\ (\forall a \in \Sigma),$$

$$[\![x]\!] \in [\mathcal{P}(\mathcal{R})^V \to \mathcal{P}(\mathcal{R})] \text{ defined by } [\![x]\!](\sigma) = \sigma(x),\ (\forall x \in V),$$

$$\begin{aligned}[\![|_R]\!] :\ & [\mathcal{P}(\mathcal{R})^V \to \mathcal{P}(\mathcal{R})] \to [\mathcal{P}(\mathcal{R})^V \to \mathcal{P}(\mathcal{R})] \text{ by } f \mapsto f[\![|_R]\!] \\ & \text{with } f[\![|_R]\!](\sigma) = f(\sigma) \cap R, (\forall R \subseteq \mathcal{R}),\end{aligned}$$

$$\begin{aligned}[\![\circ]\!] :\ & [\mathcal{P}(\mathcal{R})^V \to \mathcal{P}(\mathcal{R})]^2 \to [\mathcal{P}(\mathcal{R})^V \to \mathcal{P}(\mathcal{R})] \text{ by } f, g \mapsto f[\![\circ]\!]g \\ & \text{with } (f[\![\circ]\!]g)(\sigma) = f(\sigma) \cup g(\sigma),\end{aligned}$$

$$\begin{aligned}[\![\underset{C}{\|}]\!] :\ & [\mathcal{P}(\mathcal{R})^V \to \mathcal{P}(\mathcal{R})]^2 \to [\mathcal{P}(\mathcal{R})^V \to \mathcal{P}(\mathcal{R})] \text{ by } f, g \mapsto f[\![\underset{C}{\|}]\!]g \\ & \text{with } (f[\![\underset{C}{\|}]\!]g)(\sigma) = f(\sigma) \cup g(\sigma),\end{aligned}$$

$$\begin{aligned}[\![\mathrm{rec}\, x]\!] :\ & [\mathcal{P}(\mathcal{R})^V \to \mathcal{P}(\mathcal{R})] \to [\mathcal{P}(\mathcal{R})^V \to \mathcal{P}(\mathcal{R})] \text{ by } f \mapsto [\![\mathrm{rec}\, x]\!].f \\ & \text{with } ([\![\mathrm{rec}\, x]\!].f)(\sigma) = (\nu S. f(\sigma[x \mapsto S])),\ (\forall x \in V).\end{aligned}$$

With this view, the mapping $p \mapsto \mathrm{res}(p, -)$ is the unique Ω-algebra map from $\mathcal{L}$ to $[\mathcal{P}(\mathcal{R})^V \to \mathcal{P}(\mathcal{R})]$. Indeed, we have to show that these interpretations are well defined, i.e. that for each operator ω of arity n, $[\![\omega]\!](f_1, \ldots, f_n)$ is continuous assuming that $f_1, \ldots, f_n$ are continuous themselves.

Proposition 4.5 *If $\omega \in \Omega$ is an operator of arity n, then the operator*

$$[\![\omega]\!] : [\mathcal{P}(\mathcal{R})^V \to \mathcal{P}(\mathcal{R})]^n \to [\mathcal{P}(\mathcal{R})^V \to \mathcal{P}(\mathcal{R})]$$

is well-defined and continuous.

Proof. We have to show that $[\![\omega]\!](f_1, \ldots, f_n)$ is indeed continuous assuming that $f_1, \ldots, f_n$ are continuous themselves and in addition that $(f_1, \ldots, f_n) \mapsto [\![\omega]\!](f_1, \ldots, f_n)$ is continuous (this last part only when $n > 0$). This is clear for the constants of STOP and $a \in \Sigma$, since these give constant maps and also for $x \in V$ since projections are continuous.

It is a standard result from domain theory that continuous operators on a domain D lift to continuous operators on function spaces over D (cf. [1] or Proposition 6.1). Therefore, the results for restriction, weak sequential composition and parallel composition follow since the mappings $- \cup - : (\mathcal{P}(\mathcal{R}), \supseteq)^2 \to (\mathcal{P}(\mathcal{R}), \supseteq)$ and $- \cap - : (\mathcal{P}(\mathcal{R}), \supseteq)^2 \to (\mathcal{P}(\mathcal{R}), \supseteq)$ are continuous.

For recursion, it is not so clear since usually the greatest fixed point operator is not continuous. Fortunately, $\mathcal{P}(\mathcal{R})$ is finite, hence the greatest fixed point is obtained after a fixed number of iterations which allows to deduce the claim. More precisely, we consider the continuous selfmap $\Phi_{f,\sigma} : \mathcal{P}(\mathcal{R}) \to \mathcal{P}(\mathcal{R})$ by $\Phi_{f,\sigma}(R) = f(\sigma[x \mapsto R])$. Then, $([\![\mathrm{rec}\, x]\!].f)(\sigma) = \Phi^N_{f,\sigma}(\emptyset)$ where $N = |\mathcal{P}(\mathcal{R})|$. Since the mapping from $[\mathcal{P}(\mathcal{R})^V \to \mathcal{P}(\mathcal{R})] \times \mathcal{P}(\mathcal{R})^V$ to $[\mathcal{P}(\mathcal{R}) \to \mathcal{P}(\mathcal{R})]$ defined by $(f, \sigma) \mapsto \Phi_{f,\sigma}$ is continuous, we can show easily that the mapping $[\mathcal{P}(\mathcal{R})^V \to \mathcal{P}(\mathcal{R})] \times \mathcal{P}(\mathcal{R})^V \to \mathcal{P}(\mathcal{R})$ defined by $(f, \sigma) \mapsto \Phi^N_{f,\sigma}(\emptyset)$ is continuous. The claim that $[\![\mathrm{rec}\, x]\!]$

is well-defined and continuous follows. □

5 Operational Semantics

In this section we present an operational semantics for our language. In fact, we give somewhat more than usual – we give an operational semantics for all terms $p \in \mathcal{L}$, even those with free variables. This is necessitated by our desire to use something other than the usual least fixed point semantics of recursion that domain theory offers. The reason for this will be clarified later on – for now, we confine our discussion to presenting the transition rules for our language, and on deriving results about the resulting behavior of terms from $\mathcal{L}$ under these rules.

The now-traditional method for presenting an operational semantics for a language has its origins in the seminal work of Plotkin [14], where the notion of a *structural operational semantics* was first put forth. Such a semantics consists of a set of *transition rules* that indicate what the "next step" of a computation is, according to the form of the term being executed. This means we must have rules for each of the clauses in the BNF we gave in Section 2 that defines our language. But the fact that we must also give operational rules for open terms (ones with free variables) means we must have some way of interpreting those variables. The key is what we already have done in the previous section to interpret variables in the context of a denotational model: we use environments. Once again we rely on the mappings $\sigma\colon V \to \mathcal{P}(\mathcal{R})$ to aid us, and so our transition rules tell us what next steps are possible for a term *in a given environment* σ.

We must make an additional assumption to define our transition rules. We are interested in supporting synchronization over a set $C \subseteq \mathcal{R}$ which we view as the channels over which synchronization can occur. We therefore assume that the alphabet Σ has a synchronization operation $\|\colon \Sigma \times \Sigma \to \Sigma$ which describes the result of synchronizing a matching pair of actions. There is no extra resource introduced by synchronization and no hiding of resources either, hence we require that for all $(a_1, a_2) \in \Sigma^2$,

$$\mathrm{res}(a_1 \| a_2) = \mathrm{res}(a_1) \cup \mathrm{res}(a_2).$$

We introduce a notation used in the rule of the parallel composition. Let $p_1, p_2 \in \mathcal{L}$ be processes, $\sigma \in \mathcal{P}(\mathcal{R})^V$ be an environment and $C \subseteq \mathcal{R}$ be a set of *channels*. Then, a pair $(a_1, a_2) \in \Sigma^2$ may be synchronized in $p_1 \underset{C}{\|} p_2$ if $\mathrm{res}(a_i) \subseteq \mathrm{res}(p_i, \sigma)$ for $i = 1, 2$ and

$$\mathrm{res}(a_1) \cap \mathrm{res}(p_2, \sigma) = \mathrm{res}(a_2) \cap \mathrm{res}(p_1, \sigma) = \mathrm{res}(a_1) \cap C = \mathrm{res}(a_2) \cap C \neq \emptyset$$

We denote by $\mathrm{Sync}_{C,\sigma}(p_1, p_2)$ the set of those pairs satisfying this condition.

We now are ready for the transition rules which are the basis for the operational semantics for our language $\mathcal{L}$. We present them in natural deduction style in Table 1. We denote by SKIP the process $\mathrm{STOP}|_{\emptyset}$ which does nothing and claims no resources.

We need a number of results about the rules in Table 1 before we can define the operational behavior of a term $p \in \mathcal{L}$. In order to improve the readability, they are stated below without proofs so that we can define immediately the operational behavior. Then the proofs are given except for one of the results whose proof is much easier once we have defined the

$$(1)\quad \frac{}{a \xrightarrow[\sigma]{a} \mathrm{SKIP}}$$

$$(2a)\quad \frac{p_1 \xrightarrow[\sigma]{a} p_1'}{p_1 \circ p_2 \xrightarrow[\sigma]{a} p_1' \circ p_2} \qquad (2b)\quad \frac{p_2 \xrightarrow[\sigma]{a} p_2',\ \mathrm{res}(a) \cap \mathrm{res}(p_1, \sigma) = \emptyset}{p_1 \circ p_2 \xrightarrow[\sigma]{a} p_1 \circ p_2'}$$

$$(3)\quad \frac{p \xrightarrow[\sigma]{a} p',\ \mathrm{res}(a) \subseteq R}{p|_R \xrightarrow[\sigma]{a} p'|_R}$$

$$(4a)\quad \frac{p_1 \xrightarrow[\sigma]{a} p_1',\ \mathrm{res}(a) \cap (\mathrm{res}(p_2, \sigma) \cup C) = \emptyset}{p_1 \underset{C}{\|} p_2 \xrightarrow[\sigma]{a} p_1' \underset{C}{\|} p_2}$$

$$(4b)\quad \frac{p_2 \xrightarrow[\sigma]{a} p_2',\ \mathrm{res}(a) \cap (\mathrm{res}(p_1, \sigma) \cup C) = \emptyset}{p_1 \underset{C}{\|} p_2 \xrightarrow[\sigma]{a} p_1 \underset{C}{\|} p_2'}$$

$$(4c)\quad \frac{p_1 \xrightarrow[\sigma]{a_1} p_1',\ p_2 \xrightarrow[\sigma]{a_2} p_2',\ (a_1, a_2) \in \mathrm{Sync}_{C,\sigma}(p_1, p_2)}{p_1 \underset{C}{\|} p_2 \xrightarrow[\sigma]{a_1 \| a_2} p_1' \underset{C}{\|} p_2'}$$

$$(5)\quad \frac{p \xrightarrow[\sigma']{a} p',\ \sigma' = \sigma[x \mapsto \mathrm{res}(\mathrm{rec}\, x.p, \sigma)]}{\mathrm{rec}\, x.p \xrightarrow[\sigma]{a} p'[\mathrm{rec}\, x.p/x]}$$

Table 1
The Transition Rules for $\mathcal{L}$

denotational semantics of our language in the following section.

Proposition 5.1 *Let $p, p', p'' \in \mathcal{L}$ be processes, $\sigma \in \mathcal{P}(\mathcal{R})^V$ be a syntactic environment, and $a, b \in \Sigma$. If $a \neq b$, $p \xrightarrow[\sigma]{a} p'$ and $p \xrightarrow[\sigma]{b} p''$, then $a\, I\, b$ and $\exists p''' \in \mathcal{L}$ with $p' \xrightarrow[\sigma]{b} p'''$ and $p'' \xrightarrow[\sigma]{a} p'''$.*

Proposition 5.2 *Let $p, p', p'' \in \mathcal{L}$ be processes, $\sigma \in \mathcal{P}(\mathcal{R})^V$ be a syntactic environment, and $a \in \Sigma$. Then*

$$p \xrightarrow[\sigma]{a} p' \text{ and } p \xrightarrow[\sigma]{a} p'' \quad \Rightarrow \quad p' = p''.$$

Proposition 5.3 *Let $p, p', p'' \in \mathcal{L}$ be processes, $\sigma \in \mathcal{P}(\mathcal{R})^V$ be a syntactic environment, and $a, b \in \Sigma$. If $a\, I\, b$, $p \xrightarrow[\sigma]{a} p'$, $p' \xrightarrow[\sigma]{b} p''$, then $\exists p''' \in \mathcal{L}$ with $p \xrightarrow[\sigma]{b} p'''$ and $p''' \xrightarrow[\sigma]{a} p''$.*

Proposition 5.2 implies that our transition system is deterministic. Adding Proposition 5.1, we know that it is strongly locally confluent, whence Church-Rosser. Since we want a truly concurrent semantics, it should be possible for a process to execute independent events concurrently – i.e., independently. This is reflected by Proposition 5.3 in our transition system. From this we derive by induction

Corollary 5.4 *Let $u, v \in \Sigma^*$ with $u \sim v$. Then $p \xrightarrow[\sigma]{u} q$ iff $p \xrightarrow[\sigma]{v} q$. Hence $p \xrightarrow[\sigma]{s} q$ is well-defined for finite traces $s \in \mathbb{M}$.* □

In an interleaving semantics, the possible operational behaviors of a process p in the environment $\sigma \in \mathcal{P}(\mathcal{R})$ would consist of the set

$$X_{\Sigma^*}(p,\sigma) = \{u \in \Sigma^* \mid \exists q \in \mathcal{L}, p \xrightarrow[\sigma]{u} q\}.$$

Thanks to Corollary 5.4, we actually can define the possible concurrent behaviors as

$$X_{\mathbb{M}}(p,\sigma) = \{t \in \mathbb{M} \mid \exists q \in \mathcal{L}, p \xrightarrow[\sigma]{t} q\}.$$

But, knowing only a possible real (finite) trace that can be executed does not provide enough information to know how the process can be continued or composed with another process. Hence we need to bring resources into the picture.

Definition 5.5 Let $p \in \mathcal{L}$ be a process and $\sigma \in \mathcal{P}(\mathcal{R})^V$ be a resource environment. The set of resource trace behaviors of p in σ is

$$X_{\mathbb{F}}(p,\sigma) = \{(s, \mathrm{res}(q,\sigma)) \in \mathbb{F} \mid \exists q \in \mathcal{L}, p \xrightarrow[\sigma]{s} q\}.$$

The meaning is that $(s,S) \in X_{\mathbb{F}}(p,\sigma)$ if p can concurrently execute the trace s and then still claim the resources in S.

Actually, we will prove later (Theorem 7.10) that the set $X_{\mathbb{F}}(p,\sigma)$ is directed. The interpretation is that p has a unique maximal behavior in the environment σ which is the least upper bound of $X_{\mathbb{F}}(p,\sigma)$: $B_{\mathbb{F}}(p,\sigma) = \sqcup X_{\mathbb{F}}(p,\sigma)$. This is precisely what tells us that our semantics of parallelism does not involve nondeterministic choice.

Proposition 5.3 will be proved in Section 7 using results from the denotational semantics. The remainder of this section is devoted to the proofs of Propositions 5.2 and 5.3. We start with some useful results.

Proposition 5.6 *Let $p, p' \in \mathcal{L}$ be processes, $\sigma \in \mathcal{P}(\mathcal{R})^V$ be a syntactic environment, and $u \in \Sigma^*$. Then*

$$p \xrightarrow[\sigma]{u} p' \quad \Rightarrow \quad \mathrm{res}(p,\sigma) = \mathrm{res}(p',\sigma) \cup \mathrm{res}(u).$$

Proof. We first prove by structural induction on p that the property holds when $u = a \in \Sigma$ is a single letter. Then the only forms that p could take are $p = a$, $p = p_1 \circ p_2$, $p = p_1|_R$, $p = p_1 \underset{C}{\|} p_2$, or $p = \mathrm{rec}\, x.p_1$. We consider these in turn.

• If $p = a \in \Sigma$, then $p' = \mathrm{SKIP}$, and so

$$\mathrm{res}(p,\sigma) = \mathrm{res}(a) = \mathrm{res}(a) \cup \mathrm{res}(\mathrm{SKIP},\sigma).$$

• If $p = p_1 \circ p_2$, then either $p_1 \xrightarrow[\sigma]{a} p_1'$ or $p_2 \xrightarrow[\sigma]{a} p_2'$ and $\mathrm{res}(a) \cap \mathrm{res}(p_1,\sigma) = \emptyset$. In the first case, $p' = p_1' \circ p_2$, and so

$$\begin{aligned}
\mathrm{res}(p_1 \circ p_2,\sigma) &= \mathrm{res}(p_1,\sigma) \cup \mathrm{res}(p_2,\sigma) \\
&= \mathrm{res}(a) \cup \mathrm{res}(p_1',\sigma) \cup \mathrm{res}(p_2,\sigma) \qquad \text{by structural induction} \\
&= \mathrm{res}(a) \cup \mathrm{res}(p_1' \circ p_2,\sigma).
\end{aligned}$$

In the second case, $p' = p_1 \circ p_2'$, and

$$\mathrm{res}(p_1 \circ p_2, \sigma) = \mathrm{res}(p_1, \sigma) \cup \mathrm{res}(a) \cup \mathrm{res}(p_2', \sigma) = \mathrm{res}(a) \cup \mathrm{res}(p_1 \circ p_2', \sigma).$$

• If $p = p_1|_R$, then $p_1 \xrightarrow[\sigma]{a} p_1'$ and $\mathrm{res}(a) \subseteq R$ and $p' = p_1'|_R$. Then

$$\begin{aligned}\mathrm{res}(p, \sigma) &= \mathrm{res}(p_1) \cap R = (\mathrm{res}(a) \cup \mathrm{res}(p_1', \sigma)) \cap R \\ &= \mathrm{res}(a) \cup (\mathrm{res}(p_1', \sigma) \cap R) = \mathrm{res}(a) \cup \mathrm{res}(p_1'|_R, \sigma).\end{aligned}$$

• If $p = p_1 \underset{C}{\|} p_2$, then $p \xrightarrow[\sigma]{a} p'$ means either $p_1 \xrightarrow[\sigma]{a} p_1'$ and $\mathrm{res}(a) \cap (C \cup \mathrm{res}(p_2, \sigma)) = \emptyset$, or $p_2 \xrightarrow[\sigma]{a} p_2'$ and $\mathrm{res}(a) \cap (C \cup \mathrm{res}(p_1, \sigma)) = \emptyset$, or $a = a_1 \| a_2$ for some $(a_1, a_2) \in \mathrm{Sync}_{C,\sigma}(p_1, p_2)$ and $p_1 \xrightarrow[\sigma]{a_1} p_1'$, $p_2 \xrightarrow[\sigma]{a_2} p_2'$. The last case is the most interesting, and the proof in this case is

$$\begin{aligned}\mathrm{res}(p_1 \underset{C}{\|} p_2, \sigma) &= \mathrm{res}(p_1, \sigma) \cup \mathrm{res}(p_2, \sigma) \\ &= (\mathrm{res}(a_1) \cup \mathrm{res}(p_1', \sigma)) \cup (\mathrm{res}(a_2) \cup \mathrm{res}(p_2', \sigma)) \\ &= \mathrm{res}(a) \cup \mathrm{res}(p_1' \underset{C}{\|} p_2', \sigma),\end{aligned}$$

the middle equality following from the inductive hypothesis, and the last following from the assumption on the mapping $\| : \Sigma^2 \to \Sigma$.

• The final form is $p = \mathrm{rec}\, x.p_1$, in which case we have $p_1 \xrightarrow[\sigma']{a} p_1'$ and $p' = p_1'[\mathrm{rec}\, x.p_1/x]$ where $\sigma' = \sigma[x \mapsto \mathrm{res}(\mathrm{rec}\, x.p_1, \sigma)]$, and so

$$\begin{aligned}\mathrm{res}(p, \sigma) &= \mathrm{res}(\mathrm{rec}\, x.p_1, \sigma) = \mathrm{res}(p_1, \sigma') = \mathrm{res}(a) \cup \mathrm{res}(p_1', \sigma') \\ &= \mathrm{res}(a) \cup \mathrm{res}(p_1'[\mathrm{rec}\, x.p_1/x], \sigma),\end{aligned}$$

the second equality following from Proposition 4.3 and the last from Lemma 4.4.

Finally, the proof of Proposition 5.6 for an arbitrary $u \in \Sigma^*$ is done by induction on the length of u. If $u = \epsilon$ is the empty word, then $p' = p$, and there is nothing to prove. If $u = au'$ for some $u' \in \Sigma^*$ and $p \xrightarrow[\sigma]{u} p'$, then there is some $q \in \mathcal{L}$ with $p \xrightarrow[\sigma]{a} q \xrightarrow[\sigma]{u'} p'$, and so

$$\begin{aligned}\mathrm{res}(p, \sigma) &= \mathrm{res}(a) \cup \mathrm{res}(q, \sigma) \\ &= \mathrm{res}(a) \cup \mathrm{res}(u') \cup \mathrm{res}(p', \sigma) \\ &= \mathrm{res}(u) \cup \mathrm{res}(p', \sigma),\end{aligned}$$

the first equality following from the proof just given, and the middle equality following by the inductive hypothesis on the length of u. □

Lemma 5.7 *Let $p, p', q \in \mathcal{L}$ be processes, $\sigma, \sigma' \in \mathcal{P}(\mathcal{R})^V$ be syntactic environments, $a \in \Sigma$, and $x \in V$. Then*

$$p \xrightarrow[\sigma']{a} p', \sigma' = \sigma[x \mapsto \mathrm{res}(q, \sigma)] \quad \Rightarrow \quad p[q/x] \xrightarrow[\sigma]{a} p'[q/x].$$

Proof. We proceed by structural induction on p, and since we are concerned with processes $p \xrightarrow[\sigma']{a} p'$, we know p cannot be STOP or a variable $x \in V$ or a letter $b \neq a$.

• If $p = a \in \Sigma$, then $p[q/x] = p$ and $p' = \mathrm{SKIP}$. Then

$$p[q/x] = a \xrightarrow[\sigma]{a} \mathrm{SKIP} = \mathrm{SKIP}[q/x],$$

which proves the result in this case.

• If $p = p_1 \circ p_2$, then either $p_1 \xrightarrow[\sigma']{a} p_1'$ or else $p_2 \xrightarrow[\sigma']{a} p_2'$ and $\mathrm{res}(a) \cap \mathrm{res}(p_1, \sigma') = \emptyset$. In the first case, $p_1[q/x] \xrightarrow[\sigma]{a} p_1'[q/x]$ by the inductive hypothesis, and so

$$(p_1 \circ p_2)[q/x] = p_1[q/x] \circ p_2[q/x] \xrightarrow[\sigma]{a} p_1'[q/x] \circ p_2[q/x] = (p_1' \circ p_2)[q/x].$$

A similar argument applies in the second case. Since by Lemma 4.4 we have $\mathrm{res}(p_1[q/x], \sigma) = \mathrm{res}(p_1, \sigma')$, we deduce that $\mathrm{res}(a) \cap \mathrm{res}(p_1[q/x], \sigma) = \emptyset$. Therefore

$$(p_1 \circ p_2)[q/x] = p_1[q/x] \circ p_2[q/x] \xrightarrow[\sigma]{a} p_1[q/x] \circ p_2'[q/x] = (p_1 \circ p_2')[q/x].$$

• If $p = p_1|_R$, then $\mathrm{res}(a) \subseteq R$ and $p' = p_1'|_R$ with $p_1 \xrightarrow[\sigma']{a} p_1'$. Then, we have $p_1[q/x] \xrightarrow[\sigma]{a} p_1'[q/x]$ by the inductive hypothesis, so

$$(p_1|_R)[q/x] = (p_1[q/x])|_R \xrightarrow[\sigma]{a} (p_1'[q/x])|_R = (p_1'|_R)[q/x].$$

• If $p = p_1 \underset{C}{\|} p_2 \xrightarrow[\sigma']{a} p'$, then either $p_1 \xrightarrow[\sigma']{a} p_1'$ and $\mathrm{res}(a) \cap (C \cup \mathrm{res}(p_2, \sigma')) = \emptyset$, or $p_2 \xrightarrow[\sigma']{a} p_2'$ and $\mathrm{res}(a) \cap (C \cup \mathrm{res}(p_1, \sigma')) = \emptyset$, or $a = a_1 \| a_2$ for some $(a_1, a_2) \in \mathrm{Sync}_{C,\sigma'}(p_1, p_2)$ and $p_1 \xrightarrow[\sigma]{a_1} p_1'$, $p_2 \xrightarrow[\sigma]{a_2} p_2'$. The first two cases are easier to verify – we consider only the last one, which is the most interesting. In this case, the inductive hypothesis implies $p_1[q/x] \xrightarrow[\sigma]{a_1} p_1'[q/x]$ and $p_2[q/x] \xrightarrow[\sigma]{a_2} p_2'[q/x]$. Moreover, by Lemma 4.4 we have $\mathrm{res}(p_1, \sigma') = \mathrm{res}(p_1[q/x], \sigma)$ and $\mathrm{res}(p_2, \sigma') = \mathrm{res}(p_2[q/x], \sigma)$ and we deduce that $(a_1, a_2) \in \mathrm{Sync}_{C,\sigma}(p_1[q/x], p_2[q/x])$. Then

$$(p_1 \underset{C}{\|} p_2)[q/x] = p_1[q/x] \underset{C}{\|} p_2[q/x] \xrightarrow[\sigma]{a} p_1'[q/x] \underset{C}{\|} p_2'[q/x] = (p_1' \underset{C}{\|} p_2')[q/x].$$

• If $p = \mathrm{rec}\, x.p_1$, then $p' = p_1'[\mathrm{rec}\, x.p_1/x]$ with $p_1 \xrightarrow[\sigma'']{a} p_1'$ and $\sigma'' = \sigma'[x \mapsto \mathrm{res}(\mathrm{rec}\, x.p_1, \sigma')]$. By Lemma 4.1 we have $\mathrm{res}(\mathrm{rec}\, x.p_1, \sigma') = \mathrm{res}(\mathrm{rec}\, x.p_1, \sigma)$, hence $\sigma'' = \sigma[x \mapsto \mathrm{res}(\mathrm{rec}\, x.p_1, \sigma)]$. Therefore $\mathrm{rec}\, x.p_1 \xrightarrow[\sigma]{a} p_1'[\mathrm{rec}\, x.p_1/x]$. The result follows since x occurs free neither in $\mathrm{rec}\, x.p_1$ nor in $p_1'[\mathrm{rec}\, x.p_1/x]$.

• If $p = \mathrm{rec}\, y.p_1$ with $y \neq x$, then $p' = p_1'[\mathrm{rec}\, y.p_1/y]$ with $p_1 \xrightarrow[\sigma'']{a} p_1'$ and $\sigma'' = \sigma'[y \mapsto \mathrm{res}(p, \sigma')] = \sigma[x \mapsto \mathrm{res}(q, \sigma), y \mapsto \mathrm{res}(p, \sigma')]$. We may assume that y is not free in q. Let $\sigma''' = \sigma[y \mapsto \mathrm{res}(p, \sigma')]$, since y is not free in q we have by Lemma 4.1 $\mathrm{res}(q, \sigma) = \mathrm{res}(q, \sigma''')$ and therefore $\sigma'' = \sigma'''[x \mapsto \mathrm{res}(q, \sigma''')]$. Using the inductive hypothesis we obtain $p_1[q/x] \xrightarrow[\sigma''']{a} p_1'[q/x]$. Now, using Lemma 4.4 and that y is not free in q, we obtain $\mathrm{res}(p, \sigma') = \mathrm{res}(p[q/x], \sigma) = \mathrm{res}(\mathrm{rec}\, y.(p_1[q/x]), \sigma)$. Hence, $\sigma''' = \sigma[y \mapsto \mathrm{res}(\mathrm{rec}\, y.(p_1[q/x]), \sigma)]$. Therefore,

$$\begin{aligned}(\mathrm{rec}\, y.p_1)[q/x] = \mathrm{rec}\, y.(p_1[q/x]) &\xrightarrow[\sigma]{a} \\ &(p_1'[q/x])[\mathrm{rec}\, y.(p_1[q/x])/y] = (p_1'[\mathrm{rec}\, y.p_1/y])[q/x].\end{aligned}$$

□

Lemma 5.8 *Let $p_1, p_2 \in \mathcal{L}$ be processes, $\sigma \in \mathcal{P}(\mathcal{R})^V$ be an environment and $(a_1, a_2), (b_1, b_2) \in$*

$\mathrm{Sync}_{C,\sigma}(p_1, p_2)$ *be synchronization pairs. If* $\mathrm{res}(a_1) \cap \mathrm{res}(b_2) \neq \emptyset$ *then*

$$\mathrm{res}(a_1) \cap \mathrm{res}(b_2) = \mathrm{res}(b_1) \cap \mathrm{res}(a_2) \subseteq C.$$

In particular, the four letters a_1, a_2, b_1, b_2 *are pairwise dependent.*

Proof. We have $\mathrm{res}(a_1) \cap \mathrm{res}(b_2) \subseteq \mathrm{res}(a_1) \cap \mathrm{res}(p_2, \sigma) = \mathrm{res}(a_2) \cap C$ and $\mathrm{res}(a_1) \cap \mathrm{res}(b_2) \subseteq \mathrm{res}(p_1, \sigma) \cap \mathrm{res}(b_2) = \mathrm{res}(b_1) \cap C$. Therefore, $\mathrm{res}(a_1) \cap \mathrm{res}(b_2) \subseteq \mathrm{res}(a_2) \cap \mathrm{res}(b_1) \cap C$. Similarly, we get the reverse inclusion and we obtain $\emptyset \neq \mathrm{res}(a_1) \cap \mathrm{res}(b_2) = \mathrm{res}(b_1) \cap \mathrm{res}(a_2) \subseteq C$. ☐

Proof of Proposition 5.1. We use an induction on p. The premises are impossible to satisfy for the basis cases STOP, $x \in V$ or $c \in \Sigma$.

$p = p_1|_R$: We must have $\mathrm{res}(ab) \subseteq R$, $p_1 \xrightarrow[\sigma]{a} p_1'$ and $p_1 \xrightarrow[\sigma]{b} p_1''$. By induction we deduce that $a\, I\, b$ and there exists $p_1''' \in \mathcal{L}$ with $p_1' \xrightarrow[\sigma]{b} p_1'''$ and $p_1'' \xrightarrow[\sigma]{a} p_1'''$. The result follows with $p''' = p_1'''|_R$.

$p = \mathrm{rec}\, x.p_1$: Let $\sigma' = \sigma[x \mapsto \mathrm{res}(p, \sigma)]$. We must have $p_1 \xrightarrow[\sigma']{a} p_1'$ and $p_1 \xrightarrow[\sigma']{b} p_1''$. By induction we deduce that $a\, I\, b$ and there exists $p_1''' \in \mathcal{L}$ with $p_1' \xrightarrow[\sigma']{b} p_1'''$ and $p_1'' \xrightarrow[\sigma']{a} p_1'''$. By Lemma 5.7 we deduce that $p' = p_1'[p/x] \xrightarrow[\sigma]{b} p_1'''[p/x] = p'''$. Similarly we have $p'' = p_1''[p/x] \xrightarrow[\sigma]{a} p'''$.

$p = p_1 \circ p_2$: There are three cases to consider.

- If $p_1 \xrightarrow[\sigma]{a} p_1'$ and $p_1 \xrightarrow[\sigma]{b} p_1''$ then by induction we deduce that $a\, I\, b$ and there exists $p_1''' \in \mathcal{L}$ with $p_1' \xrightarrow[\sigma]{b} p_1'''$ and $p_1'' \xrightarrow[\sigma]{a} p_1'''$. The result follows with $p''' = p_1''' \circ p_2$.
- If $p_2 \xrightarrow[\sigma]{a} p_2'$, $p_2 \xrightarrow[\sigma]{b} p_2''$ and $\mathrm{res}(ab) \cap \mathrm{res}(p_1, \sigma) = \emptyset$. Then by induction we deduce that $a\, I\, b$ and there exists $p_2''' \in \mathcal{L}$ with $p_2' \xrightarrow[\sigma]{b} p_2'''$ and $p_2'' \xrightarrow[\sigma]{a} p_2'''$. The result follows with $p''' = p_1 \circ p_2'''$.
- If $p_1 \xrightarrow[\sigma]{a} p_1'$, $p_2 \xrightarrow[\sigma]{b} p_2''$ and $\mathrm{res}(b) \cap \mathrm{res}(p_1, \sigma) = \emptyset$. By Proposition 5.6 we have $\mathrm{res}(p_1, \sigma) = \mathrm{res}(a) \cup \mathrm{res}(p_1', \sigma)$ and we deduce that $a\, I\, b$ and $\mathrm{res}(b) \cap \mathrm{res}(p_1', \sigma) = \emptyset$. The result follows with $p''' = p_1' \circ p_2''$.

$p = p_1 \underset{C}{\|} p_2$: Parallel composition is the most difficult operator to deal with. We distinguish four cases.

- If $\mathrm{res}(ab) \cap (C \cup \mathrm{res}(p_2, \sigma)) = \emptyset$. Then $p_1 \xrightarrow[\sigma]{a} p_1'$ and $p_1 \xrightarrow[\sigma]{b} p_1''$ and we conclude easily by induction
- If $\mathrm{res}(a) \cap (C \cup \mathrm{res}(p_2, \sigma)) = \emptyset$ and $\mathrm{res}(b) \cap (C \cup \mathrm{res}(p_1, \sigma)) = \emptyset$. Then $p_1 \xrightarrow[\sigma]{a} p_1'$ and $p_2 \xrightarrow[\sigma]{b} p_2''$. By Proposition 5.6 we have $\mathrm{res}(p_1, \sigma) = \mathrm{res}(a) \cup \mathrm{res}(p_1', \sigma)$ hence $a\, I\, b$ and $\mathrm{res}(b) \cap (C \cup \mathrm{res}(p_1', \sigma)) = \emptyset$. Therefore, $p' = p_1' \underset{C}{\|} p_2 \xrightarrow[\sigma]{b} p_1' \underset{C}{\|} p_2'' = p'''$. Similarly, we show that $p'' = p_1 \underset{C}{\|} p_2'' \xrightarrow[\sigma]{a} p'''$.
- If $a = a_1 \| a_2$ with $(a_1, a_2) \in \mathrm{Sync}_{C,\sigma}(p_1, p_2)$ and $\mathrm{res}(b) \cap (C \cup \mathrm{res}(p_2, \sigma)) = \emptyset$. Then $p_1 \xrightarrow[\sigma]{a_1} p_1'$, $p_2 \xrightarrow[\sigma]{a_2} p_2'$ and $p' = p_1' \underset{C}{\|} p_2'$. Moreover, $p_1 \xrightarrow[\sigma]{b} p_1''$ and $p'' = p_1'' \underset{C}{\|} p_2$. By induction, we deduce that $a_1\, I\, b$ and there exists $p_1''' \in \mathcal{L}$ with $p_1' \xrightarrow[\sigma]{b} p_1'''$ and $p_1'' \xrightarrow[\sigma]{a_1} p_1'''$.

  By Proposition 5.6, $\mathrm{res}(p_2, \sigma) = \mathrm{res}(a_2) \cup \mathrm{res}(p_2', \sigma)$ hence $a_2\, I\, b$ and $\mathrm{res}(b) \cap (C \cup$

$\mathrm{res}(p_2', \sigma)) = \emptyset$. Therefore, $p' = p_1' \underset{C}{\|} p_2' \xrightarrow[\sigma]{b} p_1''' \underset{C}{\|} p_2' = p'''$.

Now $\mathrm{res}(p_1, \sigma) = \mathrm{res}(b) \cup \mathrm{res}(p_1'', \sigma)$ by Proposition 5.6. Since $a_2 \, I \, b$ we deduce that $\mathrm{res}(a_2) \cap \mathrm{res}(p_1, \sigma) = \mathrm{res}(a_2) \cap \mathrm{res}(p_1'', \sigma)$ and therefore $a = (a_1, a_2) \in \mathrm{Sync}_{C,\sigma}(p_1'', p_2)$. It follows $p'' = p_1'' \underset{C}{\|} p_2 \xrightarrow[\sigma]{a} p'''$.

- If $a = a_1 \| a_2$, $b = b_1 \| b_2$ with $(a_1, a_2), (b_1, b_2) \in \mathrm{Sync}_{C,\sigma}(p_1, p_2)$. Then $p_1 \xrightarrow[\sigma]{a_1} p_1'$, $p_2 \xrightarrow[\sigma]{a_2} p_2'$ and $p' = p_1' \underset{C}{\|} p_2'$. Also $p_1 \xrightarrow[\sigma]{b_1} p_1''$, $p_2 \xrightarrow[\sigma]{b_2} p_2''$ and $p'' = p_1'' \underset{C}{\|} p_2''$. By induction, we have $a_1 \, I \, b_1$ and there exists $p_1''' \in \mathcal{L}$ with $p_1' \xrightarrow[\sigma]{b_1} p_1'''$ and $p_1'' \xrightarrow[\sigma]{a_1} p_1'''$. By induction, we also have $a_2 \, I \, b_2$ and there exists $p_2''' \in \mathcal{L}$ with $p_2' \xrightarrow[\sigma]{b_2} p_2'''$ and $p_2'' \xrightarrow[\sigma]{a_2} p_2'''$. Using Lemma 5.8 we deduce that $a_1 \, I \, b_2$ and $a_2 \, I \, b_1$ and we deduce that $a \, I \, b$ since $\mathrm{res}(a) = \mathrm{res}(a_1 \| a_2) = \mathrm{res}(a_1) \cup \mathrm{res}(a_2)$ and similarly for b.

  By Proposition 5.6, $\mathrm{res}(p_2, \sigma) = \mathrm{res}(b_2) \cup \mathrm{res}(p_2'', \sigma)$ and we deduce that $\mathrm{res}(a_1) \cap \mathrm{res}(p_2, \sigma) = \mathrm{res}(a_1) \cap \mathrm{res}(p_2'', \sigma)$. Similarly, $\mathrm{res}(a_2) \cap \mathrm{res}(p_1, \sigma) = \mathrm{res}(a_2) \cap \mathrm{res}(p_1'', \sigma)$. We deduce that $(a_1, a_2) \in \mathrm{Sync}_{C,\sigma}(p_1'', p_2'')$ and $p'' = p_1'' \underset{C}{\|} p_2'' \xrightarrow[\sigma]{a} p_1''' \underset{C}{\|} p_2''' = p'''$. We obtain similarly $p' = p_1' \underset{C}{\|} p_2' \xrightarrow[\sigma]{b} p'''$.

□

Proof of Proposition 5.2. We proceed by induction. The hypothesis is impossible to fulfill for STOP, $x \in V$ or $b \neq a$. The result is trivial for $p = a$ and follows immediately by induction for restriction, recursion and weak sequential composition. The most interesting case is parallel composition: $p = p_1 \underset{C}{\|} p_2$. There are two cases.

- If $\mathrm{res}(a) \cap (C \cup \mathrm{res}(p_2, \sigma)) = \emptyset$ then necessarily (Proposition 5.6), $p_1 \xrightarrow[\sigma]{a} p_1'$, $p_1 \xrightarrow[\sigma]{a} p_1''$, $p' = p_1' \underset{C}{\|} p_2$ and $p'' = p_1'' \underset{C}{\|} p_2$. By induction we obtain $p_1' = p_1''$ and we deduce $p' = p''$.
- If $a = a_1 \| a_2 = b_1 \| b_2$ with $(a_1, a_2), (b_1, b_2) \in \mathrm{Sync}_{C,\sigma}(p_1, p_2)$. Then $p_1 \xrightarrow[\sigma]{a_1} p_1'$, $p_2 \xrightarrow[\sigma]{a_2} p_2'$ and $p' = p_1' \underset{C}{\|} p_2'$. Also $p_1 \xrightarrow[\sigma]{b_1} p_1''$, $p_2 \xrightarrow[\sigma]{b_2} p_2''$ and $p'' = p_1'' \underset{C}{\|} p_2''$.

  We claim that $a_1 = b_1$ and $a_2 = b_2$. Indeed, we have $\emptyset \neq \mathrm{res}(a) = \mathrm{res}(a_1) \cup \mathrm{res}(a_2) = \mathrm{res}(b_1) \cup \mathrm{res}(b_2)$. Either $\mathrm{res}(a_1) \cap \mathrm{res}(b_1) \neq \emptyset$ or $\mathrm{res}(a_1) \cap \mathrm{res}(b_2) \neq \emptyset$. We deduce from Lemma 5.8 that in both cases a_1 and b_1 are dependent. It follows $a_1 = b_1$ from Proposition 5.1 which proves the claim.

  By induction, we obtain $p_1' = p_1''$ and $p_2' = p_2''$, whence $p' = p''$.

□

We conclude this section with a quite natural result. As one might expect, the operational behaviour of a term only depends on the values taken by the syntactic environments at free variables of the term.

Proposition 5.9 *Let $p \in \mathcal{L}$ be processes and $\sigma, \sigma' \in \mathcal{P}(\mathcal{R})^V$ be syntactic environments such that $\sigma(y) = \sigma'(y)$ $\forall y \in \mathrm{F}(p)$. Then, $X_{\mathbb{F}}(p, \sigma) = X_{\mathbb{F}}(p, \sigma')$.*

Proof. In order to show this we first prove that under the hypotheses of the proposition, we have $p \xrightarrow[\sigma]{a} p' \quad \Leftrightarrow \quad p \xrightarrow[\sigma']{a} p'$. This can be shown by structural induction. It is also an easy corollary of Lemma 5.7: for each $x \notin \mathrm{F}(p)$ we apply this lemma with $q = \mathrm{STOP}|_{\sigma'(x)}$. Next we use an induction on the length to obtain a similar result for finite traces: $p \xrightarrow[\sigma]{s} p' \Leftrightarrow p \xrightarrow[\sigma']{s} p'$. Finally, we conclude using Lemma 4.1. $\square$

6 Denotational Semantics

In this section, we define a denotational semantics for our language that we later show is adequate and fully abstract with respect to the operational semantics given by the transition system we presented in the previous section. The semantics takes its values in the family $[\mathbb{F}^V \to \mathbb{F}]$ of continuous maps from $\mathbb{F}^V$ to the underlying domain $\mathbb{F}$ of resource traces. As was the case with the resources model of Section 4, the semantics of a closed process p is a constant map, which means it is simply a resource trace. But, in order to give the semantics of recursion, we also have to consider terms with free variables.

We begin by defining the family of *semantic environments* to be the mappings $\sigma \colon V \to \mathbb{F}$, and we endow this with the domain structure from the target domain $\mathbb{F}$, regarding $\mathbb{F}^V$ as a product on V-copies of $\mathbb{F}$. The semantics of an arbitrary process $p \in \mathcal{L}$ is a continuous map from $\mathbb{F}^V$ to $\mathbb{F}$. The semantics of a recursive process $\mathrm{rec}\, x.p$ is obtained by considering some fixed point of the semantic map associated with the body p of the recursion. We obtain a compositional semantics by defining the structure of an Ω-algebra on $[\mathbb{F}^V \to \mathbb{F}]$. We begin with the simplest operations – the nullary operators.

The denotational semantics of constants and of variables are defined by the maps:

$$\begin{aligned} [\![\mathrm{STOP}]\!] &\in [\mathbb{F}^V \to \mathbb{F}] \quad \text{defined by} \quad [\![\mathrm{STOP}]\!](\sigma) = (1, \mathcal{R}) \\ [\![a]\!] &\in [\mathbb{F}^V \to \mathbb{F}] \quad \text{defined by} \quad [\![a]\!](\sigma) = (a, \emptyset) \\ [\![x]\!] &\in [\mathbb{F}^V \to \mathbb{F}] \quad \text{defined by} \quad [\![x]\!](\sigma) = \sigma(x) \end{aligned}$$

The first two clearly are continuous, since they are constant maps. As for the last, this mapping amounts to projection of the element $\sigma \in \mathbb{F}^V$ onto its x-component, and since we endow $\mathbb{F}^V$ with the product topology, this mapping also is continuous.

Next we define the semantics of restriction, weak sequential composition and parallel composition. Rather than define the interpretations of these operators directly at the level of $[\mathbb{F}^V \to \mathbb{F}]$, we instead define continuous interpretations on $\mathbb{F}$, and then extend to $[\mathbb{F}^V \to \mathbb{F}]$ in a pointwise fashion. Proposition 6.1 is the link that shows this approach induces continuous interpretations on $[\mathbb{F}^V \to \mathbb{F}]$.

Proposition 6.1 (cf. [1]) *Let $\omega \in \Omega_n$ be an n-ary operator of our language and assume that we have defined a corresponding* continuous *operation $\overline{\omega} : \mathbb{F}^n \to \mathbb{F}$. We define the interpretation of the operation ω on $[\mathbb{F}^V \to \mathbb{F}]$ in a pointwise fashion:*

$$\widetilde{\omega} \colon [\mathbb{F}^V \to \mathbb{F}]^n \to [\mathbb{F}^V \to \mathbb{F}] \text{ defined by} \widetilde{\omega}(f_1, \dots, f_n)(\sigma) = \overline{\omega}(f_1(\sigma), \dots, f_n(\sigma)).$$

Then

(i) $\widetilde{\omega}(f_1, \dots, f_n) : \mathbb{F}^V \to \mathbb{F}$ *is a continuous map, and*

(ii) $\widetilde{\omega} : [\mathbb{F}^V \to \mathbb{F}]^n \to [\mathbb{F}^V \to \mathbb{F}]$ *also is continuous.* $\square$

We use this approach for weak sequential composition, restrictions and parallel composition. Hence, we obtain continuous intrepretations on $[\mathbb{F}^V \to \mathbb{F}]$ of these operators by defining only continuous interpretations on $\mathbb{F}$. Then, the semantics of a compound process is defined by

$$[\![\omega(p_1,\ldots,p_n)]\!] = \widetilde{\omega}([\![p_1]\!],\ldots,[\![p_n]\!]),$$

and this semantics $[\![\omega(p_1,\ldots,p_n)]\!]$ is automatically continuous.

In the following we may use the same notation for ω, $\overline{\omega}$ and $\widetilde{\omega}$. The actual operation should always be clear from the context.

6.1 Weak Sequential Composition

For weak sequential composition, we use the following result about the concatenation of resource traces.

Proposition 6.2 ([5]) *Concatenation over resource traces is a continuous operation. Moreover, for all $(x_1, x_2) \in \mathbb{F}^2$, we have*

$$\mathrm{res}(x_1 \cdot x_2) = \mathrm{res}(x_1) \cup \mathrm{res}(x_2). \qquad \square$$

The interpretation of $\circ$ on $[\mathbb{F}^V \to \mathbb{F}]$ is then

$$\circ\colon [\mathbb{F}^V \to \mathbb{F}]^2 \to [\mathbb{F}^V \to \mathbb{F}] \quad \text{defined by} \quad (f_1 \circ f_2)(\sigma) = f_1(\sigma) \cdot f_2(\sigma).$$

6.2 Restriction

For restriction and parallel composition, we need to define new operations on traces since they have not been introduced so far. We start with restriction, which is the easier of the two. We obtain restriction as the composition of two continuous maps.

Let $R \subseteq \mathcal{R}$ be a fixed resource set. We first introduce

$$\mathbb{F}_R = \{x \in \mathbb{F} \mid \mathrm{res}(\mathrm{Re}(x)) \subseteq R\},$$

the set of resource traces whose real parts use resources from R only. Note that if some set $X \subseteq \mathbb{F}_R$ is pairwise consistent in $\mathbb{F}$, then its least upper bound in $\mathbb{F}$ exists and actually belongs to $\mathbb{F}_R$. Therefore, $\mathbb{F}_R$ is also a consistently complete domain. Recall also that $\uparrow x = \{y \in \mathbb{F} \mid x \sqsubseteq y\}$ denotes the upper set of $x \in \mathbb{F}$. Now we define

$$f\colon \mathbb{F} \to \mathbb{F}_R \quad \text{by} \quad x \mapsto \sqcup\{y \in \mathbb{F}_R \mid y \sqsubseteq x\},$$

and

$$g\colon \mathbb{F}_R \to \uparrow(1, R) \subseteq \mathbb{F} \quad \text{by} \quad (s, S) \mapsto (s, S \cap R),$$

and finally,

$$|_R = g \circ f\colon \mathbb{F} \to \mathbb{F}.$$

Note first that all these mappings are well-defined. Indeed, the set $Y = \{y \in \mathbb{F}_R \mid y \sqsubseteq x\}$ is bounded above in $\mathbb{F}$, so its sup exists and belongs to $\mathbb{F}_R$. To show that g is well-defined, one only has to observe that $\mathrm{resinf}(s) \subseteq S \cap R$ when $(s, S) \in \mathbb{F}_R$. Therefore, $|_R$ is well-defined, too. We now investigate the properties of these mappings; in the following, we denote the complement of $R \subseteq \mathcal{R}$ by $\overline{R}$.

Lemma 6.3

(i) *f is continuous.*

(ii) $\mathrm{Re}(f(x)) = \mu_{\overline{R}}(x) = \sqcup\{r \in \mathbb{R} \mid \mathrm{res}(r) \subseteq R$ *and* $r \leq \mathrm{Re}(x)\}$,
$\mathrm{Im}(f(x)) = \sigma_{\overline{R}}(x) = \mathrm{Im}(x) \cup \mathrm{res}(\mathrm{Re}(f(x))^{-1}\mathrm{Re}(x))$.

(iii) $\mathrm{res}(f(x)) = \mathrm{res}(x)$.

Proof. (i) f is clearly monotone. Now, let $X \subseteq \mathbb{F}$ be a directed set. Since f is monotone, we know that $f(X)$ is bounded above by $f(\sqcup X)$, whence $\sqcup f(X) \sqsubseteq f(\sqcup X)$. Conversely, let k be a compact (finite) trace such that

$$k \sqsubseteq f(\sqcup X) = \sqcup\{y \in \mathbb{F}_R \mid y \sqsubseteq \sqcup X\}.$$

We have $k \sqsubseteq y$ for some $y \in \mathbb{F}_R$ such that $y \sqsubseteq \sqcup X$. Therefore, $k \in \mathbb{F}_R$ and $k \sqsubseteq \sqcup X$. Let $x \in X$ be such that $k \sqsubseteq x$. We have $k \sqsubseteq f(x) \sqsubseteq \sqcup f(X)$ which concludes the proof of the first point.

(ii) Let $x \in \mathbb{F}$. We define $s = \sqcup\{r \in \mathbb{R} \mid \mathrm{res}(r) \subseteq R \text{ and } r \leq \mathrm{Re}(x)\}$. Then indeed $s \leq \mathrm{Re}(x)$ and $\mathrm{res}(s) \subseteq R$. Let $S = \mathrm{Im}(x) \cup \mathrm{res}(s^{-1}\mathrm{Re}(x))$. Clearly we have $(s, S) \sqsubseteq x$ hence $(s, S) \sqsubseteq f(x)$. Conversely, let $y \in \mathbb{F}_R$ be such that $y \sqsubseteq x$. We have $\mathrm{Re}(y) \leq \mathrm{Re}(x)$ and $\mathrm{res}(\mathrm{Re}(y)) \subseteq R$, hence $\mathrm{Re}(y) \leq s$. Now, $\mathrm{Im}(y) \supseteq \mathrm{Im}(x) \cup \mathrm{res}(\mathrm{Re}(y)^{-1}\mathrm{Re}(x)) = S \cup \mathrm{res}(\mathrm{Re}(y)^{-1}s)$ and we also have $y \sqsubseteq (s, S)$. Therefore, $f(x) = (s, S)$ which proves the second point.

(iii) follows directly from the previous point. $\square$

Lemma 6.4 *The mapping g is continuous and* $\mathrm{res}(g(x)) = \mathrm{res}(x) \cap R$ *for all* $x \in \mathbb{F}_R$.

Proof. Let $(s, S) \sqsubseteq (t, T)$ be resource traces in $\mathbb{F}_R$. Then $s \leq t$ and $T \supseteq S \cup \mathrm{res}(s^{-1}t)$. Therefore $T \cap R \supseteq (S \cap R) \cup \mathrm{res}(s^{-1}t)$ which shows that g is monotone.

Let $X \subseteq \mathbb{F}_R$ be directed. Then,

$$\begin{aligned}
\mathrm{Re}(g(\sqcup X)) &= \mathrm{Re}(\sqcup X) = \bigsqcup_{x \in X} \mathrm{Re}(x) = \bigsqcup_{x \in X} \mathrm{Re}(g(x)) = \mathrm{Re}(\sqcup g(X)) \\
\mathrm{Im}(g(\sqcup X)) &= \mathrm{Im}(\sqcup X) \cap R = \bigcap_{x \in X} \mathrm{Im}(x) \cap R = \bigcap_{x \in X} \mathrm{Im}(g(x)) = \mathrm{Im}(\sqcup g(X))
\end{aligned}$$

which shows that g is continuous.

The last part of the proposition is clear. $\square$

From the previous lemmas, we obtain directly

Proposition 6.5 *The mapping* $|_R : \mathbb{F} \to \mathbb{F}$ *defined by* $x|_R = g \circ f(x)$ *is continuous. Moreover, we have* $\mathrm{res}(x|_R) = \mathrm{res}(x) \cap R$ *for all* $x \in \mathbb{F}$. $\square$

From this, we obtain the interpretation on $[\mathbb{F}^V \to \mathbb{F}]$ of the restriction operator:

$$|_R \colon [\mathbb{F}^V \to \mathbb{F}] \to [\mathbb{F}^V \to \mathbb{F}] \quad \text{defined by} \quad (f|_R)(\sigma) = f(\sigma)|_R.$$

6.3 Parallel Composition

We now turn to the semantics of parallel composition. This requires some preliminary definitions and results before we get to the definition of the parallel operation over resource traces.

We use the results from Section 2.3 about alphabetic mappings to define the semantics of the parallel composition. Recall first that we assumed the existence of a parallel composition over actions of the alphabet: $\| : \Sigma^2 \to \Sigma$ that satisfies $\mathrm{res}(a_1\|a_2) = \mathrm{res}(a_1) \cup \mathrm{res}(a_2)$ for all $(a_1, a_2) \in \Sigma^2$. The action $a_1\|a_2$ represents the result of synchronizing a_1 and a_2 in a parallel composition.

We introduce the alphabet $\Sigma' = (\Sigma \cup \{1\})^2 \setminus \{(1, 1)\}$ with the resource map $\mathrm{res}'(a_1, a_2) = \mathrm{res}(a_1) \cup \mathrm{res}(a_2)$ and the associated dependence relation D'. Then we consider the sets $\mathbb{R}(\Sigma', \mathrm{res}')$ and $\mathbb{F}(\Sigma', \mathrm{res}')$ of real traces and of resource traces over the resource alphabet $\mathrm{res}' : \Sigma' \to \mathcal{P}(\mathcal{R})$. We define the alphabetic mappings

$$
\begin{array}{lll}
\Pi_1 : \Sigma' \to \Sigma \cup \{1\} & \text{by} & \Pi_1(a_1, a_2) = a_1, \\
\Pi_2 : \Sigma' \to \Sigma \cup \{1\} & \text{by} & \Pi_2(a_1, a_2) = a_2, \text{ and} \\
\Pi\ \ : \Sigma' \to \Sigma & \text{by} & \Pi(a_1, a_2) = a_1\|a_2.
\end{array}
$$

where we set $a\|1 = 1\|a = a$. Note that $\mathrm{res}(\Pi_1(a_1, a_2)) \subseteq \mathrm{res}'(a_1, a_2)$, $\mathrm{res}(\Pi_2(a_1, a_2)) \subseteq \mathrm{res}'(a_1, a_2)$ and $\mathrm{res}(\Pi(a_1, a_2)) = \mathrm{res}'(a_1, a_2)$. Therefore, the three mappings extend to continuous morphisms over real traces (Proposition 2.3) and to continuous maps over resource traces. Moreover, Π is also a morphism over resource traces (Proposition 2.4).

Now, we consider a subset $C \subseteq \mathcal{R}$ of resources on which we want to synchronize. Recall that we view these resources as *channels*.

We fix two resource traces $x_1 = (s_1, S_1)$ and $x_2 = (s_2, S_2)$ of $\mathbb{F}(\Sigma, \mathrm{res})$ and we want to define a resource trace $x_1 \underset{C}{\|} x_2$ which represents the parallel composition of x_1 and x_2 with synchronization on the channels of C.

We first define a resource trace $\varphi_C(x_1, x_2) \in \mathbb{F}(\Sigma', \mathrm{res}')$ which represents the parallel composition of x_1 and x_2. Then we set $x_1 \underset{C}{\|} x_2 = \Pi(\varphi_C(x_1, x_2))$. Since the mapping Π is continuous, in order to obtain a continuous semantics for parallel composition, we only need to show that the mapping $\varphi_C : \mathbb{F}(\Sigma, \mathrm{res})^2 \to \mathbb{F}(\Sigma', \mathrm{res}')$ is continuous as well.

In analogy to the set $\mathrm{Sync}_{C,\sigma}(p_1, p_2)$ for terms $p_1, p_2 \in \mathcal{L}$, given resource traces $x_1 = (s_1, S_1)$ and $x_2 = (s_2, S_2)$, we can define the *synchronization set* $\mathrm{Sync}_C(x_1, x_2)$ as the set of pairs $(a_1, a_2) \in \mathrm{alph}(s_1) \times \mathrm{alph}(s_2)$ satisfying

$$\mathrm{res}(a_1) \cap C = \mathrm{res}(a_2) \cap C = \mathrm{res}(a_1) \cap \mathrm{res}(x_2) = \mathrm{res}(a_2) \cap \mathrm{res}(x_1) \neq \emptyset.$$

Then the set $\Sigma'_C(x_1, x_2)$ of actions which may occur in $\varphi_C(x_1, x_2)$ is defined as

$$
\begin{aligned}
\Sigma'_C(x_1, x_2) = \{&(a_1, 1) \in \mathrm{alph}(s_1) \times \{1\} \mid \mathrm{res}(a_1) \cap (C \cup \mathrm{res}(x_2)) = \emptyset\} \\
&\cup \{(1, a_2) \in \{1\} \times \mathrm{alph}(s_2) \mid \mathrm{res}(a_2) \cap (C \cup \mathrm{res}(x_1)) = \emptyset\} \\
&\cup \mathrm{Sync}_C(x_1, x_2),
\end{aligned}
$$

The first two sets in this union correspond to *local events:* these should not use any channel on which we want to synchronize ($\mathrm{res}(a_1) \cap C = \emptyset$). In addition, the condition $\mathrm{res}(a_1) \cap \mathrm{res}(x_2) = \emptyset$ implies that a local event does not conflict with any event of the other component, which ensures parallel composition does not involve nondeterministic choice. The set $\mathrm{Sync}_{C,\sigma}(x_1, x_2)$ corresponds to synchronization events. In order to synchronize, two events must use exactly the same channels and, in order to assure determinism, neither should conflict with resources of the other component.

Lemma 6.6

(i) *Let* $(a_1, a_2) \in \mathrm{Sync}_C(x_1, x_2)$. *Then*

$$\begin{aligned}\mathrm{res}(a_1) \cap \mathrm{res}(a_2) &= \mathrm{res}(a_1) \cap C = \mathrm{res}(a_2) \cap C \\ &= \mathrm{res}(a_1) \cap \mathrm{res}(x_2) = \mathrm{res}(a_2) \cap \mathrm{res}(x_1) \neq \emptyset.\end{aligned}$$

(ii) *Let* (a_1, a_2) *and* (b_1, b_2) *be letters in* $\Sigma'_C(x_1, x_2)$. *If* $\mathrm{res}(a_1) \cap \mathrm{res}(b_2) \neq \emptyset$ *then*

$$\mathrm{res}(a_1) \cap \mathrm{res}(b_2) = \mathrm{res}(b_1) \cap \mathrm{res}(a_2) \subseteq C.$$

In particular, the four letters a_1, a_2, b_1, b_2 *are pairwise dependent.*

(iii) *The mapping*

$$\begin{aligned}\Sigma'_C : (\mathbb{F}(\Sigma, \mathrm{res}), \sqsubseteq)^2 &\to (\mathcal{P}(\Sigma'), \subseteq) \\ (x_1, x_2) &\mapsto \Sigma'_C(x_1, x_2)\end{aligned}$$

is continuous.

Proof. (i) We have

$$\begin{aligned}\mathrm{res}(a_1) \cap \mathrm{res}(a_2) &\subseteq \mathrm{res}(a_1) \cap \mathrm{res}(x_2) \\ &= \mathrm{res}(a_1) \cap C = \mathrm{res}(a_2) \cap C \subseteq \mathrm{res}(a_1) \cap \mathrm{res}(a_2).\end{aligned}$$

(ii) The proof is similar to that of Lemma 5.8 once we have noted that $\mathrm{res}(a_1) \cap \mathrm{res}(b_2) \neq \emptyset$ implies $(a_1, a_2), (b_1, b_2) \in \mathrm{Sync}_C(x_1, x_2)$.

(iii) For $i = 1, 2$, let $x_i \sqsubseteq y_i$ with $x_i = (s_i, S_i)$ and $y_i = (t_i, T_i)$. Then, $\mathrm{alph}(s_i) \subseteq \mathrm{alph}(t_i)$ and $\mathrm{res}(y_i) \subseteq \mathrm{res}(x_i)$. It follows that $(a_1, 1) \in \Sigma'_C(x_1, x_2)$ implies $(a_1, 1) \in \Sigma'_C(y_1, y_2)$.

Now, let $(a_1, a_2) \in \mathrm{Sync}_C(x_1, x_2)$. We have, $\mathrm{res}(a_1) \cap \mathrm{res}(x_2) = \mathrm{res}(a_2) \cap C \subseteq \mathrm{res}(s_2) \subseteq \mathrm{res}(y_2) \subseteq \mathrm{res}(x_2)$. Therefore, $\mathrm{res}(a_1) \cap \mathrm{res}(x_2) = \mathrm{res}(a_1) \cap \mathrm{res}(y_2)$ and we deduce that $(a_1, a_2) \in \Sigma'_C(y_1, y_2)$.

Now, let $Y \subseteq \mathbb{F}(\Sigma, \mathrm{res})^2$ be directed and let $(y_1, y_2) = \sqcup Y$ with $y_i = (t_i, T_i)$. Since Σ'_C is monotone, we already know that $\Sigma'_C(Y)$ is directed and $\sqcup \Sigma'_C(Y) \subseteq \Sigma'_C(\sqcup Y)$. Conversely, let $(a_1, a_2) \in \Sigma'_C(\sqcup Y)$. Since Y is directed, we can choose $(x_1, x_2) \in Y$ such that $x_i = (s_i, T_i)$ with $a_i \in \mathrm{alph}(s_i)$ and $\mathrm{res}(x_i) = \mathrm{res}(y_i)$. Therefore, $(a_1, a_2) \in \Sigma'_C(x_1, x_2)$ which concludes the proof. □

Now we introduce the set

$$R_C(x_1, x_2) = \{r \in \mathbb{R}(\Sigma'_C(x_1, x_2), \mathrm{res}') \mid \Pi_i(r) \leq \mathrm{Re}(x_i) \text{ for } i = 1, 2\},$$

whose least upper bound will be the real part of the parallel composition of x_1 and x_2. The following proposition shows that indeed this least upper bound exists.

Proposition 6.7 *The set* $R_C(x_1, x_2)$ *is pairwise consistent. Moreover, it is the lower set of its least upper bound:*

$$R_C(x_1, x_2) = \downarrow(\sqcup R_C(x_1, x_2)).$$

Proof. Since Π_1 and Π_2 are monotone, it is clear that $R_C(x_1, x_2)$ is a lower set. Assume that there exist two real traces $r, r' \in R_C(x_1, x_2)$ which are inconsistent. Since $R_C(x_1, x_2)$ is a lower set, we may also assume that s and r' are consistent for all $s < r$, and that s'

and r are consistent for all $s' < r'$. Let us write $r = (r \wedge r')sa$ and $r' = (r \wedge r')s'b$ with $a = (a_1, a_2) \in \Sigma'$ and $b = (b_1, b_2) \in \Sigma'$ (this writing turns out to be unique). Clearly $(r \wedge r')s \wedge (r \wedge r')s'b = r \wedge r'$ and since $(r \wedge r')s$ is consistent with $r' = (r \wedge r')s'b$ it follows that $\mathrm{res}(s) \cap \mathrm{res}(s'b) = \emptyset$. Similarly, we obtain $\mathrm{res}(s') \cap \mathrm{res}(sa) = \emptyset$.

Let $t = (r \wedge r')ss' = (r \wedge r')s \vee (r \wedge r')s'$. Note that ta and tb are not consistent since $r \leq ta$ and $r' \leq tb$ are not. Hence, $a \neq b$ and $a\, D\, b$. We have

$$\begin{aligned}\Pi_i(ta) &= \Pi_i(r \wedge r')\Pi_i(sa)\Pi_i(s') = \Pi_i(r \wedge r')\Pi_i(sa) \vee \Pi_i(r \wedge r')\Pi_i(s') \\ &= \Pi_i(r) \vee \Pi_i((r \wedge r')s') \leq s_i\end{aligned}$$

Therefore $ta \in R_C(x_1, x_2)$ and similarly $tb \in R_C(x_1, x_2)$. Now, $\Pi_i(ta) = \Pi_i(t)a_i \leq s_i$ and $\Pi_i(tb) = \Pi_i(t)b_i \leq s_i$, hence we have either $a_i = b_i$ or $a_i\, I\, b_i$. Since $a\, D\, b$, we have for instance $a_1\, D\, b_1$ (Lemma 6.6 (ii)) and then $a_1 = b_1$. Since $a \neq b$, it is not possible to have $a_2 = b_2 = 1$. Hence for instance $a_2 \neq 1$ and we obtain $\mathrm{res}(b_1) \cap C = \mathrm{res}(a_1) \cap C = \mathrm{res}(a_2) \cap C \neq \emptyset$. It follows that $\mathrm{res}(b_2) \cap C = \mathrm{res}(b_1) \cap C = \mathrm{res}(a_2) \cap C \neq \emptyset$. Therefore $a_2\, D\, b_2$ and we get $a_2 = b_2$ which contradicts $a \neq b$.

We have proved that $R_C(x_1, x_2)$ is consistent, hence it's least upper bound exists. Since Π_i are continuous it is clear that $\sqcup R_C(x_1, x_2) \in R_C(x_1, x_2)$. □

Finally, we define the set

$$\begin{aligned}X_C(x_1, x_2) = \{(t, T) \in \mathbb{F}(\Sigma', \mathrm{res}') \mid \mathrm{alph}(t) \subseteq \Sigma'_C(x_1, x_2) \ \text{ and} \\ \Pi_i(t, T) \sqsubseteq x_i \text{ for } i = 1, 2\}\end{aligned}$$

Proposition 6.8 *The set $X_C(x_1, x_2)$ has a least upper bound $x = (r, R)$ which is given by*

$$\begin{aligned}r &= \sqcup R_C(x_1, x_2) \\ R &= S_1 \cup S_2 \cup \mathrm{res}(r_1^{-1}s_1) \cup \mathrm{res}(r_2^{-1}s_2)\end{aligned}$$

where $r_i = \Pi_i(r)$.

Moreover, $X_C(x_1, x_2) = {\downarrow} x$ is the lower set of its least upper bound and $\mathrm{res}(x) = \mathrm{res}(x_1) \cup \mathrm{res}(x_2)$.

Proof. Since Π_i are monotone, the set $X_C(x_1, x_2)$ is clearly a lower set. Now, it is easy to check that $x \in X_C(x_1, x_2)$. Conversely, let $x' = (r', R') \in X_C(x_1, x_2)$. Clearly, $r' \in R_C(x_1, x_2)$ and we obtain $r' \leq r$. Then,

$$S_i \cup \mathrm{res}({r'_i}^{-1}s_i) = S_i \cup \mathrm{res}({r'_i}^{-1}r_i) \cup \mathrm{res}(r_i^{-1}s_i) \subseteq R'.$$

Since $\mathrm{res}({r'}^{-1}r) = \mathrm{res}({r'_1}^{-1}r_1) \cup \mathrm{res}({r'_2}^{-1}r_2)$, we deduce that $R \cup \mathrm{res}({r'}^{-1}r) \subseteq R'$ which proves that $x' \sqsubseteq x$.

Finally, since $\mathrm{res}(r) = \mathrm{res}(r_1) \cup \mathrm{res}(r_2)$, we deduce that $\mathrm{res}(x) = \mathrm{res}(x_1) \cup \mathrm{res}(x_2)$. □

Proposition 6.9 *The mapping*

$$\varphi_C \colon \mathbb{F}(\Sigma, \mathrm{res})^2 \to \mathbb{F}(\Sigma', \mathrm{res}') \quad \textit{defined by} \quad \varphi_C(x_1, x_2) = \sqcup X_C(x_1, x_2)$$

is continuous.

Proof. If $(x_1, x_2) \sqsubseteq (x'_1, x'_2)$ then we have $\Sigma'_C(x_1, x_2) \subseteq \Sigma'_C(x'_1, x'_2)$ by Lemma 6.6 (iii) and it follows that $X_C(x_1, x_2) \subseteq X_C(x'_1, x'_2)$ which proves that φ_C is monotone.

Let $Y \subseteq \mathbb{F}(\Sigma, \mathrm{res})^2$ be directed and let $(y_1, y_2) = \sqcup Y$. We may assume that $Y = Y_1 \times Y_2$ with $y_1 = \sqcup Y_1$ and $y_2 = \sqcup Y_2$. We already know that $\varphi_C(Y)$ is directed and that $\sqcup\varphi_C(Y) \sqsubseteq \varphi_C(\sqcup Y) = \varphi_C(y_1, y_2)$. Conversely, let $x = (r, R) \sqsubseteq \varphi_C(y_1, y_2)$ be compact. Then, $x \in X_C(y_1, y_2)$ (Proposition 6.8) and we have $\Pi_i(x) \sqsubseteq y_i = \sqcup Y_i$. Hence, $\Pi_i(x) \sqsubseteq x_i$ for some $x_i \in Y_i$. Since Σ'_C is continuous (Lemma 6.6) and $\{\Sigma'_C(z_1, z_2) \mid (z_1, z_2) \in Y\}$ is a directed finite set, we have $\Sigma'_C(y_1, y_2) \in \{\Sigma'_C(z_1, z_2) \mid (z_1, z_2) \in Y\}$ and we may assume that $\Sigma'_C(x_1, x_2) = \Sigma'_C(y_1, y_2)$. Therefore, we obtain $x \in X_C(x_1, x_2)$ and $x \sqsubseteq \varphi_C(x_1, x_2) \sqsubseteq \sqcup\varphi_C(Y)$ which concludes the proof. $\square$

As announced earlier, we now define the semantics of the parallel composition by $\underset{C}{\|} = \Pi \circ \varphi_C$ and from the above results we obtain directly:

Corollary 6.10 *The mapping*

$$\underset{C}{\|} : \mathbb{F}(\Sigma, \mathrm{res})^2 \to \mathbb{F}(\Sigma, \mathrm{res}) \quad \textit{defined by} \quad x_1 \underset{C}{\|} x_2 = (\Pi \circ \varphi_C)(x_1, x_2)$$

is continuous. Moreover, for all $(x_1, x_2) \in \mathbb{F}^2$, we have

$$\mathrm{res}(x_1 \underset{C}{\|} x_2) = \mathrm{res}(x_1) \cup \mathrm{res}(x_2). \qquad \square$$

As with weak sequential composition and restriction, the interpretation of parallel composition on $[\mathbb{F}^V \to \mathbb{F}]$ is the mapping

$$\underset{C}{\|} : [\mathbb{F}^V \to \mathbb{F}]^2 \to [\mathbb{F}^V \to \mathbb{F}] \quad \text{defined by} \quad (f_1 \underset{C}{\|} f_2)(\sigma) = f_1(\sigma) \underset{C}{\|} f_2(\sigma).$$

6.4 Recursion

Finally, we give the denotational semantics for recursion. For each variable $x \in V$, we define a continuous selfmap $\mathrm{rec}\, x$ of $[\mathbb{F}^V \to \mathbb{F}]$; we use a fixed point of a continuous selfmap from $\mathbb{F}$ to $\mathbb{F}$, but contrary to the classical approach, we do not use the least fixed point semantics.

Example 6.11 We begin with an example. Let $\Sigma = \{a, b, c\}$ with $\mathrm{res}(a) = \{\alpha\}$, $\mathrm{res}(b) = \{\alpha, \gamma\}$ and $\mathrm{res}(c) = \{\gamma\}$. Consider the process $q = \mathrm{rec}\, x.p$ with $p = (a \circ x)$. From the previous sections we know that the semantics of p is the continuous map $[\![p]\!] : \mathbb{F}^V \to \mathbb{F}$ defined by $[\![p]\!](\sigma) = (a, \emptyset) \cdot \sigma(x)$. The semantics of q will be a fixed point of the continuous selfmap from $\mathbb{F}$ to $\mathbb{F}$ defined by $x \mapsto (a, \emptyset) \cdot x$. The fixed point is obtained as follows. Let $x_0 = (1, \mathrm{res}(a))$ and $x_{n+1} = (a, \emptyset) \cdot x_n = (a^{n+1}, \mathrm{res}(a))$. This sequence is increasing and its least upper bound $x_\omega = (a^\omega, \mathrm{res}(a))$ is the semantics of the process q: $[\![q]\!] : \mathbb{F}^V \to \mathbb{F}$ is the constant map which assigns x_ω to any environment $\sigma \in \mathbb{F}^V$. Note that the resource set claimed by q (Section 4) is exactly the resource set used by its semantics: $\mathrm{res}([\![q]\!](\sigma)) = \mathrm{res}(a) = \mathrm{res}(q, \mathrm{res}(\sigma))$.

We do not use a least fixed point semantics since we started with $x_0 = (1, \mathrm{res}(a))$, which is not the least element $(1, \mathcal{R})$ of our domain $\mathbb{F}$. One reason we did not use a least fixed point semantics is that it would claim too many resources for the recursive process. Indeed, let $y_0 = (1, \mathcal{R})$ and $y_{n+1} = (a, \emptyset) \cdot y_n = (a^{n+1}, \mathcal{R})$. Then the least fixed point of the continuous selfmap from $\mathbb{F}$ to $\mathbb{F}$ defined by $x \mapsto (a, \emptyset) \cdot x$ is the least upper bound $y_\omega = (a^\omega, \mathcal{R})$ of the sequence $(y_n)_{n\geq 0}$. Now, $\mathrm{res}(q, \mathrm{res}(\sigma)) = \mathrm{res}(a) \subsetneq \mathcal{R} = \mathrm{res}(y_\omega)$ which leads to problems when we further compose this process.

In order to explain this fact we anticipate a bit. We will see in Proposition 7.9 that, for a process $p \in \mathcal{L}$, any sequence of actions which is allowed by the operational semantics for p is actually a linearization of some prefix of the *real part* of the denotational semantics of p (and vice versa). For example, consider the process $r = q \circ c$, whose denotational semantics is (the constant map) $[\![r]\!] = [\![q]\!] \cdot [\![c]\!] = x_\omega \cdot (c, \emptyset) = (ca^\omega, \mathrm{res}(a))$. Indeed, since q claims only the resource α for its execution, the operational semantics allows any sequence of the form a^n and any sequence of the form $a^n c a^m$ as well. All these sequences are prefixes of the real part of $[\![r]\!]$. If we had used a least fixed point semantics, then the semantics of r would have been $y_\omega \cdot (c, \emptyset) = y_\omega$ and the sequences of the form $a^n c a^m$ allowed by the operational semantics would not have been prefixes of the real part of the denotational semantics.

Since we are not using the classical least fixed point semantics, we have to explain in some detail how our semantics works. We first define the two mappings

$$\varphi\colon [\mathbb{F}^V \to \mathbb{F}] \times \mathbb{F}^V \to [\mathbb{F} \to \mathbb{F}] \quad \text{by} \quad (f, \sigma) \mapsto \varphi_{f,\sigma}$$
$$\psi : [\mathbb{F}^V \to \mathbb{F}] \times \mathbb{F}^V \to [(\mathcal{P}(\mathcal{R}), \supseteq) \to (\mathcal{P}(\mathcal{R}), \supseteq)] \quad \text{by} \quad (f, \sigma) \mapsto \psi_{f,\sigma}$$

where

$$\varphi_{f,\sigma}(y) = f(\sigma[x \mapsto y]),$$
$$\psi_{f,\sigma}(R) = \mathrm{res}(f(\sigma[x \mapsto (1, R)])).$$

Proposition 6.12 *The two mappings φ and ψ are well-defined and continuous.*

Proof. First, the mapping

$$\mathbb{F}^V \times \mathbb{F} \to \mathbb{F}^V \quad \text{defined by} \quad (\sigma, y) \mapsto \sigma[x \mapsto y]$$

which overrides the value of σ at x is clearly continuous. Since applying a function is also a continuous operation in both the function and its argument, we conclude that the mapping

$$[\mathbb{F}^V \to \mathbb{F}] \times \mathbb{F}^V \times \mathbb{F} \to \mathbb{F} \quad \text{defined by} \quad (f, \sigma, y) \mapsto f(\sigma[x \mapsto y])$$

is continuous, being the composition of continuous maps. Since the category DCPO is cartesian closed [1],

$$[[\mathbb{F}^V \to \mathbb{F}] \times \mathbb{F}^V \times \mathbb{F} \to \mathbb{F}] \simeq [[\mathbb{F}^V \to \mathbb{F}] \times \mathbb{F}^V \to [\mathbb{F} \to \mathbb{F}]],$$

and we deduce that $\varphi_{f,\sigma}$ is continuous for all (f, σ) (whence φ is well-defined) and that φ is continuous.

Now, $(\mathcal{P}(\mathcal{R}), \supseteq)$ is isomorphic to a subdomain of the domain $(\mathbb{F}, \sqsubseteq)$ under the continuous embedding defined by $R \mapsto (1, R)$ (recall that we always consider $\mathcal{P}(\mathcal{R})$ with the reverse containment, precisely so that $\mathcal{P}(\mathcal{R})$ can be seen as a sub-domain of $\mathbb{F}$).

The mapping $\mathrm{res} : (\mathbb{F}, \sqsubseteq) \to (\mathcal{P}(\mathcal{R}), \supseteq)$ is continuous (Proposition 2.2). Therefore, we obtain that the mapping

$$\begin{array}{rcl} [\mathbb{F}^V \to \mathbb{F}] \times \mathbb{F}^V \times \mathcal{P}(\mathcal{R}) & \to & \mathcal{P}(\mathcal{R}) \\ (f, \sigma, R) & \mapsto & \mathrm{res}(f(\sigma[x \mapsto (1, R)])) \end{array}$$

also is continuous. Again,

$$[[\mathbb{F}^V \to \mathbb{F}] \times \mathbb{F}^V \times \mathcal{P}(\mathcal{R}) \to \mathcal{P}(\mathcal{R})] \simeq [[\mathbb{F}^V \to \mathbb{F}] \times \mathbb{F}^V \to [\mathcal{P}(\mathcal{R}) \to \mathcal{P}(\mathcal{R})]],$$

so we deduce that ψ is well-defined and continuous. □

In order to have a compositional semantics for the recursion, we need to define a continuous map $\mathrm{rec}\, x : [\mathbb{F}^V \to \mathbb{F}] \to [\mathbb{F}^V \to \mathbb{F}]$, and then we will set $[\![\mathrm{rec}\, x.p]\!] = \mathrm{rec}\, x.[\![p]\!]$. So we start with a continuous map $f \in [\mathbb{F}^V \to \mathbb{F}]$ and we explain the construction of the map $\mathrm{rec}\, x.f \in [\mathbb{F}^V \to \mathbb{F}]$. For $\sigma \in \mathbb{F}^V$, we define $(\mathrm{rec}\, x.f)(\sigma)$ as a fixed point of the continuous map $\varphi_{f,\sigma}$. As explained above, we do not use the least fixed point of $\varphi_{f,\sigma}$. Instead, we start the iteration yielding the fixed point from a resource trace $\bot_{f,\sigma}$ which depends on f and σ.

We define the mapping

$$R\colon [\mathbb{F}^V \to \mathbb{F}] \times \mathbb{F}^V \to \mathcal{P}(\mathcal{R}) \quad \text{by} \quad (f,\sigma) \mapsto R_{f,\sigma} = \nu S.\psi_{f,\sigma}(S)$$

which assigns to each pair (f,σ) the *greatest fixed point* of the monotone map $\psi_{f,\sigma}$. The starting point for the iteration is simply the resource trace $\bot_{f,\sigma} = (1, R_{f,\sigma})$. Therefore, we also have a mapping

$$\bot\colon [\mathbb{F}^V \to \mathbb{F}] \times \mathbb{F}^V \to \mathbb{F} \quad \text{defined by} \quad (f,\sigma) \mapsto \bot_{f,\sigma} = (1, R_{f,\sigma}).$$

Lemma 6.13 *The maps* $R : [\mathbb{F}^V \to \mathbb{F}] \times \mathbb{F}^V \to \mathcal{P}(\mathcal{R})$ *and* $\bot : [\mathbb{F}^V \to \mathbb{F}] \times \mathbb{F}^V \to \mathbb{F}$ *are continuous.*

Proof. Since $\mathcal{R}$ is finite, the greatest fixed point of $\psi_{f,\sigma}$ is obtained by starting from the greatest element $\emptyset$, and iterating at most $N = |\mathcal{P}(\mathcal{R})|$ times, so $R_{f,\sigma} = \psi^N_{f,\sigma}(\emptyset)$. Since ψ is continuous and application of a function is continuous in both the function and its argument, it follows that the mapping R is continuous as well. □

We now are ready to define the interpretation of recursion on $[\mathbb{F}^V \to \mathbb{F}]$:

$$\mathrm{rec}\, x\colon [\mathbb{F}^V \to \mathbb{F}] \to [\mathbb{F}^V \to \mathbb{F}] \quad \text{defined by} \quad (\mathrm{rec}\, x.f)(\sigma) = \bigsqcup_{n\geq 0} \varphi^n_{f,\sigma}(\bot_{f,\sigma}).$$

Proposition 6.14 *The mapping* $\mathrm{rec}\, x\colon [\mathbb{F}^V \to \mathbb{F}] \to [\mathbb{F}^V \to \mathbb{F}]$ *is well defined and continuous. Moreover, for* $f \in [\mathbb{F}^V \to \mathbb{F}]$ *and* $\sigma \in \mathbb{F}^V$, $(\mathrm{rec}\, x.f)(\sigma)$ *is the least fixed point of* $\varphi_{f,\sigma}$ above $\bot_{f,\sigma}$.

Proof. First, we have $R_{f,\sigma} = \psi_{f,\sigma}(R_{f,\sigma}) = \mathrm{res}(\varphi_{f,\sigma}(\bot_{f,\sigma}))$. Therefore $\bot_{f,\sigma} \sqsubseteq \varphi_{f,\sigma}(\bot_{f,\sigma})$ and the sequence $(\varphi^n_{f,\sigma}(\bot_{f,\sigma}))_{n\geq 0}$ is increasing. It follows that the least upper bound $\bigsqcup_{n\geq 0} \varphi^n_{f,\sigma}(\bot_{f,\sigma})$ exists. Since $\varphi_{f,\sigma}$ is continuous, this least upper bound is a least fixed point of $\varphi_{f,\sigma}$. Moreover, it is the least one *above* $\bot_{f,\sigma}$.

We now show simultaneously that both $\mathrm{rec}\, x.f$ and $\mathrm{rec}\, x$ are continuous. Let $\overline{\mathbb{N}} = \mathbb{N} \cup \{\infty\}$. We claim that the following mapping is continuous:

$$\begin{aligned} \Phi : \overline{\mathbb{N}} \times [\mathbb{F}^V \to \mathbb{F}] \times \mathbb{F}^V \quad &\to \quad \mathbb{F} \\ (n, f, \sigma) \quad &\mapsto \quad \begin{cases} \varphi^n_{f,\sigma}(\bot_{f,\sigma}) & \text{if } n \neq \infty \\ \bigsqcup_{m\geq 0} \Phi(m, f, \sigma) & \text{otherwise.} \end{cases} \end{aligned}$$

Φ is continuous with respect to its first argument $\overline{\mathbb{N}}$ by its very definition. We show by induction on n that it is continuous as well with respect to the other two arguments. Let $X \subseteq [\mathbb{F}^V \to \mathbb{F}]$ and $Y \subseteq \mathbb{F}^V$ be directed. For $n = 0$, it is just the continuity of the mapping $\perp$ (Lemma 6.13):

$$\Phi(0, \sqcup X, \sqcup Y) = \perp_{\sqcup X, \sqcup Y} = \bigsqcup_{f \in X, \sigma \in Y} \perp_{f,\sigma} = \bigsqcup_{f \in X, \sigma \in Y} \Phi(0, f, \sigma).$$

Assume now that $\Phi(n, -, -)$ is continuous for some $n \geq 0$ and we will show that $\Phi(n + 1, -, -)$ is continuous as well.

$$\begin{aligned}
\Phi(n+1, \sqcup X, \sqcup Y) &= \varphi_{\sqcup X, \sqcup Y}(\Phi(n, \sqcup X, \sqcup Y)) \\
&= \bigsqcup_{f \in X, \sigma \in Y} \varphi_{f,\sigma}(\Phi(n, \sqcup X, \sqcup Y)) && \text{by continuity of } \varphi \\
&= \bigsqcup_{f \in X, \sigma \in Y} \varphi_{f,\sigma}\left(\bigsqcup_{f' \in X, \sigma' \in Y} \Phi(n, f', \sigma') \right) && \text{by induction} \\
&= \bigsqcup_{f \in X, \sigma \in Y} \bigsqcup_{f' \in X, \sigma' \in Y} \varphi_{f,\sigma}(\Phi(n, f', \sigma')) && \text{by continuity of } \varphi_{f,\sigma} \\
&= \bigsqcup_{f \in X, \sigma \in Y} \varphi_{f,\sigma}(\Phi(n, f, \sigma)) && \text{by diagonalization} \\
&= \bigsqcup_{f \in X, \sigma \in Y} \Phi(n+1, f, \sigma)
\end{aligned}$$

Therefore, we know that $\Phi(n, -, -)$ is continuous for all $n \geq 0$. From this, we will deduce that $\Phi(\infty, -, -)$ is continuous as well:

$$\begin{aligned}
\Phi(\infty, \sqcup X, \sqcup Y) &= \bigsqcup_{n \geq 0} \Phi(n, \sqcup X, \sqcup Y) \\
&= \bigsqcup_{n \geq 0} \bigsqcup_{f \in X, \sigma \in Y} \Phi(n, f, \sigma) \\
&= \bigsqcup_{f \in X, \sigma \in Y} \bigsqcup_{n \geq 0} \Phi(n, f, \sigma) \\
&= \bigsqcup_{f \in X, \sigma \in Y} \Phi(\infty, f, \sigma)
\end{aligned}$$

Note that by definition of Φ we have $\Phi(\infty, f, \sigma) = \operatorname{rec} x.f(\sigma)$. We have just shown that the mapping

$$\begin{aligned}
\Phi(\infty, -, -) : [\mathbb{F}^V \to \mathbb{F}] \times \mathbb{F}^V &\to \mathbb{F} \\
(f, \sigma) &\mapsto \Phi(\infty, f, \sigma) = \operatorname{rec} x.f(\sigma)
\end{aligned}$$

is continuous. We deduce that the mapping

$$\operatorname{rec} x.f : \mathbb{F}^V \to \mathbb{F} \quad \text{defined by} \quad \sigma \mapsto \operatorname{rec} x.f(\sigma)$$

is continuous, and since

$$[[\mathbb{F}^V \to \mathbb{F}] \times \mathbb{F}^V \to \mathbb{F}] \simeq [[\mathbb{F}^V \to \mathbb{F}] \to [\mathbb{F}^V \to \mathbb{F}]],$$

we deduce that the mapping

$$\operatorname{rec} x \colon [\mathbb{F}^V \to \mathbb{F}] \to [\mathbb{F}^V \to \mathbb{F}] \quad \text{defined by} \quad f \mapsto \operatorname{rec} x.f$$

is continuous as well. □

6.5 Link with the Resource Mapping

We relate the semantic resources of a process to the syntactic resource of the process defined in Section 4. The semantic resource set of the process $p \in \mathcal{L}$ in some environment $\sigma \in \mathbb{F}^V$ is given by $\operatorname{res}(\llbracket p \rrbracket(\sigma))$. In order to relate this semantic resource set to the syntactic resource set defined in Section 4, we introduce the map

$$\operatorname{res}^V : \mathbb{F}^V \to \mathcal{P}(\mathcal{R})^V \quad \text{defined by} \quad \operatorname{res}^V(\sigma)(x) = \operatorname{res}(\sigma(x)).$$

Proposition 6.15 *Let $p \in \mathcal{L}$ be a process. Then,*

$$\forall \sigma \in \mathbb{F}^V, \qquad \operatorname{res}(\llbracket p \rrbracket(\sigma)) = \operatorname{res}(p, \operatorname{res}^V(\sigma)). \tag{1}$$

Proof. As usual, the proof is by induction on p. The result is trivial from the definitions for the basic processes STOP, $a \in \Sigma$ and $x \in V$. It follows directly from the definition of the syntactic resources (Section 4) and from Propositions 6.2 and 6.5 and Corollary 6.10 for weak sequential composition, restriction and parallel composition. The only non-trivial case at this point is recursion.

Let $p \in \mathcal{L}$ be a process which satisfies Eq. (1), we show that $\operatorname{rec} x.p$ satisfies Eq. (1) as well. Let $R = \operatorname{res}(\llbracket p \rrbracket(\sigma[x \mapsto (1, \emptyset)]))$. Using Eq. (1) we deduce that

$$R = \operatorname{res}(p, \operatorname{res}^V(\sigma)[x \mapsto \emptyset]) = \operatorname{res}(\operatorname{rec} x.p, \operatorname{res}^V(\sigma)).$$

From Proposition 4.3, we know that $R = \nu S.\operatorname{res}(p, \operatorname{res}^V(\sigma)[x \mapsto S])$. It follows using again Eq. (1) that

$$\operatorname{res}(\llbracket p \rrbracket(\sigma[x \mapsto (1, R)])) = \operatorname{res}(p, \operatorname{res}^V(\sigma)[x \mapsto R]) = R,$$

that is, $R = \nu S.\operatorname{res}(\llbracket p \rrbracket(\sigma[x \mapsto (1, S)]))$. Therefore, the semantics of $\operatorname{rec} x.p$ is given by $\llbracket \operatorname{rec} x.p \rrbracket(\sigma) = \sqcup_{n \geq 0} x_n$ where $x_0 = (1, R)$ and $x_{n+1} = \llbracket p \rrbracket(\sigma[x \mapsto x_n])$. We show by induction on n that $\operatorname{res}(x_n) = R$ for all $n \geq 0$. The result is clear for $n = 0$. Now, assume that $\operatorname{res}(x_n) = R$ for some $n \geq 0$. Using Eq. (1) for the second equality, we deduce

$$\begin{aligned} \operatorname{res}(x_{n+1}) &= \operatorname{res}(\llbracket p \rrbracket(\sigma[x \mapsto x_n])) = \operatorname{res}(p, \operatorname{res}^V(\sigma[x \mapsto \operatorname{res}(x_n)])) \\ &= \operatorname{res}(p, \operatorname{res}^V(\sigma)[x \mapsto R]) = R. \end{aligned}$$

Now, it follows that

$$\operatorname{res}(\llbracket \operatorname{rec} x.p \rrbracket(\sigma)) = \operatorname{res}\left(\bigsqcup_{n \geq 0} x_n\right) = \bigcap_{n \geq 0} \operatorname{res}(x_n) = R = \operatorname{res}(\operatorname{rec} x.p, \operatorname{res}^V(\sigma)),$$

which concludes the proof. □

During the proof of the above proposition, we have also shown that the greatest fixed point of $\psi_{f,\sigma}$ which is used for the starting point of the denotational semantics of recursion is actually obtained after the first iteration for maps $f = [\![p]\!]$ which are semantics of processes. Hence we have

Corollary 6.16 *Let $p \in \mathcal{L}$ be a process and $x \in V$ be a variable. The semantics of* $\operatorname{rec} x.p$ *is given for all environment $\sigma \in \mathbb{F}^V$ by*

$$[\![\operatorname{rec} x.p]\!](\sigma) = \bigsqcup_{n \geq 0} x_n$$

where $x_0 = (1, R)$ with $R = \operatorname{res}(\operatorname{rec} x.p, \operatorname{res}^V(\sigma)) = \operatorname{res}([\![p]\!](\sigma[x \mapsto (1, \emptyset)]))$, and $x_{n+1} = [\![p]\!](\sigma[x \mapsto x_n])$.

We remark that the greatest fixed point of the mapping $\psi_{f,\sigma}$ is not necessarily attained on the first iteration for arbitrary mappings $f \in [\mathbb{F}^V \to \mathbb{F}]$. It is easy to show that, if for all $\sigma \in \mathbb{F}^V$ we have

$$\operatorname{res}(f(\sigma)) \subseteq \operatorname{res}(f(\sigma[x \mapsto (1, \emptyset)])) \cup \operatorname{res}(\sigma(x)),$$

then the fixed point is attained on the first iteration: $R_{f,\sigma} = \psi_{f,\sigma}(\emptyset)$.

6.6 Summary

We conclude the discussion of the denotational semantics of our language by giving a summary of the semantics for the processes in $\mathcal{L}$. We have defined our denotational semantics as a compositional mapping $[\![-]\!] \colon \mathcal{L} \to [\mathbb{F}^V \to \mathbb{F}]$; the work in this section has validated that such a mapping exists, since $\mathcal{L}$ is the initial Ω-algebra, and we have given a continuous interpretation in $[\mathbb{F}^V \to \mathbb{F}]$ for each of the operators $\omega \in \Omega$ in the signature of our language. To summarize, the semantics of a process $p \in \mathcal{L}$ is the continuous map $[\![p]\!]$ defined inductively by:

$$\begin{aligned}
[\![\mathrm{STOP}]\!](\sigma) &= (1, \mathcal{R}) \\
[\![a]\!](\sigma) &= (a, \emptyset) \\
[\![x]\!](\sigma) &= \sigma(x) \\
[\![p \circ q]\!](\sigma) &= [\![p]\!](\sigma) \cdot [\![q]\!](\sigma) \\
[\![p \underset{C}{\|} q]\!](\sigma) &= [\![p]\!](\sigma) \underset{C}{\|} [\![q]\!](\sigma) \\
[\![p|_R]\!](\sigma) &= ([\![p]\!](\sigma))|_R \\
[\![\operatorname{rec} x.p]\!](\sigma) &= (\operatorname{rec} x.[\![p]\!])(\sigma) = \bigsqcup_{n \geq 0} x_n
\end{aligned}$$

where $x_0 = (1, R)$ with $R = \operatorname{res}(\operatorname{rec} x.p, \operatorname{res}^V(\sigma)) = \operatorname{res}([\![p]\!](\sigma[x \mapsto (1, \emptyset)]))$, and $x_{n+1} = [\![p]\!](\sigma[x \mapsto x_n])$.

7 Relating Semantics: The Congruence Theorem

In this section we complete the picture by showing that the operational behavior of a process defined in Section 5 is essentially the same as the Ω-algebra map we defined for the denotational model in the last section. In the next subsection we prove some rather technical lemmas. In a first reading one may skip these lemmas and go directly to the next subsection.

7.1 Auxiliary Results

The first lemma states that the denotational semantics of a process p only depends on the free variables of p.

Lemma 7.1 *Let $p \in \mathcal{L}$ be a process and $\sigma, \sigma' \in \mathbb{F}^V$ be environments satisfying $\sigma(x) = \sigma'(x)$ for all $x \in \mathrm{F}(p)$. Then $[\![p]\!](\sigma) = [\![p]\!](\sigma')$.*

Proof. We proceed by induction on p. The result is trivial for the basis cases STOP, $a \in \Sigma$, and $y \in V$. It follows directly by induction for restriction, weak sequential composition and parallel composition. Now we assume that the result hold for some process $p \in \mathcal{L}$ and we prove it for $\mathrm{rec}\, x.p$. Using the notations of Corollary 6.16, we have $[\![\mathrm{rec}\, x.p]\!](\sigma) = \sqcup_{n\geq 0} x_n$ and $[\![\mathrm{rec}\, x.p]\!](\sigma') = \sqcup_{n\geq 0} x'_n$. From Lemma 4.1 we obtain

$$\mathrm{res}(\mathrm{rec}\, x.p, \mathrm{res}^V(\sigma)) = \mathrm{res}(\mathrm{rec}\, x.p, \mathrm{res}^V(\sigma'))$$

and we deduce that $x_0 = x'_0$. Now, assume that $x_n = x'_n$ for some $n \geq 0$. Using the induction hypothesis on p we derive immediately

$$x_{n+1} = [\![p]\!](\sigma[x \mapsto x_n]) = [\![p]\!](\sigma'[x \mapsto x'_n]) = x'_{n+1},$$

which proves the lemma. □

Next, we explain the effect on the denotational semantics of substituting a process for a variable. This is in the same spirit as Lemmas 4.4 and 5.7.

Lemma 7.2 *Let $p, q \in \mathcal{L}$ and $\sigma \in \mathbb{F}^V$. Then*

$$[\![p[q/x]]\!](\sigma) = [\![p]\!](\sigma[x \mapsto [\![q]\!](\sigma)]).$$

Proof. We proceed by induction on p. Let $\sigma' = \sigma[x \mapsto [\![q]\!](\sigma)])$. If $x \notin \mathrm{F}(p)$, then the result follows from Lemma 7.1, and the result also is clear if $p = x$.

If $p = p_1|_R$, then $p[q/x] = p_1[q/x]|_R$, so

$$[\![p[q/x]]\!](\sigma) = ([\![p_1[q/x]]\!](\sigma))|_R = ([\![p_1]\!](\sigma'))|_R = [\![p]\!](\sigma'),$$

where the second equality follows from the inductive hypothesis.

Similarly, if $p = p_1 \circ p_2$ or $p = p_1 \underset{C}{\|} p_2$, then the result again follows directly by induction.

If $p = \mathrm{rec}\, y.p_1$ with $y \neq x$, then we can assume $y \notin \mathrm{F}(q)$ so that we have $(\mathrm{rec}\, y.p_1)[q/x] = \mathrm{rec}\, y.(p_1[q/x])$. By Corollary 6.16 we have $[\![p[q/x]]\!](\sigma) = [\![\mathrm{rec}\, y.p_1[q/x]]\!](\sigma) = \sqcup_{n\geq 0} y_n$ where the sequence (y_n) is defined by $y_0 = (1, R)$ with $R = \mathrm{res}(\mathrm{rec}\, y.p_1[q/x], \mathrm{res}^V(\sigma))$, and $y_{n+1} = [\![p_1[q/x]]\!](\sigma[y \mapsto y_n])$. Similarly, we have $[\![p]\!](\sigma') = [\![\mathrm{rec}\, y.p_1]\!](\sigma') = \sqcup_{n\geq 0} y'_n$ where $y'_0 = (1, R')$ with $R' = \mathrm{res}(\mathrm{rec}\, y.p_1, \mathrm{res}^V(\sigma'))$, and $y'_{n+1} = [\![p_1]\!](\sigma'[y \mapsto y'_n])$. Now,

$$\begin{aligned} R &= \mathrm{res}((\mathrm{rec}\, y.p_1)[q/x], \mathrm{res}^V(\sigma)) \\ &= \mathrm{res}(\mathrm{rec}\, y.p_1, \mathrm{res}^V(\sigma)[x \mapsto \mathrm{res}(q, \mathrm{res}^V(\sigma))]) && \text{by Lemma 4.4} \\ &= \mathrm{res}(\mathrm{rec}\, y.p_1, \mathrm{res}^V(\sigma')) && \text{by Proposition 6.15} \\ &= R'. \end{aligned}$$

Therefore $y_0 = (1, R) = (1, R') = y'_0$. Now, assume that $y_n = y'_n$ for some $n \geq 0$. Then,

$$\begin{aligned} y_{n+1} &= [\![p_1[q/x]]\!](\sigma[y \mapsto y_n]) \\ &= [\![p_1[q/x]]\!](\sigma[y \mapsto y'_n]) && \text{by induction on } n \\ &= [\![p_1]\!](\sigma'[y \mapsto y'_n]) && \text{by induction on } p \\ &= y_{n+1}. \end{aligned}$$

Therefore $[\![\mathrm{rec}\, y.p_1[q/x]]\!](\sigma) = \sqcup_{n \geq 0} y_n = \sqcup_{n \geq 0} y'_n = [\![\mathrm{rec}\, y.p_1]\!](\sigma')$. □

The following corollary will not be used in the paper but it shows that our definition for the semantics of recursion is legitimate.

Corollary 7.3 *Let $p \in \mathcal{L}$, $p_0 = x \in V$, and $p_{n+1} = p[p_n/x]$. Let $\sigma \in \mathbb{F}^V$ with $\sigma(x) = (1, \mathrm{res}(\mathrm{rec}\, x.p, \mathrm{res}^V(\sigma))) = x_0$, and let $x_{n+1} = [\![p]\!](\sigma[x \mapsto x_n])$. Then $x_n = [\![p_n]\!](\sigma)$ for all $n \geq 0$.*

Proof. We prove this by induction on n. For $n = 0$ the result is clear. For $n \geq 0$, we have

$$\begin{aligned} [\![p_{n+1}]\!](\sigma) &= [\![p[p_n/x]]\!](\sigma) = [\![p]\!](\sigma[x \mapsto [\![p_n]\!](\sigma)]) \\ &= [\![p]\!](\sigma[x \mapsto x_n]) = x_{n+1}. \end{aligned}$$ □

Next, we relate the Ω-algebra structure to the prefix ordering on $\mathbb{F}$. Recall that the prefix ordering is defined by $x \leq z$ if $z = xy$ for some y. Real traces are embedded into resource traces by the canonical mapping $s \mapsto (s, \mathrm{resinf}(s))$. In particular, a finite real trace $s \in \mathbb{M}$ is mapped to $(s, \emptyset)$. To simplify the notation, we simply write s for $(s, \mathrm{resinf}(s))$ when we consider a real trace as a resource trace. Note that if $r \in \mathbb{R}$ and $x = (s, S) \in \mathbb{F}$, then $r \leq x$ iff $r \leq s$.

Lemma 7.4 *Let $x_1, x_2 \in \mathbb{F}$, $R \subseteq \mathcal{R}$ and $a \in \Sigma$. Then*

(i) *$a \leq x_1 x_2$ iff $a \leq x_1$ or ($\mathrm{res}(a) \cap \mathrm{res}(x_1) = \emptyset$ and $a \leq x_2$).*
Moreover, $(ax_1)x_2 = a(x_1 x_2)$,
and if $\mathrm{res}(a) \cap \mathrm{res}(x_1) = \emptyset$, then $x_1(ax_2) = a(x_1 x_2)$.

(ii) *$a \leq x_1|_R$ iff $a \leq x_1$ and $\mathrm{res}(a) \subseteq R$.*
If $\mathrm{res}(a) \subseteq R$, then $(ax_1)|_R = a(x_1|_R)$.

(iii) $a \leq (x_1 \underset{C}{\|} x_2)$ iff $a \leq x_1 \ \wedge\ \mathrm{res}(a) \cap (C \cup \mathrm{res}(x_2)) = \emptyset$,
or $a \leq x_2 \ \wedge\ \mathrm{res}(a) \cap (C \cup \mathrm{res}(x_1)) = \emptyset$,
or $a = a_1 \| a_2$ with $(a_1, a_2) \in \mathrm{Sync}_C(x_1, x_2)$
and $1 \neq a_1 \leq x_1, 1 \neq a_2 \leq x_2$.

If $\mathrm{res}(a) \cap (C \cup \mathrm{res}(x_2)) = \emptyset$, then $(ax_1) \underset{C}{\|} x_2 = a(x_1 \underset{C}{\|} x_2)$.
If $\mathrm{res}(a) \cap (C \cup \mathrm{res}(x_1)) = \emptyset$, then $x_1 \underset{C}{\|} (ax_2) = a(x_1 \underset{C}{\|} x_2)$.
If $(a_1, a_2) \in \mathrm{Sync}_C(a_1 x_1, a_2 x_2)$, then $(a_1 x_1) \underset{C}{\|} (a_2 x_2) = (a_1 \| a_2)(x_1 \underset{C}{\|} x_2)$.

Proof. Parts (i) and (ii) follow easily from the definitions of $\cdot$ and $|_R$ on $\mathbb{F}$. The proof of (iii) is not so easy. In the following, we let $x_1 = (s_1, S_1)$ and $x_2 = (s_1, S_2)$.

Assume that $a \leq (x_1 \underset{C}{\|} x_2)$. We have $x_1 \underset{C}{\|} x_2 = \Pi(x)$ with $x = \sqcup X_C(x_1, x_2)$. Hence $a = \Pi(a')$ with $a' = (a_1, a_2) \in \Sigma'_C(x_1, x_2)$ and $a' \leq x$. We deduce that $a' \leq \mathrm{Re}(x) = \sqcup R_C(x_1, x_2)$. Hence $a' \in R_C(x_1, x_2)$ by Proposition 6.7. Now, there are three cases for $a' = (a_1, a_2) \in \Sigma'_C(x_1, x_2)$:

- If $a_2 = 1$ then $\mathrm{res}(a_1) \cap (C \cup \mathrm{res}(x_2)) = \emptyset$. In this case, $a_1 \leq s_1$ and $a = \Pi(a') = (a_1 \| 1) = a_1 \leq x_1$.
- The case $a_1 = 1$ is similar.
- If $a_1 \neq 1 \neq a_2$ then $(a_1, a_2) \in \mathrm{Sync}_C(x_1, x_2)$. In this case, $a_1 \leq s_1$, $a_2 \leq s_2$, whence $a_1 \leq x_1$ and $a_2 \leq x_2$.

The converse implication is rather easy to prove but we do not include the proof since it follows also from the remaining points (though their proofs are harder). These remaining points are needed for the proof of Proposition 7.9.

Assume now that $\mathrm{res}(a) \cap (C \cup \mathrm{res}(x_2)) = \emptyset$. We show that $(ax_1) \underset{C}{\|} x_2 = a(x_1 \underset{C}{\|} x_2)$. First, it is easy to show by a case distinction that $\mathrm{res}(a) \cap \mathrm{res}(x_2) = \emptyset$ implies $\Sigma'_C(x_1, x_2) \subseteq \Sigma'_C(ax_1, x_2)$. Now, let $r \in R_C(x_1, x_2)$, we claim that $a'r \in R_C(ax_1, x_2)$ with $a' = (a, 1)$. Indeed, $\Pi_1(a'r) = a\Pi_1(r) \leq as_1 = \mathrm{Re}(ax_1)$ and $\Pi_2(a'r) = \Pi_2(r) \leq s_2$. Moreover $a' \in \Sigma'_C(ax_1, x_2)$ and it follows from the above remark that $\mathrm{alph}(a'r) \subseteq \Sigma'_C(ax_1, x_2)$. Therefore $a'r \in R_C(ax_1, x_2)$ as claimed above. We deduce that

$$a' \cdot (\sqcup R_C(x_1, x_2)) = \sqcup(a' \cdot R_C(x_1, x_2)) \leq \sqcup R_C(ax_1, x_2).$$

We show now the converse inequality. Clearly, $a' \in R_C(ax_1, x_2)$. Now, let $r'' \in R_C(ax_1, x_2)$, consider $r' = a' \vee r''$ and write $r' = a'r$. We claim that $r \in R_C(x_1, x_2)$. First, $\Pi_1(r') = a\Pi_1(r) \leq as_1$, whence $\Pi_1(r) \leq s_1$. Second $\Pi_2(r') = \Pi_2(r) \leq s_2$. Third we show that $\mathrm{alph}(r) \subseteq \Sigma'_C(x_1, x_2)$. The most interesting case is when $(b_1, b_2) \in \mathrm{alph}(r)$ with $b_1 \neq 1 \neq b_2$. Since $\mathrm{alph}(r) \subseteq \Sigma'_C(ax_1, x_2)$ we deduce that $(b_1, b_2) \in \mathrm{Sync}_C(ax_1, x_2)$. Since $\mathrm{res}(a) \cap \mathrm{res}(b_2) \subseteq \mathrm{res}(a) \cap \mathrm{res}(x_2) = \emptyset$, we deduce that $(b_1, b_2) \in \mathrm{Sync}_C(x_1, x_2)$ which proves the claim.

We obtain $r'' \leq r' = a'r \leq a' \cdot (\sqcup R_C(x_1, x_2))$ and therefore $\sqcup R_C(ax_1, x_2) \leq a' \cdot (\sqcup R_C(x_1, x_2))$. Together with the inequality above we deduce

$$\sqcup R_C(ax_1, x_2) = a' \cdot (\sqcup R_C(x_1, x_2)).$$

We are almost done for this case. Let $x' = (r', R') = \sqcup X_C(ax_1, x_2)$ and let $x = (r, R) = \sqcup X_C(x_1, x_2)$. Using Proposition 6.8 we deduce that

$$r' = \sqcup R_C(ax_1, x_2) = a' \cdot (\sqcup R_C(x_1, x_2)) = a'r$$

and since $\Pi_1(r')^{-1}(as_1) = \Pi_1(r)^{-1}s_1$ and $\Pi_2(r')^{-1}(s_2) = \Pi_2(r)^{-1}s_2$ we also have $R' = R$. Therefore, $x' = a'x$ and

$$(ax_1) \underset{C}{\|} x_2 = \Pi(x') = a\Pi(x) = a(x_1 \underset{C}{\|} x_2).$$

The case $\mathrm{res}(a) \cap (C \cup \mathrm{res}(x_1)) = \emptyset$ is indeed similar. Finally, assume that $(a_1, a_2) \in$

$\mathrm{Sync}_C(a_1x_1, a_2x_2)$. We have to show that

$$(a_1x_1) \underset{C}{\|} (a_2x_2) = (a_1\|a_2)(x_1 \underset{C}{\|} x_2).$$

Again, we first show that $\Sigma'_C(x_1, x_2) \subseteq \Sigma'_C(a_1x_1, a_2x_2)$.

- If $(b_1, 1) \in \Sigma'_C(x_1, x_2)$ then $b_1 \in \mathrm{alph}(s_1) \subseteq \mathrm{alph}(a_1s_1)$ and $\mathrm{res}(b_1) \cap (C \cup \mathrm{res}(x_2)) = \emptyset$. We have $\mathrm{res}(b_1) \cap \mathrm{res}(a_2) \subseteq \mathrm{res}(a_1x_1) \cap \mathrm{res}(a_2) \subseteq C$ since $(a_1, a_2) \in \mathrm{Sync}_C(a_1x_1, a_2x_2)$. It follows that $\mathrm{res}(b_1) \cap \mathrm{res}(a_2) \subseteq \mathrm{res}(b_1) \cap C = \emptyset$, and therefore $\mathrm{res}(b_1) \cap (C \cup \mathrm{res}(a_2x_2)) = \emptyset$ which proves that $(b_1, 1) \in \Sigma'_C(a_1x_1, a_2x_2)$.
- If $(b_1, b_2) \in \mathrm{Sync}_C(x_1, x_2)$ then for $i = 1, 2$ we have $b_i \in \mathrm{alph}(s_i) \subseteq \mathrm{alph}(a_is_i)$ and we obtain as above that $\mathrm{res}(b_1) \cap \mathrm{res}(a_2) \subseteq \mathrm{res}(b_1) \cap C$ and $\mathrm{res}(b_2) \cap \mathrm{res}(a_1) \subseteq \mathrm{res}(b_2) \cap C$. We deduce easily that $(b_1, b_2) \in \mathrm{Sync}_C(a_1x_1, a_2x_2)$.

Second, let $r \in R_C(x_1, x_2)$, we claim that $a'r \in R_C(a_1x_1, a_2x_2)$ with $a' = (a_1, a_2)$. Indeed, $\Pi_i(a'r) = a_i\Pi_i(r) \leq a_is_i =$ for $i = 1, 2$. Moreover $a' \in \Sigma'_C(a_1x_1, a_2x_2)$ and we obtain $\mathrm{alph}(a'r) \subseteq \Sigma'_C(a_1x_1, a_2x_2)$. Therefore $a'r \in R_C(a_1x_1, a_2x_2)$ as claimed above. We deduce that

$$a' \cdot (\sqcup R_C(x_1, x_2)) = \sqcup(a' \cdot R_C(x_1, x_2)) \leq \sqcup R_C(a_1x_1, a_2x_2).$$

For the converse inequality, we first note that $a' \in R_C(a_1x_1, a_2x_2)$. Now, let $r'' \in R_C(a_1x_1, a_2x_2)$, consider $r' = a' \vee r''$ and write $r' = a'r$. We claim that $r \in R_C(x_1, x_2)$. First, for $i = 1, 2$ we have $\Pi_i(r') = a_i\Pi_i(r) \leq a_is_i$, whence $\Pi_i(r) \leq s_i$. Second we show that $\mathrm{alph}(r) \subseteq \Sigma'_C(x_1, x_2)$.

- If $(b_1, 1) \in \mathrm{alph}(r)$ then $b_1 \in \mathrm{alph}(\Pi_1(r)) = \mathrm{alph}(s_1)$ and since $\mathrm{alph}(r) \subseteq \Sigma'_C(a_1x_1, a_2x_2)$ we also have $\mathrm{res}(b_1) \cap (C \cup \mathrm{res}(x_2)) = \emptyset$ and we deduce $(b_1, 1) \in \Sigma'_C(x_1, x_2)$.
- If $(b_1, b_2) \in \mathrm{alph}(r)$ with $b_1 \neq 1 \neq b_2$ then $b_i \in \mathrm{alph}(\Pi_i(r)) = \mathrm{alph}(s_i)$. Since $\mathrm{alph}(r) \subseteq \Sigma'_C(a_1x_1, a_2x_2)$ we also have $(b_1, b_2) \in \mathrm{Sync}_C(a_1x_1, a_2x_2)$. Since $(a_1, a_2) \in \mathrm{Sync}_C(a_1x_1, a_2x_2)$, we have

  $$\mathrm{res}(b_1) \cap \mathrm{res}(a_2) \subseteq \mathrm{res}(a_1x_1) \cap \mathrm{res}(a_2) \subseteq C.$$

  Hence,

  $$\mathrm{res}(b_1) \cap \mathrm{res}(a_2) \subseteq \mathrm{res}(b_1) \cap C = \mathrm{res}(b_2) \cap C \subseteq \mathrm{res}(x_2),$$

  and we deduce that $\mathrm{res}(b_1) \cap \mathrm{res}(a_2x_2) = \mathrm{res}(b_1) \cap \mathrm{res}(x_2)$. Therefore $(b_1, b_2) \in \mathrm{Sync}_C(x_1, x_2)$.

We obtain $r'' \leq r' = a'r \leq a' \cdot (\sqcup R_C(x_1, x_2))$. Hence, $\sqcup R_C(a_1x_1, a_2x_2) \leq a' \cdot (\sqcup R_C(x_1, x_2))$. Together with the inequality above we deduce

$$\sqcup R_C(a_1x_1, a_2x_2) = a' \cdot (\sqcup R_C(x_1, x_2)).$$

Let $x' = (r', R') = \sqcup X_C(a_1x_1, a_2x_2)$ and let $x = (r, R) = \sqcup X_C(x_1, x_2)$. Using Proposition 6.8 we deduce that

$$r' = \sqcup R_C(a_1x_1, a_2x_2) = a' \cdot (\sqcup R_C(x_1, x_2)) = a'r$$

and since $\Pi_i(r')^{-1}(a_is_i) = \Pi_i(r)^{-1}s_i$ for $i = 1, 2$ we also have $R' = R$. Therefore, $x' = a'x$ and

$$(a_1x_1) \underset{C}{\|} (a_2x_2) = \Pi(x') = \Pi(a')\Pi(x) = (a_1\|a_2)(x_1 \underset{C}{\|} x_2).$$

$\square$

The last technical lemma that we need to proceed with the congruence theorem states that each minimal action of a recursive process is already a minimal action of the body of the recursion.

Lemma 7.5 *Let* $p \in \mathcal{L}$ *and* $\sigma \in \mathbb{F}^V$ *with* $\sigma(x) = (1, \mathrm{res}(\mathrm{rec}\, x.p, \mathrm{res}^V(\sigma)))$. *Then*

$$a \leq [\![\mathrm{rec}\, x.p]\!](\sigma) \;\Rightarrow\; a \leq [\![p]\!](\sigma).$$

The remainder of this subsection is devoted to the proof of this lemma. We start with a process $p \in \mathcal{L}$ and an environment $\sigma \in \mathbb{F}^V$ with $\sigma(x) = (1, \mathrm{res}(\mathrm{rec}\, x.p, \mathrm{res}^V(\sigma)))$.

We say that a process p is in normal form if no variable occurs both free and bound in p and if each bound variable is bound only once, that is, if $\mathrm{rec}\, y.q$ and $\mathrm{rec}\, y'.q'$ are different subterms of p then $y \neq y'$. Using fresh variables, we may assume that the process p is in normal form.

We say that an environment $\sigma \in \mathbb{F}^V$ is in normal form for a process r if for each subterm $\mathrm{rec}\, y.q$ of r we have $\sigma(y) = (1, \mathrm{res}(\mathrm{rec}\, y.q, \mathrm{res}^V(\sigma)))$. By Lemma 7.1 and since p is in normal form, we may assume that our environment σ is in normal form for $\mathrm{rec}\, x.p$. For convenience, the syntactic environment $\mathrm{res}^V(\sigma)$ is denoted by τ.

We view a term $p \in \mathcal{L}$ as a syntax tree and we consider subterms of p as nodes in the syntax tree of p. In the following, q denotes a node in p, that is, a subterm of p. If $q \neq p$ then there is a unique father r of q in p.

We define inductively the set S_q of *available resources at* q *in* p by $S_p = \mathcal{R}$ and if r is the father of $q \neq p$ then

- $r = q|_R \;\Rightarrow\; S_q = S_r \cap R$,
- $r = q' \circ q \;\Rightarrow\; S_q = S_r \setminus \mathrm{res}(q', \tau)$,
- and $S_q = S_r$ in all other cases.

Lemma 7.6 *Let* $q \neq p$ *be a node in the syntax tree of* p *and let* r *be the father of* q *in* p. *Then,* $\mathrm{res}(q, \tau) \cap S_q \subseteq \mathrm{res}(r, \tau) \cap S_r$.

Proof.

$r = q|_R :$ Then, $\mathrm{res}(r, \tau) \cap S_r = \mathrm{res}(q, \tau) \cap R \cap S_r = \mathrm{res}(q, \tau) \cap S_q$.

$r = q' \circ q :$ Then, $\mathrm{res}(q, \tau) \cap S_q \subseteq \mathrm{res}(q, \tau) \cap S_r \subseteq \mathrm{res}(r, \tau) \cap S_r$.

$r = q \circ q' :$ Then, $\mathrm{res}(q, \tau) \cap S_q = \mathrm{res}(q, \tau) \cap S_r \subseteq \mathrm{res}(r, \tau) \cap S_r$.

$r = q \underset{C}{\|} q'$ and $r = q' \underset{C}{\|} q$ are dealt with similarly.

$r = \mathrm{rec}\, y.q :$ Then, $\mathrm{res}(r, \tau) = \mathrm{res}(q, \tau[y \mapsto \mathrm{res}(r, \tau)])$ by Proposition 4.3. Since σ is in normal form for $\mathrm{rec}\, x.p$ we have $\tau(y) = \mathrm{res}(\sigma(y)) = \mathrm{res}(r, \tau)$ and we deduce that $\mathrm{res}(r, \tau) = \mathrm{res}(q, \tau)$. The result follows since $S_q = S_r$.

$\square$

By Corollary 6.16, the denotational semantics of $\mathrm{rec}\, x.p$ is given by

$$[\![\mathrm{rec}\, x.p]\!](\sigma) = \bigsqcup_{n \geq 0} x_n$$

where $x_0 = (1, \mathrm{res}(\mathrm{rec}\, x.p, \tau)) = \sigma(x)$ and $x_{n+1} = [\![p]\!](\sigma[x \mapsto x_n])$. Recall that we have seen in the proof of Proposition 6.15 that $\mathrm{res}(x_n) = \mathrm{res}(\mathrm{rec}\, x.p, \tau) = \tau(x)$ for all $n \geq 0$.

When q is a node in p we denote by $\mathcal{P}(p, q, n, \sigma)$ the condition that,

$$a \leq [\![q]\!](\sigma[x \mapsto x_n]) \ \wedge \ \mathrm{res}(a) \subseteq S_q \quad \Rightarrow \quad a \leq x_n \ \vee \ a \leq [\![q]\!](\sigma)$$

for all $a \in \Sigma$.

Lemma 7.7 *$\mathcal{P}(p, q, n, \sigma)$ holds for all $p \in \mathcal{L}$ in normal form, all $\sigma \in \mathbb{F}^V$ in normal form for $\mathrm{rec}\, x.p$, all q subterm of p and all $n \geq 0$.*

Proof. We proceed by induction on p, q and n. Let $\sigma' = \sigma[x \mapsto x_n]$. We have seen that $\mathrm{res}(x_n) = \mathrm{res}(\sigma(x))$, hence $\mathrm{res}^V(\sigma') = \mathrm{res}^V(\sigma) = \tau$. Therefore, Proposition 6.15 implies that

$$\forall r \in \mathcal{L}, \quad \mathrm{res}([\![r]\!](\sigma)) = \mathrm{res}([\![r]\!](\sigma')). \tag{2}$$

In the following, we assume that $a \leq [\![q]\!](\sigma')$ and $\mathrm{res}(a) \subseteq S_q$.

We begin with some easy special cases which constitute the basis of the induction. First, if x is not free in q then by Lemma 7.1 we deduce that $[\![q]\!](\sigma') = [\![q]\!](\sigma)$ and therefore $\mathcal{P}(p, q, n, \sigma)$ holds. Second, if $n = 0$ then $\sigma' = \sigma$ and we get the same conclusion. The result is also trivial if $q = x$ since $[\![x]\!](\sigma') = x_n$. Note that $p = x$ implies $q = x$ and since p is in normal form, $x \notin \mathrm{F}(p)$ implies $x \notin \mathrm{F}(q)$.

$q = q_1|_R$: Then $a \leq [\![q_1]\!](\sigma')$ and $\mathrm{res}(a) \subseteq S_q \cap R = S_{q_1}$. The inductive hypothesis $\mathcal{P}(p, q_1, n, \sigma)$ implies that $a \leq x_n$ or $a \leq [\![q_1]\!](\sigma)$. Since $\mathrm{res}(a) \subseteq R$, the second case implies $a \leq [\![q]\!](\sigma)$.

$q = q_1 \circ q_2$: Then either $a \leq [\![q_1]\!](\sigma') \ \wedge \ \mathrm{res}(a) \subseteq S_q = S_{q_1}$, or else $a \leq [\![q_2]\!](\sigma') \ \wedge \ \mathrm{res}(a) \subseteq S_q \setminus \mathrm{res}([\![q_1]\!](\sigma')) = S_{q_2}$. In the first case, $\mathcal{P}(p, q_1, n, \sigma)$ holds by induction, so $a \leq x_n$ or $a \leq [\![q_1]\!](\sigma)$, which implies $a \leq [\![q]\!](\sigma)$. In the second case, $\mathcal{P}(p, q_2, n, \sigma)$ holds by induction, which implies that $a \leq x_n$ or $a \leq [\![q_2]\!](\sigma)$. Since $\mathrm{res}(a) \cap \mathrm{res}([\![q_1]\!](\sigma')) = \emptyset$, we deduce using Eq. (2) that once again $a \leq [\![q]\!](\sigma)$.

$q = \mathrm{rec}\, y.q_1$: Recall that since σ is in normal form for $\mathrm{rec}\, x.p$ we have $\sigma(y) = (1, \mathrm{res}(\mathrm{rec}\, y.q_1, \tau))$. Now, p is in normal form and we have $y \neq x$. Hence $\sigma'(y) = \sigma(y) = (1, \mathrm{res}(\mathrm{rec}\, y.q_1, \tau))$. By Corollary 6.16 we deduce that $[\![q]\!](\sigma) = \sqcup_{m \geq 0} y_m$ with $y_0 = \sigma(y)$ and $y_{m+1} = [\![q_1]\!](\sigma[y \mapsto y_m])$ and also $[\![q]\!](\sigma') = \sqcup_{m \geq 0} y'_m$ with $y'_0 = \sigma'(y)$ and $y'_{m+1} = [\![q_1]\!](\sigma'[y \mapsto y'_m])$. Since $a \leq [\![q]\!](\sigma') = \sqcup_m y'_m$ and $a \not\leq y'_0$, we find some $m \geq 0$ such that $a \not\leq y'_m \ \wedge \ a \leq y'_{m+1} = [\![q_1]\!](\sigma'[y \mapsto y'_m])$. Note that the set of available resources at q_1 in q_1 is $\mathcal{R} \supseteq \mathrm{res}(a)$ and since by induction $\mathcal{P}(q_1, q_1, m, \sigma')$ holds we obtain $a \leq [\![q_1]\!](\sigma')$. Now, still by induction $\mathcal{P}(p, q_1, n, \sigma)$ holds and since $S_q = S_{q_1}$ we deduce that $a \leq x_n$ or $a \leq [\![q_1]\!](\sigma) = y_1 \sqsubseteq \sqcup_n y_n = [\![q]\!](\sigma)$. Thus $\mathcal{P}(q, q_1, n, \sigma)$ holds.

$q = q_1 \underset{C}{\|} q_2$: This is the most difficult case. We consider two subcases.

- $\mathrm{res}(a) \cap C = \emptyset$: Then a is a local event, and without of loss of generality we may assume $a \leq [\![q_1]\!](\sigma')$ and $\mathrm{res}(a) \cap (C \cup \mathrm{res}([\![q_2]\!](\sigma'))) = \emptyset$. From Eq. (2) we get $\mathrm{res}(a) \cap (C \cup \mathrm{res}([\![q_2]\!](\sigma))) = \emptyset$. By induction, $\mathcal{P}(p, q_1, n, \sigma)$ holds and since $S_q = S_{q_1}$ we deduce that $a \leq x_n$ or $a \leq [\![q_1]\!](\sigma)$ which implies $a \leq [\![q]\!](\sigma)$.
- $\mathrm{res}(a) \cap C \neq \emptyset$: Then a is a synchronization event, so $a = a_1 \| a_2$ with $a_i \leq [\![q_i]\!](\sigma')$ and $(a_1, a_2) \in \mathrm{Sync}_C([\![q_1]\!](\sigma'), [\![q_2]\!](\sigma'))$. From Eq. (2) we get $\mathrm{Sync}_C([\![q_1]\!](\sigma'), [\![q_2]\!](\sigma')) =$

$\mathrm{Sync}_C([\![q_1]\!](\sigma), [\![q_2]\!](\sigma))$. We have $\mathrm{res}(a_i) \subseteq \mathrm{res}(a) \subseteq S_q = S_{q_i}$. The inductive hypothesis implies that either $a_i \leq x_n$ or $a_i \leq [\![q_i]\!](\sigma)$ for each $i = 1, 2$.

If $a_i \leq [\![q_i]\!](\sigma)$ for $i = 1, 2$, then Lemma 7.4 implies $a \leq [\![q]\!](\sigma)$.

We assume now that $a_1 \leq x_n$. Since $a_1 \neq 1$, we must have $n > 0$ and $x_n = [\![p]\!](\sigma[x \mapsto x_{n-1}])$. Let $\sigma'' = \sigma[x \mapsto x_{n-1}]$. Note that $\mathrm{res}^V(\sigma'') = \mathrm{res}^V(\sigma') = \mathrm{res}^V(\sigma) = \tau$ since $\mathrm{res}(x_{n_1}) = \mathrm{res}(\sigma(x))$. We need the following

Claim: Let $B = \{b \in \Sigma \mid \mathrm{res}(b) \cap \mathrm{res}(a) \cap C \neq \emptyset\}$. Then for all ancestors r of q in p,

$$(\exists b \in B)\ b \leq [\![r]\!](\sigma'') \quad \Rightarrow \quad (\exists b' \in B)\ b' \leq [\![q]\!](\sigma'').$$

We first show that this claim allows us to conclude the proof. Since $\mathrm{res}(a_1) \subseteq \mathrm{res}(a)$ and $\mathrm{res}(a_1) \cap C \neq \emptyset$, we deduce that $a_1 \in B$. Now, $a_1 \leq x_n = [\![p]\!](\sigma'')$ and p is an ancestor of q hence we can apply the claim and we obtain $b \leq [\![q]\!](\sigma'')$ for some $b \in B$.

Then $\mathrm{res}(b) \cap C \neq \emptyset$ and from Lemma 7.4 we deduce that $b = b_1 \| b_2$ with $b_i \leq [\![q_i]\!](\sigma'')$ and $(b_1, b_2) \in \mathrm{Sync}_C([\![q_1]\!](\sigma''), [\![q_2]\!](\sigma''))$.

Since $x_{n-1} \sqsubseteq x_n$, we have $[\![q_i]\!](\sigma'') \sqsubseteq [\![q_i]\!](\sigma')$ and $b_i \leq [\![q_i]\!](\sigma')$.

From $\mathrm{res}(a_i) \cap C = \mathrm{res}(a) \cap C$ and $\mathrm{res}(b_i) \cap C = \mathrm{res}(b) \cap C$ we deduce that $\mathrm{res}(a_i) \cap \mathrm{res}(b_i) \cap C = \mathrm{res}(a) \cap \mathrm{res}(b) \cap C \neq \emptyset$ and $a_i \, D \, b_i$. Using $a_i \leq [\![q_i]\!](\sigma')$ and $b_i \leq [\![q_i]\!](\sigma')$ we get $a_i = b_i$ and therefore $a = b$.

Finally, $a = b \leq [\![q]\!](\sigma'') = [\![q]\!](\sigma[x \mapsto x_{n-1}])$. By induction $\mathcal{P}(p, q, n-1, \sigma)$ holds, so either $a \leq [\![q]\!](\sigma)$ or $a \leq x_{n-1}$. Since $x_{n-1} \sqsubseteq x_n$, the last case implies $a \leq x_n$.

Proof of the Claim: We proceed by induction on r following the branch from q to p in the syntax tree of p. The result is clear for the basis case $r = q$.

Recall that $\mathrm{res}^V(\sigma'') = \mathrm{res}^V(\sigma') = \mathrm{res}^V(\sigma) = \tau$. We will use several times in the proof that if r_1 is an ancestor of q in p then

$$\begin{aligned} \mathrm{res}(a) &\subseteq \mathrm{res}([\![q]\!](\sigma')) \cap S_q && \text{Hypothesis of } \mathcal{P}(p,q,n,\sigma) \\ &= \mathrm{res}(q, \tau) \cap S_q && \text{Proposition 6.15} \\ &\subseteq \mathrm{res}(r_1, \tau) \cap S_{r_1} && \text{Lemma 7.6} \\ &= \mathrm{res}([\![r_1]\!](\sigma'')) \cap S_{r_1} && \text{Proposition 6.15} \end{aligned}$$

Throughout, we assume that $b \leq [\![r]\!](\sigma'')$ for some $b \in B$ and we consider the possible cases.

$r = r_1|_R$: We have $b \leq [\![r]\!](\sigma'') = [\![r_1]\!](\sigma'')|_R$. Then $b \leq [\![r_1]\!](\sigma'')$ and we apply the induction hypothesis on r_1 to conclude.

$r = \mathrm{rec}\, y.r_1$: Since p is in normal form and σ is in normal form for $\mathrm{rec}\, x.p$ we have $\sigma''(y) = \sigma(y) = (1, \mathrm{res}(r, \tau))$. By Corollary 6.16 we have $[\![r]\!](\sigma'') = \sqcup_{k \geq 0} y_k$ where $y_0 = \sigma(y)$ and $y_{k+1} = [\![r_1]\!](\sigma''[y \mapsto y_k])$. Then, $b \leq [\![r_1]\!](\sigma'')$ and $b \not\leq y_0$ imply $(\exists k)\ b \not\leq y_k \ \wedge \ b \leq y_{k+1}$. Note that the set of available resources at r_1 in r_1 is the whole set of resources $\mathcal{R} \supseteq \mathrm{res}(a)$. Since by induction, $\mathcal{P}(r_1, r_1, k, \sigma'')$ holds we deduce that $b \leq [\![r_1]\!](\sigma'')$. Then, using the induction on r_1, there is some $b' \in b$ with $b' \leq [\![q]\!](\sigma)$.

$r = r_1 \circ r_2$ and r_1 is an ancestor of q: Then, either $b \leq [\![r_1]\!](\sigma'')$ or $b \leq [\![r_2]\!](\sigma'')$ and $\mathrm{res}(b) \cap \mathrm{res}([\![r_1]\!](\sigma'')) = \emptyset$. In the first case, the inductive hypothesis on r_1 implies there is some $b' \in B$ with $b \leq [\![q]\!](\sigma'')$. The second case is impossible. Indeed, $\emptyset \neq \mathrm{res}(b) \cap \mathrm{res}(a) \subseteq \mathrm{res}([\![r_1]\!](\sigma''))$ is in contradiction with $\mathrm{res}(b) \cap \mathrm{res}([\![r_1]\!](\sigma'')) = \emptyset$.

$r = r_1 \circ r_2$ and r_2 is an ancestor of q: If $b \leq [\![r_2]\!](\sigma'')$ then we conclude by induction on r_2. We show that $b \leq [\![r_1]\!](\sigma'')$ is impossible. Indeed, it implies that $\mathrm{res}(b) \subseteq \mathrm{res}([\![r_1]\!](\sigma'')) = \mathrm{res}(r_1, \tau)$ which contradicts

$$\emptyset \neq \mathrm{res}(b) \cap \mathrm{res}(a) \subseteq \mathrm{res}([\![r_2]\!](\sigma'')) \cap S_{r_2} \subseteq \mathcal{R} \setminus \mathrm{res}(r_1, \tau).$$

$r = r_1 \underset{E}{\|} r_2$: Without loss of generality, r_1 is an ancestor of q. There are three cases:

- $\mathrm{res}(b) \cap (E \cup \mathrm{res}([\![r_2]\!](\sigma''))) = \emptyset$: Then $b \leq [\![r_1]\!](\sigma'')$, and the inductive hypothesis on r_1 implies the result.
- $\mathrm{res}(b) \cap (E \cup \mathrm{res}([\![r_1]\!](\sigma''))) = \emptyset$ is impossible since $\emptyset \neq \mathrm{res}(b) \cap \mathrm{res}(a) \subseteq \mathrm{res}([\![r_1]\!](\sigma''))$.
- Finally, assume that $b = b_1 \| b_2$ with $(b_1, b_2) \in \mathrm{Sync}_E([\![r_1]\!](\sigma''), [\![r_2]\!](\sigma''))$ and $b_i \leq [\![r_i]\!](\sigma'')$. If $b_1 \in B$, then the result follows by the inductive hypothesis applied to r_1. If $b_1 \notin B$, then $\mathrm{res}(b) = \mathrm{res}(b_1) \cup (\mathrm{res}(b_2) \setminus E)$ and we obtain a contradiction with
$$\begin{aligned}\emptyset \neq \mathrm{res}(b) \cap \mathrm{res}(a) \cap C &= (\mathrm{res}(b_2) \setminus E) \cap \mathrm{res}(a) \cap C \\ &\subseteq (\mathrm{res}(b_2) \cap \mathrm{res}([\![r_1]\!](\sigma''))) \setminus E = \emptyset,\end{aligned}$$
where the last inequality holds since $(b_1, b_2) \in \mathrm{Sync}_E([\![r_1]\!](\sigma''), [\![r_2]\!](\sigma''))$.

This concludes the proof of the claim and of Lemma 7.7. $\square$

Proof of Lemma 7.5. Let $x_0 = \sigma(x)$ and $x_{n+1} = [\![p]\!](\sigma[x \mapsto x_n])$. Then $[\![\mathrm{rec}\, x.p]\!](\sigma) = \sqcup_n x_n$, and so $a \leq \sqcup_n x_n$. Since $a \not\leq x_0$, it follows that there is some n with $a \not\leq x_n$ and $a \leq x_{n+1} = [\![p]\!](\sigma[x \mapsto x_n])$. Since $S_p = \mathcal{R}$, Lemma 7.7 implies $a \leq [\![p]\!](\sigma)$. $\square$

7.2 The Congruence Theorem

We want first to relate the initial actions of the denotational semantics of a process $p \in \mathcal{L}$ with the initial actions allowed for p by the operational semantics. This is done in the following two crucial propositions. In order to simplify the notation, we view $\mathcal{P}(\mathcal{R})$ as a subset of $\mathbb{F}$ with the canonical embedding $S \mapsto (1, S)$.

Proposition 7.8 *Let $p \in \mathcal{L}$, $a \in \Sigma$ and $\sigma \in \mathcal{P}(\mathcal{R})^V \subseteq \mathbb{F}^V$. Then*

$$a \leq [\![p]\!](\sigma) \ \Rightarrow\ p \xrightarrow[\sigma]{a} .$$

Proof. We proceed by structural induction on p. Suppose $a \leq [\![p]\!](\sigma)$ for $a \in \Sigma, p \in \mathcal{L}$ and $\sigma \in \mathcal{P}(\mathcal{R})^V$. The cases $p = \mathrm{STOP}$, x or $b \neq a$ are clearly impossible, while the result is obvious if $p = a$.

If $p = q|_R$, then $a \leq [\![p]\!](\sigma)$ iff $a \leq [\![q]\!](\sigma)$ and $\mathrm{res}(a) \subseteq R$ (Lemma 7.4). Then the inductive hypothesis implies $q \xrightarrow[\sigma]{a}$, which in turn implies $p = q|_R \xrightarrow[\sigma]{a}$ since $\mathrm{res}(a) \subseteq R$.

If $p = p_1 \circ p_2$, then either $a \leq [\![p_1]\!](\sigma)$ or $a \leq [\![p_2]\!](\sigma)$ and $\mathrm{res}(a) \cap \mathrm{res}([\![p_1]\!](\sigma)) = \emptyset$ (Lemma 7.4). In the first case, we have $p_1 \xrightarrow[\sigma]{a}$ and so $p \xrightarrow[\sigma]{a}$. In the second case, we obtain $p_2 \xrightarrow[\sigma]{a}$ from the inductive hypothesis. Using Proposition 6.15 we deduce that $\mathrm{res}(a) \cap \mathrm{res}(p_1, \sigma) = \emptyset$, from which $p \xrightarrow[\sigma]{a}$ follows.

If $p = \mathrm{rec}\, x.q$, then we can assume that $\sigma(x) = \mathrm{res}(\mathrm{rec}\, x.q, \sigma)$ (Lemma 7.1). Then $a \leq [\![\mathrm{rec}\, x.q]\!](\sigma)$ and Lemma 7.5 imply $a \leq [\![q]\!](\sigma)$, and then induction implies $q \xrightarrow[\sigma]{a}$. Hence

$p \xrightarrow[\sigma]{a}$ by the operational rule for recursion.

If $p = p_1 \parallel_C p_2$, then either $\mathrm{res}(a) \cap (C \cup \mathrm{res}(\llbracket p_2 \rrbracket(\sigma))) = \emptyset$ and $a \le \llbracket p_1 \rrbracket(\sigma)$, or $\mathrm{res}(a) \cap (C \cup \mathrm{res}(\llbracket p_1 \rrbracket(\sigma))) = \emptyset$ and $a \le \llbracket p_2 \rrbracket(\sigma)$, or $a = a_1 \| a_2$ with $a_i \le \llbracket p_i \rrbracket(\sigma)$ and $(a_1, a_2) \in \mathrm{Sync}_C(\llbracket p_1 \rrbracket(\sigma), \llbracket p_2 \rrbracket(\sigma))$ (Lemma 7.4). In the first case, we have $p_1 \xrightarrow[\sigma]{a}$ by induction, and this implies $p \xrightarrow[\sigma]{a}$ since $\mathrm{res}(p_2, \sigma) = \mathrm{res}(\llbracket p_2 \rrbracket(\sigma))$ by Proposition 6.15. A similar argument works for the second case, and the third follows from the fact that $a_i \le \llbracket p_i \rrbracket(\sigma)$ implies $p_i \xrightarrow[\sigma]{a_i}$ by induction, and this in turn implies $p \xrightarrow[\sigma]{a}$ since by Proposition 6.15 we obtain $(a_1, a_2) \in \mathrm{Sync}_{C,\sigma}(p_1, p_2)$. $\square$

Proposition 7.9 *Let $p, q \in \mathcal{L}$, $a \in \Sigma$ and $\sigma \in \mathbb{F}^V$. Then*

$$p \xrightarrow[\tau]{a} q \;\Rightarrow\; \llbracket p \rrbracket(\sigma) = a \cdot \llbracket q \rrbracket(\sigma),$$

where $\tau = \mathrm{res}^V(\sigma)$.

Proof. We again proceed by induction on p, and we begin by noting that $p = \mathrm{STOP}$, $p = x \in V$ or $p = b \ne a$ are impossible. Moreover, the result is obvious for $p = a$.

$p = p_1|_R$: We must have $p_1 \xrightarrow[\tau]{a} q_1$, $\mathrm{res}(a) \subseteq R$ and $q = q_1|_R$. Then, using the inductive hypothesis and Lemma 7.4 we obtain

$$\llbracket p \rrbracket(\sigma) = \llbracket p_1 \rrbracket(\sigma)|_R = (a \cdot \llbracket q_1 \rrbracket(\sigma))|_R = a \cdot (\llbracket q_1 \rrbracket(\sigma)|_R) = a \cdot \llbracket q \rrbracket(\sigma).$$

$p = p_1 \circ p_2$: There are two cases. Either $p_1 \xrightarrow[\tau]{a} q_1$ and $q = q_1 \circ p_2$. Using the inductive hypothesis we deduce

$$\llbracket p_1 \circ p_2 \rrbracket(\sigma) = \llbracket p_1 \rrbracket(\sigma) \cdot \llbracket p_2 \rrbracket(\sigma) = a \cdot \llbracket q_1 \rrbracket(\sigma) \cdot \llbracket q_2 \rrbracket(\sigma) = a \cdot \llbracket q \rrbracket(\sigma).$$

Or $p_2 \xrightarrow[\tau]{a} q_2$, $\mathrm{res}(a) \cap \mathrm{res}(p_1, \tau) = \emptyset$, and $q = p_1 \circ q_2$. By Proposition 6.15 we have $\mathrm{res}(p_1, \tau) = \mathrm{res}(\llbracket p_1 \rrbracket(\sigma))$ and therefore, $\mathrm{res}(a) \cap \mathrm{res}(\llbracket p_1 \rrbracket(\sigma)) = \emptyset$. Using the inductive hypothesis and Lemma 7.4 it follows

$$\llbracket p_1 \circ p_2 \rrbracket(\sigma) = \llbracket p_1 \rrbracket(\sigma) \cdot a \cdot \llbracket q_2 \rrbracket(\sigma) = a \cdot \llbracket p_1 \rrbracket(\sigma) \cdot \llbracket q_2 \rrbracket(\sigma) = a \cdot \llbracket q \rrbracket(\sigma).$$

$p = p_1 \parallel_C p_2$: Again, we consider two cases. If $\mathrm{res}(a) \cap (C \cup \mathrm{res}(p_2, \tau)) = \emptyset$, then $p_1 \xrightarrow[\tau]{a} q_1$, and $q = q_1 \parallel_C p_2$. Using the inductive hypothesis and Lemma 7.4 we obtain

$$\llbracket p \rrbracket(\sigma) = \llbracket p_1 \rrbracket(\sigma) \parallel_C \llbracket p_2 \rrbracket(\sigma) = (a \cdot \llbracket q_1 \rrbracket(\sigma)) \parallel_C \llbracket p_2 \rrbracket(\sigma) = a \cdot \llbracket q \rrbracket(\sigma).$$

If $a = a_1 \| a_2$ with $(a_1, a_2) \in \mathrm{Sync}_{C,\tau}(p_1, p_2)$, then $p_i \xrightarrow[\tau]{a_i} q_i$ and $q = q_1 \parallel_C q_2$. As above, we obtain using the inductive hypothesis and Lemma 7.4

$$\llbracket p \rrbracket(\sigma) = (a_1 \cdot \llbracket q_1 \rrbracket(\sigma)) \parallel_C (a_2 \cdot \llbracket q_2 \rrbracket(\sigma)) = a \cdot \llbracket q \rrbracket(\sigma).$$

$p = \mathrm{rec}\, x.p_1$: Let $\sigma' = \sigma[x \mapsto \llbracket p \rrbracket(\sigma)])$, and note that $\tau' = \mathrm{res}^V(\sigma') = \tau[x \mapsto \mathrm{res}(p, \tau)]$ by Proposition 6.15. Then $p_1 \xrightarrow[\tau']{a} q_1$ and $q = q_1[p/x]$. Now, using successively the fixed point property of Proposition 6.14, the inductive hypothesis and Lemma 7.2 we get

$$[\![\mathrm{rec}\, x.p_1]\!](\sigma) = [\![p_1]\!](\sigma') = a \cdot [\![q_1]\!](\sigma') = a \cdot [\![q]\!](\sigma).$$

□

As promised in Section 5, we give now the

Proof of Proposition 5.3. Assume that $p \xrightarrow[\sigma]{a} p' \xrightarrow[\sigma]{b} p''$ with $a\, I\, b$. By Proposition 7.9 we have $[\![p]\!](\sigma) = a \cdot b \cdot [\![p'']\!](\sigma) = b \cdot a[\![p'']\!](\sigma)$. Hence $b \leq [\![p]\!](\sigma)$, and by Proposition 7.8, we get $p \xrightarrow[\sigma]{b} p'''$. Then Propositions 5.1 and 5.2 allow to conclude the proof. □

The definitions of the operational behaviors given in Section 5 are now fully justified.

Using the above propositions, we show now that each possible (operational) resource trace behavior of p (cf. Definition 5.5) in some environment $\sigma \in \mathcal{P}(\mathcal{R})^V$ corresponds to some compact resource trace below $[\![p]\!](\sigma)$, and, conversely, that each compact resource trace below $[\![p]\!](\sigma)$ approximates some (operational) resource trace behavior of p in σ.

More precisely, in order to relate the operational and the denotational semantics, we use the mapping $\chi : \mathbb{F} \to \mathcal{P}(K(\mathbb{F}))$ defined by $\chi(x) = \{(s, S) \in K(\mathbb{F}) \mid s \leq x, S = \mathrm{res}(s^{-1} \cdot x)\}$. Note that for all $x \in \mathbb{F}$ we have $\chi(x) \subseteq K(x) \subseteq \downarrow\chi(x)$ and therefore $\chi(x)$ is directed and $\sqcup\chi(x) = x$. Recall that for $x \in \mathbb{F}$, we denote by $K(x)$ the set of compact resource traces below x.

Theorem 7.10 (Congruence Theorem) *For all $p \in \mathcal{L}$ and $\sigma \in \mathcal{P}(\mathcal{R})^V$, we have*

$$X_{\mathbb{F}}(p, \sigma) = \chi([\![p]\!](\sigma))$$

and

$$B_{\mathbb{F}}(p, \sigma) = \sqcup X_{\mathbb{F}}(p, \sigma) = [\![p]\!](\sigma).$$

Proof. Let $(s, S) \in X_{\mathbb{F}}(p, \sigma)$. We have $p \xrightarrow[\sigma]{s} q$ for some process $q \in \mathcal{L}$ such that $S = \mathrm{res}(q, \sigma)$. Then Proposition 7.9 implies $[\![p]\!](\sigma) = s \cdot [\![q]\!](\sigma)$ by an easy induction on the length of s. We deduce that $\mathrm{res}(s^{-1}[\![p]\!](\sigma)) = \mathrm{res}([\![q]\!](\sigma)) = \mathrm{res}(q, \sigma) = S$ by Proposition 6.15. Therefore, $(s, S) \in \chi([\![p]\!](\sigma))$.

Conversely, let $(s, S) \in \chi([\![p]\!](\sigma))$. Then $s \in \mathbb{M}$ is a finite prefix of $[\![p]\!](\sigma)$. An induction on the length of s using Proposition 7.8 gives $p \xrightarrow[\sigma]{s} q$ for some process $q \in \mathcal{L}$. Therefore we obtain $[\![p]\!](\sigma) = s \cdot [\![q]\!](\sigma)$ by induction on the length of s using Proposition 7.9. We conclude as above since Proposition 6.15 implies $S = \mathrm{res}(s^{-1}[\![p]\!](\sigma)) = \mathrm{res}([\![q]\!](\sigma)) = \mathrm{res}(q, \sigma)$. Therefore, $(s, S) \in X_{\mathbb{F}}(p, \sigma)$. □

We use $\overline{\mathcal{L}}$ to denote the family of *closed terms* of $\mathcal{L}$ – i.e., those terms in which every variable x falls within the scope of a binding operator $\mathrm{rec}\, x$. For $p \in \mathcal{L}$ and $\xi \in \overline{\mathcal{L}}^V$, we denote by $p(\xi)$ the term p with $\xi(z)$ substituted for z for each variable $z \in V$. Note that the order in which the substitutions are performed does not matter since the processes $\xi(z)$ are closed. Also, we let $[\![\xi]\!] \in \mathbb{F}^V$ be defined componentwise by $[\![\xi]\!](z) = [\![\xi(z)]\!]$ for all $z \in V$.

Remark 7.11 The congruence theorem also may be stated in the following equivalent ways:

(i) For all $p \in \mathcal{L}$ and $\sigma \in \mathcal{P}(\mathcal{R})^V$, we have $X_{\mathbb{F}}(p, \sigma) = \chi([\![p]\!](\sigma))$ and $B_{\mathbb{F}}(p, \sigma) = \sqcup X_{\mathbb{F}}(p, \sigma) = [\![p]\!](\sigma)$.

(ii) For all $p \in \overline{\mathcal{L}}$, we have $X_{\mathbb{F}}(p) = \chi([\![p]\!])$ and $B_{\mathbb{F}}(p) = \sqcup X_{\mathbb{F}}(p) = [\![p]\!]$.

(iii) For all $p \in \mathcal{L}$ and $\xi \in \overline{\mathcal{L}}^V$, we have $X_{\mathbb{F}}(p(\xi)) = \chi([\![p]\!]([\![\xi]\!]))$ and $B_{\mathbb{F}}(p(\xi)) = \sqcup X_{\mathbb{F}}(p(\xi)) = [\![p]\!]([\![\xi]\!])$.

Proof. Part (ii) follows directly from (i). We derive (iii) from (ii) as follows. The process $p(\xi) \in \overline{\mathcal{L}}$ is closed. Using (ii) we have $X_{\mathbb{F}}(p(\xi)) = \chi([\![p(\xi)]\!])$. Lemma 7.2 then implies that $[\![p]\!]([\![\xi]\!]) = [\![p(\xi)]\!]$, which is (iii). Finally, note that (iii) implies (i) using $\xi(z) = \mathrm{STOP}|_{\sigma(z)}$. □

8 Adequacy and Full Abstraction

In this section, we first prove that our denotational semantics $[\![-]\!]$ is adequate with respect to both operational semantics $X_{\mathbb{F}}$ and $X_{\mathbb{M}}$. Then we show that it is also fully abstract with respect to the operational semantics $X_{\mathbb{F}}$. Since the denotational semantics includes the resources that are still claimed by the process after some execution, it would be surprising if it were always fully abstract with respect to the operational semantics $X_{\mathbb{M}}$ which observes only the executions and not the resources still claimed after them. But with some resource maps, these claimed resources may be probed by putting the process in some context, and in these cases we can obtain full abstraction with respect to $X_{\mathbb{M}}$. We give an exact characterization of the resource maps for which this holds.

Recall first that a denotational semantics $[\![-]\!]$ is adequate with respect to an operational semantics $\mathcal{B}$ if $\forall p, p' \in \overline{\mathcal{L}}$, $[\![p]\!] = [\![p']\!]$ implies $\mathcal{B}(p) = \mathcal{B}(p')$.

Theorem 8.1 *The denotational semantics $[\![-]\!]$ is adequate both with respect to the operational semantics $X_{\mathbb{F}}$ and $X_{\mathbb{M}}$.*

Proof. This is a direct corollary of Theorem 7.10. Indeed, for all $p \in \overline{\mathcal{L}}$, we have $X_{\mathbb{F}}(p) = \chi([\![p]\!])$ and $X_{\mathbb{M}}(p) = \mathrm{Re}(X_{\mathbb{F}}(p))$. □

Actually, we can prove stronger results.

Proposition 8.2

(i) $\forall p, p' \in \overline{\mathcal{L}}$, $[\![p]\!] = [\![p']\!]$ *implies* $X_{\mathbb{F}}(p) = X_{\mathbb{F}}(p')$.

(ii) $\forall p, p' \in \overline{\mathcal{L}}$, $[\![p]\!] = [\![p']\!]$ *implies for all* $q \in \mathcal{L}$ *with* $F(q) = \{z\}$, $X_{\mathbb{F}}(q[p/z]) = X_{\mathbb{F}}(q[p'/z])$.

(iii) $\forall p, p' \in \mathcal{L}$, $[\![p]\!] = [\![p']\!]$ *implies for all* $\sigma \in \mathcal{P}(\mathcal{R})^V$, $X_{\mathbb{F}}(p, \sigma) = X_{\mathbb{F}}(p', \sigma)$.

(iv) $\forall p, p' \in \mathcal{L}$, $[\![p]\!] = [\![p']\!]$ *implies for all* $\xi \in \overline{\mathcal{L}}^V$, $X_{\mathbb{F}}(p(\xi)) = X_{\mathbb{F}}(p'(\xi))$.

(v) $\forall p, p' \in \mathcal{L}$, $[\![p]\!] = [\![p']\!]$ *implies for all* $q \in \mathcal{L}$ *with* $z \in F(q)$ *and for all* $\xi \in \overline{\mathcal{L}}^V$, $X_{\mathbb{F}}(q[p/z](\xi)) = X_{\mathbb{F}}(q[p'/z](\xi))$.

Moreover, the same results hold for $X_{\mathbb{M}}$ in place of $X_{\mathbb{F}}$.

Proof. The proof consists of first showing that all of the statements are equivalent. The result then follows from Theorem 8.1, since it shows that (i) holds.

We prove the conditions are equivalent for $X_{\mathbb{F}}$; the proof that they are equivalent for $X_{\mathbb{M}} = \mathrm{Re}(X_{\mathbb{F}})$ then follows from this result. It is clear that (v) implies (ii) and (iv), (ii) implies (i), (iv) implies (i), and (iii) implies (i).

We can also show that (i) implies (v) using Lemma 7.2: Let $p, p' \in \mathcal{L}$ with $[\![p]\!] = [\![p']\!]$. Let $q \in \mathcal{L}$ with $z \in F(q)$ and let $\xi \in \overline{\mathcal{L}}^V$. Then, $r = q[p/z](\xi)$ and $r' = q[p'/z](\xi)$ are closed. Moreover, Lemma 7.2 implies that

$$[\![r]\!] = [\![q[p/z]]\!]([\![\xi]\!]) = [\![q]\!]([\![\xi]\!][z \mapsto [\![p]\!]([\![\xi]\!])]).$$

We have a similar equation for r' and since $[\![p]\!] = [\![p']\!]$, we deduce that $[\![r]\!] = [\![r']\!]$. Now (i) implies $X_{\mathbb{F}}(q[p/z](\xi)) = X_{\mathbb{F}}(q[p'/z](\xi))$.

Finally, (iv) implies (iii). Take $\xi(z) = \mathrm{STOP}|_{\sigma(z)}$. First, Lemma 7.2 implies $[\![p(\xi)]\!] = [\![p]\!]([\![\xi]\!]) = [\![p]\!](\sigma)$. Now, using Theorem 7.10 twice we deduce

$$X_{\mathbb{F}}(p(\xi)) = \chi([\![p(\xi)]\!]) = \chi([\![p]\!](\sigma)) = X_{\mathbb{F}}(p, \sigma)$$

and similarly for p'. The result follows. $\square$

Now we turn to full abstraction and we recall that a denotational semantics $[\![-]\!]$ is fully abstract with respect to an operational semantics $\mathcal{B}$ if $\forall p, p' \in \overline{\mathcal{L}}$, $[\![p]\!] \neq [\![p']\!]$ implies $\exists q \in \mathcal{L}$ with $F(q) = \{z\}$ and $\mathcal{B}(q[p/z]) \neq \mathcal{B}(q[p'/z])$. (The term $q \in \mathcal{L}$ with the single free variable z is usually called a *context*.)

Theorem 8.3 *The denotational semantics $[\![-]\!]$ is fully abstract with respect to the operational semantics $X_{\mathbb{F}}$.*

Proof. Again, this is a trivial corollary of Theorem 7.10. Indeed, for all $p \in \overline{\mathcal{L}}$, we have $\chi([\![p]\!]) = X_{\mathbb{F}}(p)$ and for all resource trace $x \in \mathbb{F}$, it holds $x = \sqcup\chi(x)$. Therefore, $[\![p]\!] \neq [\![p']\!]$ implies $X_{\mathbb{F}}(p) \neq X_{\mathbb{F}}(p')$ and we do not need any context to see that p and p' are different operationally. $\square$

The operational semantics $X_{\mathbb{F}}$ assumes that we are able to observe the resources that are still claimed after some partial execution of a process. One may argue that resources claimed are not observable. We do not agree with this statement and we think that resources claimed are observable in the same way divergence is considered observable in most other semantics, such as in CSP and CCS. An intuitive idea could be that there are lights on the machine indicating which resources are still claimed for the rest of the execution. Such a light will be automatically switched off when the resource is no longer claimed, never to be switched on again during the execution.

A more pragmatic argument showing that observing resources claimed is natural is that the denotational semantics is not fully abstract with respect to $X_{\mathbb{M}}$ for the simple and natural resource alphabet defined by $\Sigma = \{a, b, c, d\}$, $\mathcal{R} = \{\alpha, \beta, \gamma, \delta\}$ with $\mathrm{res}(a) = \{\delta, \alpha\}$, $\mathrm{res}(b) = \{\alpha, \beta\}$, $\mathrm{res}(c) = \{\beta, \gamma\}$, and $\mathrm{res}(d) = \{\gamma, \delta\}$. This fact is a consequence of the next theorem which characterizes the resource alphabets for which the denotational semantics is fully abstract with respect to $X_{\mathbb{M}}$. Indeed, the sets $\{\alpha, \gamma\}$ and $\{\beta, \delta\}$ are indistinguishable (definition just below) but different.

We start with a definition. Two resource sets $S, T \subseteq \mathcal{R}$ are *indistinguishable* if for all $a \in \Sigma$ we have

$$\mathrm{res}(a) \cap S = \emptyset \Longleftrightarrow \mathrm{res}(a) \cap T = \emptyset.$$

We denote by $\equiv$ this indistinguishability relation. Note that if $S \equiv T$ then we have $\mu_S = \mu_T$

and also $\sigma_S = \sigma_T$. This will be crucial later as well as the fact that $\equiv$ is a congruence with respect to union: $S \equiv T$ implies $S \cup U \equiv T \cup U$.

Theorem 8.4 *The denotational semantics $[\![-]\!]$ is fully abstract with respect to the operational semantics $X_{\mathbb{M}}$ if and only if for all sets $S, T \subseteq \mathcal{R}$ we have $S \equiv T \iff S = T$.*

Proof. The rest of this section is devoted to the proof of this theorem. We first show the condition is sufficient. Assume we have two closed processes p and p' such that for all contexts q with only one free variable z, we have $X_{\mathbb{M}}(q[p/z]) = X_{\mathbb{M}}(q[p'/z])$ (note that we do not need environments since $q[p/z]$ and $q[p'/z]$ are closed terms). We have to show that $[\![p]\!] = [\![p']\!]$. Let $[\![p]\!] = (s, S)$ and $[\![p']\!] = (t, T)$. By Theorem 7.10 we have $s = \mathrm{Re}(\sqcup X_{\mathbb{F}}(p)) = \sqcup \mathrm{Re}(X_{\mathbb{F}}(p)) = \sqcup X_{\mathbb{M}}(p)$ and similarly $t = \sqcup X_{\mathbb{M}}(p')$. Considering the context $q = z$ we deduce that $s = t$.

Now, let $a \in \Sigma$ and consider the context $q = z \circ a$. We have $[\![q[p/z]]\!] = (s, S)(a, \emptyset)$ and as above we get $s\mu_S(a) = \mathrm{Re}([\![q[p/z])]\!] = \sqcup X_{\mathbb{M}}(q[p/z])$. Similarly, $s\mu_T(a) = \mathrm{Re}([\![q[p'/z])]\!] = \sqcup X_{\mathbb{M}}(q[p'/z])$ and we deduce that $s\mu_S(a) = s\mu_T(a)$ and therefore, $S \cap \mathrm{res}(a) = \emptyset$ iff $T \cap \mathrm{res}(a) = \emptyset$. Since this holds for all $a \in \Sigma$ we deduce that $S = T$ and therefore $[\![p]\!] = [\![p']\!]$.

In order to show that the condition in Theorem 8.4 is also necessary, we will consider two cases. The first one is when $\mathrm{res}(\Sigma) \neq \mathcal{R}$. Then, let $S = \mathcal{R} \setminus \mathrm{res}(\Sigma)$ and consider the processes $p = \mathrm{STOP}|_S$ and $p' = \mathrm{SKIP}$. We have $[\![p]\!] = (1, S) \neq [\![p']\!] = (1, \emptyset)$. Note that $S \neq \emptyset$ but $S \equiv \emptyset$. We want to show that p and p' are indistinguishable in any context q.

To this end we introduce a notation and prove a lemma. Let $x = (s, S)$ and $y = (t, T)$ be two resource traces. We write $x \approx y$ when $s = t$ and $S \cap \mathrm{res}(\Sigma) = T \cap \mathrm{res}(\Sigma)$. This notation is extended to $\mathbb{F}^V$ component by component: for $\sigma, \sigma' \in \mathbb{F}^V$, we write $\sigma \approx \sigma'$ if $\sigma(z) \approx \sigma'(z)$ for all $z \in V$. Note that if $S \cap \mathrm{res}(\Sigma) = T \cap \mathrm{res}(\Sigma)$ then clearly $S \equiv T$.

Lemma 8.5 *Let $q \in \mathcal{L}$ and $\sigma, \sigma' \in \mathbb{F}^V$ be such that $\sigma \approx \sigma'$. Then we have $[\![q]\!](\sigma) \approx [\![q]\!](\sigma')$.*

Proof. We proceed by structural induction on q. The cases $q = \mathrm{STOP}$, $q = a \in \Sigma$ and $q = z \in V$ are trivial.

• Assume that $q = p|_R$. Using the induction hypothesis, we have $[\![p]\!](\sigma) = (s, S)$ and $[\![p]\!](\sigma') = (s, T)$ with $S \cap \mathrm{res}(\Sigma) = T \cap \mathrm{res}(\Sigma)$. By definition we get

$[\![q]\!](\sigma) = (\mu_{\overline{R}}(s), (S \cup \sigma_{\overline{R}}(s)) \cap R)$ and

$[\![q]\!](\sigma') = (\mu_{\overline{R}}(s), (T \cup \sigma_{\overline{R}}(s)) \cap R)$. We deduce that $[\![q]\!](\sigma) \approx [\![q]\!](\sigma')$.

• Assume now that $q = p_1 \circ p_2$ and for $i = 1, 2$, let $x_i = [\![p_i]\!](\sigma) = (s_i, S_i)$ and $y_i = [\![p_i]\!](\sigma') = (t_i, T_i)$. We have $x_1x_2 = (s_1\mu_{S_1}(s_2), S_1 \cup S_2 \cup \sigma_{S_1}(s_2))$ and $y_1y_2 = (t_1\mu_{T_1}(t_2), T_1 \cup T_2 \cup \sigma_{T_1}(t_2))$. By the inductive hypothesis, we have $s_i = t_i$ and $S_i \cap \mathrm{res}(\Sigma) = T_i \cap \mathrm{res}(\Sigma)$. We deduce that $\mu_{S_1} = \mu_{T_1}$ and $\sigma_{S_1} = \sigma_{T_1}$ and we obtain $x_1x_2 \approx y_1y_2$.

• The next case is when $q = p_1 \underset{C}{\|} p_2$. Using the inductive hypothesis we have $x_i = [\![p_i]\!](\sigma) = (s_i, S_i)$ and $y_i = [\![p_i]\!](\sigma') = (s_i, T_i)$ with $S_i \cap \mathrm{res}(\Sigma) = T_i \cap \mathrm{res}(\Sigma)$. We deduce that $\mathrm{res}(x_i) \cap \mathrm{res}(\Sigma) = \mathrm{res}(y_i) \cap \mathrm{res}(\Sigma)$ and therefore $\mathrm{res}(x_i) \cap \mathrm{res}(a) = \mathrm{res}(y_i) \cap \mathrm{res}(a)$ for all $a \in \Sigma$. Hence we have $\Sigma'_C(x_1, x_2) = \Sigma'_C(y_1, y_2)$. Since $\mathrm{Re}(x_i) = \mathrm{Re}(y_i)$ we deduce that $R_C(x_1, x_2) = R_C(y_1, y_2)$

and therefore $\mathrm{Re}(x_1 \underset{C}{\|} x_2) = \mathrm{Re}(y_1 \underset{C}{\|} y_2)$. Now, using the notations of Section 6.3 we have

$$\mathrm{Im}(x_1 \underset{C}{\|} x_2) = S_1 \cup \mathrm{res}(r_1^{-1}s_1) \cup S_2 \cup \mathrm{res}(r_2^{-1}s_2)$$
$$\mathrm{Im}(y_1 \underset{C}{\|} y_2) = T_1 \cup \mathrm{res}(r_1^{-1}s_1) \cup T_2 \cup \mathrm{res}(r_2^{-1}s_2).$$

Therefore, $[\![q]\!](\sigma) = x_1 \underset{C}{\|} x_2 \approx y_1 \underset{C}{\|} y_2 = [\![q]\!](\sigma')$.

• The last case is $q = \mathrm{rec}\, z.p$. Note that $\sigma[z \mapsto (1,\emptyset)] \approx \sigma'[z \mapsto (1,\emptyset)]$. By the inductive hypothesis we obtain $[\![p]\!](\sigma[z \mapsto (1,\emptyset)]) \approx [\![p]\!](\sigma'[z \mapsto (1,\emptyset)])$. Hence, $x_0 \approx y_0$ with $x_0 = (1, \mathrm{res}([\![p]\!](\sigma[z \mapsto (1,\emptyset)])))$ and $y_0 = (1, \mathrm{res}([\![p]\!](\sigma'[z \mapsto (1,\emptyset)])))$. Now, assume that $x_n \approx y_n$. Then, $\sigma[z \mapsto x_n] \approx \sigma'[z \mapsto y_n]$ and by the inductive hypothesis we obtain $x_{n+1} = [\![p]\!](\sigma[z \mapsto x_n]) \approx [\![p]\!](\sigma'[z \mapsto y_n]) = y_{n+1}$. Therefore, for all $n \geq 0$ we have $x_n \approx y_n$. We deduce easily that

$$\mathrm{Re}([\![q]\!](\sigma)) = \sqcup \mathrm{Re}(x_n) = \sqcup \mathrm{Re}(y_n) = \mathrm{Re}([\![q]\!](\sigma'))$$

and

$$\mathrm{Im}([\![q]\!](\sigma)) \cap \mathrm{res}(\Sigma) = \bigcap (\mathrm{Im}(x_n) \cap \mathrm{res}(\Sigma)) = \bigcap (\mathrm{Im}(y_n) \cap \mathrm{res}(\Sigma)) = \mathrm{Im}([\![q]\!](\sigma')).$$

□

Using this lemma we now show that the denotational semantics is not fully abstract with respect to the operational semantics $X_{\mathbb{M}}$ when $\mathrm{res}(\Sigma) \neq \mathcal{R}$. Let $S = \mathcal{R} \setminus \mathrm{res}(\Sigma)$ and consider the processes $p = \mathrm{STOP}|_S$ and $p' = \mathrm{SKIP}$. We have $[\![p]\!] = (1, S) \approx [\![p']\!] = (1, \emptyset)$. Let $q \in \mathcal{L}$ with $F(q) = \{z\}$. Using Lemma 7.2 we have $[\![q[p/z]]\!] = [\![q]\!]([z \mapsto [\![p]\!]])$ and $[\![q[p'/z]]\!] = [\![q]\!]([z \mapsto [\![p']\!]])$. It follows directly from Lemma 8.5 that $[\![q[p/z]]\!] \approx [\![q[p'/z]]\!]$. Using Theorem 7.10 we deduce that

$$X_{\mathbb{M}}(q[p/z]) = {\downarrow}\mathrm{Re}([\![q[p/z]]\!]) = {\downarrow}\mathrm{Re}([\![q[p'/z]]\!]) = X_{\mathbb{M}}(q[p'/z]).$$

The second and last case is when $\mathrm{res}(\Sigma) = \mathcal{R}$ and the indistinguishability relation is not equality. We have to show that full abstraction does not hold. Again, we introduce a notation and prove a crucial lemma. Note that the hypotheses of this second case are not used for the lemma; they will be needed only in the example following the lemma which shows that full abstraction does not hold in this case.

If $T \subseteq \mathcal{R}$, we extend σ_T to resource traces by $\sigma_T(s, S) = S \cup \sigma_T(s)$. For resource traces $x = (s, S)$ and $y = (s', S')$, we write $x \simeq y$ if the following conditions hold

(i) $s = s'$,

(ii) $S \equiv S'$,

(iii) $\mathrm{res}(x) = \mathrm{res}(y)$, and

(iv) $\emptyset \neq T \subseteq \mathcal{R}$ implies $\sigma_T(x) = \sigma_T(y)$.

The notation $\simeq$ is extended to $\mathbb{F}^V$ component by component: for $\sigma, \sigma' \in \mathbb{F}^V$, we write $\sigma \simeq \sigma'$ if $\sigma(z) \simeq \sigma'(z)$ for all $z \in V$.

Lemma 8.6 *Let $q \in \mathcal{L}$ and $\sigma, \sigma' \in \mathbb{F}^V$ satisfy $\sigma \simeq \sigma'$. Then we have $[\![q]\!](\sigma) \simeq [\![q]\!](\sigma')$.*

Proof. We proceed by structural induction on q. The cases $q = \text{STOP}$, $q = a \in \Sigma$ and $q = z \in V$ are trivial.

• Assume that $q = p|_R$ and let $x = [\![p]\!](\sigma)$ and $y = [\![p]\!](\sigma')$. If $x = y$ or if $R = \mathcal{R}$ then the result is trivial. Otherwise, using the inductive hypothesis we can write $x = (s, S)$ and $y = (s, S')$. We have $[\![q]\!](\sigma) = (\mu_{\overline{R}}(s), \sigma_{\overline{R}}(x) \cap R)$ and $[\![q]\!](\sigma') = (\mu_{\overline{R}}(s), \sigma_{\overline{R}}(y) \cap R)$. Since $\overline{R} \neq \emptyset$, we deduce from the inductive hypothesis that $[\![q]\!](\sigma) = [\![q]\!](\sigma')$.

• Assume now that $q = p_1 \circ p_2$ and for $i = 1, 2$, let $x_i = [\![p_i]\!](\sigma) = (s_i, S_i)$ and $y_i = [\![p_i]\!](\sigma') = (s_i, S'_i)$. Since $\mu_{S_1} = \mu_{S'_1}$ and $\sigma_{S_1} = \sigma_{S'_1}$, we have

$$\begin{aligned} x = x_1 x_2 &= (s_1 \mu_{S_1}(s_2), S_1 \cup S_2 \cup \sigma_{S_1}(s_2)) \\ y = y_1 y_2 &= (s_1 \mu_{S_1}(s_2), S'_1 \cup S'_2 \cup \sigma_{S_1}(s_2)). \end{aligned}$$

Since $\equiv$ is a congruence with respect to union, we deduce that $\text{Im}(x) \equiv \text{Im}(y)$. It follows also directly from the inductive hypothesis that $\text{res}(x) = \text{res}(y)$.

Now, note that for $T \subseteq \mathcal{R}$ we have $\sigma_T(s_1) \subseteq \sigma_T(s_1 \mu_{S_1}(s_2))$ and $\sigma_T(s_2) \subseteq \sigma_T(\mu_{S_1}(s_2)) \cup \sigma_{S_1}(s_2) \subseteq \sigma_T(s_1 \mu_{S_1}(s_2)) \cup \sigma_{S_1}(s_2)$. Therefore, $\sigma_T(x_1) \cup \sigma_T(x_2) \subseteq \sigma_T(x)$. Clearly, we also have $S_1 \cup S_2 \subseteq \sigma_T(x_1) \cup \sigma_T(x_2)$ and we deduce

$$\sigma_T(x) = \sigma_T(x_1) \cup \sigma_T(x_2) \cup \sigma_T(s_1 \mu_{S_1}(s_2)) \cup \sigma_{S_1}(s_2).$$

Similarly

$$\sigma_T(y) = \sigma_T(y_1) \cup \sigma_T(y_2) \cup \sigma_T(s_1 \mu_{S_1}(s_2)) \cup \sigma_{S_1}(s_2).$$

We deduce from the inductive hypothesis that $\sigma_T(x) = \sigma_T(y)$.

• The next case is when $q = p_1 \underset{C}{\|} p_2$. Using the inductive hypothesis we have $x_i = [\![p_i]\!](\sigma) = (s_i, S_i)$ and $y_i = [\![p_i]\!](\sigma') = (s_i, S'_i)$. We have $\text{res}(x_i) = \text{res}(y_i)$ and we deduce that $\Sigma'_C(x_1, x_2) = \Sigma'_C(y_1, y_2)$. It follows directly that $R_C(x_1, x_2) = R_C(y_1, y_2)$ and therefore $\text{Re}(x) = \text{Re}(y)$ where $x = x_1 \underset{C}{\|} x_2$ and $y = y_1 \underset{C}{\|} y_2$. Clearly, we have $\text{res}(x) = \text{res}(y)$. Now, using the notation of Section 6.3 we have

$$\begin{aligned} \text{Im}(x) &= S_1 \cup \text{res}(r_1^{-1} s_1) \cup S_2 \cup \text{res}(r_2^{-1} s_2) \\ \text{Im}(y) &= S'_1 \cup \text{res}(r_1^{-1} s_1) \cup S'_2 \cup \text{res}(r_2^{-1} s_2). \end{aligned}$$

Therefore, $\text{Im}(x) \equiv \text{Im}(y)$.

Now, let $T \subseteq \mathcal{R}$. We have $\sigma_T(r_1) \subseteq \sigma_T(r) = \sigma_T(\text{Re}(x))$. Hence, we deduce that

$$\sigma_T(x_1) = \sigma_T(s_1) \cup S_1 \subseteq \sigma_T(r_1) \cup \text{res}(r_1^{-1} s_1) \cup S_1 \subseteq \sigma_T(\text{Re}(x)) \cup \text{Im}(x) = \sigma_T(x).$$

Similarly, we obtain $\sigma_T(x_2) \subseteq \sigma_T(x)$ and we conclude as in the case of weak sequential composition.

• The last case is $q = \text{rec}\, z.p$. Note that $\sigma[z \mapsto (1, \emptyset)] \simeq \sigma'[z \mapsto (1, \emptyset)]$. By the inductive hypothesis we deduce that $y \simeq y'$ with $y = [\![p]\!](\sigma[z \mapsto (1, \emptyset)])$ and $y' = [\![p]\!](\sigma'[z \mapsto (1, \emptyset)])$. Hence, we have $\text{res}(y) = \text{res}(y')$ and we deduce $x_0 = (1, \text{res}(y)) = (1, \text{res}(y')) = y_0$. Now, assume that $x_n \simeq y_n$. Then, $\sigma[z \mapsto x_n] \simeq \sigma'[z \mapsto y_n]$ and we deduce by inductive hypothesis that $x_{n+1} = [\![p]\!](\sigma[z \mapsto x_n]) \simeq [\![p]\!](\sigma'[z \mapsto y_n]) = y_{n+1}$. Therefore, for all $n \geq 0$ we have $x_n \simeq y_n$.

We have $x = [\![q]\!](\sigma) = \sqcup x_n$ and $y = [\![q]\!](\sigma') = \sqcup y_n$. We deduce easily that

$$\mathrm{Re}([\![q]\!](\sigma)) = \sqcup \mathrm{Re}(x_n) = \sqcup \mathrm{Re}(y_n) = \mathrm{Re}([\![q]\!](\sigma')).$$

Since $\mathcal{R}$ is finite there exists $n \geq 0$ such that

- $\mathrm{Im}(x) = \mathrm{Im}(x_n)$ and $\mathrm{Im}(y) = \mathrm{Im}(y_n)$,
- $\mathrm{res}(x) = \mathrm{res}(x_n)$ and $\mathrm{res}(y) = \mathrm{res}(y_n)$,
- $\sigma_T(\mathrm{Re}(x)) = \sigma_T(\mathrm{Re}(x_n))$ and $\sigma_T(\mathrm{Re}(y)) = \sigma_T(\mathrm{Re}(y_n))$ for all $T \subseteq \mathcal{R}$.

We conclude easily using the inductive hypothesis and $x_n \simeq y_n$. □

We need an additional result.

Lemma 8.7 *Assume that $\Sigma = A \cup B$ with $\mathrm{res}(A) \cap \mathrm{res}(B) = \emptyset$. Let $R, S \subseteq \mathcal{R}$. Then, $R \equiv S$ if and only if $R \cap \mathrm{res}(A) \equiv S \cap \mathrm{res}(A)$ and $R \cap \mathrm{res}(B) \equiv S \cap \mathrm{res}(B)$.*

Proof. Assume first that $R \equiv S$ and let $a \in \Sigma$. Either $a \in A$ and $R \cap \mathrm{res}(A) \cap \mathrm{res}(a) = R \cap \mathrm{res}(a) \neq \emptyset$ iff $S \cap \mathrm{res}(A) \cap \mathrm{res}(a) = S \cap \mathrm{res}(a) \neq \emptyset$. Or $a \in B$ and $R \cap \mathrm{res}(A) \cap \mathrm{res}(a) = \emptyset = S \cap \mathrm{res}(A) \cap \mathrm{res}(a)$. The converse can be shown similarly. □

We are now ready to complete the proof of Theorem 8.4. Assume that $\mathrm{res}(\Sigma) = \mathcal{R}$ and let $S, S' \subseteq \mathcal{R}$ be such that $S \equiv S'$ and $S \neq S'$. Since $\mathrm{res}(\Sigma) = \mathcal{R}$ we can find a connected component $A \subseteq \Sigma$ with respect to the dependence relation D such that $S \cap \mathrm{res}(A) \neq S' \cap \mathrm{res}(A)$. By Lemma 8.7 we have $S \cap \mathrm{res}(A) \equiv S' \cap \mathrm{res}(A)$. Therefore, without loss of generality, we can assume that $S, S' \subseteq \mathrm{res}(A)$, $S \neq S'$ and $S \equiv S'$.

Now consider the two processes $p = (a_1 \circ \cdots \circ a_k)^k \circ \mathrm{STOP}|_S$ and $p' = (a_1 \circ \cdots \circ a_k)^k \circ \mathrm{STOP}|_{S'}$ where $A = \{a_1, \ldots, a_k\}$. Let $x = [\![p]\!] = (s, S)$ and $x' = [\![p']\!] = (s, S')$. We have $x \simeq x'$. Indeed, $\mathrm{res}(x) = \mathrm{res}(A) = \mathrm{res}(x')$. Moreover, since the subalphabet A is connected, we have $\sigma_T(s) = \mathrm{res}(A)$ if $T \neq \emptyset$ and therefore $\sigma_T(x) = \mathrm{res}(A) = \sigma_T(x')$.

We can now conclude as in the first case using Lemma 8.6 instead of Lemma 8.5 that for any context $q \in \mathcal{L}$ with $F(q) = \{z\}$, we have $[\![q[p/z]]\!] \simeq [\![q[p'/z]]\!]$ and therefore, $X_{\mathbb{M}}(q[p/z]) = X_{\mathbb{M}}(q[p'/z])$. □

9 Closing Remarks

In this paper we have presented a simple language including a number of interesting and related operators: weak sequential composition, deterministic parallel composition, restriction and recursion. We have given both an operational and a truly concurrent denotational semantics for the language and shown they are equivalent:

- The Congruence Theorem 7.10 shows that the behavior map associates to each process a unique resource trace it can execute, and this is the same resource trace that the denotational mapping associates to the process. Because the denotational mapping is compositional, this immediately implies that processes with the same denotational meaning have the same observable behavior in any context.
- Theorem 8.4 characterizes the relatively mild conditions under which processes have the same denotational meaning if their observable behavior is the same in all contexts.

A novel feature of our semantics is that it does not involve nondeterministic choice for parallel composition, as in other approaches to modelling concurrency. Others (e.g., [2,11,12,13]) who have studied true concurrency have focused exclusively on the operational side, showing how to describe the behavior of a (concurrent) transition system without appealing to interleavings. While such semantics are important, they do not address issues like compositionality. Thus, this work is the first effort at putting a truly concurrent denotational semantics on the same footing as the more prevalent interleaving models of parallel composition.

The main focus of this paper is not so much the language being studied, *per se*, but rather the underlying mathematical techniques that have been employed to establish the results about its semantics. In particular, the denotation of recursion, relying not on the usual least fixed point semantics of domain theory, but instead on a least fixed point semantics *on a subdomain that varies from process to process* is new. The fact that this can be carried out, and that the recursion operators themselves are continuous are notable. We expect that similar interesting alterations to the traditional semantics methods of domain theory will be needed as we expand this language to one that is more expressive. Indeed, as we commented early on, this work represents the beginning of a longer-range effort to build up a truly concurrent denotational semantics for a complete parallel language, and to eventually compare it to the more traditional interleaving based models. The next step is to add a hiding operator (*à la CSP*) and nondeterministic choice to the language.

Both of the operators just mentioned – hiding and nondeterministic choice – appear to require more abstract models than resource traces provide. For example, if $a\ D\ b\ D\ c$ but $a\ I\ c$, then hiding b in the trace $a \cdot b \cdot c$ does not yield a trace. Our research to this point indicates that pomsets may be useful here; these are a generalization of traces which provide a potential setting in which to model both hiding and deterministic choice. The latter is needed in order to model some of the most basic situations – Hoare's vending machines provide obvious examples. We hope to extend the language to include these operators, and to obtain a full abstraction theorem for the extended language just as we have done for the simple language we presented here. Once this work is complete, we believe we will be in a position to carry out a comparison with the more traditional process algebras supporting concurrency, and we also expect to find many interesting areas where this language will prove very useful in providing a notation and rigorous mathematical setting in which to model physical processes.

References

[1] Abramsky, S and A. Jung, *Domain Theory*, in: "Handbook of Computer Science and Logic," Volume **3**, Clarendon Press, 1995.

[2] Best, E., Frank S. de Boer and Catuscia Palamidessi, *Partial Order and SOS Semantics for Linear Constraint Programs,* in: "Coordination Models and Languages," pp. 256-273, 1997.

[3] Diekert, V. and P. Gastin, *Approximating traces*, *Acta Informatica* **35** (1998), pp. 567–593.

[4] Diekert, V. and G. Rozenberg, editors, "The Book of Traces," World Scientific, Singapore (1995).

[5] Gastin, P. and D. Teodosiu, *Resource traces: a domain for process sharing exclusive resources*, *Theoretical Computer Science*, to appear.

[6] Gierz, G., K. H. Hofmann, K. Keimel, J. Lawson, M. Mislove and D. Scott, “A Compendium of Continuous Lattices,” Springer-Verlag, Berlin, Heidelberg, New York (1980), 376pp.

[7] Hennessy, M. and G. D. Plotkin, *Full abstraction for a simple parallel programming language*, Lecture Notes in Computer Science **74** (1979), Springer-Verlag

[8] Mazurkiewicz, A., *Trace theory*, Lecture Notes in Computer Science **255** (1987), pp. 279–324.

[9] Milner, R., ”Communication and Concurrency”, Prentice-Hall, 1989.

[10] Mislove, M. W. and F. J. Oles, *Full abstraction and recursion, Theoretical Computer Science* **158** (1995), pp.

[11] Nielsen, M. and P. S. Thiagarajan, *Degrees of non-determinism and concurrency: A Petri net view,* In Proc. 4th Conf. Found. of Software Technology and Theor. Comp. Sci., Bangalore, India, Dec. 1984, volume 181 of Lecture Notes in Computer Science, pages 89–117. Springer, 1984.

[12] Olderog, E.-R., “Nets, Terms and Formulas,” Volume 23 of Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1991.

[13] Pratt, V., *On the composition of processes,* Proceedings of the Ninth POPL (1982).

[14] Plotkin, G. D., *A structural approach to operational semantics*, Report DAIMI–FN–19, 1981.

[15] Reisig, W., “Petri Nets,” EATCS Monographs in Theoretical Computer Science **4** (1985), Springer-Verlag.

[16] Roscoe, A.W., *CSP and determinism in security modelling*, Proceedings of the 1995 IEEE Symposium on Security and Privacy, IEEE Computer Society Press, 1995.

[17] Winskel, G., “Events in Computation,” Ph.D. Thesis, University of Cambridge, 1980.