Continuation-Passing Semantics for Concurrency
— The Companion Technical Report —

Eneia Nicolae Todoran
Technical University of Cluj-Napoca
Department of Computer Science
Baritiu Str. 28, 400027, Cluj-Napoca, Romania.
{Eneia.Todoran@cs.utcluj.ro}

Nikolaos S. Papaspyrou
National Technical University of Athens
School of Electrical and Computer Engineering
Software Engineering Laboratory
Polytechnioupoli, 15780 Zografou, Greece.
{nickie@softlab.ntua.gr}

Abstract

We investigate the formal design of concurrent languages in continuation-passing style (CPS). We present a continuation-based denotational semantics for an abstract language providing operators for nondeterministic choice, sequential and parallel composition, and a general mechanism of interaction between multisets of distributed actions. We show that the basic laws of concurrent systems are satisfied in this semantics. Next, by customizing the behavior of continuations we obtain denotational semantics for a couple of concurrent languages and a nature-inspired formalism. The languages discussed include Hoare's communication sequential processes (CSP), and two formalisms based on multiparty interactions: a version of CSP extended with communication and synchronization on multiple channels, and a language similar to a process algebra for DNA computing introduced by Cardelli. We accomplish the semantic investigation in the mathematical framework of complete metric spaces.

Keywords: continuation-passing style, denotational semantics, metric spaces, concurrency semantics

1 Introduction

Continuations are a very powerful control-flow mechanism. As abstract representations of control flow, they were introduced in the 1960s and soon came into prominence in the field of denotational semantics for modelling the semantics of programming languages [24, 38, 34]. In a few words, a continuation represents control flow by capturing the notion of "the rest of the program", after the current point during the program's execution. Some functional languages, such as Scheme [23] and SML/NJ [37], provide direct support for capturing and invoking continuations, which are treated as first-class values. In such languages, continuations can be used to implement a variety of advanced control flow constructs. They roughly correspond to "jumps" to different places; programmers can use them as a low-level mechanism to implement disciplined (or undisciplined) "goto", resumable exceptions, or even coroutines.

Continuation-passing style (CPS) is the style of programming in which control is passed explicitly in the form of continuations, in contrast to *direct style*, which is the usual style of programming. In direct style, a function is called with a number of arguments, it evaluates its result, and returns it to the caller. In CPS, on the other hand, functions never return to the caller. The receive as an additional argument a continuation (or more than one, if necessary), which they invoke with the value of their result, as soon as that is computed. Programs in direct style can be automatically translated to CPS [14]. Indeed CPS is commonly employed in the development of optimizing compilers for languages with higher-order functions [3].

In recent years, CPS has found a significant practical use for of synchronous or asynchronous web programming [31, 32, 13, 22]. Continuations do not only generalize the notion of a "callback" function that is standard in web programming. They can be used to circumvent the statelessness of the HTTP protocol which, in the traditional model of web programming, typically leads to poorly structured code. Several continuation-aware web servers and web frameworks have been developed [43, 41, 44, 35] and have known varying degrees of popularity.

In the semantics of programming languages, continuations can be used to model a variety of advanced control concepts, including nonlocal exits, exceptions, coroutines [19] and even multitasking [42, 15]. Various forms of *delimited continuations* [16, 17, 36, 33, 14, 20, 26] provide even finer control than traditional continuations; they represent a part of the rest of what remains to be computed and they allow (or impose) a more structured way to the use of continuations.

However, it is generally believed that traditional continuations do not work well in the presence of concurrency [20]. In the words of Mosses, from his more recent survey of programming language description languages [27]:

> "Although continuation-passing style is sometimes regarded as a standard style to use for denotational semantics, it is inadequate for describing languages that involve non-determinism or concurrent processes."

We argue that the fault lies not with the use of CPS, either as a programming style or as a tool for modeling denotational semantics, but with the standard, traditional notion of continuation, which is inadequate to describe certain aspects of concurrency.

To alleviate this problem, in our previous publications we have introduced the *Continuation Semantics for Concurrency* (CSC) technique and we have used it for defining the semantics of several concurrent languages [39, 9]. CSC provides a general tool for modelling control in concurrent systems and it can be used to design both operational and denotational semantic models. When used for the latter, the distinctive characteristic of the CSC technique is that continuations are modelled as appropriate structures of computations, representative of the control flow concepts of the languages under study. The concurrency control concepts themselves are modelled as operations manipulating continuations. Unlike other semantic models of concurrency [30, 5], in the CSC approach the final yield of the semantic functions is a simple collection of observations; all concurrency control concepts are modelled as operations manipulating continuations.

In this paper, we propose a different solution by focusing on a much simpler structure for continuations, more reminiscent of traditional continuations. Our notion of a *synchronous continuation* roughly coincides with the traditional one: a function that awaits a value and represents the "rest of the program" by yielding its final result. However, the awaited value encompasses an *asynchronous continuation*, which may contain a suspended computation, in a way similar to that used by resumption-based approaches to concurrency [12].

We argue that this enhanced notion of continuation provides the necessary structure that is missing from the traditional approach. The results reported in this paper, briefly outlined below, support this claim. We hope to collect sufficient evidence that CPS can be used as a general tool for designing denotational semantics for concurrent languages.

In the main body of this report (sections 3 and 4), we define an abstract concurrent language $\mathcal{L}$, which supports nondeterministic choice, sequential and parallel composition, and a general mechanism of interaction between multisets of distributed actions. We provide a denotational semantics for this language, designed in CPS, and we prove that our semantics preserves the basic laws of concurrent systems. Following de Bakker and de Vink [5], we use the mathematical framework of complete metric spaces for defining the semantics, which allows us to define mathematical objects and establish the precise relationship between them by making use of Banach's theorem, stating that contracting functions on complete metric spaces have unique fixed points. In the two sections that follow, we shift our focus to two different families of concurrent languages and we show that the same technique can be used to describe their semantics, just by customizing the behavior of continuations. Section 5 discusses two languages based on Hoare's communication sequential processes (CSP), whereas section 6 discusses a concurrent language inspired by DNA computations.

A smaller version of this technical report has been submitted for possible publication to *Fundamenta Informaticae*.

2 Mathematical preliminaries

The notation $(x \in)X$ introduces the set X with typical element x ranging over X. By $\mathcal{P}(X)$ we denote the power set of X, i.e., the set of all subsets of X. The notation $\mathcal{P}_\pi(X)$ denotes the subset of the power set of X whose elements have property π. For example, $\mathcal{P}_{\mathit{finite}}(X)$ is the set of all finite subsets of X. Let $f \in X \to Y$ be a function. The function $[\, f \mid x \mapsto y \,] : X \to Y$, is defined (for $x, x' \in X, y \in Y$) by: $[\, f \mid x \mapsto y \,](x') =$ $\mathsf{if}\ x'{=}x\ \mathsf{then}\ y\ \mathsf{else}\ f(x')$. Instead of $[\,[\, f \mid x_1 \mapsto y_1 \,] \cdots \mid x_n \mapsto y_n \,]$ we write $[\, f \mid x_1 \mapsto y_1 \mid \cdots \mid x_n \mapsto y_n \,]$. Higher-order functions have a prominent role in the semantic models given in this paper. We assume the reader is familiar with the λ calculus notation.

2.1 Multisets

A *multiset* is a generalization of a set, intuitively, a collection in which an element may occur more than once. One can represent the concept of a multiset of elements of type X by using functions from $X \to \mathbb{N}$. A finite multiset of X can be represented by a function $m \in A \to \mathbb{N}^+$, where $A \in \mathcal{P}_{\mathit{finite}}(X)$ and $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$ (the set of positive natural numbers). Let $(x \in)X$ be a countable set. We define:

$$[X] \ \stackrel{\text{not.}}{=} \ \bigcup_{A \in \mathcal{P}_{\mathit{finite}}(X)} \{m \mid m \in A \to \mathbb{N}^+\}$$

As X is countable, $\mathcal{P}_{\mathit{finite}}(X)$ is also countable. $[X]$ is the set of all finite multisets of elements of type X. An element $m \in [X]$ is a function $m : A \to \mathbb{N}^+$, for some finite subset $A \subseteq X$, such that $\forall x \in A : m(x) > 0$. $m(x)$ is called the *multiplicity* (number of occurrences) of x in m.

We represent a multiset $m \in [X]$ by enumerating its elements between brackets '[' and ']'. The elements in a multiset are *not* ordered; to give yet another intuition, a multiset is an unordered list of elements. For example, [] is the empty multiset, i.e., the function with empty graph. As another example $[x_1, x_1, x_2] = [x_1, x_2, x_1] = [x_2, x_1, x_1]$ is the multiset with two occurrences of x_1 and one occurrence of x_2, i.e., the function $m : \{x_1, x_2\} \to \mathbb{N}^+, m(x_1) = 2, m(x_2) = 1$.

One can define various operations on multisets $m_1, m_2 \in [X]$. Below, $\mathsf{dom}(m)$ is the domain of function m. We use the multiset sum operation $m_1 \uplus m_2 \quad (\uplus : [X] \times [X] \to [X])$, defined by $\mathsf{dom}(m_1 \uplus m_2) = \mathsf{dom}(m_1) \cup \mathsf{dom}(m_2)$, and

$$(m_1 \uplus m_2)(x) = \begin{cases} m_1(x) + m_2(x) & \mathsf{if} \quad x \in \mathsf{dom}(m_1) \cap \mathsf{dom}(m_2) \\ m_1(x) & \mathsf{if} \quad x \in \mathsf{dom}(m_1) \setminus \mathsf{dom}(m_2) \\ m_2(x) & \mathsf{if} \quad x \in \mathsf{dom}(m_2) \setminus \mathsf{dom}(m_1) \end{cases}$$

Note that the multiset sum operation $\uplus$ is associative and commutative. We write $m_1 = m_2$ to express that the multisets m_1 and m_2 are equal. $m_1 = m_2$ iff $\mathsf{dom}(m_1) = \mathsf{dom}(m_2)$ and $\forall x \in \mathsf{dom}(m_1) : m_1(x) = m_2(x)$. We also write $x \in m$ to express that $x \in \mathsf{dom}(m)$. More about the mathematics of multisets can be found in [1].

2.2 Metric spaces

The denotational models given in this paper are built within the mathematical framework of *1-bounded complete metric spaces*. We work with the following notions which we assume known: *metric* and *ultra-metric* space, *isometry* (distance preserving bijection between metric spaces, denoted by '$\cong$'), *complete* metric space, and *compact* set. For details the reader may consult, e.g., the monograph [5].

We recall that if (X, d_X), (Y, d_Y) are metric spaces, a function $f : X \to Y$ is a *contraction* if $\exists c \in \mathbb{R}$, $0 \le c < 1$, $\forall x_1, x_2 \in X : d_Y(f(x_1), f(x_2)) \le c \cdot d_X(x_1, x_2)$. In metric semantics it is customary to attach a contracting factor of $c = \frac{1}{2}$ to each computation step. When $c = 1$ the function f is called *nonexpansive*. We denote the set of all nonexpansive functions from X to Y by $X \xrightarrow{1} Y$. Let $f : X \to X$ be a function. If $f(x) = x$ we call x a *fixed point* of f. When this fixed point is unique we write $x = \mathsf{fix}(f)$. The following contraction mapping theorem is at the core of metric semantics.

Theorem 2.1 (Banach) *Let (X, d_X) be a nonempty complete metric space. Each contraction $f : X \to X$ has a* unique *fixed point.*

If $(x, y \in)X$ is any nonempty set, one can define the *discrete metric* on X $(d : X \times X \to [0, 1])$ as follows: $d(x, y) = 0$ if $x = y$, and $d(x, y) = 1$ otherwise. (X, d) is a complete ultrametric space. Other composed metric spaces can be built up using the composite metrics given in Definition 2.2.

Definition 2.2 *Let $(X, d_X), (Y, d_Y)$ be (ultra) metric spaces. On $(x \in)X$, $(f\in)X \to Y$ (the function space), $(x, y)\in X\times Y$ (the Cartesian product), $u, v\in X + Y$ (the disjoint union of X and Y, defined by: $X + Y = (\{1\} \times X) \cup (\{2\} \times Y)$), and $U, V\in\mathcal{P}(X)$ (the power set of X) one can define the following metrics:*

(a) $d_{\frac{1}{2}\cdot X} : X \times X \to [0, 1], \quad d_{\frac{1}{2}\cdot X}(x_1, x_2) = \frac{1}{2} \cdot d_X(x_1, x_2)$

(b) $d_{X\to Y} : (X \to Y) \times (X \to Y) \to [0, 1], \quad d_{X\to Y}(f_1, f_2) = \sup_{x\in X} d_Y(f_1(x), f_2(x))$

(c) $d_{X\times Y} : (X \times Y) \times (X \times Y) \to [0, 1]$
$d_{X\times Y}((x_1, y_1), (x_2, y_2)) = \max\{d_X(x_1, x_2), d_Y(y_1, y_2)\}$

(d) $d_{X+Y} : (X + Y) \times (X + Y) \to [0, 1]$
$d_{X+Y}(u, v) =$ if $(u, v \in X)$ then $d_X(u, v)$ else if $(u, v\in Y)$ then $d_Y(u, v)$ else 1

(e) $d_H : \mathcal{P}(X) \times \mathcal{P}(X) \to [0, 1], \quad d_H(U, V) = \max\{\sup_{u\in U} d(u, V), \sup_{v\in V} d(v, U)\}$, *where* $d(u, W) = \inf_{w\in W} d(u, w)$. *By convention* $\sup \emptyset = 0$, $\inf \emptyset = 1$ *(d_H is the* Hausdorff *metric).*

We use the abbreviation $\mathcal{P}_{nco}(X)$ to denote the power set of *non-empty compact* subsets of X. Also, we often suppress the metric part in domain definitions, and write, e.g., $\frac{1}{2} \cdot X$ instead of $(X, d_{\frac{1}{2}\cdot X})$.

Remark 2.3 *Let $(X, d_X), (Y, d_Y), d_{\frac{1}{2}\cdot X}$, $d_{X\to Y}, d_{X\times Y}$, d_{X+Y} and d_H be as in Definition 2.2. In case d_X, d_Y are ultrametrics, so are $d_{\frac{1}{2}\cdot X}, d_{X\to Y}, d_{X\times Y}, d_{X+Y}$ and d_H. Moreover, if $(X, d_X), (Y, d_Y)$ are complete then $\frac{1}{2} \cdot X$, $X \to Y$, $X \xrightarrow{1} Y$, $X \times Y$,$X + Y$, and $\mathcal{P}_{\mathrm{nco}}(X)$ (with the metrics defined above) are also complete metric spaces [5].*

3 Syntax and Semantics of Interaction

We consider an abstract language $\mathcal{L}$ whose semantics is defined based on a general notion of interaction between multisets of distributed actions. We assume given a set $(a \in)A$ of *elementary statements* or *atomic statements*, and a set $(y \in)Y$ of *recursion variables*. ;, + and $\|$ are operators for sequential, nondeterministic and parallel composition, respectively. $\|$ is also called a *merge* operator. $\lfloor\!\lfloor$ is the *left merge* operator and $|$ is the *synchronization merge* operator. $\lfloor$ is the *left synchronization merge* operator, that is specific of the continuation semantics presented in this paper.

Definition 3.1 *(Syntax of $\mathcal{L}$)*

(a) *(Statements)* $\quad s(\in \mathit{Stat}) ::= a \mid y \mid s;s \mid s + s \mid s \parallel s \mid s|s \mid s\lfloor\!\lfloor s \mid s\lfloor s$

(b) *(Guarded statements)* $\quad g(\in \mathit{GStat}) ::= a \mid g;s \mid g + g \mid g \parallel g \mid g|g \mid g\lfloor\!\lfloor s \mid g\lfloor g$

(c) (Declarations) $(D \in) Decl = Y \to GStat$

(d) (Programs) $(\pi \in) \mathcal{L} = Decl \times Stat$

Remark 3.2 *Following [5], we employ an approach to recursion based on declarations and guarded statements. In a guarded statement each recursive call is preceded by at least one elementary statement, which guarantees the fact that the semantic operators are contracting functions in the present metric setting. For the sake of brevity (and without loss of generality) in what follows we assume a fixed declaration $D \in Decl$. All considerations in any given argument refer to this fixed D. Having a fixed declaration D, instead of an $\mathcal{L}$ program $\pi = (D, s)$ we simply mention the statement s.*

For inductive proofs we introduce a complexity measure ς that decreases upon recursive calls. It is easy to check that ς is well defined due to our restriction to guarded recursion.

Definition 3.3 *(Complexity measure) The function $\varsigma : \mathcal{L} \to \mathbb{N}$ is given by:*

$$\begin{array}{rcll} \varsigma(a) & = & 1 & \\ \varsigma(y) & = & 1 + \varsigma(D(y)) & \\ \varsigma(s_1 \text{ op } s_2) & = & 1 + \varsigma(s_1) & \text{op} \in \{;, \lfloor\!\lfloor\} \\ \varsigma(s_1 \text{ op } s_2) & = & 1 + \max\{\varsigma(s_1), \varsigma(s_2)\} & \text{op} \in \{+, \|, |, \lfloor\} \end{array}$$

Operators $;, +, \|, \lfloor\!\lfloor, |$ are taken from classic process algebra theories [6]. Operator $\lfloor$ is included in the semantic model of $\mathcal{L}$ for technical reasons, that will be explained later. $\mathcal{L}$ is an abstract language. To illustrate the use of CPS as a language design tool, we also consider a couple of concrete languages, obtained from $\mathcal{L}$ by customizing the behavior of elementary actions and the set of operators.

For the concrete languages presented in this paper it is convenient to describe the behavior associated to the elementary statements with the aid of a set Act of *actions*, and we assume given a mapping $i : A \to Act$. Intuitively, an action is an intermediate representation of an elementary statement. The set of actions may be needed for technical reasons, in order to obtain a clear separation between syntax and semantics.

Let $(w \in) W = [Act]$ be the set of all finite multisets of actions, that we name *interaction multisets*. Let also $(\theta \in) \Theta$ be a set of interaction observable effects. There is a special value $\uparrow \in \Theta$, representing a failed interaction (or no interaction at all). The semantics of $\mathcal{L}$ is defined based on a mapping $interact : W \to \Theta$, which describes a general notion of interaction between multisets of distributed actions.[1] Customizing the interaction mapping and the set of operators we obtain various interaction mechanisms, including binary and multiparty CSP-like synchronous communication [21, 10] and the multiparty interaction mechanism employed in the DNA inspired language presented in [8]. These models are presented in Section 5 and Section 6, respectively.

4 Continuation-Passing Semantics for Concurrency

We introduce a denotational semantics designed in continuation-passing style (CPS) for the language $\mathcal{L}$. In this section we assume given a metric domain $(r \in) \mathbf{R}$ whose elements are used as final yields of our denotational semantics. We also assume that $\mathbf{R}$ is endowed with a binary operator $\oplus : \mathbf{R} \times \mathbf{R} \stackrel{1}{\to} \mathbf{R}$, which is nonexpansive, commutative, associative and idempotent. $\oplus$ is an abstract semantic operator for nondeterministic choice in $\mathcal{L}$. Note that, there is no need to impose a branching structure upon $\mathbf{R}$. It

[1] In general, an interaction may also depend upon the current state of the system, in which case the mapping $interact$ should take an additional parameter representing the current state. We do not describe formally such a language in this paper, but we consider an example from the literature. The asynchronous communication mechanism presented in [4] involves no synchronous interaction between two or more distributed actions, but there is a suspension mechanism and the elementary actions are interpreted as state transformations.

can be a simple linear time domain, whose elements are collections of (sequences of) observables. In subsequent sections we will present several possible instantiations of $\mathbf{R}$ and the semantic operator $\oplus$.

The denotational semantics of $\mathcal{L}$ is defined as a mapping $[\![\cdot]\!] : \mathcal{L} \to \mathbf{D}$, where $\mathbf{D}$ is defined by the following domain equation (isometry between complete metric spaces, to be precise):

$$\begin{array}{rcl}(\phi \in)\mathbf{D} & \cong & \mathbf{F} \xrightarrow{1} \mathbf{R} \\ (\varphi \in)\mathbf{F} & = & (W \times \mathbf{K}) \xrightarrow{1} \mathbf{R} \\ (\kappa \in)\mathbf{K} & = & \{\kappa_0\} + \dfrac{1}{2} \cdot \mathbf{D}\end{array}$$

$\mathbf{D}$ is the domain of *denotations* or *computations*. $\mathbf{F}$ is the domain of *synchronous continuations*. $\mathbf{K}$ is the domain of *asynchronous continuations*. A denotation ϕ is a function which receives as parameter a synchronous continuation φ and returns a value of the final domain $\mathbf{R}$. A synchronous continuation receives as parameter a pair (w, κ), consisting of an interaction multiset w and an asynchronous continuation κ and yields a value of the final domain. An asynchronous continuation is either the empty continuation κ_0, or a denotation stored in the space $\frac{1}{2} \cdot \mathbf{D}$.

In the above equations the set W is endowed with the discrete metric which is an ultrametric. The composed metric spaces are built up using the composite metrics given in Definition 2.2. According to the general theory developed in [2], these domain equations have a solution which is *unique* (up to isometry). Note that continuations are elements of a complete space which is the solution of a domain equation where the domain variable occurs in the left hand side of a function space construction. The solutions for $\mathbf{D}$, $\mathbf{F}$ and $\mathbf{K}$ are obtained as complete ultrametric spaces.

Definition 4.1 *Let* $(\omega \in)Op = \mathbf{D} \times \mathbf{D} \xrightarrow{1} \mathbf{D}$*. For any* $\omega \in Op$*, we denote by* $\widehat{\omega}$ *the operator* $\widehat{\omega} : \mathbf{K} \times \mathbf{K} \xrightarrow{1} \mathbf{K}$ *defined, for any* $\phi, \phi_1, \phi_2 \in \mathbf{D}$ *by:* $\widehat{\omega}(\kappa_0, \kappa_0) = \kappa_0$*,* $\widehat{\omega}(\kappa_0, \phi) = \widehat{\omega}(\phi, \kappa_0) = \phi$*, and* $\widehat{\omega}(\phi_1, \phi_2) = \omega(\phi_1, \phi_2)$*. We define* $\Omega_{;}, \Omega_{\lfloor}, \Omega_{\|} : Op \to Op$ *as follows, and we take* $; = \mathsf{fix}(\Omega_{;})$*,* $\| = \mathsf{fix}(\Omega_{\|})$*,* $\|\!\lfloor = \Omega_{;}(\|)$*,* $\lfloor = \Omega_{\lfloor}(\|)$*, and* $| = \Omega_{|}(\|)$*.*

$$\Omega_{;}(\omega)(\phi_1, \phi_2) = \lambda\varphi \,.\, \phi_1(\lambda(w_1, \kappa_1) \,.\, \varphi(w_1, \widehat{\omega}(\kappa_1, \phi_2)))$$

$$\Omega_{\lfloor}(\omega)(\phi_1, \phi_2) = \lambda\varphi \,.\, \phi_1(\lambda(w_1, \kappa_1) \,.\, \phi_2(\lambda(w_2, \kappa_2) \,.\, \varphi(w_1 \uplus w_2, \widehat{\omega}(\kappa_1, \kappa_2))))$$

$$\Omega_{|}(\omega)(\phi_1, \phi_2) = \lambda\varphi \,.\, \Omega_{\lfloor}(\omega)(\phi_1, \phi_2)(\varphi) \oplus \Omega_{\lfloor}(\omega)(\phi_2, \phi_1)(\varphi)$$

$$\Omega_{\|}(\omega)(\phi_1, \phi_2) = \lambda\varphi \,.\, \Omega_{;}(\omega)(\phi_1, \phi_2)(\varphi) \oplus \Omega_{;}(\omega)(\phi_2, \phi_1)(\varphi) \oplus \Omega_{|}(\omega)(\phi_1, \phi_2)(\varphi)$$

In combination with Banach's Theorem 2.1, Lemma 4.2 shows that the semantic operators $;$, $\|$, $\|\!\lfloor$, $|$ and $\lfloor$ are well defined.

Lemma 4.2 *The higher-order mappings* $\Omega_{;}$ *and* $\Omega_{\|}$ *are* $\frac{1}{2}$*- contractive (in* ω*).*

Proof Omitted. The properties stated by this Lemma are an easy consequence of the fact that in the right hand side of each equation defining $\Omega_{;}$, $\Omega_{\|}$, $\Omega_{|}$ and $\Omega_{\lfloor}$, the computations $\widehat{\omega}(\kappa_1, \phi_2)$ and $\widehat{\omega}(\kappa_1, \kappa_2)$ are stored in the space $\frac{1}{2} \cdot \mathbf{D}$. $\square$

Some explanations are necessary. The semantic operators $;$, $\|\!\lfloor$ and $\lfloor$ behave as follows:

$$\phi_1 ; \phi_2 = \lambda\varphi \,.\, \phi_1(\lambda(w_1, \kappa_1) \,.\, \varphi(w_1, \kappa_1 \widehat{;} \phi_2))$$

$$\phi_1 \|\!\lfloor \phi_2 = \lambda\varphi \,.\, \phi_1(\lambda(w_1, \kappa_1) \,.\, \varphi(w_1, \kappa_1 \widehat{\|} \phi_2))$$

$$\phi_1 \lfloor \phi_2 = \lambda\varphi \,.\, \phi_1(\lambda(w_1, \kappa_1) \,.\, \phi_2(\lambda(w_2, \kappa_2) \,.\, \varphi(w_1 \uplus w_2, \kappa_1 \widehat{\|} \kappa_2)))$$

A sequential composition $\phi_1 ; \phi_2$ is evaluated with respect to a synchronous continuation φ. The computation ϕ_1 yields a pair (w_1, κ_1) representing its decomposition into an interaction multiset w_1 and an asynchronous continuation κ_1. Control is then transmitted to the synchronous continuation φ, which receives as parameters the interaction multiset w_1 and an asynchronous continuation consisting of a sequential composition between κ_1 and the computation ϕ_2. The latter does not contribute to the interaction; its execution is started only after the completion of the asynchronous continuation κ_1, representing the rest of the computation ϕ_1. The left merge composition $\phi_1 \lfloor\!\lfloor \phi_2$ is evaluated in a similar manner, but in this case the asynchronous continuation κ_1 is executed in parallel with the computation ϕ_2. The left synchronization operation $\phi_1 \lfloor \phi_2$ evaluates both computations ϕ_1 and ϕ_2; they both contribute to the interaction and their asynchronous continuations are executed in parallel.

The operator for parallel composition $\|$ can be expressed (based on $\lfloor\!\lfloor$ and $\lfloor$) as a nondeterministic choice between alternative computations. It can express all possible interleavings of two or more concurrent processes and synchronize an arbitrary number of concurrent processes, taking into consideration all possible interactions.

$$\phi_1 \,|\, \phi_2 = \lambda\varphi \,.\, (\phi_1 \lfloor \phi_2)(\varphi) \oplus (\phi_2 \lfloor \phi_1)(\varphi)$$

$$\phi_1 \;\|\; \phi_2 = \lambda\varphi \,.\, (\phi_1 \lfloor\!\lfloor \phi_2)(\varphi) \oplus (\phi_2 \lfloor\!\lfloor \phi_1)(\varphi) \oplus (\phi_1 \,|\, \phi_2)(\varphi)$$

Remark 4.3

(a) *Operators* $;\,, \|, \lfloor\!\lfloor\,, |\,, \lfloor : \mathbf{D}\times\mathbf{D} \xrightarrow{1} \mathbf{D}$, *and* $\widehat{;}\,, \widehat{\|} : \mathbf{K}\times\mathbf{K} \xrightarrow{1} \mathbf{K}$ *are nonexpansive in both arguments. They behave as follows:* $\kappa_0 \widehat{;} \kappa_0 = \kappa_0 \widehat{\|} \kappa_0 = \kappa_0$, $\kappa_0 \widehat{;} \phi = \phi \widehat{;} \kappa_0 = \kappa_0 \widehat{\|} \phi = \phi \widehat{\|} \kappa_0 = \phi$, $\phi_1 \widehat{;} \phi_2 = \phi_1 ; \phi_2$, *and* $\phi_1 \widehat{\|} \phi_2 = \phi_1 \| \phi_2$, *for any* $\phi, \phi_1, \phi_2 \in \mathbf{D}$.

(b) *Operator* $\lfloor$ *is specific of the continuation semantics that we present in this paper. It was introduced in order to obtain symmetric definitions for operators* $\|$ *and* $|$. *By using this symmetry and the fact that the nondeterministic choice operator* $\oplus$ *is commutative, it follows that operators* $\|$ *and* $|$ *are also commutative:* $\phi_1 \;\|\; \phi_2 = \phi_2 \;\|\; \phi_1$, *and* $\phi_1 \,|\, \phi_2 = \phi_2 \,|\, \phi_1$, *for any* $\phi_1, \phi_2 \in \mathbf{D}$. *The commutativity of* $\|$ *extends easily to the whole class of asynchronous continuations:* $\kappa_1 \widehat{\|} \kappa_2 = \kappa_2 \widehat{\|} \kappa_1$, *for any* $\kappa_1, \kappa_2 \in \mathbf{K}$.

In Definition 4.4 we assume given a set Act of *actions*, and a mapping $i : A \to Act$, which behaves as explained in Section 3. We recall that $W = [Act]$.

Definition 4.4 (Denotational semantics of $\mathcal{L}$) *Let* $(S \in)Sem = \mathcal{L} \to \mathbf{D}$. *We define* $\Phi : Sem \to Sem$ *as follows. We also take* $[\![\cdot]\!] = \mathsf{fix}(\Phi)$ *to be the denotational semantics of* $\mathcal{L}$ *(*$[\![\cdot]\!] : \mathcal{L} \to \mathbf{D}$*).*

$$\begin{array}{rcl}
\Phi(S)(a)(\varphi) & = & \varphi([i(a)], \kappa_0) \\
\Phi(S)(y)(\varphi) & = & \Phi(D(y))(\varphi) \\
\Phi(S)(s_1 + s_2)(\varphi) & = & \Phi(S)(s_1)(\varphi) \oplus \Phi(S)(s_2)(\varphi) \\
\Phi(S)(s_1 ; s_2)(\varphi) & = & (\Phi(S)(s_1) ; S(s_2))(\varphi) \\
\Phi(S)(s_1 \| s_2)(\varphi) & = & (\Phi(S)(s_1) \lfloor\!\lfloor S(s_2))(\varphi) \;\oplus \\
 & & (\Phi(S)(s_2) \lfloor\!\lfloor S(s_1))(\varphi) \;\oplus \\
 & & (\Phi(S)(s_1) \,|\, \Phi(S)(s_2))(\varphi) \\
\Phi(S)(s_1 \,|\, s_2)(\varphi) & = & (\Phi(S)(s_1) \,|\, \Phi(S)(s_2))(\varphi) \\
\Phi(S)(s_1 \lfloor\!\lfloor s_2)(\varphi) & = & (\Phi(S)(s_1) \lfloor\!\lfloor S(s_2))(\varphi) \\
\Phi(S)(s_1 \lfloor s_2)(\varphi) & = & (\Phi(S)(s_1) \lfloor \Phi(S)(s_2))(\varphi)
\end{array}$$

The denotational semantics $[\![\cdot]\!]$ is defined as the (unique) fixed point of Φ. It may not be obvious why on the right-hand sides of the equations given in Definition 4.4(b), in some places we use $\Phi(S)$ while in other places we use S. The definition of $\Phi(S)$ is organized by induction on $\varsigma(s)$ (see Definition 3.3). Intuitively, Φ is a contraction (hence it has a *unique* fixed point) because in the definition of Φ, each computation $S(s)$ occurs as a parameter of an operator ; or $\lfloor\!\lfloor$, which stores $S(s)$ in the space $\mathbf{K} = \{\kappa_0\} + \frac{1}{2} \cdot \mathbf{D}$. Definition 4.4 is justified by Lemma 4.5, which is easily established.

Lemma 4.5 *The mapping Φ is well defined and $\frac{1}{2}$ - contractive (in S).*

Proof It suffices to show that

$$d(\Phi(S_1)(s)(\varphi), \Phi(S_2)(s)(\varphi)) \leq d(S_1, S_2)$$

for any $s \in Stat$, $\varphi \in \mathbf{F}$. We proceed by induction on $\varsigma(s)$. We treat two subcases.

Case $s = a$.

$$\begin{aligned}
& d(\Phi(S_1)(a)(\varphi), \Phi(S_2)(a)(\varphi)) \\
& = d(\varphi([i(a)], \kappa_0), \varphi([i(a)], \kappa_0)) = 0 \\
& \leq d(S_1, S_2)
\end{aligned}$$

Case $s = s_1\,;\, s_2$.

$$\begin{aligned}
& d(\Phi(S_1)(s_1\,;\, s_2)(\varphi), \Phi(S_2)(s_1\,;\, s_2)(\varphi)) \\
& = d((\Phi(S_1)(s_1)\,;\, S_1(s_2))(\varphi), (\Phi(S_2)(s_1)\,;\, S_2(s_2))(\varphi)) \\
& = d(\Phi(S_1)(s_1)(\lambda(w_1, \kappa_1)\,.\,\varphi(w_1, \kappa_1 \widehat{;} S_1(s_2))), \Phi(S_2)(s_1)(\lambda(w_1, \kappa_1)\,.\,\varphi(w_1, \kappa_1 \widehat{;} S_2(s_2)))) \\
& \leq max\{d(\Phi(S_1)(s_1)(\lambda(w_1, \kappa_1)\,.\,\varphi(w_1, \kappa_1 \widehat{;} S_1(s_2))), \\
& \qquad\qquad \Phi(S_1)(s_1)(\lambda(w_1, \kappa_1)\,.\,\varphi(w_1, \kappa_1 \widehat{;} S_2(s_2))))^{\#.1}, \\
& \qquad\quad d(\Phi(S_1)(s_1)(\lambda(w_1, \kappa_1)\,.\,\varphi(w_1, \kappa_1 \widehat{;} S_2(s_2))), \\
& \qquad\qquad \Phi(S_2)(s_1)(\lambda(w_1, \kappa_1)\,.\,\varphi(w_1, \kappa_1 \widehat{;} S_2(s_2))))^{\#.2}\}
\end{aligned}$$

Taking into account that $\Phi(S_1)(s_1)$ is nonexpansive, for $^{\#.1}$ we can compute as follows:

$^{\#.1} \leq \sup_{(w_1,\kappa_1)\in(W\times\mathbf{K})} d(\varphi(w_1, \kappa_1 \widehat{;} S_1(s_2)), \varphi(w_1, \kappa_1 \widehat{;} S_2(s_2)))$ [φ is nonexpansive]

$\leq \sup_{(w_1,\kappa_1)\in(W\times\mathbf{K})} d((w_1, \kappa_1 \widehat{;} S_1(s_2)), (w_1, \kappa_1 \widehat{;} S_2(s_2)))$

[$\mathbf{K} = \{\kappa_0\} + \frac{1}{2} \cdot \mathbf{D}$]

$\leq d_{\frac{1}{2}\cdot\mathbf{D}}(S_1(s_2), S_2(s_2)) \leq \frac{1}{2} \cdot d_{\mathbf{D}}(S_1, S_2)$

Also, for $^{\#.2}$ we have:

$^{\#.2} \leq d(\Phi(S_1)(s_1), \Phi(S_2)(s_1))$ [Induction hypothesis, $\varsigma(s_1) < \varsigma(s_1\,;\, s_2) = \varsigma(s)$]

$\leq \frac{1}{2} \cdot d(S_1, S_2)$

□

4.1 Concurrency Laws in Continuation Semantics

We present a method of describing the behavior of concurrent systems in denotational models designed in continuation-passing style. For the abstract language $\mathcal{L}$ we show that the semantic operators satisfy the usual laws of concurrency theories, such as commutativity and associativity of parallel composition. Various properties can be proved by simple manipulations of the semantic equations. Some properties require arguments of the kind "$\varepsilon \leq \frac{1}{2} \cdot \varepsilon \Rightarrow \varepsilon = 0$," which are standard in metric semantics [5]. The main concurrency laws that can be established for $\mathcal{L}$ are given in Theorem 4.15.

4.1.1 Some auxiliary properties

The following lemma is useful in various contexts.

Lemma 4.6 *For any $s \in Stat$, and $\varphi_1, \varphi_2 \in \mathbf{F}$, $\phi_1, \phi_2 \in \mathbf{D}$:*

(a) $[\![s]\!](\lambda(w, \kappa)\,.\,\varphi_1(w, \kappa) \oplus \varphi_2(w, \kappa)) = [\![s]\!](\varphi_1) \oplus [\![s]\!](\varphi_2)$

(b) $[\![s]\!](\lambda(w,\kappa)\,.\,\phi_1(\varphi_1)\oplus\phi_2(\varphi_2)) = [\![s]\!](\lambda(w,\kappa)\,.\,\phi_1(\varphi_1))\oplus[\![s]\!](\lambda(w,\kappa)\,.\,\phi_2(\varphi_2))$

Proof We only handle Lemma 4.6(a). We proceed by induction on $\varsigma(s)$. Two subcases.

Case $s = a$.

$$\begin{aligned}&[\![a]\!](\lambda(w,\kappa)\,.\,\varphi_1(w,\kappa)\oplus\varphi_2(w,\kappa))\\ &=\varphi_1([i(a)],\kappa_0)\oplus\varphi_2([i(a)],\kappa_0)\\ &=[\![a]\!](\varphi_1)\oplus[\![a]\!](\varphi_2)\end{aligned}$$

Case $s = s_1 \lfloor s_2$.

$$\begin{aligned}&[\![s_1 \lfloor s_2]\!](\lambda(w,\kappa)\,.\,\varphi_1(w,\kappa)\oplus\varphi_2(w,\kappa))\\ &=[\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,[\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,\varphi_1(w_1\uplus w_2,\kappa_1\widehat{\|}\kappa_2)\oplus\varphi_2(w_1\uplus w_2,\kappa_1\widehat{\|}\kappa_2)))^{\$.0}\end{aligned}$$

Let $\psi_1,\psi_2 \in (W\times\mathbf{K})\xrightarrow{1}\mathbf{F}$ be given by: $\psi_1(w,\kappa) = \lambda(w_2,\kappa_2)\,.\,\varphi_1(w\uplus w_2,\kappa\widehat{\|}\kappa_2)$, and $\psi_2(w,\kappa) = \lambda(w_2,\kappa_2)\,.\,\varphi_2(w\uplus w_2,\kappa\widehat{\|}\kappa_2)$. It is easy to check that ψ_1,ψ_2 are well-defined ($\psi_1(w,\kappa),\psi_2(w,\kappa)\in\mathbf{F}$, for any $w\in W,\kappa\in\mathbf{K}$) and nonexpansive. Therefore:

$$\begin{aligned}&{}^{\$.0}=[\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,[\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,\psi_1(w_1,\kappa_1)(w_2,\kappa_2)\oplus\psi_2(w_1,\kappa_1)(w_2,\kappa_2)))\\ &\text{[Induction hypothesis for } s_2\ (\varsigma(s_2)<\varsigma(s_1\lfloor s_2))]\\ &=[\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,[\![s_2]\!](\psi_1(w_1,\kappa_1))\oplus[\![s_2]\!](\psi_2(w_1,\kappa_1)))\\ &=[\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,[\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,\varphi_1(w_1\uplus w_2,\kappa_1\widehat{\|}\kappa_2))\oplus\\ &\qquad\qquad\qquad[\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,\varphi_2(w_1\uplus w_2,\kappa_1\widehat{\|}\kappa_2)))^{*.0}\end{aligned}$$

Let $\varphi_1,\varphi_2\in\mathbf{F}$ be given by: $\varphi_1' = \lambda(w_1,\kappa_1)\,.\,[\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,\varphi_1(w_1\uplus w_2,\kappa_1\widehat{\|}\kappa_2))$, and $\varphi_2' = \lambda(w_1,\kappa_1)\,.\,[\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,\varphi_2(w_1\uplus w_2,\kappa_1\widehat{\|}\kappa_2))$. In the sequel we compute as follows:

$$\begin{aligned}&{}^{*.0}=[\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,\varphi_1'(w_1,\kappa_1)\oplus\varphi_2'(w_1,\kappa_1))\\ &\text{[Induction hypothesis for } s_1\ (\varsigma(s_1)<\varsigma(s_1\lfloor s_2))]\\ &=[\![s_1]\!](\varphi_1')\oplus[\![s_1]\!](\varphi_2')\\ &=[\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,[\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,\varphi_1(w_1\uplus w_2,\kappa_1\widehat{\|}\kappa_2)))\oplus\\ &\quad[\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,[\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,\varphi_2(w_1\uplus w_2,\kappa_1\widehat{\|}\kappa_2)))\\ &=[\![s_1\lfloor s_2]\!](\varphi_1)\oplus[\![s_1\lfloor s_2]\!](\varphi_2)\end{aligned}$$

□

Lemma 4.7 *Let* $\mathbf{K}_D = \{\kappa_0\}\cup\{[\![s']\!]\mid s'\in \mathit{Stat}\}$. *For any* $s\in\mathit{Stat}$, *and* $\varphi',\varphi''\in\mathbf{F}$*:*

$$d([\![s]\!](\varphi'),[\![s]\!](\varphi''))\le\sup_{w\in W,\kappa\in\mathbf{K}_D} d(\varphi'(w,\kappa),\varphi''(w,\kappa))$$

Proof By induction on $\varsigma(s)$. Two subcases.

Case $s = a$

$$\begin{aligned}d([\![a]\!](\varphi'),[\![a]\!](\varphi'')) &= d(\varphi'([i(a)],\kappa_0),\varphi''([i(a)],\kappa_0))\\ &\le \sup_{w\in W,\kappa\in\mathbf{K}_D} d(\varphi'(w,\kappa),\varphi''(w,\kappa))\end{aligned}$$

Case $s = s_1 \lfloor\!\lfloor s_2$

$$
\begin{array}{rl}
& d([\![s_1 \lfloor\!\lfloor s_2]\!](\varphi'), [\![s_1 \lfloor\!\lfloor s_2]\!](\varphi'')) \\
= & d([\![s1]\!](\lambda(w_1,\kappa_1)\,.\,\varphi'(w_1, \kappa_1 \widehat{\|}\, [\![s_2]\!])), [\![s1]\!](\lambda(w_1,\kappa_1)\,.\,\varphi''(w_1, \kappa_1 \widehat{\|}\, [\![s_2]\!]))) \\
& \text{Induction hypothesis} \\
\leq & \sup_{w\in W, \kappa\in \mathbf{K}_D} d(\varphi'(w, \kappa \widehat{\|}\, [\![s_2]\!]), \varphi''(w, \kappa \widehat{\|}\, [\![s_2]\!])) \\
= & \max\{\sup_{w\in W} d(\varphi'(w, [\![s_2]\!]), \varphi''(w, [\![s_2]\!])), \\
& \qquad\qquad \sup_{w\in W, s'\in Stat} d(\varphi'(w, [\![s' \parallel s_2]\!]), \varphi''(w, [\![s' \parallel s_2]\!]))\} \\
\leq & \sup_{w\in W, \kappa\in \mathbf{K}_D} d(\varphi'(w,\kappa), \varphi''(w,\kappa))
\end{array}
$$

□

Lemma 4.8 *For any* $\mathsf{op}_K : \mathbf{K}\times\mathbf{K} \xrightarrow{1} \mathbf{K}$, $s_1, s_2 \in$ *Stat, and* $\varphi \in \mathbf{F}$*:*

$$
\begin{array}{rl}
& [\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,[\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,\varphi(w_1 \uplus w_2,\ \mathsf{op}_K\ (\kappa_1,\kappa_2)))) \\
= & [\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,[\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,\varphi(w_1 \uplus w_2,\ \mathsf{op}_K\ (\kappa_1,\kappa_2))))
\end{array}
$$

Proof By induction on $\varsigma(s_1)$. Three subcases.

Case $s_1 = a$.

$$
\begin{array}{l}
[\![a]\!](\lambda(w_1,\kappa_1)\,.\,[\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,\varphi(w_1 \uplus w_2,\ \mathsf{op}_K\ (\kappa_1,\kappa_2)))) \\
= [\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,\varphi([i(a)] \uplus w_2,\ \mathsf{op}_K\ (\kappa_0,\kappa_2))) \\
= [\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,[\![a]\!](\lambda(w_1,\kappa_1)\,.\,\varphi(w_1 \uplus w_2,\ \mathsf{op}_K\ (\kappa_1,\kappa_2))))
\end{array}
$$

Case $s_1 = s_1^1 + s_1^2$.

$$
\begin{array}{l}
[\![s_1^1 + s_1^2]\!](\lambda(w_1,\kappa_1)\,.\,[\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,\varphi(w_1 \uplus w_2,\ \mathsf{op}_K\ (\kappa_1,\kappa_2)))) \\
= [\![s_1^1]\!](\lambda(w_1,\kappa_1)\,.\,[\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,\varphi(w_1 \uplus w_2,\ \mathsf{op}_K\ (\kappa_1,\kappa_2))))\oplus \\
\quad [\![s_1^2]\!](\lambda(w_1,\kappa_1)\,.\,[\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,\varphi(w_1 \uplus w_2,\ \mathsf{op}_K\ (\kappa_1,\kappa_2)))) \\
\text{[Induction hypothesis, } \varsigma(s_1) = 1 + max\{\varsigma(s_1^1), \varsigma(s_1^2)\}] \\
= [\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,[\![s_1^1]\!](\lambda(w_1,\kappa_1)\,.\,\varphi(w_1 \uplus w_2,\ \mathsf{op}_K\ (\kappa_1,\kappa_2))))\oplus \\
\quad [\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,[\![s_1^2]\!](\lambda(w_1,\kappa_1)\,.\,\varphi(w_1 \uplus w_2,\ \mathsf{op}_K\ (\kappa_1,\kappa_2)))) \\
\text{[Lemma 4.6(b)]} \\
= [\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,([\![s_1^1]\!](\lambda(w_1,\kappa_1)\,.\,\varphi(w_1 \uplus w_2,\ \mathsf{op}_K\ (\kappa_1,\kappa_2)))\oplus \\
\qquad\qquad\qquad\qquad [\![s_1^2]\!](\lambda(w_1,\kappa_1)\,.\,\varphi(w_1 \uplus w_2,\ \mathsf{op}_K\ (\kappa_1,\kappa_2))))) \\
= [\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,[\![s_1^1 + s_1^2]\!](\lambda(w_1,\kappa_1)\,.\,\varphi(w_1 \uplus w_2,\ \mathsf{op}_K\ (\kappa_1,\kappa_2)))) \\
= [\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,[\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,\varphi(w_1 \uplus w_2,\ \mathsf{op}_K\ (\kappa_1,\kappa_2))))
\end{array}
$$

Case $s_1 = s_1^1 \lfloor\!\lfloor s_1^2$. Let $\mathsf{op}_K{}' : (\mathbf{K}\times\mathbf{K}) \xrightarrow{1} \mathbf{K}$ be given by: $\mathsf{op}_K{}'(\kappa_1^1, \kappa_2) = \mathsf{op}_K\ (\kappa_1^1 \widehat{\|}\, [\![s_1^2]\!], \kappa_2)$. As op_K and ; are nonexpansive, $\mathsf{op}_K{}'$ is also nonexpansive.

$$
\begin{array}{l}
[\![s_1^1 \lfloor\!\lfloor s_1^2]\!](\lambda(w_1,\kappa_1)\,.\,[\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,\varphi(w_1 \uplus w_2,\ \mathsf{op}_K\ (\kappa_1,\kappa_2)))) \\
= [\![s_1^1]\!](\lambda(w_1^1,\kappa_1^1)\,.\,[\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,\varphi(w_1^1 \uplus w_2,\ \mathsf{op}_K\ (\kappa_1^1 \widehat{\|}\, [\![s_1^2]\!],\kappa_2)))) \\
= [\![s_1^1]\!](\lambda(w_1^1,\kappa_1^1)\,.\,[\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,\varphi(w_1^1 \uplus w_2,\ \mathsf{op}_K{}'(\kappa_1^1,\kappa_2)))) \\
\text{[Induction hypothesis, } \varsigma(s_1) = 1 + \varsigma(s_1^1)] \\
= [\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,[\![s_1^1]\!](\lambda(w_1^1,\kappa_1^1)\,.\,\varphi(w_1^1 \uplus w_2,\ \mathsf{op}_K{}'(\kappa_1^1,\kappa_2)))) \\
= [\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,[\![s_1^1]\!](\lambda(w_1^1,\kappa_1^1)\,.\,\varphi(w_1^1 \uplus w_2,\ \mathsf{op}_K\ (\kappa_1^1 \widehat{\|}\, [\![s_1^2]\!],\kappa_2))))
\end{array}
$$

$$= [\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,[\![s_1^1 \mathbin{|\!\lfloor} s_1^2]\!](\lambda(w_1,\kappa_1)\,.\,\varphi(w_1 \uplus w_2,\ \mathsf{op}_K\ (\kappa_1,\kappa_2))))$$

$\square$

If we replace op_K with $\widehat{\|}$ in Lemma 4.8 we obtain immediately the following interesting result: $[\![s_1 \lfloor s_2]\!] = [\![s_2 \lfloor s_1]\!]$, for any $s_1, s_2 \in \mathit{Stat}$. Lemma 4.9 presents some simple properties of $\lfloor$.

Lemma 4.9 *For any $s_1, s_2, s_3 \in \mathit{Stat}$:*

(a) $[\![s_1 \,|\, s_2]\!] = [\![s_2 \,|\, s_1]\!] = [\![(s_1 \lfloor s_2) + (s_2 \lfloor s_1)]\!] = [\![s_1 \lfloor s_2]\!] = [\![s_2 \lfloor s_1]\!]$

(b) $[\![s_1 \lfloor (s_2 + s_3)]\!] = [\![(s_1 \lfloor s_2) + (s_1 \lfloor s_3)]\!]$

(c) $[\![(s_1 + s_2) \lfloor s_3)]\!] = [\![(s_1 \lfloor s_3) + (s_2 \lfloor s_3)]\!]$

Proof Lemma 4.9(a) is an easy consequence of Lemma 4.8. Lemma 4.9(b) can be proved by using Lemma 4.6. Here we only handle Lemma 4.9(c).

$$[\![(s_1 + s_2) \lfloor s_3)]\!] = \lambda\varphi\,.\,[\![s_1 + s_2]\!](\lambda(w,\kappa)\,.\,[\![s_3]\!](\lambda(w_3,\kappa_3)\,.\,\varphi(w \uplus w_3, \kappa \,\widehat{\|}\, \kappa_3)))$$

$$= \lambda\varphi\,.\,([\![s_1]\!](\lambda(w,\kappa)\,.\,[\![s_3]\!](\lambda(w_3,\kappa_3)\,.\,\varphi(w \uplus w_3, \kappa \,\widehat{\|}\, \kappa_3)))\oplus$$

$$[\![s_2]\!](\lambda(w,\kappa)\,.\,[\![s_3]\!](\lambda(w_3,\kappa_3)\,.\,\varphi(w \uplus w_3, \kappa \,\widehat{\|}\, \kappa_3))))$$

$$= \lambda\varphi\,.\,([\![s_1 \lfloor s_3]\!](\varphi) \oplus [\![s_2 \lfloor s_3]\!](\varphi))$$

$$= \lambda\varphi\,.\,[\![(s_1 \lfloor s_3) + (s_2 \lfloor s_3)]\!](\varphi)$$

$$= [\![(s_1 \lfloor s_3) + (s_2 \lfloor s_3)]\!]$$

$\square$

4.1.2 Syntactic contexts and basic interaction terms

We show that continuations can be used to reason in a compositional manner upon the behavior of concurrent programs. We introduce a notion of syntactic *contexts* for the class of $\mathcal{L}$ statements.

Definition 4.10 (Contexts for $\mathcal{L}$)

$$C ::= \bullet \mid a \mid y \mid C;C \mid C+C \mid C\|C \mid C \mathbin{|\!\lfloor} C \mid C|C \mid C\lfloor C$$

We denote by $C(s)$ the result of substituting s for all occurrences of $\bullet$ in C. This substitution can be defined inductively: $\bullet(s) = s$, $a(s) = a$, $y(s) = y$, and $(C_1 \ \mathsf{op}\ C_2)(s) = C_1(s)\ \mathsf{op}\ C_2(s)$, where $\mathsf{op} \in \{;, +, \|, \mathbin{|\!\lfloor}, |, \lfloor\}$.

Lemma 4.11 shows that program properties are preserved in any $\mathcal{L}$ syntactic context. It is an easy consequence of the compositionality of the denotational semantics $[\![\cdot]\!]$.

Lemma 4.11 *If $s_1, s_2 \in \mathit{Stat}$ and $[\![s_1]\!] = [\![s_2]\!]$, then $[\![C(s_1)]\!] = [\![C(s_2)]\!]$, for any context C.*

As a syntactic counterpart of the set W of interaction multisets, we introduce a class of *interaction statements* $(u \in)\,\mathit{UStat}$, with elements of the form $(a_1 \,|\, \cdots \,|\, a_n)$, where $a_i \in A$ are $\mathcal{L}$ elementary statements. The semantic operator for synchronization $|$ is commutative (see Remark 4.3(b)). We will prove that $|$ is also associative, hence any order of association of the elementary statements a_i can be used in an interaction statement.

Definition 4.12 (Interaction statements) *The class $(u \in)\,\mathit{UStat}$ is given by: $u ::= a \mid u\,|\,u$, where $a (\in A)$ is an elementary statement. We define a function $i_U : \mathit{UStat} \to W$, $i_U(a) = [i(a)]$, $i_U(u_1 \,|\, u_2) = i_U(u_1) \uplus i_U(u_2)$. Recall that $i : A \to \mathit{Act}$ is a given function mapping elementary statements to actions, and $W = [\mathit{Act}]$. $\uplus$ is the multiset sum operation, described in Section 2.1.*

In Definition 4.13 we introduce the class $(t \in) TStat$ of *basic interaction terms* ($TStat \subseteq Stat$). We can prove that every non-recursive $\mathcal{L}$ statement (i.e., not containing recursion variables $y \in Y$) is semantically equivalent to a basic interaction term (obviously, any $t \in TStat$ is a non-recursive term).

Definition 4.13 (Basic interaction terms) *We define the set* $(t \in) TStat$ *by:* $t ::= u \mid u;t \mid t+t$.

The properties stated in Lemma 4.14 can be established for any $s, s_1, s_2 \in Stat$ and for any $u \in UStat$. In particular, they can be used to reason about the behavior of basic interaction terms.

Lemma 4.14 *For any* $s, s_1, s_2 \in Stat$*, and for any* $u \in U$ *we have:* $[\![u]\!] = \lambda\varphi \,.\, \varphi(i_U(u), \kappa_0)$, $[\![u;s]\!] = \lambda\varphi \,.\, \varphi(i_U(u), [\![s]\!])$*, and* $[\![s_1 + s_2]\!] = \lambda\varphi \,.\, ([\![s_1]\!](\varphi) \oplus [\![s_2]\!](\varphi))$.

Proof We show that $[\![u]\!] = \lambda\varphi \,.\, \varphi(i_U(u), \kappa_0)$ by structural induction on u.

Case $u = a$.

$$[\![a]\!] = \lambda\varphi \,.\, \varphi([i(a)], \kappa_0) = \lambda\varphi \,.\, \varphi(i_U(a), \kappa_0)$$

Case $u = u_1 \,|\, u_2$.

$$\begin{aligned}
&[\![u_1 \,|\, u_2]\!] \qquad \text{[Lemma 4.9(a)]}\\
&= [\![u_1 \lfloor u_2]\!] = \lambda\varphi \,.\, [\![u_1]\!](\lambda(w_1, \kappa_1) \,.\, [\![u_2]\!](\lambda(w_2, \kappa_2) \,.\, \varphi(w_1 \uplus w_2, \kappa_1 \,\widehat{\|}\, \kappa_2)))\\
&\qquad \text{[Induction hypothesis for } u_2]\\
&= \lambda\varphi \,.\, [\![u_1]\!](\lambda(w_1, \kappa_1) \,.\, \varphi(w_1 \uplus i_U(u_2), \kappa_1 \,\widehat{\|}\, \kappa_0))\\
&\qquad \text{[Induction hypothesis for } u_1]\\
&= \lambda\varphi \,.\, \varphi(i_U(u_1) \uplus i_U(u_2), \kappa_0 \,\widehat{\|}\, \kappa_0) = \lambda\varphi \,.\, \varphi(i_U(u_1 \,|\, u_2), \kappa_0)\\
&= \lambda\varphi \,.\, \varphi(i_U(u), \kappa_0)
\end{aligned}$$

Next, we show that $[\![u;s]\!] = \lambda\varphi \,.\, \varphi(i_U(u), [\![s]\!])$, by using the previous result.

$$\begin{aligned}
&[\![u;s]\!] = \lambda\varphi \,.\, [\![u]\!](\lambda(w, \kappa) \,.\, \varphi(w, \kappa \,\widehat{;}\, [\![s]\!]))\\
&= \lambda\varphi \,.\, \varphi(i_U(u), \kappa_0 \,\widehat{;}\, [\![s]\!])\\
&= \lambda\varphi \,.\, \varphi(i_U(u), [\![s]\!])
\end{aligned}$$

The property $[\![s_1 + s_2]\!] = \lambda\varphi \,.\, ([\![s_1]\!](\varphi) \oplus [\![s_2]\!](\varphi))$ is obvious (by Definition 4.4). $\square$

4.1.3 Concurrency laws

In the sequel we write $s_1 \simeq s_2$ to express that $[\![C(s_1)]\!] = [\![C(s_2)]\!]$ for all $\mathcal{L}$ contexts C.

Theorem 4.15 *For all* $s, s_1, s_2, s_3 \in Stat$*, and* $u, u_1, u_2 \in UStat$*:*

$$\begin{array}{lrcl}
(A1) & s_1 + s_2 & \simeq & s_2 + s_1\\
(A2) & (s_1 + s_2) + s_3 & \simeq & s_1 + (s_2 + s_3)\\
(A3) & s + s & \simeq & s\\
(A4) & (s_1 + s_2); s_3 & \simeq & (s_1; s_3) + (s_2; s_3)\\
(A5) & (s_1; s_2); s_3 & \simeq & s_1; (s_2; s_3)\\
(CM1) & s_1 \,\|\, s_2 & \simeq & (s_1 \,\|\!\lfloor\, s_2) + (s_2 \,\|\!\lfloor\, s_1) + (s_1 \,|\, s_2)\\
(CM2) & u \,\|\!\lfloor\, s & \simeq & u; s\\
(CM3) & (u; s_1) \,\|\!\lfloor\, s_2 & \simeq & u; (s_1 \| s_2)\\
(CM4) & (s_1 + s_2) \,\|\!\lfloor\, s_3 & \simeq & (s_1 \,\|\!\lfloor\, s_3) + (s_2 \,\|\!\lfloor\, s_3)\\
(CM5) & (u_1; s) \,|\, u_2 & \simeq & (u_1 \,|\, u_2); s\\
(CM6) & u_1 \,|\, (u_2;\, s) & \simeq & (u_1 \,|\, u_2); s\\
(CM7) & (u_1; s_1) \,|\, (u_2; s_2) & \simeq & (u_1 \,|\, u_2); (s_1 \| s_2)\\
(CM8) & (s_1 + s_2) \,|\, s_3 & \simeq & (s_1 \,|\, s_3) + (s_2 \,|\, s_3)\\
(CM9) & s_1 \,|\, (s_2 + s_3) & \simeq & (s_1 \,|\, s_2) + (s_1 \,|\, s_3)
\end{array}$$

Proof According to Lemma 4.11, in each case, in order to prove that $s \simeq s'$ it is enough to show that $[\![s]\!] = [\![s']\!]$.

(A1) $[\![s_1 + s_2]\!] = \lambda\varphi \,.\, [\![s_1 + s_2]\!](\varphi)$

$= \lambda\varphi \,.\, ([\![s_1]\!](\varphi) \oplus [\![s_2]\!](\varphi))$ [$\oplus$ is commutative]

$= \lambda\varphi \,.\, ([\![s_2]\!](\varphi) \oplus [\![s_1]\!](\varphi)) = \lambda\varphi \,.\, [\![s_2 + s_1]\!](\varphi)$

$= [\![s_2 + s_1]\!]$

(A2) $[\![(s_1 + s_2) + s_3]\!] = \lambda\varphi \,.\, [\![(s_1 + s_2) + s_3]\!](\varphi)$

$= \lambda\varphi \,.\, (([\![s_1]\!](\varphi) \oplus [\![s_2]\!](\varphi)) \oplus [\![s_3]\!](\varphi))$ [$\oplus$ is associative]

$= \lambda\varphi \,.\, ([\![s_1]\!](\varphi) \oplus ([\![s_2]\!](\varphi) \oplus [\![s_3]\!](\varphi)))$

$= \lambda\varphi \,.\, [\![s_1 + (s_2 + s_3)]\!](\varphi)$

$= [\![s_1 + (s_2 + s_3)]\!]$

(A3) We leave this proof as an easy exercise for the reader. It is a consequence of the fact that $\oplus$ is idempotent.

(A4) $[\![(s_1 + s_2) ; s_3]\!] = \lambda\varphi \,.\, [\![(s_1 + s_2) ; s_3]\!](\varphi)$

$= \lambda\varphi \,.\, [\![s_1 + s_2]\!](\lambda(w, \kappa) \,.\, \varphi(w, \kappa \,\widehat{;}\, [\![s_3]\!]))$

$= \lambda\varphi \,.\, ([\![s_1]\!](\lambda(w, \kappa) \,.\, \varphi(w, \kappa \,\widehat{;}\, [\![s_3]\!])) \oplus [\![s_2]\!](\lambda(w, \kappa) \,.\, \varphi(w, \kappa \,\widehat{;}\, [\![s_3]\!])))$

$= \lambda\varphi \,.\, ([\![s_1 ; s_3]\!](\varphi) \oplus [\![s_2 ; s_3]\!](\varphi))$

$= \lambda\varphi \,.\, [\![(s_1 ; s_3) + (s_2 ; s_3)]\!](\varphi)$

$= [\![(s_1 ; s_3) + (s_2 ; s_3)]\!]$

(A5) The proof for A5 involves an argument of the kind $\varepsilon \leq \frac{1}{2} \cdot \varepsilon \Rightarrow \varepsilon = 0$. Let $\varepsilon = \sup_{s_1, s_2, s_3 \in \mathit{Stat}} d_{\mathbf{D}}([\![s_1]\!] ; ([\![s_2]\!] ; [\![s_3]\!]), ([\![s_1]\!] ; [\![s_2]\!]) ; [\![s_3]\!])$.

$$\sup_{s_1, s_2, s_3 \in \mathit{Stat}} d_{\mathbf{D}}([\![s_1 ; (s_2 ; s_3)]\!], [\![(s_1 ; s_2) ; s_3]\!])$$
$$= \varepsilon = \sup_{s_1, s_2, s_3 \in \mathit{Stat}} d_{\mathbf{D}}(\lambda\varphi \,.\, [\![s_1]\!](\lambda(w_1, \kappa_1) \,.\, \varphi(w_1, \kappa_1 \,\widehat{;}\, ([\![s_2]\!] ; [\![s_3]\!]))),$$
$$\lambda\varphi \,.\, [\![s_1]\!](\lambda(w_1, \kappa_1) \,.\, \varphi(w_1, (\kappa_1 \,\widehat{;}\, [\![s_2]\!]) \,\widehat{;}\, [\![s_3]\!]))) \quad \text{[Lemma 4.7]}$$
$$\leq \sup_{\varphi \in \mathbf{F}, w \in W, \kappa \in \mathbf{K}_D, s_2, s_3 \in \mathit{Stat}} d_{\mathbf{R}}(\varphi(w, \kappa \,\widehat{;}\, ([\![s_2]\!] ; [\![s_3]\!])), \varphi(w, (\kappa \,\widehat{;}\, [\![s_2]\!]) \,\widehat{;}\, [\![s_3]\!]))$$
$$[\varphi \text{ is nonexpansive}, \forall \varphi \in \mathbf{F}\ ]$$
$$\leq \sup_{\kappa \in \mathbf{K}_D, s_2, s_3 \in \mathit{Stat}} d_{\mathbf{K}}(\kappa \,\widehat{;}\, ([\![s_2]\!] ; [\![s_3]\!]), (\kappa \,\widehat{;}\, [\![s_2]\!]) \,\widehat{;}\, [\![s_3]\!])$$
$$[\mathbf{K}_D = \{\kappa_0\} \cup \{s_1 \mid s_1 \in \mathit{Stat}\}, \mathbf{K} = \{\kappa_0\} + (\tfrac{1}{2} \cdot \mathbf{D})]$$

$$= \max\{d_{\frac{1}{2} \cdot \mathbf{D}}([\![s_2 ; s_3]\!], [\![s_2 ; s_3]\!]), \sup_{s_1, s_2, s_3 \in \mathit{Stat}} d_{\frac{1}{2} \cdot \mathbf{D}}([\![s_1]\!] ; ([\![s_2]\!] ; [\![s_3]\!]), ([\![s_1]\!] ; [\![s_2]\!]) ; [\![s_3]\!])\}$$
$$= \tfrac{1}{2} \cdot \sup_{s_1, s_2, s_3 \in \mathit{Stat}} d_{\mathbf{D}}([\![s_1]\!] ; ([\![s_2]\!] ; [\![s_3]\!]), ([\![s_1]\!] ; [\![s_2]\!]) ; [\![s_3]\!]) = \tfrac{1}{2} \cdot \varepsilon$$

We have obtained $\varepsilon \leq \frac{1}{2} \cdot \varepsilon$, which implies $\varepsilon = 0$. Hence $[\![s_1 ; (s_2 ; s_3)]\!] = [\![(s_1 ; s_2) ; s_3]\!]$, for any $s_1, s_2, s_3 \in \mathit{Stat}$, as required.

(CM1) $[\![s_1 \parallel s_2]\!] = \lambda\varphi \,.\, [\![s_1 \parallel s_2]\!](\varphi)$

$= \lambda\varphi \,.\, (([\![s_1]\!] \mathbin{⌊\!⌊} [\![s_2]\!])(\varphi) \oplus ([\![s_2]\!] \mathbin{⌊\!⌊} [\![s_1]\!])(\varphi) \oplus ([\![s_1]\!] \mid [\![s_2]\!])(\varphi))$

$$= \lambda\varphi\,.\,(\llbracket s_1 \mathbin{\lfloor\!\lfloor} s_2\rrbracket(\varphi) \oplus \llbracket s_2 \mathbin{\lfloor\!\lfloor} s_1\rrbracket(\varphi) \oplus \llbracket s_1 \mid s_2\rrbracket(\varphi))$$

$$= \lambda\varphi\,.\,(\llbracket (s_1 \mathbin{\lfloor\!\lfloor} s_2) + (s_2 \mathbin{\lfloor\!\lfloor} s_1) + (s_1 \mid s_2)\rrbracket(\varphi))$$

$$= \llbracket (s_1 \mathbin{\lfloor\!\lfloor} s_2) + (s_2 \mathbin{\lfloor\!\lfloor} s_1) + (s_1 \mid s_2)\rrbracket$$

(CM2) For CM2 we compute as follows:

$$\begin{aligned}
&\llbracket u \mathbin{\lfloor\!\lfloor} s\rrbracket = \lambda\varphi\,.\,\llbracket u \mathbin{\lfloor\!\lfloor} s\rrbracket(\varphi) = \lambda\varphi\,.\,\llbracket u\rrbracket(\lambda(w,\kappa)\,.\,\varphi(w,\kappa \mathbin{\widehat{\|}} \llbracket s\rrbracket)) \quad \text{[Lemma 4.14]}\\
&= \lambda\varphi\,.\,\varphi(i_U(u),\llbracket s\rrbracket) = \lambda\varphi\,.\,\llbracket u\rrbracket(\lambda(w,\kappa)\,.\,\varphi(w,\kappa \mathbin{\widehat{;}} \llbracket s\rrbracket))\\
&= \lambda\varphi\,.\,\llbracket u;s\rrbracket(\varphi) = \llbracket u;s\rrbracket
\end{aligned}$$

(CM3) $\llbracket (u;s_1) \mathbin{\lfloor\!\lfloor} s_2\rrbracket = \lambda\varphi\,.\,\llbracket (u;s_1) \mathbin{\lfloor\!\lfloor} s_2\rrbracket(\varphi)$

$$= \lambda\varphi\,.\,\llbracket u;s_1\rrbracket(\lambda(w_1,\kappa_1)\,.\,\varphi(w_1,\kappa_1 \mathbin{\widehat{\|}} \llbracket s_2\rrbracket))$$

$$= \lambda\varphi\,.\,\llbracket u\rrbracket(\lambda(w,\kappa)\,.\,\varphi(w,(\kappa \mathbin{\widehat{\|}} \llbracket s_1\rrbracket) \mathbin{\widehat{\|}} \llbracket s_2\rrbracket)) \quad \text{[Lemma 4.14]}$$

$$= \lambda\varphi\,.\,\varphi(i_U(u),(\kappa_0 \mathbin{\widehat{\|}} \llbracket s_1\rrbracket) \mathbin{\widehat{\|}} \llbracket s_2\rrbracket)$$

$$= \lambda\varphi\,.\,\varphi(i_U(u),\llbracket s_1\rrbracket \parallel \llbracket s_2\rrbracket)$$

$$= \lambda\varphi\,.\,\varphi(i_U(u),\llbracket s_1 \parallel s_2\rrbracket) \quad \text{[Lemma 4.14]}$$

$$= \llbracket u;(s_1 \parallel s_2)\rrbracket$$

(CM4) $\llbracket (s_1+s_2) \mathbin{\lfloor\!\lfloor} s_3\rrbracket = \lambda\varphi\,.\,\llbracket (s_1+s_2) \mathbin{\lfloor\!\lfloor} s_3\rrbracket(\varphi)$

$$= \lambda\varphi\,.\,\llbracket s_1+s_2\rrbracket(\lambda(w,\kappa)\,.\,\varphi(w,\kappa \mathbin{\widehat{\|}} \llbracket s_3\rrbracket))$$

$$= \lambda\varphi\,.\,(\llbracket s_1\rrbracket(\lambda(w,\kappa)\,.\,\varphi(w,\kappa \mathbin{\widehat{\|}} \llbracket s_3\rrbracket)) \oplus \llbracket s_2\rrbracket(\lambda(w,\kappa)\,.\,\varphi(w,\kappa \mathbin{\widehat{\|}} \llbracket s_3\rrbracket)))$$

$$= \lambda\varphi\,.\,(\llbracket s_1 \mathbin{\lfloor\!\lfloor} s_3\rrbracket(\varphi) \oplus \llbracket s_2 \mathbin{\lfloor\!\lfloor} s_3\rrbracket(\varphi))$$

$$= \lambda\varphi\,.\,\llbracket (s_1 \mathbin{\lfloor\!\lfloor} s_3) + (s_2 \mathbin{\lfloor\!\lfloor} s_3)\rrbracket(\varphi)$$

$$= \llbracket (s_1 \mathbin{\lfloor\!\lfloor} s_3) + (s_2 \mathbin{\lfloor\!\lfloor} s_3)\rrbracket$$

(CM5) $\llbracket (u_1;s) \mid u_2\rrbracket = \lambda\varphi\,.\,\llbracket (u_1;s) \mid u_2\rrbracket(\varphi)$ [Lemma 4.9]

$$= \lambda\varphi\,.\,\llbracket (u_1;s) \lfloor u_2\rrbracket(\varphi)$$

$$= \lambda\varphi\,.\,\llbracket u_1;s\rrbracket(\lambda(w,\kappa)\,.\,\llbracket u_2\rrbracket(\lambda(w_2,\kappa_2)\,.\,\varphi(w \uplus w_2,\kappa \mathbin{\widehat{\|}} \kappa_2)))$$

$$= \lambda\varphi\,.\,\llbracket u_1\rrbracket(\lambda(w_1,\kappa_1)\,.\,\llbracket u_2\rrbracket(\lambda(w_2,\kappa_2)\,.\,\varphi(w_1 \uplus w_2,(\kappa_1 \mathbin{\widehat{;}} \llbracket s\rrbracket) \mathbin{\widehat{\|}} \kappa_2))) \quad \text{[Lemma 4.14]}$$

$$= \lambda\varphi\,.\,\llbracket u_1\rrbracket(\lambda(w_1,\kappa_1)\,.\,\varphi(w_1 \uplus i_U(u_2),(\kappa_1 \mathbin{\widehat{;}} \llbracket s\rrbracket) \mathbin{\widehat{\|}} \kappa_0)) \quad \text{[Lemma 4.14]}$$

$$= \lambda\varphi\,.\,\varphi(i_U(u_1) \uplus i_U(u_2),(\kappa_0 \mathbin{\widehat{;}} \llbracket s\rrbracket) \mathbin{\widehat{\|}} \kappa_0))$$

$$= \lambda\varphi\,.\,\varphi(i_U(u_1 \mid u_2),\llbracket s\rrbracket) \quad \text{[Lemma 4.14]}$$

$$= \llbracket (u_1 \mid u_2);s\rrbracket$$

(CM6) $\llbracket u_1 \mid (u_2;s)\rrbracket$ [Lemma 4.9]

$$= \llbracket (u_2;s) \mid u_1\rrbracket \quad \text{[(CM5), } \mathit{UStat} \subseteq \mathit{Stat}\text{, Lemma 4.9]}$$

$$= \llbracket (u_1 \mid u_2);s\rrbracket$$

(CM7) $[\![(u_1\,;\,s_1)\,|\,(u_2\,;\,s_2)]\!] = \lambda\varphi\,.\,[\![(u_1\,;\,s_1)\,|\,(u_2\,;\,s_2)]\!](\varphi)$ [Lemma 4.9]

$$= \lambda\varphi\,.\,[\![(u_1\,;\,s_1)\,\lfloor\,(u_2\,;\,s_2)]\!](\varphi)$$

$$= \lambda\varphi\,.\,[\![u_1\,;\,s_1]\!](\lambda(w_1,\kappa_1)\,.\,[\![u_2\,;\,s_2]\!](\lambda(w_2,\kappa_2)\,.\,\varphi(w_1 \uplus w_2, \kappa_1 \,\widehat{\|}\, \kappa_2)))$$

$$= \lambda\varphi\,.\,[\![u_1\,;\,s_1]\!](\lambda(w_1,\kappa_1)\,.\,[\![u_2]\!](\lambda(w_2',\kappa_2')\,.\,\varphi(w_1 \uplus w_2', \kappa_1 \,\widehat{\|}\, (\kappa_2' \widehat{;} [\![s_2]\!])))))$$

$$= \lambda\varphi\,.\,[\![u_1]\!](\lambda(w_1',\kappa_1')\,.\,[\![u_2]\!](\lambda(w_2',\kappa_2')\,.\,\varphi(w_1' \uplus w_2', (\kappa_1' \widehat{;} [\![s_1]\!]) \,\widehat{\|}\, (\kappa_2' \widehat{;} [\![s_2]\!])))))\quad \text{[Lemma 4.14]}$$

$$= \lambda\varphi\,.\,[\![u_1]\!](\lambda(w_1',\kappa_1')\,.\,\varphi(w_1' \uplus i_U(u_2), (\kappa_1' \widehat{;} [\![s_1]\!]) \,\widehat{\|}\, (\kappa_0 \widehat{;} [\![s_2]\!]))))\quad \text{[Lemma 4.14]}$$

$$= \lambda\varphi\,.\,\varphi(i_U(u_1) \uplus i_U(u_2), (\kappa_0 \widehat{;} [\![s_1]\!]) \,\widehat{\|}\, (\kappa_0 \widehat{;} [\![s_2]\!]))\quad \text{[Lemma 4.14]}$$

$$= \lambda\varphi\,.\,\varphi(i_U(u_1\,|\,u_2), [\![s_1 \parallel s_2]\!])\quad \text{[Lemma 4.14]}$$

$$= [\![(u_1\,|\,u_2)\,;\,(s_1 \parallel s_2)]\!]$$

(CM8) $[\![(s_1 + s_2)\,|\,s_3]\!]$ [Lemma 4.9]

$$= [\![(s_1 + s_2)\,\lfloor\,s_3]\!]\quad \text{[Lemma 4.9]}$$

$$= [\![(s_1\,\lfloor\,s_3) + (s_2\,\lfloor\,s_3)]\!] = \lambda\varphi\,.\,[\![(s_1\,\lfloor\,s_3) + (s_2\,\lfloor\,s_3)]\!](\varphi)$$

$$= \lambda\varphi\,.\,([\![s_1\,\lfloor\,s_3]\!](\varphi) \oplus [\![s_2\,\lfloor\,s_3]\!](\varphi))\quad \text{[Lemma 4.9]}$$

$$= \lambda\varphi\,.\,([\![s_1\,|\,s_3]\!](\varphi) \oplus [\![s_2\,|\,s_3]\!](\varphi))$$

$$= \lambda\varphi\,.\,[\![(s_1\,|\,s_3) + (s_2\,|\,s_3)]\!](\varphi)$$

$$= [\![(s_1\,|\,s_3) + (s_2\,|\,s_3)]\!]$$

(CM9) Property (CM9) follows easily by using (CM8) the fact that the synchronization operator '$|$' is commutative.

□

We can find similarities between the set of laws established in Theorem 4.15 and the axioms of the Algebra of Communication Processes (ACP) of Bergstra and Klop [7, 6]. Both systems are parameterized by a set $(a \in)A$, the elements of which are called atomic (or elementary) statements in $\mathcal{L}$, respectively atomic actions in ACP. The left synchronization merge operator $\lfloor$ of $\mathcal{L}$ is lacking from ACP, but ACP provides an encapsulation operator ∂_H ($H \subseteq A$), which is not included in $\mathcal{L}$. Also, in ACP there is a (partial) communication function $\gamma : A \times A \to A$ which describes the effect of simultaneously executing two atomic actions (considering the notation employed in [6]).

If in $\mathcal{L}$ we see the set $(u \in)\,UStat$ of interaction statements as a set of "atomic actions", then the set of laws stated in Theorem 4.15 coincides with the set of axioms of the ACP process algebra from which the axioms for deadlock and the axioms for encapsulation are eliminated (see [7, 6]). In this interpretation, the two systems behave the same if we consider a communication (interaction) function $\gamma : UStat \times UStat \to UStat$, defined for all pairs u_1, u_2 of "actions" by $\gamma(u_1, u_2) = u_1\,|\,u_2$, in which case the axioms for deadlock can also be removed (because this γ is defined everywhere).

Remark 4.16 *For any non-recursive $\mathcal{L}$ program (closed term) $s \in Stat$, there is a basic interaction term $t \in TStat$ such that $s \simeq t$.*[2]

We can prove this claim by considering the laws stated in Theorem 4.15 as a term rewriting system with rules corresponding to laws A3-A5 and CM1-CM9. Each of these laws can give rise to a

[2] In Remark 4.16 we investigate the similarities with the ACP theory [7, 6], hence we consider only $\mathcal{L}$ programs that do not contain the left synchronization merge operator $\lfloor$.

corresponding rewriting rule (by reading the law from left to right). For example, the rewriting rule corresponding to law CM1 is: $s_1 \parallel s_2 \to (s_1 \lfloor\!\lfloor s_2) + (s_2 \lfloor\!\lfloor s_1) + (s_1 \,|\, s_2)$. *It is not difficult to prove that the rewriting system corresponding to laws A3-A5 and CM1-CM9 is strongly normalizing*[3] *and confluent*[4] *modulo the laws A1 and A2 (i.e., up to the order of summands). Various similar proofs are provided, e.g., in [7, 6]. The terminology that we use in this remark is also taken from [7, 6].*

Let $(h \in)HStat$ *be given by* $h ::= u \mid h; h \mid h + h$. *Clearly,* $HStat$ *is a (strict) superset of* $TStat$ *(we have* $UStat \subseteq TStat \subseteq HStat \subseteq Stat$*). In* $TStat$ *the sequential composition is available only in the restricted (prefix) form* $u; t$*, with* $u \in UStat$ *and* $t \in TStat$*, whereas* $HStat$ *also contains statements of the form* $h_1; h_2$*, for arbitrary* $h_1, h_2 \in HStat$*. It is easy to see that, for every* $HStat$ *term* h *there is a basic interaction term* t *such that* $h \simeq t$*. This can be proved by considering a term rewriting system with rules corresponding to the laws A4 and A5 (stated in Theorem 4.15). It is not hard to see that this term rewriting system is strongly normalizing and confluent (modulo the laws A1 and A2), and that a normal form*[5] *of a closed term is a basic interaction term* $t \in TStat$*.*

More generaly, one can show that the term rewriting system corresponding to the laws A3-A5 and CM1-CM9 is strongly normalizing and confluent (modulo the laws A1 and A2) and that a normal form must be a basic interaction term $t \in TStat$*. For if a closed* $\mathcal{L}$ *term* s *is an element of* $(s \in)HStat$*, the property follows by rewriting* s *into a basic interaction term* $t \in TStat$*, as explained above. Otherwise, if a closed* $\mathcal{L}$ *term* s $(s \notin HStat)$ *has an occurrence of the parallel composition operator* $\parallel$ *then the rewriting rule corresponding to CM1 can still be applied. Finally, if the term* s *has an occurrence of* $\lfloor\!\lfloor$ *or* $|$ *and* $s \notin HStat$*, then consider the smallest subterm* s' *such that* $s' \notin HStat$ *and* s' *has an occurrence of* $\lfloor\!\lfloor$ *or* $|$*. Such a subterm* s' *has either the form* $s'_1 \lfloor\!\lfloor s'_2$ *or the form* $s'_1 \,|\, s'_2$*, where* $s'_1, s'_2 \in HStat$*; note that, if* $s' = s'_1 \,|\, s'_2$ *then either* $s'_1 \notin UStat$ *or* $s'_2 \notin UStat$*, because we assume that* $s' \notin HStat$ *(if* $s'_1 \in UStat$ *and* $s'_2 \in UStat$ *then we also have* $s'_1 \,|\, s'_2 \in UStat$*). If* s' *has the form* $s'_1 \lfloor\!\lfloor s'_2$ *then we can rewrite* $s'_1 (\in HStat)$ *into a basic interaction term and in this case one of the rewriting rules corresponding to the laws CM2-CM4 can still be applied. Otherwise, if* s' *has the form* $s'_1 \,|\, s'_2$ *then we can rewrite both* $s'_1 (\in HStat)$ *and* $s'_2 (\in HStat)$ *into corresponding basic interaction terms* $t'_1 (\in TStat)$ *and* $t'_2 (\in TStat)$ *and we obtain* $t'_1 \,|\, t'_2$*; note that in this case either* $t'_1 \notin UStat$ *or* $t'_2 \notin UStat$ *(because either* $s'_1 \notin UStat$ *or* $s'_2 \notin UStat$*), hence one of the rewriting rules corresponding to the laws CM5-CM9 is applicable.*

Proposition 4.17 establishes a couple of properties which are also well known from classic process algebra theories [6], including the associativity of operators $\parallel, |$. The left synchronization merge operator $\lfloor$ is also associative. The proof of Proposition 4.17 is laborious but it relies again on arguments of the kind "$\varepsilon \le \frac{1}{2} \cdot \varepsilon \Rightarrow \varepsilon = 0$", which are standard in metric semantics [5].

Proposition 4.17 *For all* $s, s_1, s_2, s_3 \in Stat$*:*

$$\begin{array}{rrcl}
(1) & (s_1 \parallel s_2) \parallel s_3 & \simeq & s_1 \parallel (s_2 \parallel s_3) \\
(2) & (s_1 \lfloor s_2) \lfloor s_3 & \simeq & s_1 \lfloor (s_2 \lfloor s_3) \\
(3) & (s_1 \,|\, s_2) \,|\, s_3 & \simeq & s_1 \,|\, (s_2 \,|\, s_3) \\
(4) & (s_1 \lfloor\!\lfloor s_2) \lfloor\!\lfloor s_3 & \simeq & s_1 \lfloor\!\lfloor (s_2 \parallel s_3) \\
(5) & s_1 \,|\, (s_2 \lfloor\!\lfloor s_3) & \simeq & (s_1 \,|\, s_2) \lfloor\!\lfloor s_3
\end{array}$$

Proof According to Lemma 4.11, in each case, in order to prove that $s \simeq s'$ it is enough to show that $[\![s]\!] = [\![s']\!]$. The proof of property 4.17(1) (the associativity of the parallel composition operator $\parallel$) is the most laborious one. Let $\varepsilon = \sup_{s_1,s_2,s_3 \in Stat} d_{\mathbf{D}}(([\![s_1]\!] \parallel [\![s_2]\!]) \parallel [\![s_3]\!], [\![s_1]\!] \parallel ([\![s_2]\!] \parallel [\![s_3]\!]))$. We will show that $\varepsilon \le \frac{1}{2} \cdot \varepsilon$, which implies $\varepsilon = 0$. Therefore we have $[\![(s_1 \parallel s_2) \parallel s_3]\!] = [\![(s_1 \parallel s_2) \parallel s_3]\!]$, hence we obtain property 4.17(1): $(s_1 \parallel s_2) \parallel s_3 \simeq s_1 \parallel (s_2 \parallel s_3)$, as required.

$$\varepsilon = \sup_{s_1,s_2,s_3 \in Stat} \sup_{\varphi \in \mathbf{F}} d((([\![s_1]\!] \parallel [\![s_2]\!]) \parallel [\![s_3]\!])(\varphi), ([\![s_1]\!] \parallel ([\![s_2]\!] \parallel [\![s_3]\!]))(\varphi))$$

[3] There is no infinite sequence of reductions $s^1 \to s^2 \to \cdots$.

[4] Whenever $s \to \cdots \to s^1$ and $s \to \cdots \to s^2$ we can find an s' such that $s^1 \to \cdots \to s'$ and $s^2 \to \cdots \to s'$.

[5] A term s is a *normal form* (or in normal form) if there is no term s' with $s \to s'$.

$$
\begin{aligned}
= \sup_{s_1,s_2,s_3\in Stat,\varphi\in\mathbf{F}} d((&\llbracket s_1\rrbracket \,\|\, \llbracket s_2\rrbracket)(\lambda(w,\kappa)\,.\,\varphi(w,\kappa \mathbin{\widehat{\|}} \llbracket s_3\rrbracket))\oplus \\
&\llbracket s_3\rrbracket(\lambda(w_3,\kappa_3)\,.\,\varphi(w_3,\kappa_3 \mathbin{\widehat{\|}} (\llbracket s_1\rrbracket \,\|\, \llbracket s_2\rrbracket)))\oplus \\
&(\llbracket s_1\rrbracket \,\|\, \llbracket s_2\rrbracket)(\lambda(w,\kappa)\,.\,\llbracket s_3\rrbracket(\lambda(w_3,\kappa_3)\,.\,\varphi(w\uplus w_3,\kappa \mathbin{\widehat{\|}} \kappa_3)))\oplus \\
&\llbracket s_3\rrbracket(\lambda(w_3,\kappa_3)\,.\,(\llbracket s_1\rrbracket \,\|\, \llbracket s_2\rrbracket)(\lambda(w,\kappa)\,.\,\varphi(w_3\uplus w,\kappa_3 \mathbin{\widehat{\|}} \kappa))), \\
&\llbracket s_1\rrbracket(\lambda(w_1,\kappa_1)\,.\,\varphi(w_1,\kappa_1 \mathbin{\widehat{\|}} (\llbracket s_2\rrbracket \,\|\, \llbracket s_3\rrbracket)))\oplus \\
&(\llbracket s_2\rrbracket \,\|\, \llbracket s_3\rrbracket)(\lambda(w,\kappa)\,.\,\varphi(w,\kappa \mathbin{\widehat{\|}} \llbracket s_1\rrbracket))\oplus \\
&\llbracket s_1\rrbracket(\lambda(w_1,\kappa_1)\,.\,(\llbracket s_2\rrbracket \,\|\, \llbracket s_3\rrbracket)(\lambda(w,\kappa)\,.\,\varphi(w_1\uplus w,\kappa_1 \mathbin{\widehat{\|}} \kappa)))\oplus \\
&(\llbracket s_2\rrbracket \,\|\, \llbracket s_3\rrbracket)(\lambda(w,\kappa)\,.\,\llbracket s_1\rrbracket(\lambda(w_1,\kappa_1)\,.\,\varphi(w\uplus w_1,\kappa \mathbin{\widehat{\|}} \kappa_1))))
\end{aligned}
$$

[Lemma 4.6, $\uplus$ is commutative and associative]

$$
\begin{aligned}
&= \sup_{s_1,s_2,s_3\in Stat,\varphi\in\mathbf{F}} \\
&\quad d(\llbracket s_1\rrbracket(\lambda(w_1,\kappa_1)\,.\,\varphi(w_1,(\kappa_1 \mathbin{\widehat{\|}} \llbracket s_2\rrbracket) \mathbin{\widehat{\|}} \llbracket s_3\rrbracket))^{\$.1}\oplus \\
&\quad\quad \llbracket s_2\rrbracket(\lambda(w_2,\kappa_2)\,.\,\varphi(w_2,(\kappa_2 \mathbin{\widehat{\|}} \llbracket s_1\rrbracket) \mathbin{\widehat{\|}} \llbracket s_3\rrbracket))^{\$.2}\oplus \\
&\quad\quad \llbracket s_1\rrbracket(\lambda(w_1,\kappa_1)\,.\,\llbracket s_2\rrbracket(\lambda(w_2,\kappa_2)\,.\,\varphi(w_1\uplus w_2,(\kappa_1 \mathbin{\widehat{\|}} \kappa_2) \mathbin{\widehat{\|}} \llbracket s_3\rrbracket)))^{\$.3}\oplus \\
&\quad\quad \llbracket s_2\rrbracket(\lambda(w_2,\kappa_2)\,.\,\llbracket s_1\rrbracket(\lambda(w_1,\kappa_1)\,.\,\varphi(w_1\uplus w_2,(\kappa_2 \mathbin{\widehat{\|}} \kappa_1) \mathbin{\widehat{\|}} \llbracket s_3\rrbracket)))^{\$.4}\oplus \\
&\quad\quad \llbracket s_1\rrbracket(\lambda(w_1,\kappa_1)\,.\,\llbracket s_3\rrbracket(\lambda(w_3,\kappa_3)\,.\,\varphi(w_1\uplus w_3,(\kappa_1 \mathbin{\widehat{\|}} \llbracket s_2\rrbracket) \mathbin{\widehat{\|}} \kappa_3)))^{\$.5}\oplus \\
&\quad\quad \llbracket s_2\rrbracket(\lambda(w_2,\kappa_2)\,.\,\llbracket s_3\rrbracket(\lambda(w_3,\kappa_3)\,.\,\varphi(w_2\uplus w_3,(\kappa_2 \mathbin{\widehat{\|}} \llbracket s_1\rrbracket) \mathbin{\widehat{\|}} \kappa_3)))^{\$.6}\oplus \\
&\quad\quad \llbracket s_1\rrbracket(\lambda(w_1,\kappa_1)\,.\,\llbracket s_2\rrbracket(\lambda(w_2,\kappa_2)\,.\,\llbracket s_3\rrbracket(\lambda(w_3,\kappa_3)\,.\,\varphi(w_1\uplus w_2\uplus w_3,(\kappa_1 \mathbin{\widehat{\|}} \kappa_2) \mathbin{\widehat{\|}} \kappa_3))))^{\$.7}\oplus \\
&\quad\quad \llbracket s_2\rrbracket(\lambda(w_2,\kappa_2)\,.\,\llbracket s_1\rrbracket(\lambda(w_1,\kappa_1)\,.\,\llbracket s_3\rrbracket(\lambda(w_3,\kappa_3)\,.\,\varphi(w_1\uplus w_2\uplus w_3,(\kappa_2 \mathbin{\widehat{\|}} \kappa_1) \mathbin{\widehat{\|}} \kappa_3))))^{\$.8}\oplus \\
&\quad\quad \llbracket s_3\rrbracket(\lambda(w_3,\kappa_3)\,.\,\varphi(w_3,\kappa_3 \mathbin{\widehat{\|}} (\llbracket s_1\rrbracket \,\|\, \llbracket s_2\rrbracket)))^{\$.9}\oplus \\
&\quad\quad \llbracket s_3\rrbracket(\lambda(w_3,\kappa_3)\,.\,\llbracket s_1\rrbracket(\lambda(w_1,\kappa_1)\,.\,\varphi(w_1\uplus w_3,\kappa_3 \mathbin{\widehat{\|}} (\kappa_1 \mathbin{\widehat{\|}} \llbracket s_2\rrbracket))))^{\$.10}\oplus \\
&\quad\quad \llbracket s_3\rrbracket(\lambda(w_3,\kappa_3)\,.\,\llbracket s_2\rrbracket(\lambda(w_2,\kappa_2)\,.\,\varphi(w_2\uplus w_3,\kappa_3 \mathbin{\widehat{\|}} (\kappa_2 \mathbin{\widehat{\|}} \llbracket s_1\rrbracket))))^{\$.11}\oplus \\
&\quad\quad \llbracket s_3\rrbracket(\lambda(w_3,\kappa_3)\,.\,\llbracket s_1\rrbracket(\lambda(w_1,\kappa_1)\,.\,\llbracket s_2\rrbracket(\lambda(w_2,\kappa_2)\,.\,\varphi(w_1\uplus w_2\uplus w_3,\kappa_3 \mathbin{\widehat{\|}} (\kappa_1 \mathbin{\widehat{\|}} \kappa_2)))))^{\$.12}\oplus \\
&\quad\quad \llbracket s_3\rrbracket(\lambda(w_3,\kappa_3)\,.\,\llbracket s_2\rrbracket(\lambda(w_2,\kappa_2)\,.\,\llbracket s_1\rrbracket(\lambda(w_1,\kappa_1)\,.\,\varphi(w_1\uplus w_2\uplus w_3,\kappa_3 \mathbin{\widehat{\|}} (\kappa_2 \mathbin{\widehat{\|}} \kappa_1)))))^{\$.13}, \\
&\quad\quad \llbracket s_1\rrbracket(\lambda(w_1,\kappa_1)\,.\,\varphi(w_1,\kappa_1 \mathbin{\widehat{\|}} (\llbracket s_2\rrbracket \,\|\, \llbracket s_3\rrbracket)))^{*.1}\oplus \\
&\quad\quad \llbracket s_2\rrbracket(\lambda(w_2,\kappa_2)\,.\,\varphi(w_2,(\kappa_2 \mathbin{\widehat{\|}} \llbracket s_3\rrbracket) \mathbin{\widehat{\|}} \llbracket s_1\rrbracket))^{*.2}\oplus \\
&\quad\quad \llbracket s_3\rrbracket(\lambda(w_3,\kappa_3)\,.\,\varphi(w_3,(\kappa_3 \mathbin{\widehat{\|}} \llbracket s_2\rrbracket) \mathbin{\widehat{\|}} \llbracket s_1\rrbracket))^{*.3}\oplus \\
&\quad\quad \llbracket s_2\rrbracket(\lambda(w_2,\kappa_2)\,.\,\llbracket s_3\rrbracket(\lambda(w_3,\kappa_3)\,.\,\varphi(w_2\uplus w_3,(\kappa_2 \mathbin{\widehat{\|}} \kappa_3) \mathbin{\widehat{\|}} \llbracket s_1\rrbracket)))^{*.4}\oplus \\
&\quad\quad \llbracket s_3\rrbracket(\lambda(w_3,\kappa_3)\,.\,\llbracket s_2\rrbracket(\lambda(w_2,\kappa_2)\,.\,\varphi(w_2\uplus w_3,(\kappa_3 \mathbin{\widehat{\|}} \kappa_2) \mathbin{\widehat{\|}} \llbracket s_1\rrbracket)))^{*.5}\oplus \\
&\quad\quad \llbracket s_1\rrbracket(\lambda(w_1,\kappa_1)\,.\,\llbracket s_2\rrbracket(\lambda(w_2,\kappa_2)\,.\,\varphi(w_1\uplus w_2,\kappa_1 \mathbin{\widehat{\|}} (\kappa_2 \mathbin{\widehat{\|}} \llbracket s_3\rrbracket))))^{*.6}\oplus \\
&\quad\quad \llbracket s_1\rrbracket(\lambda(w_1,\kappa_1)\,.\,\llbracket s_3\rrbracket(\lambda(w_3,\kappa_3)\,.\,\varphi(w_1\uplus w_3,\kappa_1 \mathbin{\widehat{\|}} (\kappa_3 \mathbin{\widehat{\|}} \llbracket s_2\rrbracket))))^{*.7}\oplus
\end{aligned}
$$

$$[\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,[\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,[\![s_3]\!](\lambda(w_3,\kappa_3)\,.\,\varphi(w_1 \uplus w_2 \uplus w_3, \kappa_1 \widehat{\|} (\kappa_2 \widehat{\|} \kappa_3))))))^{*.8} \oplus$$

$$[\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,[\![s_3]\!](\lambda(w_3,\kappa_3)\,.\,[\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,\varphi(w_1 \uplus w_2 \uplus w_3, \kappa_1 \widehat{\|} (\kappa_3 \widehat{\|} \kappa_2))))))^{*.9} \oplus$$

$$[\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,[\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,\varphi(w_1 \uplus w_2, (\kappa_2 \widehat{\|} [\![s_3]\!]) \widehat{\|} \kappa_1)))^{*.10} \oplus$$

$$[\![s_3]\!](\lambda(w_3,\kappa_3)\,.\,[\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,\varphi(w_1 \uplus w_3, (\kappa_3 \widehat{\|} [\![s_2]\!]) \widehat{\|} \kappa_1)))^{*.11} \oplus$$

$$[\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,[\![s_3]\!](\lambda(w_3,\kappa_3)\,.\,[\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,\varphi(w_1 \uplus w_2 \uplus w_3, (\kappa_2 \widehat{\|} \kappa_3) \widehat{\|} \kappa_1))))^{*.12} \oplus$$

$$[\![s_3]\!](\lambda(w_3,\kappa_3)\,.\,[\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,[\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,\varphi(w_1 \uplus w_2 \uplus w_3, (\kappa_3 \widehat{\|} \kappa_2) \widehat{\|} \kappa_1))))^{*.13})$$

[$\oplus$ is nonexpansive, commutative, associative and idempotent]

$$\le max\{\sup_{s_1,s_2,s_3 \in Stat, \varphi \in \mathbf{F}} d(^{\$.1}, ^{*.1}), \sup_{s_1,s_2,s_3 \in Stat, \varphi \in \mathbf{F}} d(^{\$.2}, ^{*.2}),$$

$$\sup_{s_1,s_2,s_3 \in Stat, \varphi \in \mathbf{F}} d(^{\$.3}, ^{*.6}), \sup_{s_1,s_2,s_3 \in Stat, \varphi \in \mathbf{F}} d(^{\$.4}, ^{*.10}),$$

$$\sup_{s_1,s_2,s_3 \in Stat, \varphi \in \mathbf{F}} d(^{\$.5}, ^{*.7}), \sup_{s_1,s_2,s_3 \in Stat, \varphi \in \mathbf{F}} d(^{\$.6}, ^{*.4}),$$

$$\sup_{s_1,s_2,s_3 \in Stat, \varphi \in \mathbf{F}} d(^{\$.7}, ^{*.8}), \sup_{s_1,s_2,s_3 \in Stat, \varphi \in \mathbf{F}} d(^{\$.8}, ^{*.12}),$$

$$\sup_{s_1,s_2,s_3 \in Stat, \varphi \in \mathbf{F}} d(^{\$.9}, ^{*.3}), \sup_{s_1,s_2,s_3 \in Stat, \varphi \in \mathbf{F}} d(^{\$.10}, ^{*.11}),$$

$$\sup_{s_1,s_2,s_3 \in Stat, \varphi \in \mathbf{F}} d(^{\$.11}, ^{*.5}), \sup_{s_1,s_2,s_3 \in Stat, \varphi \in \mathbf{F}} d(^{\$.12}, ^{*.13}),$$

$$\sup_{s_1,s_2,s_3 \in Stat, \varphi \in \mathbf{F}} d(^{\$.13}, ^{*.13}), \sup_{s_1,s_2,s_3 \in Stat, \varphi \in \mathbf{F}} d(^{\$.7}, ^{*.9}),$$

We treat two typical cases. First, we show that $\sup_{s_1,s_2,s_3 \in Stat, \varphi \in \mathbf{F}} d(^{\$.1}, ^{*.1}) \le \frac{1}{2} \cdot \varepsilon$. Indeed:

$$\sup_{s_1,s_2,s_3 \in Stat, \varphi \in \mathbf{F}} d(^{\$.1}, ^{*.1})$$

$$= \sup_{s_1,s_2,s_3 \in Stat, \varphi \in \mathbf{F}} d([\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,\varphi(w_1, (\kappa_1 \widehat{\|} [\![s_2]\!]) \widehat{\|} [\![s_3]\!]),$$

$$[\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,\varphi(w_1, \kappa_1 \widehat{\|} ([\![s_2]\!] \parallel [\![s_3]\!])))))$$

[Lemma 4.7]

$$\le \sup_{s_2,s_3 \in Stat, \varphi \in \mathbf{F}} \sup_{w \in W, \kappa \in \mathbf{K}_D} d(\varphi(w, (\kappa \widehat{\|} [\![s_2]\!]) \widehat{\|} [\![s_3]\!]), \varphi(w, \kappa \widehat{\|} ([\![s_2]\!] \parallel [\![s_3]\!])))$$

[φ is nonexpansive]

$$\le \sup_{s_2,s_3 \in Stat, \kappa \in \mathbf{K}_D} d_{\mathbf{K}}((\kappa \widehat{\|} [\![s_2]\!]) \widehat{\|} [\![s_3]\!], \kappa \widehat{\|} ([\![s_2]\!] \parallel [\![s_3]\!]))$$

[$\mathbf{K}_D = \{\kappa_0\} \cup \{[\![s']\!] \mid s' \in Stat\}$]

$$\le max\{\sup_{s_2,s_3 \in Stat} d_{\mathbf{K}}((\kappa_0 \widehat{\|} [\![s_2]\!]) \widehat{\|} [\![s_3]\!], \kappa_0 \widehat{\|} ([\![s_2]\!] \parallel [\![s_3]\!])),$$

$$\sup_{s',s_2,s_3 \in Stat} d_{\mathbf{K}}(([\![s']\!] \widehat{\|} [\![s_2]\!]) \widehat{\|} [\![s_3]\!], [\![s']\!] \widehat{\|} ([\![s_2]\!] \parallel [\![s_3]\!]))\}$$

[$\mathbf{K} = \{\kappa_0\} + (\frac{1}{2} \cdot \mathbf{D})$]

$$\le max\{\sup_{s_2,s_3 \in Stat} d_{\frac{1}{2} \cdot \mathbf{D}}([\![s_2]\!] \parallel [\![s_3]\!], [\![s_2]\!] \parallel [\![s_3]\!]),$$

$$\sup_{s',s_2,s_3 \in Stat} d_{\frac{1}{2} \cdot \mathbf{D}}(([\![s']\!] \parallel [\![s_2]\!]) \parallel [\![s_3]\!], [\![s']\!] \parallel ([\![s_2]\!] \parallel [\![s_3]\!]))\}$$

$$= \tfrac{1}{2} \cdot \sup_{s',s_2,s_3 \in Stat} d_{\mathbf{D}}(([\![s']\!] \parallel [\![s_2]\!]) \parallel [\![s_3]\!], [\![s']\!] \parallel ([\![s_2]\!] \parallel [\![s_3]\!]))$$

$$\le \tfrac{1}{2} \cdot \varepsilon$$

We also show that $\sup_{s_1,s_2,s_3\in Stat,\varphi\in\mathbf{F}} d(^{\$.8},^{*.12}) \le \frac{1}{2}\cdot\varepsilon$.

$$\sup_{s_1,s_2,s_3\in Stat,\varphi\in\mathbf{F}} d(^{\$.8},^{*.12})$$

$$= \sup_{s_1,s_2,s_3\in Stat,\varphi\in\mathbf{F}}$$

$$d([\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,[\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,[\![s_3]\!](\lambda(w_3,\kappa_3)\,.\,\varphi(w_1\uplus w_2\uplus w_3,(\kappa_2\,\widehat{\|}\,\kappa_1)\,\widehat{\|}\,\kappa_3)))),$$

$$[\![s_2]\!](\lambda(w_2,\kappa_2)\,.\,[\![s_3]\!](\lambda(w_3,\kappa_3)\,.\,[\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,\varphi(w_1\uplus w_2\uplus w_3,(\kappa_2\,\widehat{\|}\,\kappa_3)\,\widehat{\|}\,\kappa_1)))))$$

[Lemma 4.7]

$$\le \sup_{s_1,s_2,s_3\in Stat,\varphi\in\mathbf{F},w\in W,\kappa\in\mathbf{K}_D}$$

$$d([\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,[\![s_3]\!](\lambda(w_3,\kappa_3)\,.\,\varphi(w_1\uplus w\uplus w_3,(\kappa\,\widehat{\|}\,\kappa_1)\,\widehat{\|}\,\kappa_3))),$$

$$[\![s_3]\!](\lambda(w_3,\kappa_3)\,.\,[\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,\varphi(w_1\uplus w\uplus w_3,(\kappa\,\widehat{\|}\,\kappa_3)\,\widehat{\|}\,\kappa_1))))$$

[d is an ultrametric]

$$\le \sup_{s_1,s_2,s_3\in Stat,\varphi\in\mathbf{F},w\in W,\kappa\in\mathbf{K}_D}$$

$$max\{d([\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,[\![s_3]\!](\lambda(w_3,\kappa_3)\,.\,\varphi(w_1\uplus w\uplus w_3,(\kappa\,\widehat{\|}\,\kappa_1)\,\widehat{\|}\,\kappa_3))),$$

$$[\![s_3]\!](\lambda(w_3,\kappa_3)\,.\,[\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,\varphi(w_1\uplus w\uplus w_3,(\kappa\,\widehat{\|}\,\kappa_1)\,\widehat{\|}\,\kappa_3))))^{\$\$.1},$$

$$d([\![s_3]\!](\lambda(w_3,\kappa_3)\,.\,[\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,\varphi(w_1\uplus w\uplus w_3,(\kappa\,\widehat{\|}\,\kappa_1)\,\widehat{\|}\,\kappa_3))),$$

$$[\![s_3]\!](\lambda(w_3,\kappa_3)\,.\,[\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,\varphi(w_1\uplus w\uplus w_3,(\kappa\,\widehat{\|}\,\kappa_3)\,\widehat{\|}\,\kappa_1))))^{**.1}\}$$

Let $\mathsf{op}_K\,'' : \mathbf{K}\xrightarrow{1}((\mathbf{K}\times\mathbf{K})\xrightarrow{1}\mathbf{K})$, be given by $\mathsf{op}_K\,''(\kappa)(\kappa_1,\kappa_3) = (\kappa\,\widehat{\|}\,\kappa_1)\,\widehat{\|}\,\kappa_3$. It is easy to see that $\mathsf{op}_K\,''$ is well-defined (nonexpansive in its arguments). Let also $\varphi''\in\mathbf{F}$, $\varphi''(w',\kappa') = \varphi(w\uplus w',\kappa')$. Note that $\varphi''(w_1\uplus w_3,\ \mathsf{op}_K\,''(\kappa)(\kappa_1,\kappa_3)) = \varphi(w_1\uplus w\uplus w_3,(\kappa\,\widehat{\|}\,\kappa_1)\,\widehat{\|}\,\kappa_3)$. We have:

$$\sup_{s_1,s_2,s_3\in Stat,\varphi\in\mathbf{F},w\in W,\kappa\in\mathbf{K}_D} max\{^{\$\$.1},^{**.1}\}$$

[$\widehat{\|}$ is commutative, Remark 4.3]

$$\le \sup_{s_1,s_2,s_3\in Stat,\varphi\in\mathbf{F},w\in W,\kappa\in\mathbf{K}_D}$$

$$max\{d([\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,[\![s_3]\!](\lambda(w_3,\kappa_3)\,.\,\varphi''(w_1\uplus w_3,\ \mathsf{op}_K\,''(\kappa)(\kappa_1,\kappa_3)))),$$

$$[\![s_3]\!](\lambda(w_3,\kappa_3)\,.\,[\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,\varphi''(w_1\uplus w_3,\ \mathsf{op}_K\,''(\kappa)(\kappa_1,\kappa_3)))))^{\$\$.2},$$

$$d([\![s_3]\!](\lambda(w_3,\kappa_3)\,.\,[\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,\varphi(w_1\uplus w\uplus w_3,(\kappa_1\,\widehat{\|}\,\kappa)\,\widehat{\|}\,\kappa_3))),$$

$$[\![s_3]\!](\lambda(w_3,\kappa_3)\,.\,[\![s_1]\!](\lambda(w_1,\kappa_1)\,.\,\varphi(w_1\uplus w\uplus w_3,\kappa_1\,\widehat{\|}\,(\kappa\,\widehat{\|}\,\kappa_3)))))^{**.2}\}$$

Notice that $^{\$\$.2} = 0$, by Lemma 4.8. Also, by using Lemma 4.7 we obtain:

$$^{**.2} \le \sup_{s_1,s_2,s_3\in Stat,\varphi\in\mathbf{F},w,w',w''\in W,\kappa,\kappa',\kappa''\in\mathbf{K}_D} d(\varphi(w\uplus w'\uplus w'',(\kappa''\,\widehat{\|}\,\kappa)\,\widehat{\|}\,\kappa'),$$

$$\varphi(w\uplus w'\uplus w'',\kappa''\,\widehat{\|}\,(\kappa\,\widehat{\|}\,\kappa')))$$

[φ is nonexpansive, $\varphi\in\mathbf{F}$]

$$\le \sup_{\kappa,\kappa',\kappa''\in\mathbf{K}_D} d_{\mathbf{K}}((\kappa''\,\widehat{\|}\,\kappa)\,\widehat{\|}\,\kappa',\kappa''\,\widehat{\|}\,(\kappa\,\widehat{\|}\,\kappa'))^{**.3}$$

According to the definition of $\widehat{\|}$, if $\kappa = \kappa_0$ or $\kappa' = \kappa_0$ or $\kappa'' = \kappa_0$ then ${}^{**.3} = 0$. Otherwise, recall that $\mathbf{K}_D = \{\kappa_0\} \cup \{[\![s']\!] \mid s' \in \mathit{Stat}\}$ (see Lemma 4.7). Hence we have:

$$
\begin{aligned}
&{}^{**.3} \le \sup_{s,s',s'' \in \mathit{Stat}} d_{\frac{1}{2}\cdot\mathbf{D}}(([\![s'']\!] \,\widehat{\|}\, [\![s]\!]) \,\widehat{\|}\, [\![s']\!], [\![s'']\!] \,\widehat{\|}\, ([\![s]\!] \,\widehat{\|}\, [\![s']\!]))\\
&\le \tfrac{1}{2} \cdot \sup_{s,s',s'' \in \mathit{Stat}} d_{\mathbf{D}}(([\![s'']\!] \,\widehat{\|}\, [\![s]\!]) \,\widehat{\|}\, [\![s']\!], [\![s'']\!] \,\widehat{\|}\, ([\![s]\!] \,\widehat{\|}\, [\![s']\!]))\\
&\le \tfrac{1}{2} \cdot \varepsilon
\end{aligned}
$$

Therefore, we have $\quad \sup_{s_1,s_2,s_3 \in \mathit{Stat},\varphi \in \mathbf{F}} d({}^{\$.8}, {}^{*.12}) \le \frac{1}{2} \cdot \varepsilon$.

The other cases can be handled similarly, and we obtain $\varepsilon \le \frac{1}{2} \cdot \varepsilon$, which imples $\varepsilon = 0$. Therefore we obtain $[\![(s_1 \| s_2) \| s_3]\!] = [\![s_1 \| (s_2 \| s_3)]\!]$, for any $s_1, s_2, s_3 \in \mathit{Stat}$, which implies the property stated by Proposition 4.17(1), namely $(s_1 \| s_2) \| s_3 \simeq s_1 \| (s_2 \| s_3)$, as required.

We continue with Proposition 4.17(2). We have

$$
\begin{aligned}
&[\![(s_1 \lfloor s_2) \lfloor s_3]\!] = \lambda\varphi \,.\, [\![(s_1 \lfloor s_2) \lfloor s_3]\!](\varphi)\\
&= \lambda\varphi \,.\, [\![s_1 \lfloor s_2]\!](\lambda(w,\kappa) \,.\, [\![s_3]\!](\lambda(w_3,\kappa_3) \,.\, \varphi(w \uplus w_3, \kappa \,\widehat{\|}\, \kappa_3))) \qquad [\uplus \text{ is associative}]\\
&= \lambda\varphi \,.\, [\![s_1]\!](\lambda(w_1,\kappa_1) \,.\, [\![s_2]\!](\lambda(w_2,\kappa_2) \,.\, [\![s_3]\!](\lambda(w_3,\kappa_3) \,.\, \varphi(w_1 \uplus w_2 \uplus w_3, (\kappa_1 \,\widehat{\|}\, \kappa_2) \,\widehat{\|}\, \kappa_3))))^{\#\#.1}
\end{aligned}
$$

Similarly:

$$
\begin{aligned}
&[\![s_1 \lfloor (s_2 \lfloor s_3)]\!] = \lambda\varphi \,.\, [\![s_1 \lfloor (s_2 \lfloor s_3)]\!](\varphi)\\
&= \lambda\varphi \,.\, [\![s_1]\!](\lambda(w_1,\kappa_1) \,.\, [\![s_2 \lfloor s_3]\!](\lambda(w,\kappa) \,.\, \varphi(w_1 \uplus w, \kappa_1 \,\widehat{\|}\, \kappa)))\\
&= \lambda\varphi \,.\, [\![s_1]\!](\lambda(w_1,\kappa_1) \,.\, [\![s_2]\!](\lambda(w_2,\kappa_2) \,.\, [\![s_3]\!](\lambda(w_3,\kappa_3) \,.\, \varphi(w_1 \uplus w_2 \uplus w_3, \kappa_1 \,\widehat{\|}\, (\kappa_2 \,\widehat{\|}\, \kappa_3)))))^{\#\#.2}
\end{aligned}
$$

In the sequel we compute as follows

$$
\begin{aligned}
&\sup_{s_1,s_2,s_3 \in \mathit{Stat}} d([\![(s_1 \lfloor s_2) \lfloor s_3]\!], [\![s_1 \lfloor (s_2 \lfloor s_3)]\!])\\
&= \sup_{s_1,s_2,s_3 \in \mathit{Stat}} d({}^{\#\#.1}, {}^{\#\#.2})\\
&= \sup_{s_1,s_2,s_3 \in \mathit{Stat},\varphi \in \mathbf{F}}\\
&\qquad d([\![s_1]\!](\lambda(w_1,\kappa_1) \,.\, [\![s_2]\!](\lambda(w_2,\kappa_2) \,.\, [\![s_3]\!](\lambda(w_3,\kappa_3) \,.\, \varphi(w_1 \uplus w_2 \uplus w_3, (\kappa_1 \,\widehat{\|}\, \kappa_2) \,\widehat{\|}\, \kappa_3)))),\\
&\qquad\quad [\![s_1]\!](\lambda(w_1,\kappa_1) \,.\, [\![s_2]\!](\lambda(w_2,\kappa_2) \,.\, [\![s_3]\!](\lambda(w_3,\kappa_3) \,.\, \varphi(w_1 \uplus w_2 \uplus w_3, \kappa_1 \,\widehat{\|}\, (\kappa_2 \,\widehat{\|}\, \kappa_3))))))\\
&\quad [\text{Lemma 4.7}]\\
&\le \sup_{s_2,s_3 \in \mathit{Stat},\varphi \in \mathbf{F},w_1 \in W,\kappa_1 \in \mathbf{K}_D}\\
&\qquad d([\![s_2]\!](\lambda(w_2,\kappa_2) \,.\, [\![s_3]\!](\lambda(w_3,\kappa_3) \,.\, \varphi(w_1 \uplus w_2 \uplus w_3, (\kappa_1 \,\widehat{\|}\, \kappa_2) \,\widehat{\|}\, \kappa_3))),\\
&\qquad\quad [\![s_2]\!](\lambda(w_2,\kappa_2) \,.\, [\![s_3]\!](\lambda(w_3,\kappa_3) \,.\, \varphi(w_1 \uplus w_2 \uplus w_3, \kappa_1 \,\widehat{\|}\, (\kappa_2 \,\widehat{\|}\, \kappa_3))))) \quad [\text{Lemma 4.7}]\\
&\le \sup_{s_3 \in \mathit{Stat},\varphi \in \mathbf{F},w_1,w_2 \in W,\kappa_1,\kappa_2 \in \mathbf{K}_D}\\
&\qquad d([\![s_3]\!](\lambda(w_3,\kappa_3) \,.\, \varphi(w_1 \uplus w_2 \uplus w_3, (\kappa_1 \,\widehat{\|}\, \kappa_2) \,\widehat{\|}\, \kappa_3)),\\
&\qquad\quad [\![s_3]\!](\lambda(w_3,\kappa_3) \,.\, \varphi(w_1 \uplus w_2 \uplus w_3, \kappa_1 \,\widehat{\|}\, (\kappa_2 \,\widehat{\|}\, \kappa_3)))) \quad [\text{Lemma 4.7}]\\
&\le \sup_{\varphi \in \mathbf{F},w_1,w_2,w_3 \in W,\kappa_1,\kappa_2,\kappa_3 \in \mathbf{K}_D}\\
&\qquad d(\varphi(w_1 \uplus w_2 \uplus w_3, (\kappa_1 \,\widehat{\|}\, \kappa_2) \,\widehat{\|}\, \kappa_3), \varphi(w_1 \uplus w_2 \uplus w_3, \kappa_1 \,\widehat{\|}\, (\kappa_2 \,\widehat{\|}\, \kappa_3))) \quad [\varphi \text{ is nonexpansive}]
\end{aligned}
$$

$$\leq \sup_{\kappa_1,\kappa_2,\kappa_3 \in \mathbf{K}_D} d((\kappa_1 \mathbin{\widehat{\|}} \kappa_2) \mathbin{\widehat{\|}} \kappa_3, \kappa_1 \mathbin{\widehat{\|}} (\kappa_2 \mathbin{\widehat{\|}} \kappa_3)) \quad \text{[Proposition 4.17(1)]}$$

$$= 0$$

We recall that $\mathbf{K}_D = \{\kappa_0\} \cup \{\llbracket s' \rrbracket \mid s' \in \mathit{Stat}\}$, see Lemma 4.7. We obtain $\llbracket (s_1 \lfloor s_2) \lfloor s_3 \rrbracket = \llbracket s_1 \lfloor (s_2 \lfloor s_3) \rrbracket$, for any $s_1, s_2, s_3 \in \mathit{Stat}$. Hence, we infer immediately the property stated by Proposition 4.17(2), namely $(s_1 \lfloor s_2) \lfloor s_3 \simeq s_1 \lfloor (s_2 \lfloor s_3)$, as required.

Proposition 4.17(3) follows without difficulty by using Proposition 4.17(2) and Lemma 4.9(a). By using Lemma 4.9(a), it is easy to show that $\llbracket s_1 \mid (s_2 \mid s_3) \rrbracket = \llbracket s_1 \lfloor (s_2 \lfloor s_3) \rrbracket$ and $\llbracket (s_1 \mid s_2) \mid s_3 \rrbracket = \llbracket (s_1 \lfloor s_2) \lfloor s_3 \rrbracket$. Hence we obtain

$$\llbracket s_1 \mid (s_2 \mid s_3) \rrbracket = \llbracket s_1 \lfloor (s_2 \lfloor s_3) \rrbracket \quad \text{[Proposition 4.17(2)]}$$

$$= \llbracket (s_1 \lfloor s_2) \lfloor s_3 \rrbracket = \llbracket (s_1 \mid s_2) \mid s_3 \rrbracket$$

which implies $(s_1 \mid s_2) \mid s_3 \simeq s_1 \mid (s_2 \mid s_3)$, as required.

Next, we handle Proposition 4.17(4). We have

$$\llbracket (s_1 \mathbin{\lfloor\!\lfloor} s_2) \mathbin{\lfloor\!\lfloor} s_3 \rrbracket$$

$$= \lambda\varphi \,.\, \llbracket s_1 \mathbin{\lfloor\!\lfloor} s_2 \rrbracket(\lambda(w,\kappa) \,.\, \varphi(w, \kappa \mathbin{\widehat{\|}} \llbracket s_3 \rrbracket))$$

$$= \lambda\varphi \,.\, \llbracket s_1 \rrbracket(\lambda(w_1,\kappa_1) \,.\, \varphi(w_1, (\kappa_1 \mathbin{\widehat{\|}} \llbracket s_2 \rrbracket) \mathbin{\widehat{\|}} \llbracket s_3 \rrbracket))^{\#\#.3}$$

and

$$\llbracket s_1 \mathbin{\lfloor\!\lfloor} (s_2 \mathbin{\lfloor\!\lfloor} s_3) \rrbracket$$

$$= \lambda\varphi \,.\, \llbracket s_1 \rrbracket(\lambda(w_1,\kappa_1) \,.\, \varphi(w_1, \kappa_1 \mathbin{\widehat{\|}} \llbracket s_2 \parallel s_3 \rrbracket))^{\#\#.4}$$

Therefore

$$\sup_{s_1,s_2,s_3 \in \mathit{Stat}} d(\llbracket (s_1 \mathbin{\lfloor\!\lfloor} s_2) \mathbin{\lfloor\!\lfloor} s_3 \rrbracket, \llbracket s_1 \mathbin{\lfloor\!\lfloor} (s_2 \parallel s_3) \rrbracket)$$

$$= \sup_{s_1,s_2,s_3 \in \mathit{Stat}} d(^{\#\#.3}, ^{\#\#.4}) \quad \text{[Lemma 4.7]}$$

$$\leq \sup_{s_2,s_3 \in \mathit{Stat}, \varphi \in \mathbf{F}, w_1 \in W, \kappa_1 \in \mathbf{K}_D} d(\varphi(w_1, (\kappa_1 \mathbin{\widehat{\|}} \llbracket s_2 \rrbracket) \mathbin{\widehat{\|}} \llbracket s_3 \rrbracket), \varphi(w_1, \kappa_1 \mathbin{\widehat{\|}} \llbracket s_2 \parallel s_3 \rrbracket))$$

$$\text{[}\varphi \text{ is nonexpansive]}$$

$$\leq \sup_{s_2,s_3 \in \mathit{Stat}, \kappa_1 \in \mathbf{K}_D} d((\kappa_1 \mathbin{\widehat{\|}} \llbracket s_2 \rrbracket) \mathbin{\widehat{\|}} \llbracket s_3 \rrbracket, \kappa_1 \mathbin{\widehat{\|}} (\llbracket s_2 \rrbracket \parallel \llbracket s_3 \rrbracket)) \quad \text{[Proposition 4.17(1)]}$$

$$= 0$$

Therefore $\llbracket (s_1 \mathbin{\lfloor\!\lfloor} s_2) \mathbin{\lfloor\!\lfloor} s_3 \rrbracket = \llbracket s_1 \mathbin{\lfloor\!\lfloor} (s_2 \parallel s_3) \rrbracket$, for any $s_1, s_2, s_3 \in \mathit{Stat}$, which implies the desired property stated by Proposition 4.17(4), namely $(s_1 \mathbin{\lfloor\!\lfloor} s_2) \mathbin{\lfloor\!\lfloor} s_3 \simeq s_1 \mathbin{\lfloor\!\lfloor} (s_2 \parallel s_3)$.

Finally, we prove Proposition 4.17(5). We have:

$$\llbracket s_1 \mid (s_2 \mathbin{\lfloor\!\lfloor} s_3) \rrbracket \quad \text{[Lemma 4.9(a)]}$$

$$= \lambda\varphi \,.\, \llbracket s_1 \lfloor (s_2 \mathbin{\lfloor\!\lfloor} s_3) \rrbracket(\varphi)$$

$$= \lambda\varphi \,.\, \llbracket s_1 \rrbracket(\lambda(w_1,\kappa_1) \,.\, \llbracket s_2 \mathbin{\lfloor\!\lfloor} s_3 \rrbracket(\lambda(w,\kappa) \,.\, \varphi(w_1 \uplus w, \kappa_1 \mathbin{\widehat{\|}} \kappa)))$$

$$= \lambda\varphi \,.\, \llbracket s_1 \rrbracket(\lambda(w_1,\kappa_1) \,.\, \llbracket s_2 \rrbracket(\lambda(w_2,\kappa_2) \,.\, \varphi(w_1 \uplus w_2, \kappa_1 \mathbin{\widehat{\|}} (\kappa_2 \mathbin{\widehat{\|}} \llbracket s_3 \rrbracket)))))^{\#\#.5}$$

and

$$\llbracket (s_1 \mid s_2) \mathbin{\lfloor\!\lfloor} s_3 \rrbracket = \lambda\varphi \,.\, \llbracket (s_1 \mid s_2) \mathbin{\lfloor\!\lfloor} s_3 \rrbracket(\varphi)$$

$$= \lambda\varphi \,.\, [\![s_1 \mid s_2]\!](\lambda(w,\kappa) \,.\, \varphi(w, \kappa \mathbin{\widehat{\|}} [\![s_3]\!])) \quad \text{[Lemma 4.9(a)]}$$
$$= \lambda\varphi \,.\, [\![s_1 \lfloor s_2]\!](\lambda(w,\kappa) \,.\, \varphi(w, \kappa \mathbin{\widehat{\|}} [\![s_3]\!]))$$
$$= \lambda\varphi \,.\, [\![s_1]\!](\lambda(w_1,\kappa_1) \,.\, [\![s_2]\!](\lambda(w_2,\kappa_2) \,.\, \varphi(w_1 \uplus w_2, (\kappa_1 \mathbin{\widehat{\|}} \kappa_2) \mathbin{\widehat{\|}} [\![s_3]\!])))^{\#\#.6}$$

Therefore

$$\sup_{s_1,s_2,s_3 \in Stat} d([\![s_1 \mid (s_2 \lfloor\!\lfloor s_3)]\!], [\![(s_1 \mid s_2) \lfloor\!\lfloor s_3]\!])$$
$$= \sup_{s_1,s_2,s_3 \in Stat} d(^{\#\#.5}, ^{\#\#.6}) \quad \text{[Lemma 4.7]}$$
$$\leq \sup_{s_3 \in Stat, \varphi \in \mathbf{F}, w_1,w_2 \in W, \kappa_1,\kappa_2 \in \mathbf{K}_D} d(\varphi(w_1 \uplus w_2, \kappa_1 \mathbin{\widehat{\|}} (\kappa_2 \mathbin{\widehat{\|}} [\![s_3]\!])), \varphi(w_1 \uplus w_2, (\kappa_1 \mathbin{\widehat{\|}} \kappa_2) \mathbin{\widehat{\|}} [\![s_3]\!]))$$
$$\text{[}\varphi \text{ is nonexpansive]}$$
$$= \sup_{s_3 \in Stat, \kappa_1,\kappa_2 \in \mathbf{K}_D} d(\kappa_1 \mathbin{\widehat{\|}} (\kappa_2 \mathbin{\widehat{\|}} [\![s_3]\!]), (\kappa_1 \mathbin{\widehat{\|}} \kappa_2) \mathbin{\widehat{\|}} [\![s_3]\!]) \quad \text{[Proposition 4.17(1)]}$$
$$= 0$$

Hence $[\![s_1 \mid (s_2 \lfloor\!\lfloor s_3)]\!] = [\![(s_1 \mid s_2) \lfloor\!\lfloor s_3]\!]$, for any $s_1, s_2, s_3 \in Stat$, and thus $s_1 \mid (s_2 \lfloor\!\lfloor s_3) \simeq (s_1 \mid s_2) \lfloor\!\lfloor s_3$, as required. □

The behavior of a computation $[\![s]\!](\varphi)$ cannot be characterized solely by the set of laws stated in Theorem 4.15 and Proposition 4.17, which are all independent of the continuation parameter φ. In general, the semantics of interaction depends on the continuation φ. All successful interactions, and all deadlock situations, can be modeled based on the interaction multiset that is transmitted as a parameter to a continuation. In the following sections we present denotational semantics for a couple of languages obtained by providing concrete instantiations of the continuation parameter, which in turn depends on the mapping $interact$, defining the behavior of interaction multisets.

Remark 4.18 *There are benefits resulting from the use of continuations in concurrency semantics. No branching structure is needed. The final yield of the denotational semantics can be a simple linear time domain. In continuation semantics one can design denotational models for concurrency that produce exactly the desired observables assembled in linear models, rather than complex branching structures incorporating communication attempts and silent steps as in classic resumption-based models [30, 5]. A couple of examples are provided in subsequent sections. We can also mention some of our previous works [10, 40]. The terminology "linear time" versus "branching time" is often used in temporal logic and denotational semantics (see, e.g., [5]). Intuitively, an element of a linear time domain is a collection of traces. An element of a branching time domain is a tree-like structure whose nodes represent nondeterministic choice points.*

However, in continuation semantics it may be more difficult to design fully abstract denotational models of concurrent languages. The correctness condition of full abstractness raises no special difficulty [39]. However, in continuation semantics some properties can be established only for computations denotable by program statements; see Remark 5.6. Hence, it may be difficult to achieve the completeness condition of full abstractness. The full abstractness problem was raised by Milner [25].

5 Continuation Semantics for Concurrency and Communication

In this section we study languages where the semantics of elementary statements is defined based on a set $(\sigma \in)\Sigma$ of *states*. The final semantic domain will be $(r \in)\mathbf{R} = \Sigma \to \mathbf{P}$, where $(p \in)\mathbf{P}$ is a linear time domain. The domain $\mathbf{P}$ is defined as the (unique) solution of the domain equation given below, where set Σ is endowed with the discrete metric, which is an ultrametric. The solution for $\mathbf{P}$ is a complete ultrametric space [5].

$$(p \in)\mathbf{P} \;=\; \mathcal{P}_{nco}(\mathbf{Q})$$
$$(q \in)\mathbf{Q} \;\cong\; \{\epsilon\} + \{\delta\} + (\Sigma \times \frac{1}{2} \cdot \mathbf{Q})$$

ϵ models the empty sequence. δ is used to model *deadlock*. Instead of $(\sigma_1, (\sigma_2, \ldots (\sigma_n, \epsilon) \ldots))$, $(\sigma_1, (\sigma_2, \ldots (\sigma_n, \delta) \ldots))$ and $(\sigma_1, (\sigma_2, \ldots))$, we write $\sigma_1\sigma_2 \ldots \sigma_n$, $\sigma_1\sigma_2 \ldots \sigma_n\delta$ and $\sigma_1\sigma_2 \ldots$, respectively. We use the following notations: $\sigma \cdot q = (\sigma, q)$ and $\sigma \cdot p = \{\sigma \cdot q \mid q \in p\}$, for any $\sigma \in \Sigma, p \in \mathbf{P}$. Note that $d(\sigma \cdot p_1, \sigma \cdot p_2) = \frac{1}{2} \cdot d(p_1, p_2)$ [5].

We introduce an operator $+ : \mathbf{P} \times \mathbf{P} \xrightarrow{1} \mathbf{P}$, $p_1 + p_2 = \{q \mid q \in p_1 \cup p_2, q \neq \delta\} \cup \{\delta \mid \delta \in p_1 \cap p_2\}$. This definition expresses that $p_1 + p_2$ blocks only if both p_1 and p_2 block. Operator $+$ is well-defined, nonexpansive, associative, commutative and idempotent [5]. The operator for nondeterministic choice $\oplus : \mathbf{R} \times \mathbf{R} \xrightarrow{1} \mathbf{R}$ is defined as: $(r_1 \oplus r_2) = \lambda\sigma \,.\, (r_1(\sigma) + r_2(\sigma))$. It is easy to check that $\oplus$ is also well-defined, nonexpansive, associative, commutative and idempotent. We also use the notation: $\sigma' \cdot r = \lambda\sigma \,.\, \{\sigma' \cdot q \mid q \in r(\sigma)\}$ $(\in \mathbf{R})$, for any $\sigma' \in \Sigma$, $r \in \mathbf{R}$.

5.1 Multiple Channels Communication

We consider a CSP-like language $\mathcal{L}_{MCC}$ extended with a mechanism of communication and synchronization on multiple channels inspired by the Join calculus [18]. $\mathcal{L}_{MCC}$ was first investigated in [10] under the name *MCC*, an abbreviation for *Multiple Channels Communication*.[6] $\mathcal{L}_{MCC}$ provides the same operators as $\mathcal{L}$ and two primitives for concurrent interaction on multiple channels, namely $c!e$ and a communication pattern $(c_1?v_1\& \cdots \&c_n?v_n)$. Synchronized execution of $n+1$ statements $c_1!e_1, \ldots, c_n!e_n$ and $(c_1?v_1\& \cdots \&c_n?v_n)$ occurring in parallel processes results in the transmission of the value of each expression e_i along the channel c_i from the process executing the $c_i!e_i$ statement to the process executing the statement $(c_1?v_1\& \cdots \&c_n?v_n)$. The value of each expression e_i is transmitted along the channel c_i and assigned to the corresponding variable v_i. The whole interaction behaves like a distributed multi-assignment. When $n = 1$, the interaction is a point-to-point communication like in CSP [21]. $\mathcal{L}_{MCC}$ also provides a multi-assignment statement $(v_1, \cdots, v_n := e_1, \cdots, e_n)$, which allows several variables to be assigned in parallel, e.g., as in Python. $\mathcal{L}_{MCC}$ evaluates all the expressions $e_1, \ldots, e_n$ in the right-hand side of the multi-assignment, then it matches the values of the expressions with corresponding variables in the left-hand side. The value of each expression e_i is assigned to the corresponding variable v_i.

We assume as given a set $(v \in)V$ of *variables*, a set $(c \in)Ch$ of *communication channels* and a set $(e \in)Exp$ of *expressions* (without side effects).

Definition 5.1 *(Syntax of $\mathcal{L}_{MCC}$) The syntax of $\mathcal{L}_{MCC}$ comprises the following components:*

(a) (Communication patterns) $j(\in J) ::= c?e \mid j\&j$

(b) (Elementary statements) $a(\in A) ::= \mathsf{stop} \mid (v_1, \cdots, v_n := e_1, \cdots, e_n) \mid c!e \mid j$

(c) (Statements) $s(\in Stat) ::= a \mid y \mid s;s \mid s+s \mid s \parallel s \mid s|s \mid s \mathbin{\lfloor\!\lfloor} s \mid s \lfloor s$

To be valid, the channels $c_1, \ldots, c_n$ and the variables $v_1, \ldots, v_n$ of a communication pattern $j = (c_1?v_1\& \cdots \&c_n?v_n)$ must be pairwise distinct. Also, the variables $v_1, \ldots, v_n$ of a multi-assignment $(v_1, \cdots, v_n := e_1, \cdots, e_n)$, must be pairwise distinct, and the lists of variables $v_1, \ldots, v_n$ and expressions $e_1, \ldots, e_n$ must have the same length. The sets of guarded statements, declarations, and programs for $\mathcal{L}_{MCC}$ are defined as in Definition 3.1.

We obtain a denotational semantics for $\mathcal{L}_{MCC}$ by customizing certain components of the semantic model introduced in Section 4. The evaluation of expressions is modeled by a given valuation $\mathcal{E}[\![\cdot]\!] : Exp \to \Sigma \to Val$, where $(\xi \in) Val$ is some set of *values* and $(\sigma \in)\Sigma = V \to Val$ is the set of *states*.

The set of Act actions for $\mathcal{L}_{MCC}$ is $Act = \{\, \mathsf{stop} \,\} \cup (V \times \Xi)^+ \cup (Ch \times \Xi) \cup (Ch \times V)^+$, where $(\xi \in)\Xi = \Sigma \to Val$. $(V \times \Xi)^+$ $((Ch \times V)^+)$ is the set of all finite and non-empty sequences over $V \times \Xi$ $(Ch \times V)$. For easier readability, we denote typical elements $(v_1, \xi_1) \cdots (v_n, \xi_n) \in (V \times \Xi)^+$, $(c, \xi) \in (Ch \times \Xi)$, and $(c_1, v_1) \cdots (c_n, v_n) \in (Ch \times V)^+$ by $(v_1 := \xi_1, \cdots, v_n := \xi_n)$, $c!\xi$, and $(c_1?v_1\& \cdots c_n?v_n)$, respectively.

[6]However, *MCC* lacks the multi-assignment statement incorporated in $\mathcal{L}_{MCC}$.

The mapping $i : A \to Act$ is given by: $i(\mathsf{stop}) = \mathsf{stop}$, $i(v_1, \cdots, v_n := e_1, \cdots, e_n) = (v_1 := \mathcal{E}[\![e_1]\!], \cdots, v_n := \mathcal{E}[\![e_n]\!]) \in (V \times \Xi)^+$, $i(c!e) = c!\mathcal{E}[\![e]\!] \in (Ch \times \Xi)$, and $i(c_1?v_1\& \cdots \&c_n?v_n) = (c_1?v_1\& \cdots \&c_n?v_n) \in (Ch \times V)^+$.

The mapping $interact : W \to \Theta$, where $W = [Act]$, and $(\theta \in)\Theta = \{\uparrow\} \cup (\Sigma \to \Sigma)$, behaves as follows: $interact([(v_1 := \xi_1, \cdots, v_n := \xi_n)]) = \lambda\sigma . [\, \sigma \mid v_1 \mapsto \xi_1 \mid \cdots \mid v_n \mapsto \xi_n \,] = interact([c_1!\xi_1, \ldots, c_n!\xi_n, (c_1?v_1\& \cdots \&c_n?v_n)])$, for any $v_1, \ldots, v_n \in V$, $\xi_1, \ldots, \xi_n \in \Xi$, $c_1, \ldots, c_n \in Ch$, and $interact(w) = \uparrow$, for any other $w \in W$. Any assignment action can always be executed independently. Also, any $n+1$ actions $c_1!\xi_1, \ldots, c_n!\xi_n$ and $(c_1?v_1\& \cdots \&c_n?v_n)$ occurring in parallel processes can interact synchronously. No other interaction is possible in $\mathcal{L}_{SYN}$. We recall that the elements of a multiset are not ordered. For example, $[c!\xi, c?v] = [c?v, c!\xi]$. We use the symbol $\uparrow$ to express the impossibility of an interaction between a multiset of actions. In particular, note that the action stop cannot interact with any other action(s): if $w \in W$ and $\mathsf{stop} \in w$ then $interact(w) = \uparrow$ (in the semantic model we use the symbol δ).

We can now define an *initial continuation* $\varphi_0 \in \mathbf{F}$, as fixed point of an appropriate higher-order mapping; the domain definitions are as in Section 4, only the domain $\mathbf{R} = \Sigma \to \mathbf{P}$ is specific.

Definition 5.2 *We define $\varphi_0 \in \mathbf{F}$ by $\varphi_0 = \mathsf{fix}(\Psi)$, where $\Psi : \mathbf{F} \to \mathbf{F}$ is given by:*

$$\Psi(\varphi)(w, \kappa_0) = \begin{cases} \lambda\sigma . \{\delta\} & \text{if } interact(w) = \uparrow \\ \lambda\sigma . \{\sigma'\} & \text{if } interact(w) = \theta \neq \uparrow, \sigma' = \theta(\sigma) \end{cases}$$

$$\Psi(\varphi)(w, \phi) = \begin{cases} \lambda\sigma . \{\delta\} & \text{if } interact(w) = \uparrow \\ \lambda\sigma . (\sigma' \cdot \phi(\varphi))(\sigma') & \text{if } interact(w) = \theta \neq \uparrow, \sigma' = \theta(\sigma) \end{cases}$$

We also define $\mathcal{D}[\![\cdot]\!] : \mathcal{L}_{MCC} \to \mathbf{R}$, by $\mathcal{D}[\![s]\!] = [\![s]\!](\varphi_0)$.

Ψ is a contraction and has a unique fixed point in particular due to the "$\sigma'\cdot$" - step in its definition.

Lemma 5.3 *The mapping Ψ is well defined and $\frac{1}{2}$ - contractive (in φ).*

Proposition 5.4 is easily established. It can be used to describe compositionally the behavior of $\mathcal{D}[\![s]\!]$, which evaluates an $\mathcal{L}_{MCC}$ program s with respect to the initial (empty) continuation φ_0. We write $s_1 \sim s_2$ to express that $\mathcal{D}[\![C(s_1)]\!] = \mathcal{D}[\![C(s_2)]\!]$, for all contexts C.

Proposition 5.4 *For any $s, s_1, s_2 \in Stat$, $v_1, \ldots, v_n \in V$, $e_1, \ldots, e_n \in Exp$, $c_1, \ldots, c_n \in Ch$, and $u \in UStat$:*[7]

$$\begin{array}{lrcl} & s_1 \simeq s_2 & \Rightarrow & s_1 \sim s_2 \\ \textit{(A6)} & s + \mathsf{stop} & \sim & s \\ \textit{(A7)} & \mathsf{stop}\,; s & \sim & s \\ \textit{(MCF1)} & c_1!e_1 \mid \cdots \mid c_n!e_n \mid (c_1?v_1\& \cdots \&c_n?v_n) & \sim & (v_1, \cdots, v_n := e_1, \cdots, e_n) \\ \textit{(MCF2)} & u & \sim & \mathsf{stop} \qquad \text{if } interact(i_U(u)) = \uparrow \end{array}$$

By using Lemma 4.14 we can also prove the following:

Lemma 5.5 *For any $s, s_1, s_2 \in Stat$, and $u \in UStat$:*

$$\begin{array}{rcl} \mathcal{D}[\![u]\!] & = & \begin{cases} \lambda\sigma . \{\delta\} & \text{if } interact(i_U(u)) = \uparrow \\ \lambda\sigma . \{\sigma'\} & \text{if } interact(i_U(u)) = \theta \neq \uparrow, \sigma' = \theta(\sigma) \end{cases} \\ \mathcal{D}[\![u; s]\!] & = & \begin{cases} \lambda\sigma . \{\delta\} & \text{if } interact(i_U(u)) = \uparrow \\ \lambda\sigma . (\sigma' \cdot \mathcal{D}[\![s]\!])(\sigma') & \text{if } interact(i_U(u)) = \theta \neq \uparrow, \sigma' = \theta(\sigma) \end{cases} \\ \mathcal{D}[\![s_1 + s_2]\!] & = & \mathcal{D}[\![s_1]\!] \oplus \mathcal{D}[\![s_2]\!] \end{array}$$

[7]Note that $interact(i_U(c_1!e_1 \mid \cdots \mid c_n!e_n \mid (c_1?v_1\& \cdots \&c_n?v_n)))) \neq \uparrow$

Note that, $\mathcal{D}[\![(v_1,\cdots,v_n := e_1,\cdots,e_n)]\!] = \mathcal{D}[\![c_1!e_1 \,|\, \cdots \,|\, c_n!e_n \,|\, (c_1?v_1\&\cdots\&c_n?v_n)]\!] = \lambda\sigma\,.\,\{[\,\sigma \mid v_1 \mapsto \mathcal{E}[\![e_1]\!](\sigma) \mid \cdots \mid v_n \mapsto \mathcal{E}[\![e_n]\!](\sigma)\,]\}$. Also, $\mathcal{D}[\![\,\mathsf{stop}\,;s]\!] = \mathcal{D}[\![\,\mathsf{stop}\,]\!] = \lambda\sigma\,.\,\{\delta\}$, for any $s \in \mathit{Stat}$.

By using Proposition 5.4 and Remark 4.16, for any non-recursive statement (closed term) $s \in \mathit{Stat}$ there is a basic interaction term $t \in \mathit{TStat}$, such that $s \sim t$. Next, one can use the properties of $\mathcal{D}[\![\cdot]\!]$ stated by Lemma 5.5, which are appropriate for evaluating basic interaction terms. For example:

$$\mathcal{D}[\![c!e \,\|\, c?v]\!] = \mathcal{D}[\![(c!e \,\lfloor\!\lfloor\, c?v) + (c?v \,\lfloor\!\lfloor\, c!e) + (c!e \,|\, c?v)]\!]$$

$$= \mathcal{D}[\![(c!e; c?v) + (c?v;\, c!e) + (c!e \,|\, c?v)]\!] = \mathcal{D}[\![\,\mathsf{stop}\, + \,\mathsf{stop}\, + (c!e \,|\, c?v)]\!]$$

$$= \mathcal{D}[\![c!e \,|\, c?v]\!] = \lambda\sigma\,.\,\{[\,\sigma \mid v \mapsto \mathcal{E}[\![e]\!](\sigma)\,]\}$$

As a slightly more complicated example one can check that $\mathcal{D}[\![c_1!e_1 \,\|\, c_2!e_2 \,\|\, (c_1?v_1\&c_2?v_2)]\!] = \lambda\sigma\,.\,\{[\,\sigma \mid v_1 \mapsto \mathcal{E}[\![e_1]\!](\sigma) \mid v_2 \mapsto \mathcal{E}[\![e_2]\!](\sigma)\,]\}$.

Remark 5.6 *According to Lemma 4.9, we have $[\![s_1]\!] \,\lfloor\, [\![s_2]\!] = [\![s_2]\!] \,\lfloor\, [\![s_1]\!]$, for any $s_1, s_2 \in \mathit{Stat}$. However, $\phi_\epsilon \,\lfloor\, \phi_\delta = \lambda\varphi\,.\,\lambda\sigma\,.\,\{\epsilon\} \neq \lambda\varphi\,.\,\lambda\sigma\,.\,\{\delta\} = \phi_\delta \,\lfloor\, \phi_\epsilon$, where $\phi_\epsilon = \lambda\varphi\,.\,\lambda\sigma\,.\,\{\epsilon\}$, and $\phi_\delta = \lambda\varphi\,.\,\lambda\sigma\,.\,\{\delta\}$. One can easily verify that $\phi_\epsilon, \phi_\delta \in \mathbf{D}$, but there is no $\mathcal{L}$ statement $s \in \mathit{Stat}$ such that $[\![s]\!] = \phi_\epsilon$ or $[\![s]\!] = \phi_\delta$. Lemma 4.9 is needed in the proof of Theorem 4.15. In general, the properties stated by Theorem 4.15 hold only for computations denotable by program statements.*

5.2 Binary Communication

Starting from $\mathcal{L}_{MCC}$ it is straightforward to obtain a CSP-like language with (only) binary communication. In the syntax Definition 5.1 of $\mathcal{L}_{MCC}$ we replace the general multi-assignment statement with the the simple assignment $v := e$ and the communication pattern with the receive statement $c?v$. The constructions $c!e$ and $c?v$ are now as in Occam [28]. Next, the mapping $\mathit{interact} : W \to \Theta$ specializes as follows: $\mathit{interact}([v := \xi]) = \lambda\sigma\,.\,[\,\sigma \mid v \mapsto \xi\,] = \mathit{interact}([c!\xi, c?v])$, and $\mathit{interact}(w) = \uparrow$, for any other $w \in W$. Here, $(\theta \in)\Theta = \{\uparrow\} \cup (\Sigma \to \Sigma)$, and $W = [\mathit{Act}]$, where $\mathit{Act} = \{\,\mathsf{stop}\,\} \cup (V \times \Xi) \cup (\mathit{Ch} \times \Xi) \cup (\mathit{Ch} \times V)$. The mapping $i : A \to \mathit{Act}$ is given by: $i(v := e) = (v, \mathcal{E}[\![e]\!]) \in (V \times \Xi)$, $i(c?v) = (c, v) \in (\mathit{Ch} \times V)$, $i(c!e) = (c, \mathcal{E}[\![e]\!]) \in (\mathit{Ch} \times \Xi)$ and $i(\,\mathsf{stop}\,) = \,\mathsf{stop}\,$. Let's call this language restricted to binary communication $\mathcal{L}_{SYN}$.

The other components of the denotational semantics remain unchanged. The denotational model of $\mathcal{L}_{SYN}$ satisfies all concurrency laws stated in Theorem 4.15. However, we note that the interaction statements u of the form $u = a_1 \,|\, \cdots \,|\, a_n$, with $n > 2$, will all block in $\mathcal{L}_{SYN}$, a property known as *handshaking* [6]. Obviously, the properties stated by Theorem 4.15 still hold if we replace each $u \in \mathit{UStat}$ by $a \in A$ in CM2, CM3, CM5-CM7 (because $A \subseteq \mathit{UStat}$). The properties A6 and A7 given in Proposition 5.4 are also satisfied for $\mathcal{L}_{SYN}$, and MCF1 and MCF2 specialize to the following:

(CF1)	$c!e \,|\, c?v$	$\sim$	$v := e$	
(CF2)	u	$\sim$	stop	if $\mathit{interact}(i_U(u)) = \uparrow$

for any $u \in UStat, c \in Ch, v \in V, e \in Exp$.

Now consider the set $(a \in)A$ of elementary statements in $\mathcal{L}_{SYN}$ as a set of atomic actions and (following the notation in [6]) a partial communication function $\gamma : A \times A \to A$, given by $\gamma(c!e, c?v) = \gamma(c?v, c!v) = (v := e)$, and $\gamma(a_1, a_2)$ undefined, otherwise. According to Proposition 5.4 (specialized for $\mathcal{L}_{SYN}$), the relation $\sim$ inherits all properties (A1-A5 and CM1-CM9) of the relation $\simeq$, given in Theorem 4.15. Also, note that, the collection of laws A1-A7, CF1, CF2, CM1-CM9, satisfied by $\sim$, coincides with the set of axioms of the process algebra ACP for synchronous communication, (for this specific communication function γ and) without encapsulation [6].

6 Continuation Semantics for a Nature-Inspired Formalism

In this section we employ the domain of continuations introduced in Section 4 for defining a denotational semantics designed in CPS, for the language $\mathcal{L}_{DNA}$ inspired by DNA computing. $\mathcal{L}_{DNA}$ is derived from the combinatorial strand algebra $\mathcal{P}$, introduced in [8]. The relevance of the $\mathcal{P}$ formalism for DNA computing is explained in [8]. Our present aim is to offer an investigation in this area of natural computing, by using methods and techniques consecrated in the tradition of programming languages semantics, viz. denotational semantics and continuations.

The elementary components in $\mathcal{L}_{DNA}$ are called *signals* and *gates*. We use a countable set $(x \in)X$ for the class of signals. A gate is a pair of multisets of signals written as $[x_1, \ldots, x_n].[x'_1, \ldots, x'_m]$ in [8]. We use a slightly different notation, adding enclosing parentheses: $([x_1, \ldots, x_n].[x'_1, \ldots, x'_m])$. For easier readability, we use the same notation for gates both in the syntactic representation and in semantic computations. We define the class of gates by $(g \in)G = ([X] \setminus []) \times [X]$. Note that the input part $[x_1, \ldots, x_n]$ of a gate $([x_1, \ldots, x_n].[x'_1, \ldots, x'_m])$ cannot be the empty multiset.

Definition 6.1 *We define the set $(a \in)A$ of elementary components by: $a ::= x \mid g$. The syntax of $(P \in)\mathcal{L}_{DNA}$ is given below (where $n \in \mathbb{N}$):*

$$P ::= \mathbf{0} \mid a \mid P^n \mid P \| P$$

$\mathbf{0}$ is the inert component. An elementary component $a \in A$ may be a signal $x \in X$, or a gate $g \in G$. $P_1 \| P_2$ is the parallel composition of P_1 and P_2. P^n is a construction for finite populations, an abbreviation for n concurrent copies of P, with $P^0 = \mathbf{0}$, and $P^{n+1} = P \| P^n$, for any $n \in \mathbb{N}$. The reader may wonder why we use the semantic notion of a multiset in the syntax definition of $\mathcal{L}_{DNA}$. It would be easy to make a complete separation between syntax and semantics. We could define the set of gates as a purely syntactic class, e.g., by modeling a gate as a pair of sequences of signals (rather than multisets). Instead, we use multisets because the order of signals in a gate is irrelevant. By using the operators for parallel composition and populations, signals and gates can be combined into a multiset (a "soup") of concurrent components that can interact. When a gate $([x_1, \ldots, x_n].[x'_1, \ldots, x'_m])$ is combined with n concurrent signals $x_1, \ldots, x_n$ the multiset $[x_1, \ldots, x_n]$ behaves as a join pattern [18]. The gate joins the signals $x_1, \ldots, x_n$, forks the signals $x'_1, \ldots, x'_m$ and is consumed in the interaction [8]. The behavior of such a system is expressed formally in [8] by a binary reaction relation $\rightarrow (\subseteq \mathcal{P} \times \mathcal{P})$ and the following rule, where $g = ([x_1, \ldots, x_n].[x'_1, \ldots, x'_m])$:

$$x_1 \parallel \ldots \parallel x_n \parallel g \quad \longrightarrow \quad x'_1 \parallel \ldots \parallel x'_m$$

In [11] the reaction relation is extended to a ternary relation, using gates as observable items capturing the semantics of DNA interactions; the above rule is written as $x_1 \parallel \cdots \parallel x_n \parallel g \xrightarrow{g} y_1 \parallel \cdots \parallel y_m$. The gates become observable items that capture the information expressed by DNA interactions (reactions). It may not be obvious that this notion of an observable contains sufficient information to capture the behavior of a DNA system specified by using the $\mathcal{P}$ process algebra. A formal justification of this semantic design decision is provided in [11]. In this section we also use the set G of gates as the set of observable items, that is used in the definition of the final domain $\mathbf{R}$, given below.

$\mathcal{L}_{DNA}$ is obtained from $\mathcal{P}$ by replacing the construction for unbounded populations P^* (incorporated in $\mathcal{P}$) with the construction for finite populations P^n. P^* can be used to generate an arbitrary number of concurrent copies of P. For simplicity, we restrict to finite populations. Apart from this, $\mathcal{L}_{DNA}$ is identical with $\mathcal{P}$. A denotational semantics designed with continuations for the (complete) language $\mathcal{P}$ introduced in [8] is provided in [40]. The denotational model given in [40] is designed by using a domain for continuations similar to the one that we employ in this paper, and it requires an additional fixed point construction (for handling unbounded populations) that we do not present here.

The denotational model for $\mathcal{L}_{DNA}$ can be based on the semantic framework introduced in Section 4. The class of actions is $Act = A$, and $i : A \rightarrow Act$, $i(a) = a$, for any $a \in A$. We take $W = [Act]$, and $(\theta \in)\Theta = \{\uparrow\} \cup G$. We define the mapping $interact : W \rightarrow \Theta$, $interact([x_1, \ldots, x_n, g]) = g$ if

$g = ([x_1, \ldots, x_n].[x'_1, \ldots, x'_m])$, i.e., if the multiset of signals $x_1, \ldots, x_n$ matches the input part of the gate g, and $interact(w) = \uparrow$ otherwise. We use the following final semantic domain:

$$\begin{array}{rcl}(r \in)\mathbf{R} & = & \mathcal{P}_{nco}(\mathbf{Q}) \\ (q \in)\mathbf{Q} & \cong & \{\epsilon\} + (G \times \frac{1}{2} \cdot \mathbf{Q})\end{array}$$

ϵ models the empty sequence (there is no notion of abnormal termination or deadlock in $\mathcal{L}_{DNA}$ [8, 10]). We use the same notation for sequences, as in Section 5, writing, e.g., $g_1 g_2 \ldots$, instead of $(g_1, (g_2, \ldots))$. Also, we use the notations: $g \cdot q = (g, q)$ and $g \cdot r = \{g \cdot q \mid q \in r\}$, for any $g \in G, q \in \mathbf{Q}$ and $r \in \mathbf{R}$. We define operator $\oplus : \mathbf{R} \times \mathbf{R} \xrightarrow{1} \mathbf{R}$ by $r_1 \oplus r_2 = \{q \mid q \in r_1 \cup r_2, q \neq \epsilon\} \cup \{\epsilon \mid \epsilon \in r_1 \cap r_2\}$.

The semantic domains $(\phi \in)\mathbf{D}$, $(\varphi \in)\mathbf{F}$, $(\kappa \in)\mathbf{K}$, and the semantic operators for parallel composition $\widehat{\|}$ and $\|$ are defined as in Section 4. We use the notation $\kappa^0 = \kappa_0$, and $\kappa^{n+1} = \kappa \,\widehat{\|}\, \kappa^n$, for any $\kappa \in \mathbf{K}, n \in \mathbb{N}$. Let $\phi_{(\cdot)} : A \to \mathbf{D}$, be given by $\phi_{(a)} = \lambda\varphi \,.\, \varphi([i(a)], \kappa_0)$. The denotational semantics $[\![\cdot]\!] : \mathcal{L}_{DNA} \to \mathbf{K}$ is defined (inductively) by the following equations:

$$\begin{array}{rcl}[\![\mathbf{0}]\!] & = & \kappa_0 \\ [\![a]\!] & = & \phi_{(a)} \\ [\![P^n]\!] & = & [\![P]\!]^n \\ [\![P_1 \parallel P_2]\!] & = & [\![P_1]\!] \,\widehat{\|}\, [\![P_2]\!]\end{array}$$

We write $P_1 \simeq P_2$ to express that $[\![C(P_1)]\!] = [\![C(P_2)]\!]$, where C is a syntactic context for $\mathcal{L}_{DNA}$, an element of the class $C ::= \bullet \mid \mathbf{0} \mid a \mid C^n \mid C \parallel C$, and the substitution $C(P)$ is defined by structural induction: $\bullet(P) = P, \mathbf{0}(P) = \mathbf{0}, a(P) = a, C^n(P) = C(P)^n$, and $(C_1 \parallel C_2)(P) = C_1(P) \parallel C_2(P)$. The following properties can be established: $P \parallel \mathbf{0} \simeq P$, $P_1 \parallel P_2 \simeq P_2 \parallel P_1$, and $(P_1 \parallel P_2) \parallel P_3 \simeq P_1 \parallel (P_2 \parallel P_3)$, for any $P, P_1, P_2, P_3 \in \mathcal{L}_{DNA}$.

We also define $\mathcal{D}[\![\cdot]\!] : \mathcal{L}_{DNA} \to \mathbf{R}$ as follows: $\mathcal{D}[\![P]\!] = \{\epsilon\}$ if $[\![P]\!] = \kappa_0$, and $\mathcal{D}[\![P]\!] = [\![P]\!](\varphi_0)$ otherwise; here $\varphi_0 = fix(\Psi)$ is the *initial continuation* defined as fixed point of the higher-order mapping $\Psi : \mathbf{F} \to \mathbf{F}$ (intuitively, Ψ is a contraction due to the "$g\cdot$"-step in its Definition [5])

$$\Psi(\varphi)(w, \kappa_0) = \begin{cases} \{\epsilon\} & \text{if } interact(w) = \uparrow \\ \{g\} & \text{if } interact(w) = g = ([x_1, \ldots, x_n].[]) \in G \\ g \cdot (\phi_{(x'_1)} \parallel \cdots \parallel \phi_{(x'_m)})(\varphi) & \text{if } interact(w) = g, \\ & \quad g = ([x_1, \ldots, x_n].[x'_1, \ldots, x'_m]) \in G \end{cases}$$

$$\Psi(\varphi)(w, \phi) = \begin{cases} \{\epsilon\} & \text{if } interact(w) = \uparrow \\ g \cdot \phi(\varphi) & \text{if } interact(w) = g([x_1, \ldots, x_n].[]) \in G \\ g \cdot (\phi \parallel \phi_{(x'_1)} \parallel \cdots \parallel \phi_{(x'_m)})(\varphi) & \text{if } interact(w) = g, \\ & \quad g = ([x_1, \ldots, x_n].[x'_1, \ldots, x'_m]) \in G \end{cases}$$

Example 6.2 *Let $P = x_1 \parallel ([x_1].[x'_1]) \parallel x_2 \parallel ([x_2].[x'_2])$, and $P' = x \parallel ([x, x_1].[x_2, x])^3 \parallel (x_1)^3$. The subterm $x \parallel ([x, x_1].[x_2, x])^3$ of P' is a catalytic system, ready to transform multiple x_1 (in this example 3 instances of x_1) to x_2, with catalyst x [8]. One can check the following:*

$$\begin{array}{rcl}\mathcal{D}[\![P]\!] & = & \{([x_1].[x'_1])([x_2].[x'_2]), ([x_2].[x'_2])([x_1].[x'_1])\} \\ \mathcal{D}[\![P']\!] & = & \{([x, x_1].[x_2, x])([x, x_1].[x_2, x])([x, x_1].[x_2, x])\}\end{array}$$

7 Implementation

We have developed prototype implementations of the denotational semantics given in this paper, using the functional language Haskell [29] (as a metalanguage for denotational semantics). A couple of $\mathcal{L}_{MCC}, \mathcal{L}_{SYN}$ and $\mathcal{L}_{DNA}$ programs (including the ones discussed in the paper) are provided, and can be tested by using the semantic interpreters (which are complete Haskell implementations of the denotational models given in the paper) available at `http://users.utcluj.ro/~eneia/fi17`

8 Conclusion

We presented a method for describing the behavior of concurrent programs, using denotational semantics and continuation-passing style (CPS). Based on the theory of metric spaces, we designed continuation-based denotational semantics for an abstract concurrent language, providing a general mechanism of interaction between multisets of distributed actions. By customizing the behavior of continuations, we obtained continuation semantics for various synchronous and asynchronous interaction mechanisms, incorporated in a couple of concurrent languages and a formalism for DNA computing. We showed that the basic laws of concurrent systems are satisfied in these semantics. The significance of the results lies mainly in the flexibility provided by the technique of continuations, which can thus be used to describe concurrent behavior in a formal framework.

We intend to investigate the relationship between the continuation-based approach presented in this paper and other approaches to concurrency semantics, in particular the direct approach to concurrency semantics, where the semantic operator for parallel composition is not designed with continuations [30, 5]. A formal relationship could be investigated, in particular with respect to the observable behavior. One option would be to study the relationship between the continuation-based models presented in this paper with the corresponding direct-style operational semantics, eventually by using intermediate continuation-based operational semantics. In this way the various views could be unified.

References

[1] A. Alexandru, G. Ciobanu. Mathematics of multisets in the Fraenkel-Mostowski framework. *Bull. Math. Soc. Sci. Math. Roumanie*, tome 58(106), no.1, pp. 3-18, 2015.

[2] P. America, J.J.M.M. Rutten. Solving Reflexive Domain Equations in a Category of Complete Metric Spaces. *J. of Comp. Syst. Sci*, vol. 39(3), pp. 343–375, 1989.

[3] A.W. Appel. *Compiling with Continuations*. Cambridge University Press, 1992.

[4] F.S. de Boer, J.N. Kok, C. Palamidessi, J.J.M.M. Rutten. A Paradigm for Asynchronous Communication and its Application to Concurrent Constraint Programming. In K.R. Apt, J.W. de Bakker, J.J.M.M. Rutten, editors, *Logic Progr. Languages: Constraints, Functions and Objects*, pp. 82–114, MIT Press, 1993.

[5] J.W. de Bakker, E.P. de Vink. *Control Flow Semantics.* MIT Press, 1996.

[6] J.C.M. Baeten, W.P. Weijland. *Process Algebra.* Cambridge Univ. Press, 1990.

[7] J.A. Bergstra, J.W. Klop. Process Algebra for Synchronous Communication. *Information and Control*, vol. 60, pp. 109–137, 1984.

[8] L. Cardelli. Strand Algebras for DNA Computing. *Natural Computing*, vol. 10(1), pp. 407–428, 2011.

[9] G. Ciobanu, E.N. Todoran, Continuation Semantics for Asynchronous Concurrency, *Fundamenta Informaticae*, vol. 131(3-4), pp. 373–388, 2014.

[10] G. Ciobanu, E.N. Todoran. Continuation Semantics for Concurrency with Multiple Channels Communication. In *Proc. 17th International Conference on Formal Engineering Methods 2015 (ICFEM 2015)*, LNCS, vol. 9407, pp. 400–416, Springer, 2015.

[11] G. Ciobanu, E.N. Todoran. Correct Metric Semantics for a Language Inspired by DNA Computing. *Concurrency and Computation: Practice and Experience*, vol. 28, pp. 3042-3060, 2016.

[12] K. Claessen A Poor Man's Concurrency Monad. *Journal of Functional Programming*, vol. 9(3), pp. 313–323, 1999.

[13] E. Cooper, S. Lindley, P. Wadler, and J. Yallop. Links: Web Programming Without Tiers. In *Proc. 5th International Conference on Formal Methods for Components and Objects*, pp. 266–296, LNCS 4709, Springer-Verlag, 2007.

[14] O. Danvy, and A. Filinski. Representing Control: a Study of the CPS Transformation. *Mathematical Structures in Computer Science*, vol. 2(4), pp. 361–391, 1992.

[15] R.K. Dybvig and R. Hieb. Engines from Continuations. *Computer Languages*, vol. 14(2), pp. 109–123, 1989.

[16] M. Felleisen. The Theory and Practice of First-Class Prompts. In *Procedings of 15th Annual ACM Symposium on Principles of Programming Languages*, pp. 180–190, ACM Press, 1988.

[17] M. Felleisen, M. Wand, D.P. Friedman, and B. Duba. Abstract Continuations: a Mathematical Semantics for Handling Full Functional Jumps. In *Proceedings of the 1988 ACM Conference on Lisp and Functional Programming*, pp. 52–62, ACM Press, 1988.

[18] C. Fournet and G. Gonthier. The Join calculus: A Language for Distributed Mobile Programming. LNCS vol. 2395, pp. 268–332, Springer-Verlag, 2002.

[19] C.T. Haynes, D.P. Friedman, and M. Wand. Obtaining Coroutines with Continuations. *Computer Languages*, vol. 11(3/4), pp. 143–153, 1986.

[20] R. Hieb, R.K. Dybvig and C.W. Anderson. Subcontinuations. *Lisp and Symbolic Computation*, vol. 7(1), pp. 83–110, 1994.

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

[22] P. Leger and H. Fukuda, Using Continuations and Aspects to Tame Asynchronous Programming on the Web. In *Companion Proc. 15th International Conference on Modularity*, pp. 79–82, ACM Press, 2016.

[23] R. Kelsey, W. Clinger and J. Rees, editors, *Revised Report on the Algorithmic Language Scheme*, 1998. Available from `http://swiss.csail.mit.edu/projects/scheme/`.

[24] A. Mazurkiewicz. Proving Algorithms by Tail Functions. *Information and Control*, vol. 18, pp. 220–226, 1971.

[25] R. Milner. Fully Abstract Models of Typed λ-Calculi. *Theoretical Computer Science*, vol. 4, pp. 1–22, 1977.

[26] L. Moreau and C. Queinnec. Partial Continuations as the Difference of Continuations: A Duumvirate of Control Operators. In *Proceedings of the 6th International Symposium on Programming Languages Implementation and Logic Programming*, LNCS vol. 844, pp. 182–197, Springer-Verlag, 1994.

[27] P.D. Mosses. Programming Language Description Languages: From Christopher Strachey to Semantics Online. In P. Boca *et al.*, editors, *Formal Methods: State of the Art and New Directions*, pp. 249–273, Springer-Verlag, 2010.

[28] INMOS Ltd. *Occam Programming Manual.* Prentice Hall, 1984.

[29] S. Peyton Jones, J. Hughes, Editors. Report on the Programming Language Haskell 98: a Non-Strict Purely Functional Language. 1999, available at `http://www.haskell.org`

[30] G. Plotkin. A Powerdomain Construction. *SIAM Journal of Computing*, vol. 5(3), pp. 452-487, 1976.

[31] C. Queinnec. The Influence of Browsers on Evaluators or, Continuations to Program Web Servers. In *Proceedings of the 5th ACM SIGPLAN International Conference on Functional Programming*, pp. 23-33, ACM Press, 2000.

[32] C. Queinnec. Inverting Back the Inversion of Control or, Continuations Versus Page-centric Programming. *SIGPLAN Notices*, vol. 38(2), pp. 57–64, 2003.

[33] C. Queinnec and B. Serpette. A Dynamic Extent Conrol Operator for Partial Continuations. In *Proc. 18th Annual ACM Symposium on Principles of Programming Languages*, pp. 174–184, ACM Press, 1991.

[34] J.C. Reynolds. The Discoveries of Continuations. *Lisp and Symbolic Computation*, vol. 6(3–4), pp. 233–247, 1993.

[35] Seaside Available from `http://www.seaside.st/`.

[36] D. Sitaram and M. Felleisen. Control Delimiters and their Hierarchies. *Lisp and Symbolic Computation*, vol. 3(1), pp. 67–99, 1990.

[37] Standard ML of New Jersey. Available from `http://www.seaside.st/`.

[38] C. Strachey and C.P. Wadsworth. Continuations: A Mathematical Semantics for Handling Full Jumps. *Higher-Order and Symbolic Computation*, vol. 13(1/2), pp. 135–152, 2000. Reprint of the technical monograph PRG-11, Oxford University Computing Laboratory (1974), with a foreword.

[39] E.N. Todoran. Metric Semantics for Synchronous and Asynchronous Communication: A Continuation-based Approach. *E*lectronic Notes in Theoretical Computer Science, vol. 28, pp. 119–146, Elsevier, 2000.

[40] E.N. Todoran, N. Papaspyrou. Experiments with Continuation Semantics for DNA Computing. *Proc. 9th IEEE Int. Conf. on Intelligent Computer Communication and Processing (ICCP 2013)*, pp. 251–258, 2013.

[41] UnCommon Web Available from `https://common-lisp.net/project/ucw/`.

[42] Wand, M. Continuation-Based Multiprocessing. *Higher-Order and Symbolic Computation*, vol. 12(3), pp. 285–299, 1999. Reprint from the Proceedings of the 1980 ACM Conference on LISP and Functional Programming, with a foreword.

[43] Web Applications in Racket Available from `http://docs.racket-lang.org/web-server/`.

[44] Weblocks Available from `https://common-lisp.net/project/cl-weblocks/`.