A Truly Concurrent Semantics for a Simple Parallel Programming Language

Paul Gastin[1] and Michael Mislove[2] *

[1] LIAFA, Université Paris 7, 2, place Jussieu, F-75251 Paris Cedex 05, France
Paul.Gastin@liafa.jussieu.fr
[2] Department of Mathematics, Tulane University, New Orleans, LA 70118
mwm@math.tulane.edu

Abstract. This paper represents the beginning of a study aimed at devising semantic models for true concurrency that provide clear distinctions between concurrency, parallelism and choice. We present a simple programming language which includes (weakly) sequential composition, asynchronous and synchronous parallel composition, a restriction operator, and that supports recursion. We develop an operational and a denotational semantics for this language, and we obtain a congruence theorem relating the behavior of a process as described by the transition system to the meaning of the process in the denotational model. This implies that the denotational model is adequate with respect to the operational model. Our denotational model is based on the resource traces of Gastin and Teodesiu, and since a single resource trace represents all possible executions of a concurrent process, we are able to model each term of our concurrent language by a single trace. Therefore we obtain a deterministic semantics for our language and we are able to model parallelism without introducing nondeterminism.

1 Introduction

The basis for building semantic models to support parallel composition of processes was laid out in the seminal work [6]. That work showed how *power domains* could be used to provide such models, but at the expense of introducing nondeterminism in the models, and also into the language. In this paper, we present an alternative approach to modeling parallelism that avoids nondeterminism. Our approach relies instead on *true concurrency.* True concurrency had its beginnings in the seminal work of Mazurkiewicz [7], who developed *trace theory* (as the area is called) in order to devise models for Petri nets, themselves a model of nondeterministic automata [11]. A great deal of research has been carried out in this area (cf. [3]), but programming semantics has not benefited from this research. The reasons are twofold: first, research in trace theory has focused on automata theory, for obvious reasons. The second reason is perhaps more telling:

* Partial support provided by the National Science Foundation and the US Office of Naval Research

the traditional models which were developed for the concatenation operator of trace theory do not support a partial order relative to which concatentation is continuous (in the sense of Scott), nor are they metric spaces relative to which concatenation is a contraction. This means that the standard methods for modeling recursion are not available in these models, and so their use in modeling programming language constructs is limited.

Even so, domain-theoretic connections to trace theory abound in the literature, owing mainly to Winskel's insight that certain domains – *prime event structures* – provide models for concurrency [12]. There also is the work of Pratt [9] that introduced *pomsets*, which are models for trace theory. But none of this work has developed a *programming semantics* approach in which there is an abstract language based on the concatenation operator.

Recently Diekert, Gastin and Teodesiu have developed models for trace theory that are cpos relative to which the concatenation operator is Scott continuous [2, 4]. This opens the way for studying a trace-theoretic approach to concurrency, using these structures as the denotational models for such a language. This paper reports the first research results into this area. It presents a truly concurrent language which supports a number of interesting operators: the concatenation operator of trace theory, a restriction operator that confines processes to a certain set of *resources*, a synchronization operator that allows processes to execute independently, synchronizing on those actions which share common channels in the synchronization set, and that includes process variables and recursion.

The basis for our language is the concatenation operator from trace theory. In trace theory, independent actions can occur concurrently while dependent actions must be ordered. Therefore the concatenation of traces is only *weakly sequential*: it allows the beginning of the second process to occur independently of the end of the first process provided they are independent. We think this is a very attractive feature that corresponds to the automatic parallelization of processes. We also can model purely sequential composition by ending a process with a terminating action which depends on all other actions.

The rest of the paper is organized as follows. In the Section 2 we provide some preliminary background on domain theory, on trace theory generally, and on the resource traces model of Gastin and Teodesiu [4]. These are the main ingredients of the semantic models for our language. The syntax of our language is the subject of the Section 3, and this is followed by Section 4 in which we explore the properties of the resource mapping that assigns to each action its set of resources; the results of this section are needed for both the operational and denotational semantics. Section 5 gives the transition system and the resulting operational semantics of our language, and also shows that the operational transition system is Church-Rosser. Section 6 is devoted to the denotational model, and the seventh section presents the congruence theorem.

Due to space limitations we cannot include all lemmas and most proofs are omitted as well. A full version can be found at the URL
`http://www.liafa.jussieu.fr/~gastin/gastmis99.ps`.

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. Similarly, for the theory of traces the reader is refered to [3]; specific results on resource traces can be found in [4].

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 $x = \vee D \in P$ exists, then $\vee f(D) \in Q$ exists and $f(\vee D) = \vee f(D)$.

If P is a dcpo, the element $k \in P$ is *compact* if, for each directed subset $D \subseteq P$, if $k \sqsubseteq \vee D$, 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 $x = \vee K(x)$ for each $x \in P$.

2.2 Resource traces

We start with a finite alphabet Σ, a finite set $\mathcal{R}$ of *resources*, and a mapping $\mathrm{res}\colon \Sigma \to \mathcal{P}(\mathcal{R})$ satisfying $\mathrm{res}(a) \neq \emptyset$ for all $a \in A$. We can then define a dependence relation on Σ by $(a, b) \in D$ iff $\mathrm{res}(a) \cap \mathrm{res}(b) \neq \emptyset$. The dependence relation is reflexive and symmetric and its complement $I = (\Sigma \times \Sigma) \setminus D$ is called the *independence relation* on Σ.

A *real trace* t over (Σ, D) is the isomorphism class of a labeled, directed 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-labelling satisfying

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

The trace t is finite if V is finite and the length of t is $|t| = |V|$. The set of real traces over (Σ, D) is denoted by $\mathbb{R}(\Sigma, D)$, and the set of finite traces by $\mathbb{M}(\Sigma, D)$.

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 $\mathrm{res}(t) = \mathrm{res}(\mathrm{Alph}(t))$. 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: Let $t_1 = [V_1, E_1, \lambda_1]$ and $t_2 = [V_2, E_2, \lambda_2]$ be 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 $t_1 \cdot t_2 = [V, E, \lambda]$, where $V = V_1 \dot\cup V_2$, $\lambda = \lambda_1 \dot\cup \lambda_2$, 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 monoid of finite traces $(\mathbb{M}(\Sigma, D), \cdot)$ is isomorphic to the quotient monoid $\Sigma^*/\equiv$ of the free monoid Σ^* of finite words over Σ, modulo the least congruence generated by $\{(ab, ba) \mid (a, b) \in I\}$.

The prefix ordering is defined on real traces by $r \leq t$ iff there exists a real trace s such that $t = rs$. When $r \leq t$, then the trace s satisfying $t = rs$ is unique and is denoted by $r^{-1}t$. $(\mathbb{R}(\Sigma, D), \leq)$ is a dcpo with the empty trace as least element. The compact of $(\mathbb{R}(\Sigma, D), \leq)$ are exactly the finite traces.

Just as in the case of the concatenation of words, the concatenation operation on $\mathbb{M}(\Sigma, D)$ is not monotone with respect to the prefix order. It is for this reason that $\mathbb{M}(\Sigma, D)$ 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 Teodesiu [2, 4]. In this paper, we will use the latter work as a basis for the denotational models for our language. The *resource trace monoid* over $(\Sigma, \mathcal{R}, \mathrm{res})$ is then defined to be the family

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

For a resource trace $x = (r, R) \in \mathbb{F}(\Sigma, D)$, we call $\mathrm{Re}(x) = r$ the *real part* of x and $\mathrm{Im}(x) = R$ the *imaginary part* of x. We endow $\mathbb{F}(\Sigma, D)$ 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 $\sigma_R(s) = \mathrm{res}(\mu_R(s)^{-1}s)$. The resource trace monoid $\mathbb{F}(\Sigma, D)$ is also endowed with a partial order called the *approximation order:*

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

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 $r \in \mathbb{M}(\Sigma, D)$ is finite.

We close this section with a simple result about the resource mapping.

Proposition 1. *The resource mapping* $\mathrm{res}\colon \Sigma \to \mathcal{P}(\mathcal{R})$ *extends to a continuous mapping* $\mathrm{res}\colon \mathbb{F}(\Sigma, D) \to (\mathcal{P}(\mathcal{R}), \supseteq)$ *defined by* $\mathrm{res}(r, R) = \mathrm{res}(r) \cup R$. □

2.3 Alphabetic mappings

The results presented in this section are new. They are useful for the denotational semantics of our parallel composition operator.

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 $\operatorname{res}'(\varphi(a)) \subseteq \operatorname{res}(a)$ for all $a \in \Sigma$. We extend φ to real traces: If $r = [V, E, \lambda] \in \mathbb{R}(\Sigma, D)$, 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.

1. *$\varphi : (\mathbb{R}(\Sigma, D), \cdot) \to (\mathbb{R}(\Sigma', D'), \cdot)$ is a morphism.*
2. *$\varphi : (\mathbb{R}(\Sigma, D), \leq) \to (\mathbb{R}(\Sigma', D'), \leq)$ is continuous.* □

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

Proposition 3.

1. *$\varphi : (\mathbb{F}(\Sigma, D), \sqsubseteq) \to (\mathbb{F}(\Sigma', D'), \sqsubseteq)$ is continuous.*
2. *If $\operatorname{res}'(\varphi(a)) = \operatorname{res}(a)\, (\forall a \in \Sigma)$, then $\varphi : (\mathbb{F}(\Sigma, D), \cdot) \to (\mathbb{F}(\Sigma', D'), \cdot)$ is a non-erasing morphism.* □

3 The Language

In this section we introduce a simple parallel programming language. We begin once again with a finite set Σ of atomic actions, a finite set $\mathcal{R}$ of resources, and a mapping $\operatorname{res}\colon \Sigma \to \mathcal{P}(\mathcal{R})$ which assigns to each $a \in \Sigma$ a *non-empty* set of resources. We view $\operatorname{res}(a)$ 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

$$p ::= \mathrm{STOP} \mid a \mid p \circ p \mid p|_R \mid p \underset{C}{\|} p \mid x \mid \operatorname{rec} x.p$$

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 $R \subseteq \mathcal{R}$. Only those actions a from p can execute for which $\operatorname{res}(a) \subseteq R$; all other actions are disabled.

– $p \underset{C}{\|} q$ denotes the parallel composition of the component processes, synchronizing on all actions a which satisfy $\mathrm{res}(a) \cap C \neq \emptyset$, where $C \subseteq \mathcal{R}$. Those actions from either component which do not have any resources in common with any of the actions in the other component nor any resources lying in C are called *local* and can execute independently. Since our semantics is deterministic, this process can only make progress as long as there are no actions from either 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.

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, namely, as the set of possible interleavings of the actions of each component. We avoid nondeterminism, and in fact our language is deterministic. But we still support parallel composition – that in which the actions of each component are independent.

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 conflicting 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 of the other process and of synchronization events which are executed in matching pairs. These synchronization events may introduce conflict when the two processes offer non-matching synchronization events. Since nondeterministic choice is unavailable, conflicting events result in deadlock in our parallel composition. Note that this situation does not occur in a cooperative parallel composition, e.g. in a parallel sorting algorithm.

We view the BNF-like syntax given above as the signature, $\Omega = \cup_n \Omega_n$, of a single sorted universal algebra, where the index n denotes the arity of the operators in the subset Ω_n. In our case, 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;

then $\mathcal{L}$ is the *initial Ω-algebra*. This means that, given any other Ω-algebra A, 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.

4 The Resource Mapping

In this section we define the resources which may be used by a process $p \in \mathcal{L}$. This is crucial for defining the operational semantics of weak sequential composition and of parallel composition. We extend the mapping res: $\Sigma \to \mathcal{P}(\mathcal{R})$ to the full language $\mathcal{L}$ with variables and recursion. In order to define the resource set associated with a process with free variables, we use a *resource environment*, a mapping $\sigma : V \to \mathcal{P}(\mathcal{R})$ 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:

$$\sigma[x \mapsto R](y) = R, \text{ if } y = x, \quad \text{and} \quad \sigma[x \mapsto R](y) = \sigma(y), \text{ otherwise},$$

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

Now, we define inductively the resources of 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 $R \subseteq \mathcal{R}$,
- $\mathrm{res}(p \circ q, \sigma) = \mathrm{res}(p, \sigma) \cup \mathrm{res}(q, \sigma)$,
- $\mathrm{res}(p \underset{q}{\|}, \sigma) = \mathrm{res}(p, \sigma) \cup \mathrm{res}(q, \sigma)$,
- $\mathrm{res}(x, \sigma) = \sigma(x)$ 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 $\mathrm{res}((\mathrm{rec}\, x.(x \circ a)) \underset{C}{\|} (\mathrm{rec}\, y.(b \circ y))) = \mathrm{res}(a) \cup \mathrm{res}(b)$.

It is easy to see that for each process $p \in \mathcal{L}$, the map $\mathrm{res}(p, -) : (\mathcal{P}(\mathcal{R}), \supseteq)^V \to (\mathcal{P}(\mathcal{R}), \supseteq)$ is continuous. A crucial result concerning the resource map states that the definition of recursion is actually a fixed point.

Proposition 4. *Let $p \in \mathcal{L}$ be a process and $\sigma \in \mathcal{P}(\mathcal{R})^V$ be a resource environment. Then,* $\mathrm{res}(\mathrm{rec}\, x.p, \sigma)$ *is the greatest fixed point of the mapping*

$$\mathrm{res}(p, \sigma[x \mapsto -]) : (\mathcal{P}(\mathcal{R}), \supseteq) \to (\mathcal{P}(\mathcal{R}), \supseteq). \qquad \square$$

In fact, 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. The constants STOP and a ($a \in \Sigma$) are interpreted as constant maps $\sigma \mapsto \mathcal{R}$ and $\sigma \mapsto \mathrm{res}(a)$, the process x is interpreted as the projection $\sigma \mapsto \sigma(x)$, restriction $|_R$ is intersection with R, the two compositions $\circ$ and $\underset{C}{\|}$ are union, and finally, recursion $\mathrm{rec}\, x$ is the greatest fixed point: it maps $f \in [\mathcal{P}(\mathcal{R})^V \to \mathcal{P}(\mathcal{R})]$ to the mapping $\sigma \mapsto (\nu R.f(\sigma[x \mapsto R]))$. With this view, the mapping $p \mapsto \mathrm{res}(p, -)$ is the unique Ω-algebra map from $\mathcal{L}$ to $[\mathcal{P}(\mathcal{R})^V \to \mathcal{P}(\mathcal{R})]$.

We use $p[q/x]$ to denote the result of substituting q for the variable x in p. We now show how to compute the resource map at the process $p[q/x]$ in terms of the resource map at p.

Lemma 1. *Let $p, q \in \mathcal{L}$ be two processes and $\sigma \in \mathcal{P}(\mathcal{R})^V$ be a resource environment. Then*

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

5 Operational Semantics

In this section we present an operational semantics for all terms $p \in \mathcal{L}$, even those with free variables. This is necessitated by our use of 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 deriving results about the resulting behavior of terms from $\mathcal{L}$ under these rules. The key is to use environments. 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 $\|\colon \Sigma \times \Sigma \to \Sigma$ that satisfies $\mathrm{res}(a_1\|a_2) = \mathrm{res}(a_1) \cup \mathrm{res}(a_2)$ for all $(a_1, a_2) \in \Sigma^2$. Moreover, for $p_1, p_2 \in \mathcal{L}$, $\sigma \in \mathcal{P}(\mathcal{R})^V$ and $C \subseteq \mathcal{R}$ we define the set $\mathrm{Sync}_{C,\sigma}(p_1, p_2)$ of pairs $(a_1, a_2) \in \Sigma^2$ such that

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

$\mathrm{Sync}_{C,\sigma}(p_1, p_2)$ consists of all pairs which may be synchronized in $p_1 \underset{C}{\|} p_2$. We present the transition rules which are the basis for the operational semantics for our language $\mathcal{L}$ in Table 1 below. We denote by SKIP the process $\mathrm{STOP}|_{\emptyset}$.

We need a number of results about the rules in Table 1 before we can define the operational behaviour of a term $p \in \mathcal{L}$. Some of the results presented here are easier to prove once we have defined the denotational semantics of our language in the following section, but we have chosen to state the results now to improve the readability of the presentation.

Proposition 5. *In the following, $p, p', p'', q \in \mathcal{L}$ are processes, $\sigma, \sigma' \in \mathcal{P}(\mathcal{R})^V$ are syntactic environments, $x \in V$, and $s \in \Sigma^*$.*

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

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

Property III $p \xrightarrow[\sigma]{a} p'$ *and* $p \xrightarrow[\sigma]{a} p''$ *imply* $p' = p''$.

Property IV $p \xrightarrow[\sigma]{a} p'$ *and* $p \xrightarrow[\sigma]{b} p''$, $a \neq b$ *imply* aIb *and* $\exists p''' \in \mathcal{L}$ *with* $p' \xrightarrow[\sigma]{b} p'''$ *and* $p'' \xrightarrow[\sigma]{a} p'''$.

Property V $p \xrightarrow[\sigma]{a} p'$, $p' \xrightarrow[\sigma]{b} p''$, *and* $a\,I\,b$ *imply* $\exists p''' \in \mathcal{L}$ *with* $p \xrightarrow[\sigma]{b} p'''$ *and* $p''' \xrightarrow[\sigma]{a} p'''$. □

Property III means that our transition system is deterministic. Adding Property IV, 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 Property V in our transition system. From this we derive by induction

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

(2a) $$\frac{p_1 \xrightarrow[\sigma]{a} p_1'}{p_1 \circ p_2 \xrightarrow[\sigma]{a} p_1' \circ p_2}$$

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

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

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

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

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

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

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

Corollary 1. *Let $u, v \in \Sigma^*$ with $u \equiv v$. Then $p \xrightarrow[\sigma]{u} q$ iff $p \xrightarrow[\sigma]{v} q$. Hence $p \xrightarrow[\sigma]{s} q$ is well-defined for finite traces $s \in \mathbb{M}(\Sigma, D)$.* □

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 1, we actually can define the possible concurrent behaviors as

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

But, knowing only a possible real (finite) trace that can be executed does not allow us to know how the process can be continued or composed with another process. Hence we need to bring resources into the picture, and so we define the resource trace behaviors by

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

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 can prove 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 exactly what tells us that our semantics of parallelism does not involve nondeterministic choice.

6 Denotational semantics

The denotational semantics for our language takes its values in the family $[\mathbb{F}^V \to \mathbb{F}]$ of continuous maps from $\mathbb{F}^V$ to the underlying domain $\mathbb{F} = \mathbb{F}(\Sigma, D)$ of resource traces. As was the case with the resources model of Section 4, the semantics of a *finitary process* (i.e., one without variables) $p \in \mathcal{L}$ is a constant map, which means it is a resource trace. More generally, the semantics of any closed process $p \in \mathcal{L}$ is simply a resource trace. But, as in the case of the semantics based on the mapping $\mathrm{res}\colon \Sigma \to \mathcal{P}(\mathcal{R})$, 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 $\mathbb{F}$, and the semantics of a recursive process $\mathrm{rec}\, x.p$ is obtained using a fixed points of the semantic map associated with p.

We obtain a compositional semantics by defining the structure of a continuous Ω-algebra on $[\mathbb{F}^V \to \mathbb{F}]$. We define the interpretations constants and variables in $[\mathbb{F}^V \to \mathbb{F}]$ directly, but with the other operators, we instead define their interpretations on $\mathbb{F}$, and then extend them to $[\mathbb{F}^V \to \mathbb{F}]$ in a pointwise fashion. This approach induces on $[\mathbb{F}^V \to \mathbb{F}]$ the structure of a continuous Ω-algebra (cf. [1]).

6.1 Constants and variables

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

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

The first two clearly are continuous, since they are constant maps. The last 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.

6.2 Weak sequential composition

We define the semantics of weak sequential composition using the following result about the concatenation of resource traces.

Proposition 6 ([4]). *Concatenation over resource traces is a continuous operation. Moreover,* $\mathrm{res}(x_1 \cdot x_2) = \mathrm{res}(x_1) \cup \mathrm{res}(x_2)$ *for all* $(x_1, x_2) \in \mathbb{F}^2$. □

The denotational semantics of $\circ$ is then defined by:

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

6.3 Restriction

For restriction and parallel composition, we need to define new operations on traces that have not been introduced so far. We start with restriction, which we obtain 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 sup 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.

Proposition 7. *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* $x \in \mathbb{F}$. □

From this, we obtain the semantics of the restriction operator by

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

6.4 Parallel composition

We require some preliminary definitions and results before we can define the parallel composition of resource traces. We use the results from Section 2.3 to define the semantics of this operation. 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 $\mathbb{R}(\Sigma', D')$ and $\mathbb{F}(\Sigma', D')$ of real traces and of resource traces over the resource alphabet $\mathrm{res}' : \Sigma' \to \mathcal{P}(\mathcal{R})$. We define the alphabetic mappings

$$\begin{aligned} \Pi_i &: \Sigma' \to \Sigma \cup \{1\} && \text{by} \quad \Pi_i(a_1, a_2) = a_i, i = 1, 2 \text{ and} \\ \Pi &: \Sigma' \to \Sigma && \text{by} \quad \Pi(a_1, a_2) = a_1 \| a_2. \end{aligned}$$

where we set $a\|1 = 1\|a = a$. Note that $\mathrm{res}(\Pi_i(a_1,a_2)) \subseteq \mathrm{res}'(a_1,a_2)$, $i = 1,2$, 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) and to continuous maps over resource traces. Moreover, Π also is a morphism of resource traces (Proposition 3).

We consider a subset $C \subseteq \mathcal{R}$ of resources on which we want to synchronize; we call these resources *channels*. We fix two resource traces $x_1 = (s_1, S_1)$ and $x_2 = (s_2, S_2)$ of $\mathbb{F}(\Sigma, D)$ 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 $\varphi(x_1,x_2) \in \mathbb{F}(\Sigma', D')$ which represents the parallel composition of x_1 and x_2. Then we set $x_1 \underset{C}{\|} x_2 = \Pi(\varphi(x_1,x_2))$. Since the mapping Π is continuous, in order to obtain a continuous semantics for parallel composition, we only need to show that the mapping $\varphi : \mathbb{F}(\Sigma, D)^2 \to \mathbb{F}(\Sigma', D')$ 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 $x_i = (s_i, S_i)$, $i = 1,2$, we can define the *synchronization set* $\mathrm{Sync}_{C,\sigma}(x_1,x_2)$ 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 $\varphi(x_1,x_2)$ is defined as

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

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 last set 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.

Now we introduce the set

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

Proposition 8. *The set $X_C(x_1,x_2)$ has a least upper bound $x = (r,R)$ given by*

$$\begin{aligned} r &= \sqcup\{r \in \mathbb{R}(\Sigma'_C(x_1,x_2)) \mid \Pi_i(r) \leq s_i \textit{ for } i = 1,2\}, \\ R &= S_1 \cup S_2 \cup \mathrm{res}(r_1^{-1}s_1) \cup \mathrm{res}(r_2^{-1}s_2), \textit{ where } r_i = \Pi_i(r). \end{aligned}$$

Moreover, $X_C(x_1,x_2) = \downarrow x$ and $\mathrm{res}(x) = \mathrm{res}(x_1) \cup \mathrm{res}(x_2)$. □

Proposition 9. *The mapping* $\varphi\colon \mathbb{F}(\Sigma, D)^2 \to \mathbb{F}(\Sigma', D')$ *defined by* $\varphi(x_1, x_2) = \sqcup X_C(x_1, x_2)$ *is continuous.* $\square$

As announced earlier, we define the semantics of parallel composition by $\underset{C}{\|} = \Pi \circ \varphi$ and the above results imply:

Corollary 2. $\underset{C}{\|} : \mathbb{F}(\Sigma, D)^2 \to \mathbb{F}(\Sigma, D)$ by $x_1 \underset{C}{\|} x_2 = (\Pi \circ \varphi)(x_1, x_2)$ *is continuous. Moreover,* $\operatorname{res}(x_1 \underset{C}{\|} x_2) = \operatorname{res}(x_1) \cup \operatorname{res}(x_2)$ *for all* $(x_1, x_2) \in \mathbb{F}^2$. $\square$

The semantics of parallel composition is defined by

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

6.5 Recursion

In order to have a compositional semantics for recursion, we need to define for each variable $x \in V$ an interpretation $\operatorname{rec} x : [\mathbb{F}^V \to \mathbb{F}] \to [\mathbb{F}^V \to \mathbb{F}]$, and then we will set $[\![\operatorname{rec} x.p]\!] = \operatorname{rec} x.[\![p]\!]$. For $f \in [\mathbb{F}^V \to \mathbb{F}]$, $\operatorname{rec} x.f$ will be defined as a fixed point of a continuous selfmap from $\mathbb{F}$ to $\mathbb{F}$, but we do not use the classical least fixed point semantics. Hence, we need to describe our approach in some detail. Starting with a continuous map $f \in [\mathbb{F}^V \to \mathbb{F}]$, we first consider the maps

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

defined by

$$\begin{aligned}
\varphi_{f,\sigma}(y) &= f(\sigma[x \mapsto y]), \\
\psi_{f,\sigma}(R) &= \operatorname{res}(f(\sigma[x \mapsto (1, R)])).
\end{aligned}$$

Proposition 10. *The two maps* φ *and* ψ *are well-defined and continuous.* $\square$

For $\sigma \in \mathbb{F}^V$, we define $(\operatorname{rec} x.f)(\sigma)$ as a fixed point of the continuous map $\varphi_{f,\sigma}$. Instead of using the least fixed point of $\varphi_{f,\sigma}$, 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 $R\colon [\mathbb{F}^V \to \mathbb{F}] \times \mathbb{F}^V \to \mathcal{P}(\mathcal{R})$ by $(f, \sigma) \mapsto R_{f,\sigma} = \operatorname{FIX}(\psi_{f,\sigma})$ which assigns to each pair (f, σ) the *greatest fixed point* of the monotone selfmap $\psi_{f,\sigma}$. The starting point for the iteration is simply the resource trace $\bot_{f,\sigma} = (1, R_{f,\sigma})$. Therefore, we also have a mapping $\bot\colon [\mathbb{F}^V \to \mathbb{F}] \times \mathbb{F}^V \to \mathbb{F}$ by $(f, \sigma) \mapsto \bot_{f,\sigma} = (1, R_{f,\sigma})$.

Lemma 2. *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.* $\square$

We define the semantics of recursion by

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

Proposition 11. *The mapping* $\operatorname{rec} x\colon [\mathbb{F}^V \to \mathbb{F}] \to [\mathbb{F}^V \to \mathbb{F}]$ *is well defined and continuous.* $\square$

6.6 Summary

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:

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

7 The Congruence Theorem

In this section we complete the picture by showing that the operational behavior of a process defined in Section 5 is the same as the Ω-algebra map we defined for the denotational model in the last section.

To begin, we relate the semantic resources of a process to the syntactic resource of the process. The semantic resource set of the process $p \in \mathcal{L}$ in some environment $\sigma \in \mathbb{F}^V$ is given by $\mathrm{res}([\![p]\!](\sigma))$. In order to relate this semantic resource set to the syntactic resource set defined in Section 4, we introduce the map $\mathrm{res}^V : \mathbb{F}^V \to \mathcal{P}(\mathcal{R})^V$ by $\mathrm{res}^V(\sigma)(x) = \mathrm{res}(\sigma(x))$.

Proposition 12. *Let $p \in \mathcal{L}$ and $\sigma \in \mathbb{F}^V$, then* $\mathrm{res}([\![p]\!](\sigma)) = \mathrm{res}(p, \mathrm{res}^V(\sigma))$. □

The following result is the key lemma for the congruence theorem. It requires an extended sequence of results to derive.

Proposition 13. *Let $p, q \in \mathcal{L}$, $a \in \Sigma$, $\sigma \in \mathbb{F}^V$ and $\tau \in \mathcal{P}(\mathcal{R})^V$. Then*

1. $p \xrightarrow[\mathrm{res}^V(\sigma)]{a} q \;\Rightarrow\; [\![p]\!](\sigma) = a \cdot [\![q]\!](\sigma)$,
2. $[\![p]\!](\tau) = a \cdot [\![q]\!](\tau) \;\Rightarrow\; p \xrightarrow[\tau]{a} q$. □

Using the above proposition, we can show that each possible operational behavior of p in some environment $\sigma \in \mathcal{P}(\mathcal{R})$ corresponds to some compact resource trace below $[\![p]\!](\sigma)$, and, conversely, that each compact resource trace below $[\![p]\!](\sigma)$ approximates some operational behavior of p in σ. From this, the congruence theorem follows.

Theorem 1. *For all $p \in \mathcal{L}$ and $\sigma \in \mathcal{P}(\mathcal{R})^V$, we have $B_{\mathbb{F}}(p, \sigma) = [\![p]\!](\sigma)$. More precisely,*

$$X_{\mathbb{F}}(p, \sigma) \subseteq K([\![p]\!](\sigma)) \subseteq {\downarrow} X_{\mathbb{F}}(p, \sigma).$$ □

8 Closing Remarks

We have presented a simple language that includes a number of interesting related operators: weak sequential composition, deterministic parallel composition, restriction and recursion. We also have presented a congruence theorem relating its operational semantics to its denotational semantics. The novel feature of our language is that the semantics of parallel composition does not involve nondeterministic choice, as in other approaches. We believe this language will have some interesting applications, among them the analysis of security protocols (where determinism has proved to be an important property) and model checking, where trace theory has been used to avoid the state explosion problem.

What remains to be done is to expand the language to include some of the missing operators from the usual approach to process algebra. Chief among these are the hiding operator of CSP and a deterministic choice operator. Both of these 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 congruence theorem for the extended language just as we have done for the simple language we presented here.

References

1. Abramsky, S and A. Jung, *Domain Theory*, in: "Handbook of Computer Science and Logic," Volume **3**, Clarendon Press, 1995.
2. Diekert, V. and P. Gastin, *Approximating traces*, *Acta Informatica* (1997), pp.
3. Diekert, V. and G. Rozenberg, editors, "The Book of Traces," World Scientific, Singapore (1995).
4. Gastin, P. and D. Teodesiu, *Resource traces: a domain for process sharing exclusive resources, Theoretical Computer Science*, submitted.
5. 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.
6. Hennessy, M. and G. D. Plotkin, *Full abstraction for a simple parallel programming language,* Lecture Notes in Computer Science **74** (1979), Springer-Verlag
7. Mazurkiewicz, A., *Trace theory,* Lecture Notes in Computer Science **255** (1987), pp. 279–324.
8. Mislove, M. W. and F. J. Oles, *Full abstraction and recursion, Theoretical Computer Science* **158** (1995), pp.
9. Pratt, V., *On the composition of processes,* Proceedings of the Ninth POPL (1982).
10. Plotkin, G. D., *Structures operational semantics,* DIKU Technical Report.
11. Reisig, W., "Petri Nets," EATCS Monographs in Theoretical Computer Science **4** (1985), Springer-Verlag.
12. Winskel, G., "Events in Computation," Ph.D. Thesis, University of Cambridge, 1980.