Experiments with Continuation Semantics for DNA Computing

Eneia Nicolae Todoran
Department of Computer Science
Technical University of Cluj-Napoca
Gh. Baritiu Street 28, 400027, Cluj-Napoca, Romania
Email: eneia.todoran@cs.utcluj.ro

Nikolaos Papaspyrou
School of Electrical and Computer Engineering
National Technical University of Athens
Polytechnioupoli, 15780 Zografou, Athens, Greece
Email: nickie@softlab.ntua.gr

***Abstract*—We investigate the semantics of a process algebra language for DNA computing. As a formal description technique we use denotational semantics and the mathematical methodology of metric semantics. We use continuations and powerdomains to represent nondeterministic behavior. An element of a powerdomain is a collection of sequences of observables representing DNA structures. We consider two notions of an observable item and we design two corresponding denotational models. As far as we know this is the first paper that employs the denotational approach in the semantic investigation of DNA computing.**

I. Introduction

We investigate the semantics of a process algebra language - that we call here L_{DNA} - which incorporates some basic concepts of DNA computing. The language was introduced in [6] where a couple of so-called 'strand algebras' are presented. These formalisms can capture the massive concurrency available at molecular level in DNA systems. The relevance of L_{DNA} process algebra for DNA computing is explained in [6].

As a formal description technique we use denotational semantics and the mathematical methodology of metric semantics [3]. We use continuations for concurrency [15] and powerdomains [12], [2] to represent nondeterministic behavior. In this paper an element of a powerdomain is a collection of sequences of observables representing L_{DNA} structures. We consider two notions of an observable item and we design two corresponding denotational models. The presentation focuses on the first denotational model where an observable is a L_{DNA} gate which captures an interaction. In the second denotational model an observable is a multiset of L_{DNA} elements.

L_{DNA} combines the following concepts: signals, gates and populations. Signals $x, y, \ldots \in X$ are symbols taken from a countable alphabet X. A gate[1] is an operator $([x_1, \ldots, x_n], [y_1, \ldots, y_m])$ that joins the signals $x_1, \ldots, x_n$ and forks the signals $y_1, \ldots, y_m$. We consider that the order of signals in $[x_1, \ldots, x_n]$ and $[y_1, \ldots, y_m]$ is irrelevant, hence, $[x_1, \ldots, x_n]$ and $[y_1, \ldots, y_m]$ are multisets.[2] A gate is a pair of multisets of signals. Signals and gates combine in a multiset of elements (a 'chemical soup') that proceed concurrently. When n signals $x_1, \ldots, x_n$ and a gate $([x_1, \ldots, x_n], [y_1, \ldots, y_m])$ are present in such a multiset they can interact. The signals $x_1, \ldots, x_n$ and the gate are consumed in the interaction and the signals $y_1, \ldots, y_m$ are released in the multiset. This interaction is described operationally in [6] by the following rule, where '$\|$' is the operator for parallel composition:

$$x_1 \parallel \cdots \parallel x_n \parallel ([x_1, \ldots, x_n], [y_1, \ldots, y_m]) \rightarrow y_1 \parallel \cdots \parallel y_m$$

Note that signals can interact with gates, but signals cannot interact with signals, nor gates with gates [6]. A gate captures the information that is processed in a L_{DNA} interaction, in the sense that it contains all signals that are joined and forked.

The signals $x_1, \ldots, x_n$ of a gate $([x_1, \ldots, x_n], [y_1, \ldots, y_m])$ represent a join pattern. Join synchronization was initially investigated in [8]. The chemical metaphor [5] served as an inspiration model for the both formalisms [8], [6]. The ability to join and fork signals and the ability to group signals and gates into finite or unbounded (inexhaustible) populations are specific of the strand algebras introduced in [6]. The construct for unbounded populations is based on the replication primitive of π-calculus [10].

Process algebras are formal languages that can describe concurrent activities of multiple processes [4]. In general, a process algebra only provides compositionality at the level of syntax in terms of operators which can be used to build more complex systems from simple components.

Denotational semantics (initially known as mathematical semantics or Scott-Stratchey semantics) is an important approach to formalizing the meanings of languages. The most important principle in denotational semantics is *compositionality*: the denotational semantics of a composite syntactic construct is expressed solely based on the denotational semantics of its syntactic components. A denotational semantics assigns mathematical objects to syntactic constructs which are elements of a given formal language. The approach employs a space of denotations, which are elements of a mathematical domain specific of the language under investigation. Denotations can be computed and analyzed independently, and then composed. In general, the denotational description of iteration and recursion relies on fixed point definitions.

In the denotational approach metric spaces [3] may be more appropriate than classic domains [9] for models that are naturally characterized by unique fixed points. On the other hand, the use of metric structures may make certain

[1] In [6] a gate is represented by using the notation $[x_1, \ldots, x_n].[y_1, \ldots, y_m]$. We avoid this notation since we use the symbol '$\cdot$' to represent a concatenation operator over sequences.

[2] Intuitively, a *multiset* is a collection in which an element may occurs more than once, an unordered list. A formal definition is provided in section II.

optimizations more difficult to achieve. In some applications it may be difficult to avoid using *hiatons* or *silent steps* to achieve the contractiveness of the semantic operators [3].

In this paper we follow the metric approach to semantics in defining two denotational semantics for L_{DNA}: $[\![\cdot]\!]_{\mathcal{G}}$ and $[\![\cdot]\!]_{\mathcal{C}}$. Each of the two denotational mappings takes two parameters: a (synchronous) continuation and a synchronization context. The main difference between $[\![\cdot]\!]_{\mathcal{G}}$ and $[\![\cdot]\!]_{\mathcal{C}}$ is given by the notion of an observable item. In the case of $[\![\cdot]\!]_{\mathcal{G}}$ an observable is an L_{DNA} gate capturing an interaction. In the case of $[\![\cdot]\!]_{\mathcal{C}}$ an observable is a multiset of elements representing a configuration of a system specified in L_{DNA}. We consider a couple of L_{DNA} examples.

Let $P_1 = (x_1 \parallel ([x_1],[y_1])) \parallel (x_2 \parallel ([x_2],[y_2]))$, $P_1 \in L_{DNA}$, f_0 be the empty continuation and $null$ be the empty synchronization context. $[\![P]\!]_{\mathcal{G}}$ behaves as follows:

$$[\![P_1]\!]_{\mathcal{G}}(f_0)(null) = \{([x_1],[y_1])([x_2],[y_2]), \\ ([x_2],[y_2])([x_1],[y_1])\}$$

The result is a collection of two sequences of gates. Each gate represents an interaction. There are two possible interactions: $([x_i],[y_i])$ interacts with x_i and releases y_i, for $i = 1, 2$. Also, there are two possible interleavings between the two interactions, caused by the nondeterminism of the system.

Let $P_2 = x \parallel (([x_1,x_2],[x_3]) \parallel ([x],[x_1,x_2])) \in L_{DNA}$.

$$[\![P_2]\!]_{\mathcal{G}}(f_0)(null) = \{([x],[x_1,x_2])([x_1,x_2],[x_3])\}$$

There is one possible execution sequence. First, $([x],[x_1,x_2])$ interacts with x, $([x],[x_1,x_2])$ and x are consumed, and x_1 and $x2$ are released. Next, follows the interaction between $([x_1,x_2],[x_3])$, x_1 and x_2, with the release of x_3.

A formal comparison between denotational semantics and operational semantics for L_{DNA} is beyond the scope of the present paper. Operationally, the semantics of P_2 can be described by the following sequence of (two) transitions [6].

$$P_2 \;\rightarrow\; x_1 \parallel x_2 \parallel ([x_1,x_2],[x_3]) \;\rightarrow\; x_3$$

A transition is an element of a relation between elements, called *configurations* [13]. In the above example, the configurations are L_{DNA} terms (P_2, $x_1 \parallel x_2 \parallel ([x_1,x_2],[x_3])$, and x_3) and each transition shows the effect of an L_{DNA} interaction. $[\![\cdot]\!]_{\mathcal{C}}$ can capture such (operational) effects denotationally:

$$[\![P_2]\!]_{\mathcal{C}}(f_0)(null) = \{[x_1,x_2,([x_1,x_2],[x_3])][x_3]\}$$

$[\![\cdot]\!]_{\mathcal{C}}$ produces as final value a collection of sequences of observable items, where each observable is a multisetof L_{DNA} elements. It is common to represent parallel composition of processes by using the concept of a multiset [5], [8], [6]. We represent a (finite) multiset by enumerating its elements between square brackets '[' and ']'. In the last example the multiset $[x_1,x_2,([x_1,x_2],[x_3])]$ is a semantic representation of the L_{DNA} term $x_1 \parallel x_2 \parallel ([x_1,x_2],[x_3])$. Note that in this representation parallel composition is automatically associative and commutative.

In general, an operational semantics is defined based on a *labeled transition system* and the relationship with a denotational semantics is established based on transition labels [3]. An operational semantics can also show systems configurations. In this paper we show how to express behavior based on system configurations in a pure compositional manner, following the discipline of denotational semantics.

A. Semantic prototype interpreters

The denotational (mathematical) specifications given in this paper were developed following a prototyping approach. We used the functional language Haskell [11] as a prototyping tool (and as a metalanguage) for denotational semantics.

The two denotational models given in this paper are implemented as 4 executable Haskell semantic interpreters available at [17]. $[\![\cdot]\!]_{\mathcal{G}}$ is implemented in the files `semgDNA.hs` and `semgDNA_fin.hs`. $[\![\cdot]\!]_{\mathcal{C}}$ is implemented in the files `semcDNA.hs` and `semcDNA_fin.hs`. `semgDNA.hs` and `semcDNA.hs` can only be used to test toy L_{DNA} "programs" that terminate after a finite number of interactions.[3]

The interpreters `semgDNA_fin.hs` and `semgDNA_fin.hs` compute only finite approximations of the semantics. They accept two parameters, an L_{DNA} "program" and a natural number n. They terminate the "execution" of the L_{DNA} "program" after at most n steps (i.e. the semantics is a collection of sequences, the length of each sequence is less than or equal with n).

B. Contribution

We offer a semantic investigation of a process algebra language for DNA computing introduced in [6]. The language combines join synchronization [6], [8], finite and unbounded populations. We use metric spaces [3] and continuations for concurrency [15] to model concurrent behavior following the discipline of denotational (compositional) semantics. To the best of our knowledge this is the first paper that employs the denotational approach in the semantic investigation of DNA computing. The final semantic domain describes behavior as a collection of sequences of observables representing DNA structures, with no silent steps interspersed.

As it is well known, at present most researchers prefer operational semantics, where behavior is expressed based on transitions between system configurations. Each transition can show the effect of an interaction. In this paper we demonstrate that such operational effects can be captured in a pure compositional manner, following the discipline of denotational semantics by using continuations for concurrency [15].

C. Structure of the paper

Section II contains some mathematical preliminaries. In section III we present the syntax of L_{DNA}. In sections IV and V we define the denotational semantics $[\![\cdot]\!]_{\mathcal{G}}$ and $[\![\cdot]\!]_{\mathcal{C}}$, respectively. The final section VI contains concluding remarks and plans for future research.

[3]In the mathematical specification $w \in W$ only if $\mu(w)$; see definition 4.1. In the Haskell implementation the condition $\mu(w)$ is verified at run time. Apart from this, we think that the denotational functions represent an accurate mathematical specification based on the Haskell prototypes.

II. MATHEMATICAL PRELIMINARIES

A *multiset* is a generalization of a set. Intuitively, a multiset is a collection in which an element may occur more than once. We can present a multiset of elements of type X by using a function from X to $\mathbb{N}$, or a partial function $m : X \to \mathbb{N}^+$, where $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$, namely the set of natural numbers without 0. $m(x)$ is called the *multiplicity* of x, representing its number of occurrences in m.

The notation $(x \in)X$ introduces the set X with typical element x ranging over X. Let X be a countable set. We denote by $[X]$ the set of all finite multisets of elements of type X, i.e., $[X] \stackrel{not.}{=} \bigcup_{A \in \mathcal{P}_{fin}(X)} \{m \mid m \in (A \to \mathbb{N}^+)\}$; $\mathcal{P}_{fin}(X)$ is the powerset of all *finite* subsets of X. Since X is countable, $\mathcal{P}_{fin}(X)$ is also countable. An element $m \in [X]$ is a multiset of elements of type X, namely a function $m : A \to \mathbb{N}^+$, where $A \in \mathcal{P}_{fin}(X)$ is such that $\forall x \in A : m(x) > 0$.

We can also represent a multiset $m \in [X]$ by enumerating its elements between parentheses '[' and ']'. Notice that the elements in a multiset are *not* ordered; intuitively, a multiset is an unordered list of elements. For example, [] is the empty multiset, i.e. the function with empty graph. 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 $m : \{x_1, x_2\} \to \mathbb{N}^+, m(x_1) = 2, m(x_2) = 1$.

We can define various operations on multisets $m_1, m_2 \in [X]$. Below, $\mathsf{dom}(\cdot)$ is the domain of function '$\cdot$'.

- *Multiset sum:* $m_1 \uplus m_2 \quad (\uplus : ([X] \times [X]) \to [X])$

  $\mathsf{dom}(m_1 \uplus m_2) = \mathsf{dom}(m_1) \cup \mathsf{dom}(m_2)$

  $(m_1 \uplus m_2)(x) =$

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

- *Multiset difference:* $m_1 \setminus m_2 \ (\setminus : ([X] \times [X]) \to [X])$

  $\mathsf{dom}(m_1 \setminus m_2) =$

  $(\, \mathsf{dom}(m_1) \setminus \mathsf{dom}(m_2) \,) \cup$

  $\{x \mid x \in \mathsf{dom}(m_1) \cap \mathsf{dom}(m_2), m_1(x) > m_2(x)\}$

  $(m_1 \setminus m_2)(x) =$

  $$\begin{cases} m_1(x) & \text{if } x \in \mathsf{dom}(m_1) \setminus \mathsf{dom}(m_2) \\ m_1(x) - m_2(x) & \text{if } x \in \mathsf{dom}(m_1) \cap \mathsf{dom}(m_2) \end{cases}$$

- *Submultiset:* $m_1 \subseteq m_2 \ (\subseteq : ([X] \times [X]) \to Bool)$

  $m_1 \subseteq m_2$ iff $(\, \mathsf{dom}(m_1) \subseteq \mathsf{dom}(m_2) \,) \wedge$

  $(\forall x \in \mathsf{dom}(m_1) : m_1(x) \leq m_2(x))$.

- *Cardinal number:* $|m| = \sum_{x \in \mathsf{dom}(m)} m(x)$

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 $\forall x \in \mathsf{dom}(m_1) : m_1(x) = m_2(x)$. Also, we write $m_1 \subset m_2$ whenever $m_1 \subseteq m_2$ and $\neg(m_1 = m_2)$.

When $(x \in)X$ is a countable set we use the following convention. We denote by $\overline{x}$ typical elements of $[X]$, i.e. $\overline{x} \in [X]$ is a multiset of elements of the type X.

Let $f : X \to X$ be a function. When $x \in X$ is such that $f(x) = x$, we call x a *fixed point* of f. When this fixed point is unique, we write $x = fix(f)$.

The denotational semantics given in this paper is defined following the mathematical methodology of metric semantics [3]. More exactly, we work within the mathematical framework of *1-bounded complete metric spaces*. We assume the following notions are known: *metric* and *ultrametric* space, *isometry* (distance preserving bijection between metric spaces, denoted by '$\cong$'), *complete* metric space, and *compact* set. For details, the reader may consult the monograph [3], for instance.

Some metrics are frequently used in metric semantics. For example, if X is any nonempty set, we can define the *discrete metric* $d : X \times X \to [0,1]$ as follows: $d(x,y) = \text{if } x = y \text{ then } 0 \text{ else } 1$. (X, d) is a complete ultrametric space.

We recall that if (X, d_X), (Y, d_Y) are metric spaces, a function $f{:}X \to Y$ is a *contraction* if $\exists c \in \mathbb{R}$, $0 \leq c < 1$, $\forall x_1, x_2 \in X : d_Y(f(x_1), f(x_2)) \leq c \cdot d_X(x_1, x_2)$. In metric semantics, it is usual to attach a contracting factor $c = \frac{1}{2}$ to each computation step. When $c = 1$ the function f is called *nonexpansive*. In what follows, we denote by $X \stackrel{1}{\to} Y$ the set of all nonexpansive functions from X to Y.

The following theorem is at the core of metric semantics.

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

Definition 2.2: Let $(X, d_X), (Y, d_Y)$ be (ultra)metric spaces. We define the following metrics over X, $X \to Y$ (function space), $X \times Y$ (Cartesian product), $X + Y$ (disjoint union defined by $X + Y = (\{1\} \times X) \cup (\{2\} \times Y)$), and $\mathcal{P}(X)$ (powerset of X), respectively.

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

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

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

We use the abbreviation $\mathcal{P}_{nco}(X)$ to denote the powerset of *non-empty and compact* subsets of X. Also, we often suppress the metrics part in domain definitions, and write only $\frac{1}{2} \cdot X$ 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. If d_X, d_Y are ultrametrics, then 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 $\frac{1}{2} \cdot X$, $X \to Y$, $X \stackrel{1}{\to} Y$, $X \times Y$, $X + Y$, and $\mathcal{P}_{nco}(X)$ with their metrics defined above are also complete metric spaces [3].

III. Syntax of L_{DNA}

Let $(x, y \in)X$ be a (countable) set of *signals* and let $(\overline{x}, \overline{y} \in)[X]$ be the set of all finite multisets of signals. We define the set of *gates* by $(g \in)G = [X] \times [X]$. A gate $g = (\overline{x}, \overline{y})(\in G)$ is a pair of multisets of signals.

Definition 3.1: (Syntax of L_{DNA})

$$P ::= 0 \mid x \mid g \mid P \parallel P \mid P^k \mid P^*$$

Formally, $(P \in)L_{DNA}$ is the set of finite trees P generated by the syntax above. We call the syntactic elements of L_{DNA} *programs* or *components*, and we use the term *execution* to refer to their behavior.

0 is the inert component. x is a signal. $g = ([x_1, \ldots, x_n], [y_1, \ldots, y_m])$ is a gate, an operator from signals to signals; it can interact with n parallel signals $x_1, \ldots, x_n$, it produces m signals $y_1, \ldots, y_m$ and is consumed in the process. The semantics of such an interaction was described in the introduction; the formal operational semantics is given in [6]. Signals and gates can be combined into a multiset by parallel composition $P_1 \parallel P_2$, and can also be combined into finite populations P^k or unbounded populations P^*. P^k is an abbreviation for k parallel copies of P, with $P^0 = 0$. P^* has the property that $P^* = P \parallel P^*$, that is, there is always one more P that can be taken from the population.

The reader may wonder why we use the semantic notion of a multiset in the syntax definition of L_{DNA}. It would be easy to make a complete separation between syntax and semantics. For example, we could define gates by $g ::= (x^*, x^*)$, where x^* is a finite sequence of signals. But we use multisets because we consider that the order of signals is irrelevant.

IV. Denotational semantics of L_{DNA} with observable interactions

We present a denotational semantics $[\![\cdot]\!]_{\mathcal{G}}$ for L_{DNA}, where each observable item is an L_{DNA} gate used in an interaction. We introduce a set $(w \in)W$ of *synchronization contexts* that expresses synchronization in continuation semantics. We present the domain definition, and the definition of $[\![\cdot]\!]_{\mathcal{G}}$.

A. Synchronization contexts

Definition 4.1: The set $(w \in)W$ of *synchronization contexts* is defined by:

$$W = \{\mu(w) \mid w \in \{null\} \cup (G \times [X])\}$$

where $\mu : \{null\} \cup (G \times [X]) \to Bool$ is given by

$$\mu(null) = true$$
$$\mu((\overline{x}, \overline{y}), \overline{x}') = (\overline{x}' \subseteq \overline{x})$$

Intuitively, $\mu(w) = true$ if w *could synchronize* but not necessarily synchronizes (already). Over $[X]$ multisets we use the operations $(\uplus, =, \subseteq, \subset, \setminus, |\cdot|)$ introduced in section II. $null$ could synchronize (if some parallel component contributes with a gate at synchronization). A synchronization context $((\overline{x}, \overline{y}), \overline{x}')$ contains a gate $(\overline{x}, \overline{y})$ and a multiset of signals $\overline{x}'$ that could 'match' with the gate in the sense that $\overline{x}' \subseteq \overline{x}$. $((\overline{x}, \overline{y}), \overline{x}')$ could synchronize if $\overline{x}' \subseteq \overline{x}$, i.e. if it is possible to obtain $\overline{x}$ from $\overline{x}'$ by adding a few signals (produced by parallel components) to the multiset $\overline{x}'$. In definition 4.2 we introduce an operator $\oplus$ which adds a multiset of signals to a synchronization context.

Definition 4.2: We define $\oplus : (W \times [X]) \to W$ by:

$$w \oplus \overline{x}'' = \begin{cases} ((\overline{x}, \overline{y}), \overline{x}' \uplus \overline{x}'') & \text{if } w = ((\overline{x}, \overline{y}), \overline{x}') \text{ and} \\ & \quad \overline{x}' \uplus \overline{x}'' \subseteq \overline{x} \\ w & \text{otherwise.} \end{cases}$$

Definition 4.3: We define $\sigma : W \to Bool$ by:

$$\sigma(null) = false$$
$$\sigma((\overline{x}, \overline{y}), \overline{x}') = (\overline{x}' = \overline{x})$$

If $w \in W$ and $\sigma(w)$ we say that w *synchronizes*.

Remark 4.4: $\sigma(w) \Rightarrow \mu(w)$.

Definition 4.5: Let $(\cdot < \cdot), [\cdot < \cdot) : (W \times W) \to Bool$,

$$(w_1 < w_2) = \begin{cases} true & \text{if } w_1 = (g_1, \overline{x}_1) \text{ and } w_2 = null \\ true & \text{if } w_1 = (g_1, \overline{x}_1), w_2 = (g_2, \overline{x}_2), \\ & \quad g_1 = g_2, \text{ and } \overline{x}_2 \subset \overline{x}_1 \\ false & \text{otherwise.} \end{cases}$$

$$[w_1 < w_2) = (w_1 < w_2) \wedge \neg(\sigma(w_1))$$

Intuitively, $(w_1 < w_2)$ if $\mu(w_1)$ and w_1 is closer of synchronization than w_2. If $\mu(w_1)$, $w_1 = (g_1, \overline{x}_1)$ and $w_2 = null$ then $(w_1 < w_2)$. Also, $(w_1 < w_2)$ if $\mu(w_1)$, $w_1 = (g_1, \overline{x}_1)$, $w_2 = (g_2, \overline{x}_2)$, $g_1 = g_2$ and $\overline{x}_2 \subset \overline{x}_1$. Otherwise, $\neg(w_1 < w_2)$. $[w_1 < w_2)$ if $(w_1 < w_2)$ and w_1 does not synchronize (yet).

Remarks 4.6:

(a) For any $w_1, w_2 \in W$, if $\sigma(w_2)$ then $\neg(w_1 < w_2)$.

(b) For any $w_1, w_2 \in W$, if $\sigma(w_2)$ then $\neg[w_1 < w_2)$.

We introduce a complexity function $c_w(w)$ that measures how far or close w is from synchronization.

Definition 4.7: We define $c_w : W \to \mathbb{N} \cup \{\infty\}$ by:

$$c_w(null) = \infty$$
$$c_w((\overline{x}, \overline{y}), \overline{x}') = |\overline{x} \setminus \overline{x}'|$$

We endow $\mathbb{N} \cup \{\infty\}$ with the total order $0 < 1 < 2 < \cdots < n < \cdots \infty$. $|\overline{x} \setminus \overline{x}'|$ is the cardinal number of the multiset $\overline{x} \setminus \overline{x}'$.

Remarks 4.8:

(a) $(w_1 < w_2) \Rightarrow c_w(w_1) < c_w(w_2)$.

(b) $\sigma(w) \Leftrightarrow c_w(w) = 0$.

B. Domain definitions

The domain of the denotational semantics $[\![\cdot]\!]_{\mathcal{G}}$ is $\mathbf{D}$:

$$(\phi \in)\mathbf{D} \cong \{d_0\} + \mathbf{Den}$$
$$(\varphi \in)\mathbf{Den} = \mathbf{F} \overset{!}{\to} W \to \mathbf{P}$$
$$(f \in)\mathbf{F} = \mathbf{K} \overset{!}{\to} W \to \mathbf{P}$$
$$(\kappa \in)\mathbf{K} = \tfrac{1}{2} \cdot \mathbf{D}$$
$$(p \in)\mathbf{P} = \mathcal{P}_{nco}(\mathbf{Q})$$
$$(q \in)\mathbf{Q} \cong \{\epsilon\} + (G \times (\tfrac{1}{2} \cdot \mathbf{Q}))$$

$\mathbf{D}$ and $\mathbf{Den}$ are domains of *computations*. $\mathbf{D}$ is also a domain of *denotations*; the denotational semantics produces values of the type $\mathbf{D}$. An element $\phi \in \mathbf{D}$ is either the inert computation d_0 or a computation $\varphi \in \mathbf{Den}$. $\mathbf{F}$ is the domain of *synchronous continuations*. $\mathbf{K}$ is the domain of *asynchronous continuations*. Intuitively, an asynchronous continuation κ stores a $\mathbf{D}$ computation; κ is a parallel composition of computations. In general, such an (asynchronous) continuation is a more complex structure, e.g., a tree of computations [15], [16]. In the case of L_{DNA}, a continuation is a multiset of computations that are packed into a single computation by means of parallel composition. In this paper, an asynchronous continuation κ is a computation, stored in the space $\mathbf{K} = \frac{1}{2} \cdot \mathbf{D}$. $\mathbf{K}$ and $\mathbf{D}$ have the same support set (see definition 2.2), only the distance between points halves in $\mathbf{K}$.

We use '$\cdot$' as a prefixing operator for $\mathbf{Q}$ sequences: $g \cdot q = (g, q)$, for $q \in \mathbf{Q}$. Instead of $(g_1, (g_2, \cdots (g_n, \epsilon) \cdots))$ we write $g_1 g_2 \cdots g_n$. Also, we use the notation $g \cdot p = \{g \cdot q \mid q \in p\}$, for any $g \in G, p \in \mathbf{P}$. $(p \in)\mathbf{P}$ is the domain of compact and nonempty collections of $\mathbf{Q}$ sequences.

In the equations given above the sets G and $[X]$ are endowed with the discrete metric which is an ultrametric. The composed metric spaces are built up using the metrics of Definition 2.2. The system of equations has a solution, which is unique up to isometry [1]. The solution for $\mathbf{D}$ is obtained as a complete ultrametric space. In [1], the family of complete (ultra)metric spaces is made into a category $\mathcal{C}$. It is proved that a generalized form of Banachs fixed point theorem holds, stating that a functor $\mathcal{F} : \mathcal{C} \to \mathcal{C}$ that is contracting (in a sense) has a unique fixed point (up to isometry). Intuitively, in the equation above the relevant functor is contracting as a consequence of the fact that the recursive occurrence of $\mathbf{D}$ is preceded by a $\frac{1}{2}$ factor. The domain equation for $\mathbf{Q}$ also has a unique solution (up to isometry) [2]. ϵ is the empty sequence, which denotes *termination*.

C. Semantic operators

Definition 4.9: The semantics of *nondeterministic choice* in L_{DNA} is given by the operator $+ : (\mathbf{P} \times \mathbf{P}) \to \mathbf{P}$,

$$p_1 + p_2 = \{q \mid q \in p_1 \cup p_2, q \neq \epsilon\} \cup \{\epsilon \mid \epsilon \in p_1 \cap p_2\}.$$

Also, we define $(:) : (Bool \times \mathbf{P}) \to \mathbf{P}$ by:

$$true : p = p$$

$$false : p = \{\epsilon\}$$

Remarks 4.10:

(a) The definition of '$+$' reflects that $p_1 + p_2$ terminates only if both p_1 and p_2 terminate.

(b) It is easy to check the following properties:

$$b : (p_1 + p_2) = (b : p_1) + (b : p_2),$$
$$(b_1 \wedge b_2) : p = b_1 : (b_2 : p) = b_2 : (b_1 : p).$$

(c) '$+$' is nonexpansive, associative, commutative and idempotent [3]. ':' is nonexpansive.

Definition 4.11: We define $\| = fix(\Psi)$, where $\Psi : \mathbf{Op} \to \mathbf{Op}$, $\mathbf{Op} = (\mathbf{D} \times \mathbf{D}) \xrightarrow{1} \mathbf{D}$, is given by:

$$\Psi(\psi)(d_0, d_0) = d_0$$

$$\Psi(\psi)(d_0, \varphi) = \varphi$$

$$\Psi(\psi)(\varphi, d_0) = \varphi$$

$$\Psi(\psi)(\varphi_1, \varphi_2) =$$

$$\lambda f.\lambda w.(\varphi_1(\lambda\kappa_1.\lambda w_1.$$
$$((w_1 < w) : f(\psi(\kappa_1, \varphi_2))\, w_1) +$$
$$([w_1 < w) :$$
$$\varphi_2(\lambda\kappa_2.f(\psi(\kappa_1, \kappa_2)))\, w_1))\, w +$$
$$\varphi_2(\lambda\kappa_2.\lambda w_2.$$
$$((w_2 < w) : f(\psi(\kappa_2, \varphi_1))\, w_2) +$$
$$([w_2 < w) :$$
$$\varphi_1(\lambda\kappa_1.f(\psi(\kappa_2, \kappa_1)))\, w_2))\, w)$$

Lemma 4.12: $\Psi : Op \xrightarrow{\frac{1}{2}} Op$ (Ψ has a unique fixed point).

The proof of lemma 4.12 is omitted. One can use the fact that in the right-hand side of the equation for $\Psi(\psi)(\varphi_1, \varphi_2)$ the occurrences of $\psi(\cdot, \cdot)$ are stored in the space $\mathbf{K} = \frac{1}{2} \cdot \mathbf{D}$.

Definition 4.13: $\lfloor : (\mathbf{Den} \times \mathbf{Den}) \to \mathbf{Den}$ is an operator that we call *left synchronization*.

$$(\varphi_1 \lfloor \varphi_2) f w =$$
$$\varphi_1(\lambda\kappa_1.\lambda w_1.((w_1 < w) : f(\kappa_1 \parallel \varphi_2)\, w_1) +$$
$$([w_1 < w) : \varphi_2(\lambda\kappa_2.f(\kappa_1 \parallel \kappa_2))\, w_1))\, w$$

We use the infix notation for $\|$ and $\lfloor$. The expression $(\phi_1 \lfloor \phi_2) f w$ reads, after adding parentheses, as follows: $(\phi_1 \lfloor \phi_2)(f)(w)$. Similar elaborations will be omitted in the sequel since the intended meaning can always be inferred from the types of the functions concerned. $(\varphi_1 \lfloor \varphi_2)$ attempts to synchronize two computations φ_1, φ_2, in this order. φ_1 is evaluated in the synchronization context w and produces a new synchronization context w_1 such that $(w_1 < w)$. φ_2 is evaluated in the synchronization context w_1 only if w_1 does not synchronize (yet), i.e. if $\neg\, \sigma(w_1)$ (see definition 4.5). No observable is produced before synchronization. Synchronization is handled by the synchronous continuation f. It is easy to check the following:

Lemma 4.14: $\|$ and $\lfloor$ are nonexpansive.

Remark 4.15: $d_0 \parallel d_0 = d_0, d_0 \parallel \varphi = \varphi \parallel d_0 = \varphi$ and for any $\varphi_1, \varphi_2 \in \mathbf{Den}$:

$$\varphi_1 \parallel \varphi_2 = \lambda f.\lambda w.((\varphi_1 \lfloor \varphi_2) f w + (\varphi_1 \lfloor \varphi_2) f w)$$

$\|$ is commutative (because $+$ is commutative).

Definition 4.16: For any $n \in \mathbb{N}$, we define $\|^n (\cdot) : \mathbf{D}^n \to \mathbf{D}$ ($\mathbf{D}^n = \mathbf{D} \times \cdots \times \mathbf{D}$ - n times, $n \geq 0$) by:

$$\|^0 () = d_0$$

$$\|^{n+1} (\phi_1, \phi_2, \ldots, \phi_{n+1}) = \phi_1 \parallel (\|^n (\phi_2, \ldots, \phi_{n+1}))$$

Definition 4.17: Let $[\![\cdot]\!]^X_{\mathcal{G}} : X \to \mathbf{D}$ be given by:

$$[\![x]\!]^X_{\mathcal{G}} =$$
$$\lambda f.\lambda w.\ \mathsf{if}\ (w = null)\ \mathsf{then}\ \{\epsilon\}$$
$$\mathsf{else}\ \ \mathsf{let}\ w' = w \oplus [x]$$
$$\mathsf{in}\ ((w' < w) : f(d_0)(w'))$$

Also, we define $[\![\cdot]\!]^G_{\mathcal{G}} : G \to \mathbf{D}$ by:

$[\![g]\!]^G_{\mathcal{G}} =$

$\lambda f.\lambda w.$ if $(w = null)$ then $f(d_0)(g, [])$ else $\{\epsilon\}$

Definition 4.18: Let $\Phi : \mathbf{F} \to \mathbf{F}$ be given by:

$$\begin{array}{l}\Phi(f)kw = \\ \quad \text{if } (\neg\sigma(w)) \text{ then } \{\epsilon\} \\ \quad \text{else let } w = (g, \overline{x}') \\ \qquad\qquad g = (\overline{x}, [y_1, \ldots, y_m]) \\ \qquad\qquad \phi = \|^{m+1}\ (\kappa, [\![y_1]\!]^X_{\mathcal{G}}, \ldots, [\![y_m]\!]^X_{\mathcal{G}}) \\ \qquad\quad \text{in if } \phi = d_0 \text{ then } \{g\} \text{ else } g \cdot \phi(f)null\end{array}$$

We define $f_0 = fix(\Phi)$.

Lemma 4.19: Φ is a contraction, i.e. $\Phi : \mathbf{F} \xrightarrow{\frac{1}{2}} \mathbf{F}$.

Proof: The proof is an easy consequence of the fact that the occurence of f is preceded by the "$g\cdot$" -step in the right-hand side of the equation defining Φ. ∎

We introduce an operator Ω that we use in the semantic equation for unbounded populations. Well-definedness of Ω follows by induction on $c_w(w)$.

Definition 4.20: We define $\Omega : \mathbf{Den} \to \mathbf{Den} \to \mathbf{Den}$ by:

$$\begin{array}{l}\Omega\varphi_1\varphi_2 f w = \\ \quad \varphi_1(\lambda\kappa_1.\lambda w_1.((w_1 < w) : f(\kappa_1 \parallel \varphi_2)\, w_1) + \\ \qquad\qquad ([\, w_1 < w) : \Omega\varphi_1\varphi_2(\lambda\kappa_2.f(\kappa_1 \parallel \kappa_2))\, w_1))\, w\end{array}$$

Lemma 4.21: $\Omega : \mathbf{Den} \xrightarrow{1} \mathbf{Den} \xrightarrow{\frac{1}{2}} \mathbf{Den}$, i.e.:

(a) $d(\Omega\varphi_1^1, \Omega\varphi_1^2) \le d(\varphi_1^1, \varphi_1^2),\ \forall \varphi_1^1, \varphi_1^2 \in \mathbf{Den}$ and

(b) $d(\Omega\varphi_1\varphi_2^1, \Omega\varphi_1\varphi_2^2) \le \frac{1}{2} \cdot d(\varphi_2^1, \varphi_2^2),\ \forall \varphi_1, \varphi_2^1, \varphi_2^2 \in \mathbf{Den}$.

Proof: We only treat 4.21(b). It suffices to show that $d(\Omega\varphi_1\varphi_2^1 f w, \Omega\varphi_1\varphi_2^2 f w) \le \frac{1}{2} \cdot d(\varphi_2^1, \varphi_2^2)$, for arbitrary $f \in \mathbf{F}, w \in W$. We proceed by induction on $c_w(w)$.

If $c_w(w) = 0$ then $d(\Omega\varphi_1\varphi_2^1 f w, \Omega\varphi_1\varphi_2^2 f w) = d(\varphi_1 f_\epsilon, \varphi_1 f_\epsilon) = 0$, where $f_\epsilon = \lambda\kappa_1.\lambda w_1.\{\epsilon\}$, because $c_w(w) = 0 \Rightarrow \sigma(w)$, hence there is no w_1 such that $(w_1 < w)$ or $[\, w_1 < w)$, according to remark 4.6.

Next, assume that $c_w(w) > 0$.

$$\begin{array}{l}d(\Omega\varphi_1\varphi_2^1 f w, \Omega\varphi_1\varphi_2^2 f w) \\ = d(\varphi_1(\lambda\kappa_1.\lambda w_1. \\ \qquad ((w_1 < w) : f(\kappa_1 \parallel \varphi_2^1)\, w_1) + \\ \qquad ([\, w_1 < w) : \\ \qquad\quad \Omega\varphi_1\varphi_2^1(\lambda\kappa_2^1.f(\kappa_1 \parallel \kappa_2^1))\, w_1))\, w \\ \quad \varphi_1(\lambda\kappa_1.\lambda w_1. \\ \qquad ((w_1 < w) : f(\kappa_1 \parallel \varphi_2^2)\, w_1) + \\ \qquad ([\, w_1 < w) : \\ \qquad\quad \Omega\varphi_1\varphi_2^2(\lambda\kappa_2^2.f(\kappa_1 \parallel \kappa_2^2))\, w_1))\, w) \\ \quad [\varphi_1, \text{'+' and ':' are nonexpansive}] \\ = sup_{\kappa_1 \in \mathbf{K}, w_1 \in W} \\ \qquad max\{d(f(\kappa_1 \parallel \varphi_2^1)w_1, f(\kappa_1 \parallel \varphi_2^2)w_1)^{(*)}, \\ \qquad\quad d(\Omega\varphi_1\varphi_2^1(\lambda\kappa_2^1.f(\kappa_1 \parallel \kappa_2^1))\, w_1, \\ \qquad\quad \Omega\varphi_1\varphi_2^2(\lambda\kappa_2^2.f(\kappa_1 \parallel \kappa_2^2))\, w_1)^{(**)}\}\end{array}$$

By using the fact that f is nonexpansive we obtain:

$$\begin{array}{l}^{(*)} = d_{\mathbf{K}}(\kappa_1 \parallel \varphi_2^1, \kappa_1 \parallel \varphi_2^2) \\ [\text{'}\parallel\text{' is nonexpansive, } \mathbf{K} = \frac{1}{2} \cdot \mathbf{D} = \frac{1}{2} \cdot (\{d_0\} + \mathbf{Den})] \\ \le \frac{1}{2} \cdot\ d_{\mathbf{D}}(\varphi_2^1, \varphi_2^2) = \frac{1}{2} \cdot\ d_{\mathbf{Den}}(\varphi_2^1, \varphi_2^2)\end{array}$$

By remark 4.8 $[\, w_1 < w) \Rightarrow c_w(w_1) < c_w(w)$. Hence, by the induction hypothesis:

$$^{(**)} \le \frac{1}{2} \cdot\ d_{\mathbf{Den}}(\varphi_2^1, \varphi_2^2)$$

∎

Remark 4.22: Let $\varphi \in \mathbf{Den}$. The mapping $\Omega(\varphi)$ is $\frac{1}{2}$ contractive (by lemma 4.21). Let $\overline{\varphi} = fix(\Omega(\varphi))$. It is straightforward to check that $\Omega(\varphi)(\overline{\varphi}) = \varphi \lfloor \overline{\varphi}$.

D. Denotational semantics

Definition 4.23: We define $[\![\cdot]\!]_{\mathcal{G}} : L_{DNA} \to \mathbf{D}$ by:

$$\begin{array}{rcl}[\![0]\!]_{\mathcal{G}} & = & d_0 \\ [\![x]\!]_{\mathcal{G}} & = & [\![x]\!]^X_{\mathcal{G}} \\ [\![g]\!]_{\mathcal{G}} & = & [\![g]\!]^G_{\mathcal{G}} \\ [\![P^k]\!]_{\mathcal{G}} & = & \|^k\ ([\![P]\!]_{\mathcal{G}}, \ldots, [\![P]\!]_{\mathcal{G}}) \\ [\![P^*]\!]_{\mathcal{G}} & = & \begin{cases} d_0 & \text{if } [\![P]\!]_{\mathcal{G}} = d_0, \\ fix(\Omega([\![P]\!]_{\mathcal{G}})) & \text{otherwise} \end{cases} \\ [\![P_1 \parallel P_2]\!]_{\mathcal{G}} & = & [\![P_1]\!]_{\mathcal{G}} \parallel [\![P_2]\!]_{\mathcal{G}}\end{array}$$

Let $\mathcal{D}_{\mathcal{G}}[\![\cdot]\!] : L_{DNA} \to \mathbf{P}$ be given, for any $P \in L_{DNA}$, by:

$$\mathcal{D}_{\mathcal{G}}[\![P]\!] = [\![P]\!]_{\mathcal{G}}(f_0)(null)$$

Remark 4.24: The operator for unbounded populations is based on the operator for replication from the π calculus [10]. There are standard domain-theoretic descriptions of replication in the literature; see, e.g, [14]. However, such solutions are conceived for binary interactions (as in the π calculus), and cannot be applied to a language with multiparty (or join) interactions as L_{DNA}. Essentially, (when $[\![P]\!]_{\mathcal{G}} \neq d_0$) in definition 4.23 we put $[\![P^*]\!]_{\mathcal{G}} = fix(\lambda\varphi.([\![P]\!]_{\mathcal{G}} \lfloor \varphi))$, which implies $[\![P^*]\!]_{\mathcal{G}} = [\![P]\!]_{\mathcal{G}} \lfloor [\![P^*]\!]_{\mathcal{G}}$. The operator for unbounded populations should satisfy the property: $[\![P^*]\!]_{\mathcal{G}} = [\![P]\!]_{\mathcal{G}} \parallel [\![P^*]\!]_{\mathcal{G}}$,[4] but we cannot simply put $fix(\lambda\varphi.([\![P]\!]_{\mathcal{G}} \parallel \varphi))$, because, in general, $\lambda\varphi.([\![P]\!]_{\mathcal{G}} \parallel \varphi)$ is *not* a contraction. In this paper we only explain informally why our solution works. Also we validate the solution experimentally, by providing L_{DNA} examples with unbounded populations that are executed properly by our semantic interpreters available at [17].

$[\![P]\!]_{\mathcal{G}} \parallel [\![P^*]\!]_{\mathcal{G}}$ is defined based on a nondeteministic choice between $[\![P]\!]_{\mathcal{G}} \lfloor [\![P^*]\!]_{\mathcal{G}}$ and $[\![P^*]\!]_{\mathcal{G}} \lfloor [\![P^*]\!]_{\mathcal{G}}$. As the operator for nondeterministic choice '+' is idempotent, it is enough to prove that $[\![P]\!]_{\mathcal{G}} \lfloor [\![P^*]\!]_{\mathcal{G}} = [\![P^*]\!]_{\mathcal{G}} \lfloor [\![P]\!]_{\mathcal{G}}$. We notice that $[\![P]\!]_{\mathcal{G}} \lfloor [\![P^*]\!]_{\mathcal{G}} = ([\![P]\!]_{\mathcal{G}} \lfloor \cdots ([\![P]\!]_{\mathcal{G}} \lfloor [\![P^*]\!]_{\mathcal{G}}) \cdots)$ and $[\![P^*]\!]_{\mathcal{G}} \lfloor [\![P]\!]_{\mathcal{G}} = ([\![P]\!]_{\mathcal{G}} \lfloor \cdots ([\![P]\!]_{\mathcal{G}} \lfloor [\![P^*]\!]_{\mathcal{G}}) \cdots) \lfloor [\![P]\!]_{\mathcal{G}}$ Intuitively, both $[\![P^*]\!]_{\mathcal{G}} \lfloor [\![P]\!]_{\mathcal{G}}$ and $[\![P]\!]_{\mathcal{G}} \lfloor [\![P^*]\!]_{\mathcal{G}}$ take as many copies of $[\![P]\!]_{\mathcal{G}}$ as necessary (but not more) to achieve a

[4] $[\![P]\!]_{\mathcal{G}} \parallel [\![P^*]\!]_{\mathcal{G}} = \lambda f w.([\![P]\!]_{\mathcal{G}} \lfloor [\![P^*]\!]_{\mathcal{G}}) f w + ([\![P^*]\!]_{\mathcal{G}} \lfloor [\![P]\!]_{\mathcal{G}}) f w.$

synchronization. The synchronization produces an observable ($\frac{1}{2}$ contraction) step. After such a synchronization step the continuations of all computations involved are executed in parallel with $[\![P^*]\!]_{\mathcal{G}} \parallel [\![P]\!]_{\mathcal{G}}$ and $[\![P^*]\!]_{\mathcal{G}}$, respectively. Based on this observation we notice that the relationship between $[\![P^*]\!]_{\mathcal{G}} \parallel [\![P]\!]_{\mathcal{G}}$ and $[\![P^*]\!]_{\mathcal{G}}$ is an *invariant* of the computation, preserved by each computation step.

A formal proof of $[\![P^*]\!]_{\mathcal{G}} = [\![P]\!]_{\mathcal{G}} \parallel [\![P^*]\!]_{\mathcal{G}}$ and other program properties could employ the technique introduced in [7]. In [7] each (nontrivial) semantic property is proved by identifying a corresponding invariant of the computation, as a relation between continuation structures. The identification of semantic properties from the invariants of the computation is common in bisimulation semantics [10]. In [7] this idea is adapted to a denotational framework, by using arguments of the kind '$\varepsilon \leq \frac{1}{2} \cdot \varepsilon \Rightarrow \varepsilon = 0$', which are standard in metric semantics [3]. ε is the distance between two behaviorally equivalent continuations, before and after a computation step, respectively. The effect of each computation step is given by the $\frac{1}{2}$ contracting factor. Hence $\varepsilon = 0$ and the desired property follows.

Examples 4.25: Let $P_1, P_2, P_3 \in L_{DNA}$,

$$P_1 = (x_1 \parallel ([x_1],[y_1])) \parallel (x_2 \parallel ([x_2],[y_2]))$$
$$P_2 = x \parallel (([x_1,x_2],[x_3]) \parallel ([x],[x_1,x_2]))$$
$$P_3 = (y \parallel ([y,x_1],[x_2,y])^*) \parallel (x_1)^3$$

P_1 and P_2 are as in section I. In P_3, $y \parallel ([y,x_1],[x_2,y])^*$ is a catalytic system ready to transform multiple x_1 to x_2 with catalyst y [6]. P_3 contains a finite population of 3 signals $(x_1)^3$, hence computation terminates after 3 steps (although P_3 also contains an unbounded population $([y,x_1],[x_2,y])^*$). One may check the following results:

$$\mathcal{D}_{\mathcal{G}}[\![P_1]\!] = \{([x_1],[y_1])([x_2],[y_2]),([x_2],[y_2])([x_1],[y_1])\}$$
$$\mathcal{D}_{\mathcal{G}}[\![P_2]\!] = \{([x],[x_1,x_2])([x_1,x_2],[x_3])\}$$
$$\mathcal{D}_{\mathcal{G}}[\![P_3]\!] = \{ggg\}, \text{ where } \; g = ([y,x_1],[x_2,y])$$

The reader may (download and) run the semantic interpreters available at [17]. For example, running (the Haskell implementation of) P_1 with the interpreter `semgDNA.hs` one obtains the following result:

```
[(["x1"],["y1"]).(["x2"],["y2"]),

(["x2"],["y2"]).(["x1"],["y1"])]
```

Let $P_4 = x^* \parallel ([x],[y])^*$. The execution of P_4 never terminates, hence P_4 cannot be tested with `semgDNA.hs`. The semantic interpreter `semgDNA_fin.hs` enforces the termination any L_{DNA} program after a finite number of steps, hence it allows you to make experiments with nonterminating L_{DNA} programs. For example, you can run P_4 enforcing its termination after at most 3 steps and you obtain:

```
[(["x"],["y"]).(["x"],["y"]).(["x"],["y"])]
```

V. Denotational semantics of L_{DNA} with observable configurations

We present an alternative denotational semantics $[\![\cdot]\!]_{\mathcal{C}}$ for L_{DNA}. In this section each observable item is a Γ configuration of a L_{DNA} system. Since the main lines of reasoning were introduced in section IV we adopt a more terse style of the presentation.

Definition 5.1: We define the class $\alpha \in A$ of L_{DNA} *elements* inductively. Any signal $x \in X$ or gate $g \in G$ is an L_{DNA} element, i.e. $X \subseteq A, G \subseteq A$. If $\alpha_1, \ldots, \alpha_n \in A$ then $(*, [\alpha_1, \ldots, \alpha_n]) \in A$. We use the notation $[\alpha_1, \ldots, \alpha_n]^* = (*, [\alpha_1, \ldots, \alpha_n])$; here, $[\alpha_1, \ldots, \alpha_n]$ is a multiset of L_{DNA} elements. We define the class $\gamma \in \Gamma$ of L_{DNA} *configurations* by $\Gamma = [A]$; a configuration is a multiset of L_{DNA} elements.

The domain of the denotational semantics $[\![\cdot]\!]_{\mathcal{G}}$ is $\mathbf{D}$:

$$(\phi \in)\mathbf{D} \cong \{d_0\} + (\Gamma \times \mathbf{Den}),$$
$$(\varphi \in)\mathbf{Den} = \mathbf{F} \stackrel{1}{\to} W \to \mathbf{P},$$
$$(f \in)\mathbf{F} = \mathbf{K} \stackrel{1}{\to} W \to \mathbf{P}, \quad (\kappa \in)\mathbf{K} = \tfrac{1}{2} \cdot \mathbf{D},$$
$$(p \in)\mathbf{P} = \mathcal{P}_{nco}(\mathbf{Q}), \quad (q \in)\mathbf{Q} \cong \{\epsilon\} + (\Gamma \times (\tfrac{1}{2} \cdot \mathbf{Q})).$$

Domain definitions are similar to the ones given in section IV. There are two important differences. In this section observable items are Γ configurations, rather than G gates. Also, a denotation may be either the inert computation d_0 or a pair consisting of a Γ configuration and a $\mathbf{Den}$ computation. The set Γ is endowed with the discrete metric.

We use the operators on W synchronization contexts and multisets introduced in the previous sections. Also, for $\mathbf{Q}$ and $\mathbf{P}$ we use the same notation and operators. The notation for prefixing is $\gamma \cdot p = \{\gamma \cdot q \mid q \in p\}$, where $\gamma \cdot q = (\gamma, q)$, $\forall \gamma \in \Gamma, q \in \mathbf{Q}$. ϵ is the empty sequence and instead of $(\gamma_1, (\gamma_2, \ldots (\gamma_n \epsilon) \ldots))$ we write $\gamma_1 \gamma_2 \ldots \gamma_n$. The operator $+ : (\mathbf{P} \times \mathbf{P}) \to \mathbf{P}$ for nondeterministic choice is $p_1 + p_2 = \{q \mid q \in p_1 \cup p_2, q \neq \epsilon\} \cup \{\epsilon \mid \epsilon \in p_1 \cap p_2\}$. Also, we put $(:) : (Bool \times \mathbf{P}) \to \mathbf{P}$, $(true : p) = p$, and $(false : p) = \{\epsilon\}$.

The semantic operator for parallel composition $\parallel : (\mathbf{D} \times \mathbf{D}) \to \mathbf{D}$ acts as a multiset sum on configurations. $d_0 \parallel d_0 = d_0$, $d_0 \parallel \phi = d_0 \parallel \phi = \phi$ and:

$$\begin{aligned}
&(\gamma_1, \varphi_1) \parallel (\gamma_2, \varphi_2) = \\
&\quad (\gamma_1 \uplus \gamma_2, \\
&\quad \lambda f.\lambda w.(\varphi_1(\lambda \kappa_1.\lambda w_1. \\
&\qquad\qquad ((w_1 < w) : f(\kappa_1 \parallel (\gamma_2, \varphi_2))\, w_1) + \\
&\qquad\qquad ([w_1 < w) : \\
&\qquad\qquad\quad \varphi_2(\lambda \kappa_2.f(\kappa_1 \parallel \kappa_2))\, w_1))\, w + \\
&\qquad \varphi_2(\lambda \kappa_2.\lambda w_2. \\
&\qquad\qquad ((w_2 < w) : f(\kappa_2 \parallel (\gamma_1, \varphi_1))\, w_2) + \\
&\qquad\qquad ([w_2 < w) : \\
&\qquad\qquad\quad \varphi_1(\lambda \kappa_1.f(\kappa_2 \parallel \kappa_1))\, w_2))\, w))
\end{aligned}$$

The mappings $[\![\cdot]\!]^X_{\mathcal{C}} : X \to \mathbf{D}$ and $[\![\cdot]\!]^G_{\mathcal{C}} : G \to \mathbf{D}$ are

$$\begin{aligned}
&[\![x]\!]^X_{\mathcal{C}} = \\
&\quad ([x], \lambda f.\lambda w.\, \mathsf{if}\ (w = null)\ \mathsf{then}\ \{\epsilon\} \\
&\qquad\qquad \mathsf{else}\ \ \mathsf{let}\ w' = w \oplus [x] \\
&\qquad\qquad\qquad \mathsf{in}\ ((w' < w) : f(d_0)(w'))
\end{aligned}$$

$$\begin{aligned}
&[\![g]\!]^G_{\mathcal{C}} = \\
&\quad ([g], \lambda f.\lambda w.\, \mathsf{if}\ (w = null)\ \mathsf{then}\ f(d_0)(g, []) \ \mathsf{else}\ \{\epsilon\})
\end{aligned}$$

The initial continuation $f_0 : \mathbf{F}$ produces an observable step which is a configuration. This information is extracted from

the asynchronous continuation and the signals that are released by the interaction step.

$$
\begin{array}{l}
f_0 kw = \mathsf{if}\ (\neg\,\sigma(w))\ \mathsf{then}\ \{\epsilon\} \\
\qquad \mathsf{else}\ \ \mathsf{let}\ w = ((\overline{x}, [y_1, \ldots, y_m]), \overline{x}') \\
\qquad\qquad \phi = \|^{m+1}\ (\kappa, [\![y_1]\!]_{\mathcal{C}}^X, \ldots, [\![y_m]\!]_{\mathcal{C}}^X) \\
\qquad\quad \mathsf{in}\ \ \mathsf{if}\ \phi = d_0\ \mathsf{then}\ \{[]\} \\
\qquad\qquad\ \ \mathsf{else}\ \ \mathsf{let}\ \phi = (\gamma, \varphi)\ \mathsf{in}\ \gamma \cdot \varphi(f_0) null
\end{array}
$$

Formally, $\|$ and f_0 can be defined as fixed points of appropriate higher-order operators, as we have shown in section IV. The n-ary operators for parallel composition can also be defined inductively: $\|^0\ () = d_0$ and $\|^{n+1}\ (\phi_1, \phi_2, \ldots, \phi_{n+1}) = \phi_1 \parallel (\|^n\ (\phi_2, \ldots, \phi_{n+1}))$.

We define the semantics of unbounded populations based on the operator $\Omega : \Gamma \to \mathbf{D} \to \mathbf{D} \to \mathbf{D}$,

$$
\begin{array}{l}
\Omega\gamma_2\varphi_1\varphi_2 f w = \\
\quad \varphi_1(\lambda\kappa_1.\lambda w_1.((w_1 < w) : f(\kappa_1 \parallel (\gamma_2, \varphi_2))\, w_1) + \\
\qquad\qquad ([w_1 < w) : \\
\qquad\qquad\quad \Omega\gamma_2\varphi_1\varphi_2(\lambda\kappa_2.f(\kappa_1 \parallel \kappa_2))\, w_1))\, w
\end{array}
$$

For any $\gamma \in \Gamma, \varphi \in \mathbf{Den}$ one can show that $\Omega\gamma\varphi$ is $\frac{1}{2}$ contractive. If $\overline{\varphi} = fix(\Omega\gamma\varphi)$ then $\Omega\gamma\varphi\overline{\varphi} = \varphi \lfloor \overline{\varphi}$, where

$$
\begin{array}{l}
(\varphi_1 \lfloor \varphi_2) f w = \\
\quad \varphi_1(\lambda\kappa_1.\lambda w_1.((w_1 < w) : f(\kappa_1 \parallel (\gamma_2, \varphi_2))\, w_1) + \\
\qquad\qquad ([w_1 < w) : \varphi_2(\lambda\kappa_2.f(\kappa_1 \parallel \kappa_2))\, w_1))\, w
\end{array}
$$

Finally, we define the denotational semantics $[\![\cdot]\!]_{\mathcal{C}}$.

Definition 5.2: We define $[\![\cdot]\!]_{\mathcal{C}} : L_{DNA} \to \mathbf{D}$ by:

$$
\begin{array}{rcl}
[\![0]\!]_{\mathcal{C}} & = & d_0 \\
[\![x]\!]_{\mathcal{C}} & = & [\![x]\!]_{\mathcal{C}}^X \\
[\![g]\!]_{\mathcal{C}} & = & [\![g]\!]_{\mathcal{C}}^G \\
[\![P^k]\!]_{\mathcal{C}} & = & \|^k\ ([\![P]\!]_{\mathcal{C}}, \ldots, [\![P]\!]_{\mathcal{C}}) \\
[\![P^*]\!]_{\mathcal{C}} & = & \begin{cases} d_0 & \text{if } [\![P]\!]_{\mathcal{C}} = d_0, \\ ([\gamma^*], fix(\Omega[\gamma^*]\varphi) & \text{if } [\![P]\!]_{\mathcal{C}} = (\gamma, \varphi) \end{cases} \\
[\![P_1 \parallel P_2]\!]_{\mathcal{C}} & = & [\![P_1]\!]_{\mathcal{C}} \parallel [\![P_2]\!]_{\mathcal{C}}
\end{array}
$$

Let $\mathcal{D}_{\mathcal{G}}[\![\cdot]\!] : L_{DNA} \to \mathbf{P}$ be given, for any $P \in L_{DNA}$, by:

$$\mathcal{D}_{\mathcal{C}}[\![P]\!] = [\![P]\!]_{\mathcal{C}}(f_0)(null)$$

Examples 5.3: Let P_1, P_2, P_3 be the L_{DNA} example programs considered in 4.25. One can check the following:

$$
\begin{array}{l}
\mathcal{D}_{\mathcal{C}}[\![P_1]\!] = \{[x_2, y_1, ([x_2], [y_2])][y_1, y_2], \\
\qquad\qquad [x_1, y_2, ([x_1], [y_1])][y_1, y_2]\} \\
\mathcal{D}_{\mathcal{C}}[\![P_2]\!] = \{[x_1, x_2, ([x_1, x_2], [x_3])][x_3]\} \\
\mathcal{D}_{\mathcal{C}}[\![P_3]\!] = \{\gamma_1\gamma_2\gamma_3\}
\end{array}
$$

where

$$
\begin{array}{l}
\gamma_1 = [x_1, x_1, x_2, y, [([y, x_1], [x_2, y])]^*] \\
\gamma_2 = [x_1, x_2, x_2, y, [([y, x_1], [x_2, y])]^*] \\
\gamma_3 = [x_2, x_2, x_2, y, [([y, x_1], [x_2, y])]^*]
\end{array}
$$

These and other L_{DNA} example programs can be tested by using the semantic interpreters `semcDNA.hs` and `semcDNA_fin.hs` available at [17]. P_1, P_2 and P_3 can be tested by using `semcDNA.hs`. `semcDNA_fin.hs` terminates automatically any L_{DNA} program after a specified number of steps, hence it provides testing support for nonterminating L_{DNA} programs, such as $P_4 = x^* \parallel ([x], [y])^*$.

VI. Concluding remarks and future research

We report on the first stage of an investigation of the denotational semantics of DNA computing. We work in the mathematical framework of metric semantics [3]. We use continuations [15] and powerdomains to represent nondeterministic behavior. In this paper an element of a powerdomain is a collection of sequences of observables representing DNA structures. We consider two notions of an observable item and we design two corresponding denotational models.

We intend to continue the research concerning the behavior of DNA systems by using methods in the tradition of programming languages semantics. We will investigate the possibility to define a metric denotational semantics designed with continuations for the stochastic strand algebra given in [6]. We will study the formal relationship between the denotational semantics and the operational semantics of the process algebras for DNA computing given in [6].

References

[1] P.America, J.J.M.M. Rutten, Solving reflexive domain equations in a category of complete metric spaces, *J. of Comput. System Sci.*, 39:343-375, 1989.

[2] J.W. de Bakker, J.I. Zucker, Processes and the denotational semantics of concurrency. *Information and Control*, 54:70-120, 1982.

[3] J.W. de Bakker, E.P. de Vink, *Control flow semantics.* MIT Press, 1996.

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

[5] G. Berry, G. Baudol, The chemical abstract machine, *Theoretical Computer Science*, 96:217–248, 1992.

[6] L. Cardelli, Strand algebras for DNA computing, *Natural Computing* 10(1): 407-428, 2011.

[7] G. Ciobanu, E.N. Todoran, Continuation semantics for asynchronous concurrency, *Fundamenta Informaticae* (in press).

[8] C. Fournet, G. Gonthier, The Join calculus: a language for distributed mobile programming, *LNCS* 25:268–332, 2002.

[9] G. Gierz, D.S. Scott, *Continuous lattices and domains.* Cambridge Univ. Press, 2003.

[10] R. Milner. *Communicating and mobile systems: the π caculus.* Cambridge Univ. Press, 1999.

[11] S. Peyton Jones, J. Hughes (Eds.), Report on the Programming Language Haskell 98: a Non-Strict Purely Functional Language, 1999. Available at `http://www.haskell.org/`.

[12] G. Plotkin, A powerdomain construction, *SIAM Journal of Computing* 5(3):452–487, 1976.

[13] G. Plotkin, A structural approach to operational semantics, *J. Log. Algebr. Program.* (60-61):17–139, 2004.

[14] I. Stark, A fully abstract domain model for the π-calculus, Proc. of *LICS*, pages 36–42, 1996.

[15] E.N.Todoran, Metric semantics for synchronous and asynchronous communication: a continuation-based approach, *ENTCS* 28:119–146, 2000.

[16] E.N.Todoran, N.Papaspyrou, Continuations for prototyping concurrent languages: yet another study of concurrency with emphasis on control-flow and communication mechanisms, Technical Report CSD-SW-TR-1-06, National Technical University of Athens, Software Engineering Laboratory, 2006.

[17] `ftp://ftp.utcluj.ro/pub/users/gc/`.