Combinatorial Strand Algebra in Insertion Modeling System

Dmitriy M. Klionov[1]

[1] Kherson State University, 40 rokiv Zhovtnya st. 27, Kherson, Ukraine
soulslayermaster@gmail.com

Abstract. This article is focused on the Insertion Modeling System developed by A.A. Letichevsky of the department 100/105 of the Glushkov Institute of Cybernetics, National Academy of Science of Ukraine, Kyiv, Ukraine Insertion Modeling System (IMS)[1] is buit on the Algebraic Programming System (APS) that also was developed by A.A. Letichevsky in 1987. and on the way of implementation of strand algebras – a process algebra for DNA computing devised by Luca Cardelli in order to compile other formal systems into the algebra, and compilation of this algebra into DNA structures. We focus on the basic strand algebra – combinatorial strand algebra, which is equivalent to the place-transition Petri nets, and on the version of the model driver of the Insertion Modeling System, based on the Petri nets.

Keywords: insertion modeling, system biology, strand algebras

Key Terms: Computation, Model, InsertionModeling, Algebra.

1 Introduction

DNA technology is reaching the point where one can envision automatically compiling high-level formalisms to DNA computational structures [11]. There are three compilation processes for concurrent languages, described by Cardelli in paper[10]. First, is the compilation of a low-level combinatorial algebra to a certain class of composable DNA structures [12]: this is intended to be a direct (but not quite trivial) mapping, which provides an algebraic notation for writing concurrent molecular programs. Second, is the compilation of a higher-level expression-based algebra to the lower-level combinatorial algebra, as a paradigm for compiling expressions of arbitrary complexity to `assembly language' DNA combinators. Third is translating concurrent interacting automata [13] to molecular structures. There is no clear way to implement such system, because one must decompose concurrent communication patterns into a form suitable for molecular interactions (a quadratic process that is described in [13]), and then one must find some suitable `general programmable matter' as a physical substrate. Some solution of this problem, based on the combinatorial DNA algebra, was given by Cardelli in paper[10].

Process algebras are formal languages designed to describe and analyze the concurrent activities of multiple processes. The standard technical presentation of process algebras was initially inspired by a chemical metaphor [14], and it is therefore natural, as a tutorial, to see how the chemistry of diluted well-mixed solutions can itself be presented as a process algebra. Having chemistry in this form also facilitates relating it to other process algebras.

Take a set C of chemical solutions denoted by P,Q,R. Two binary relations are defined on this set. The first relation, mixing, $P \equiv Q$ is an equivalence relation: its purpose is to describe reversible events that amount to `chemical mixing'; that is, to bringing components close to each other (syntactically) so that they can conveniently react by the second relation. Its basic algebraic laws are the commutative monoid laws of + and 0, where + is the chemical combination symbol and 0 represents the empty solution. The second relation, reaction, $P \to Q$, describes how a (sub-) solution P becomes a different solution Q. A reaction $P \to Q$ operates under a dilution assumption; namely, that adding some R to P does not make it then impossible for P to become Q (although R may enable additional reactions that overall quantitatively repress by interfering with P). The two relations of mixing and reaction are connected by a rule that says that the solution is well mixed: for any reaction to happen it is sufficient to mix the solution so that the reagents can interact. In first instance, the reaction relation does not have chemical rates. However, from the initial solution, from the rates of the base reactions, and from the relation $\to$ describing whole-system transitions, one can generate a continuous time Markov chain representing the kinetics of the system. In terms of system evolution, it is also useful to consider the symmetric and transitive closure, $\to^*$, representing sequences of reactions.

As process algebra, chemistry therefore obeys the following general laws, shown lower:

$$P \equiv P; P \equiv Q \Rightarrow Q \equiv P; P \equiv Q, Q \equiv R \Rightarrow P \equiv R \quad (1)$$

Equivalence

$$P \equiv Q \Rightarrow P + R \equiv Q + R \quad (2)$$

Congruence

$$P + Q \equiv Q + P; P + (Q + R) \equiv (P + Q) + R; P + 0 \equiv P \quad (3)$$

Diffusion

$$P \to Q \Rightarrow P + R \to Q + R \quad (4)$$

Dilusion

$$P \equiv P', P' \to Q', Q' \equiv Q \Rightarrow P \to Q \quad (5)$$

well mixing

Algebra is about equations, but in process algebra equations are usually a derived concept. Instead of axiomatizing a set of equations, we can use the reaction relation to study the equations that hold in a given algebra, meaning that $P = Q$ holds if P and Q produce the same reactions [15]. The complexity of these derived equational theories varies with the algebra. A simple instance here is the equation P + 0 = P, whose validity requires verifying that in definition of $\rightarrow$ there is no reaction for 0, nor for 0 combined with something else.

This way, chemistry can be presented as process algebra. But the algebra of chemical `+' is one among many: there are other process algebras that can suit biochemistry more directly [16,17] or, that can suit DNA computing. In the same way the strand algebra represent DNA strands, DNA gates and operations among them allowing the higher-level formalisms to be compiled to the DNA structures. I this paper we will show the way to represent the simplest strand algebra – combinatorial strand algebra as insertion model for the Insertion Modeling System[1], using the fact that combinatorial strand algebra is equivalent to place transition Petri nets. There is a representation of Petri nets, given by A.A. Letichevsky in form of insertion machines. We use this representation in order to build a specified analytical model driver for the combinatorial strand algebra.

2 Insertion Modeling System

2.1 The Architecture of Insertion Modeling System

Insertion Modeling System(IMS) [1] developed by A.A. Letichevsky of the Glushkov Institute of Cybernetics, National Academy of Science of Ukraine, Kyiv, Ukraine. Insertion modeling is the technology of system design founded on the theory of interaction of agents and environments. It is based on process algebra and is intended for the unification of different models of interaction and computation (such as CCS, CSP, π- calculus, mobile ambients etc.).

Insertion model of a system represent this system as a composition of environment and agents inserted into it. The insertion function is usually denoted as E[u] were E is the state of environment and u is the state of an agent. E[u] is a new environment state after insertion an agent u. All agents and environments are labeled or attributed transition systems (labeled systems with states labeled by attribute labels [9]). The states of transition systems are considered up to bisimilarity. The main invariant of bisimilarity is the behavior $beh[E]$ of transition system in the state E (an oriented tree with edges labeled by actions and nodes labeled by attribute labels). Behaviors themselves can be considered as states of transition systems.

The general architecture of insertion machine is represented on the fig. 1.

The main component of insertion machine is model driver, the component which controls the machine movement along the behavior tree of a model. The state of a model is represented as a text in the input language of insertion machine and is considered as an algebraic expression. The input language include the recursive definitions of agent behaviors, the notation for insertion function, and possibly some compositions for environment states. The state of a system must be reduced to the form $E[u_1,u_2,...]$. This functionality is performed by the module called agent behavior unfolder.

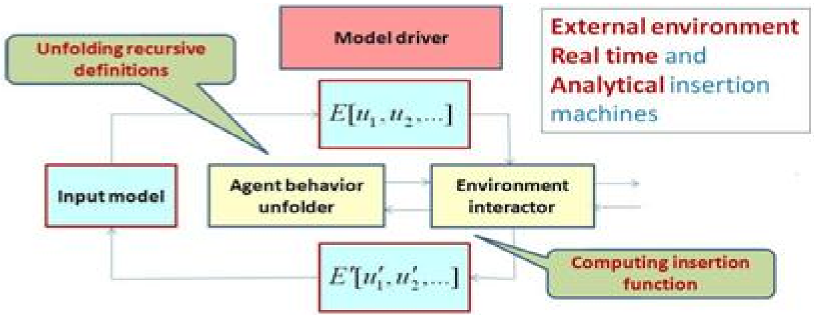

Fig. 1. Architecture of Insertion Machine

Two kinds of insertion machines are considered: real type or interactive and analytical insertion machines. The first ones exist in the real or virtual environment, interacting with it in the real or virtual time. Analytical machines intended for model analyses, investigation of its properties, solving problems etc. The drivers for two kinds of machines correspondingly are also divided on interactive and analytical drivers.

Insertion machine with interactive driver operates as an agent inserted into external environment with insertion function defining the laws of functioning of this environment.

Interactive driver can be organized in a rather complex way. If it has criteria of successful functioning in external environment intellectual driver can accumulate the information about its past, develop the models of external environment, improve the algorithms of selecting actions to increase the level of successful functioning.

Analytical insertion machine as opposed to interactive one can consider different variants of making decision about performed actions, returning to choice points (as in logic programming) and consider different paths in the behavior tree of a model. The model of a system can include the model of external environment of this system, and the driver performance depends on the goals of insertion machine. In the general case analytical machine solves the problems by search of states, having the corresponding properties(goal states) or states in which given safety properties are violated.

2.2 The Decomposition of Petri Nets into the Composition of Agents and Environments

Petri net (place/transition net) is one of several mathematical modeling languages for the description of distributed systems. A Petri net is a directed bipartite graph, in which the nodes represent transitions (i.e. events that may occur, signified by bars), and places (i.e. conditions, signified by circles). The directed arcs describe which places are pre/and/or postconditions for which transitions (signified by arrows). Like the industry standards such as UML activity diagrams, BPMN and EPCs, Petri nets offer graphical notation for stepwise processes that include choice, iteration, and concurrent execution. Unlike these standards, Petri nets have an exact mathematical definition of their execution semantics, with a well-developed theory for process analysis.

A Petri net consists of places, transitions, and arcs. Arcs run from a place to transition or vice versa, newer between places or between transitions. The places from which an arc runs to a transition are called the input places of the transition; the places to which arcs run from a transition are called the output places of the transition. Places in a Petri net may contain a discrete number of marks called tokens. Any distribution of tokens over the places will represent a configuration of the net called a marking. In abstract sense relating to Petri nets diagram, a transition of a Petri net may fire whenever there are sufficient tokens at the start of all input arcs; when it fires, it consumes this tokens, and places them at the end of all output arcs. A firing is atomic, i.e., a single non-interruptible step.

Execution of Petri nets s nondeterministic: when multiple transitions are enabled at the same time, any of them may fire, so multiple tokens may be represented anywhere in the net (even in the same place). Petri nets are well suited for modeling the concurrent behavior of distributed systems.

Petri nets are formally defined as a state-transition systems that extend a class of nets called elementary nets. Formal definition is represented lower.

1. P is a set of states, called places.
2. T is a set of transitions
3. F where $F \subset (P \times T) \cup (T \times P)$ is a set of relations called arcs
4. $N = (P, T, F)$ is a net
5. C is such that $C \subseteq P$ is a configuration
6. M so that $M : P \to Z$ is a place multiset, where Z is a countable set.
7. W so that $W : F \to Z$ is an arc multiset
8. $PN = (N, M, W)$ is a Petri net

Fig 2. Formal definition of Petri net.

Petri net is bipartite graph, where P is one partition and T is the other. Moreover, for every t in T there exist p and q in P so that (p, t) and (t, q) are in F, and for every p and q in P, if (p, t) and (t, q) are in F then p≠q.

The set are the new elements. The set of places define the local states of a net, however, the global state of a net can be defined by place subsets.

In order to represent Petri nets as a composition of agents and environments, we represent transitions T as actions, tokens as agents, and places as states. The behavior of tokens located in the enabled place s is written as:

$$u(s) = \sum_{W(s,t)>0} t.\Delta \tag{6}$$

The states of Petri environment are equal to the marks (configurations) of the net.

$$M : S \rightarrow Nat \Leftrightarrow E(M) = \| \{u(s) |^{M(s)} | s \in S\} \tag{7}$$

The insertion function is defined as:

$$E(M)[u] = E(M) \| u \tag{8}$$

The environment transitions are defined as:

$$\frac{u \xrightarrow{t} u'}{E[u] \xrightarrow{t} E[u']} \tag{9}$$

3 Combinatorial Strand Algebra

Strand algebra is a process algebra [18] where the main components represent DNA strands, DNA gates, and their interactions. The basic algebra is non-deterministic algebra, and the further extension is a stochastic variant [10]. Strand algebras may look very similar to either chemical reactions, or Petri nets, or multiset-rewriting systems. The difference here is that the equivalent of, respectively, reactions, transitions, and rewrites, do not live outside the system, but rather are part of the system itself and are consumed by their own activity, reflecting their DNA implementation. A process algebra formulation is particularly appropriate for such an internal representation of active elements.

The Combinatorial Strand Algebra, P – basic strand algebra has some atomic elements (signals and gates), and only two combinators: parallel (concurrent) composition $P | Q$, and populations P^*. An inexhaustible population P^* has the property that $P^* = P | P^*$; that is, there is always one more P that can be taken from the population. The set P is formally the set of finite trees P generated by the syntax shown below; we freely use parentheses when representing these trees linearly as strings. Up to the algebraic equations described below, each P is a multiset, i.e., a solution. The signals x,y,... are taken from a countable set.

$$P ::= x; [x_1,...,x_n].[y_1,...,y_m]; 0; P_1 | P_2; P^* n \geq 1, m \geq 0 \tag{10}$$

A gate is an operator from signals to signals: $[x_1,...,x_n].[y_1,...,y_m]$ is a gate that binds signals $x_1,...,x_n$, produces signals $y_1,...,y_m$, and is consumed in the process. We say that this gate joins n signals and then forks m signals; some special cases are shown on the fig 4. An inert component is indicated by 0. Signals and gates can be combined into a `soup' by parallel composition $P_1 | P_2$ (a commutative and associative operator, similar to chemical `+'), and can also be assembled into inexhaustible populations, P^*. Square brackets are omitted for single inputs or outputs.
Explanation of the Syntax and Abbreviations:

$$x \tag{11}$$

Signal

$$0 \tag{12}$$

Inert

$$x_1.x_2 \triangleq [x_1].[x_2] \tag{13}$$

transduser gate

$$P_1 \mid P_2 \tag{14}$$

Composition

$$x.[x_1,...,x_m] \triangleq [x].[x_1,...,x_m] \tag{15}$$

fork gate

$$P^* \tag{16}$$

Population

$$[x_1,...,x_m].x \triangleq [x_1,...,x_m].[x]$$

The relation $\equiv \subseteq P \times P$, called mixing, is the smallest relation satisfying the following properties; it is a substitutive equivalence relation axiomatizing a well-mixed solution[3]given lower:

$$P \equiv P \tag{17}$$

Equivalence

$$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R \tag{18}$$

Congruence

$$P \equiv Q \Rightarrow Q \equiv P \tag{19}$$

$$P \equiv Q \Rightarrow P^* \equiv Q^* \tag{20}$$

$$P \equiv Q, Q \equiv R \Rightarrow P \equiv R \tag{21}$$

$$P^* \equiv P^* \mid P \tag{22}$$

population

$$P \mid 0 \equiv P \tag{23}$$

Diffusion

$$0^* \equiv 0 \tag{24}$$

$$P \mid Q \equiv Q \mid P \tag{25}$$

$$(P \mid Q)^* \equiv P^* \mid Q^* \tag{26}$$

$$P \mid (Q \mid R) \equiv (P \mid Q) \mid R \tag{27}$$

$$P^{**} \equiv P^* \tag{28}$$

The relation $\rightarrow \subseteq P \times P$, called reaction, is the smallest relation satisfying the following properties. In addition, $\rightarrow^*$, reaction sequence, is the symmetric and transitive closure of $\rightarrow$. Reaction is shown lower:

$$x_1 \mid .. \mid x_n \mid [x_1,...,x_n].[y_1,...,y_m] \rightarrow y_1 \mid .. \mid y_m \text{ gate } \quad n \geq 1, m \geq 0 \tag{29}$$

$$P \rightarrow Q \Rightarrow P \mid R \rightarrow Q \mid R \tag{30}$$

Dilution

$$P \equiv P', P' \rightarrow Q', Q' \equiv Q \Rightarrow P \rightarrow Q \tag{31}$$

well mixing

The first reaction (gate) forms the core of the semantics: the other rules allow reactions to happen in context. Note that the special case of the gate rule for m = 0 is $x_1 \mid .. \mid x_n \mid [x_1,...,x_n].[] \rightarrow 0$. And, in particular, x.[] annihilates an x signal. We can choose any association of operators in the formal gate rule: because of the associativity of parallel composition under the exact choice is not important. Since is a relation, reactions are in general nondeterministic.

Note that signals can interact with gates but signals cannot interact with signals, nor gates with gates. As we shall see, in the DNA implementation the input part of a gate is the Watson-Crick dual of the corresponding signal strand, so that the inputs are always `negative' and the outputs are always `positive'. This Watson-Crick duality need not be exposed in the syntax: it is implicit in the separation between signals and gates, so we use the same x1 both for the `positive' signal strand and for the complementary `negative' gate input in a reaction like $x_1 \mid x_1.x_2 \rightarrow x_2$.

4 Insertion Machine for Combinatorial Strand Algebra

The representation of Petri nets as a composition of agents and environments was discussed in section 2.2. This will allow us to build a model driver based on the Petri nets. Consider a place-transition Petri Net with places xi; then, a transition with incoming arcs from places $x_1..x_n$,and outgoing arcs to places $y_1..y_m$ is represented in the combinatorial strand algebra as $([x_1..x_n].[y_1..y_m])^*$, where an unbounded population of gates ensures that the transition can fire repeatedly. The initial token marking $x_1,...,x_k$ (a multiset of places) is represented as $x_1 \mid .. \mid x_k$. Conversely, a signal in strand algebra can be represented as a marked place in a Petri net, and a gate $[x_1,...x_n].[y_1..y_m]$ as a transition with an additional marked `one-shot' place on the input that makes it fire only once; then, P^* can be represented by connecting the transitions of P to refresh the one-shot places (this was suggested by Cosimo Laneve). Therefore, the combinatorial strand algebra is equivalent to place-transition Petri nets, and can be easily implemented into the Insertion Modeling System, by using the model driver based on the Petri nets.

Conclusions

Strand algebras in general would allow the compilation of a high-level formalism into the DNA structures, using the methods advised by Cardelli in [10]. We have shown the way of implementation of the basic strand algebra – combinatorial strand algebra, in the Insertion Modeling System, by constructing the model driver for the system, based on the Petri nets. Combinatorial strand algebra deals with countable sets of signals/gates and so on(as well as Petri nets), we can extend it in future, to make it able to handle infinite sets, using the possibilities of insertion modeling, that works with infinite models. Implementation of combinatorial strand algebra to the insertion modeling system can be considered as the first step for building the insertion models of biological systems. However the further extensions of combinatorial strand algebra: Nested strand algebra [6], and Stochastic strand algebra, require a constructing of a probabilistic model driver, in order to implement them in the Insertion Modeling System. The stochastic semantics can be taken for example from the Stochastic Petri nets, which are just nets with rates on transitions and with an induced Continuous Time Markov Chain semantics.

References

1. Alexander Letichevsky, Olexander Letichevskyi, Vladimir Peschanenko, Igor Blinov and Dmitriy Klionov: (en) Insertion Modeling System And Constraint Programming. In: Ermolayev, V. et al. (eds.) Proc. 7-th Int. Conf. ICTERI 2011, Kherson, Ukraine, May 4-7, 2011, CEUR-WS.org/Vol-716, ISSN 1613-0073, pp. 51-64 (2011), http://ceur-ws.org/Vol-716
2. Gilbert D.R., Letichevsky A.A. : A universal interpreter for nondeterministic concurrent programming languages.In M. Gabbrielli (eds.), Fifth Compulog network area meeting on language design and semantic analysis methods (1996).
3. Letichevsky A.A., D.R. Gilbert: A general theory of action languages.Cybernetics and System Analyses, vol. 1, pp. 16--36 (1998).
4. Letichevsky A.A. and Gilbert D.R.: A Model for Interaction of Agents and Environments. In D. Bert, C. Choppy, P. Moses, (eds.). Recent Trends in Algebraic Development Techniques. LNCS, vol. 1827, pp.11--58. Springer (1999).
5. Letichevsky A.A.: Algebra of behavior transformations and its applications. In V.B.Kudryavtsev and I.G.Rosenberg (eds). Structural theory of Automata, Semigroups, and Universal Algebra, NATO Science Series II. Mathematics, Physics and Chemistry, vol. 207, pp. 241--272. Springer (2005).
6. Baranov S., Jervis C., Kotlyarov V., Letichevsky A., and Weigert T.: Leveraging UML to Deliver Correct Telecom Applications. In L. Lavagno, G. Martin, B.Selic (eds.). UML for Real: Design of Embedded Real-Time Systems. Kluwer Academic Publishers. Amsterdam (2003).
7. Letichevsky A., Kapitonova J., Letichevsky A. Jr., Volkov V., Baranov S., Kotlyarov V., Weigert T.: Basic Protocols, Message Sequence Charts, and the Verification of Requirements Specifications. Computer Networks, vol. 47, 662–675 (2005).
8. Kapitonova J., Letichevsky A., Volkov V., and Weigert T.: Validation of Embedded Systems. In R. Zurawski, (eds.). The Embedded Systems Handbook, CRC Press, Miami (2005).
9. Letichevsky A., Kapitonova J., Volkov V., Letichevsky A., jr., Baranov S., Kotlyarov V., Weigert T.: System Specification with Basic Protocols. Cybernetics and System Analyses, vol. 4, pp. 479--493 (2005).
10. Cardelli L.: Strand Algebras for DNA Computing (Preliminary version). DNA Computing and Molecular Programming, 15th International Conference, DNA 15. LNCS 5877:12-24, Springer (2009).
11. P. Yin, H. M.T. Choi, Calvert C.R.,Pierce N.A.: Programming Biomolecular Selfassembly Pathways. Nature, 451:318-322 (2008).
12. Soloveichik D., Seelig G., Winfree E.: DNA as a Universal Substrate for Chemical Kinetics. PNAS, March 4, 2010, doi: 10.1073/pnas.0909380107.
13. Cardelli L.: On Process Rate Semantics. Theoretical Computer Science 391(3) 190-215,
14. Marathe A., Condon A.E., R.M.: Corn. On Combinatorial DNA Word Design. J. Comp. Biology 8(3), vol. pp. 201--219, (2001).
15. Milner R.: Communicating and Mobile Systems: The π-Calculus. Cambridge University Press (1999)
16. Danos V., Laneve C.: Formal molecular biology. Theoretical Computer Science 325(1) pp. 69-110 (2004)
17. Regev A., Panina E.M., Silverman W., Cardelli L., Shapiro E.: BioAmbients: An Abstraction for Biological Compartments. Theoretical Computer Science, vol. 325(1), pp. 141--167 (2004)
18. Cardelli L.: Artificial Biochemistry. In: A. Condon, D. Harel, J.N. Kok, A. Salomaa, E.Winfree (eds.). Algorithmic Bioprocesses. Springer (2009)