Assemblies as Graph Processes

Dirk Janssens

Abstract. This short paper explores the potential of embedding-based graph rewriting as a tool for understanding natural computing, and in particular self-assembly. The basic point of view is that aggregation steps in self-assembly can be adequately described by graph rewriting steps in an embedding-based graph transformation system: the building blocks of an assembly correspond to occurrences of rewriting rules, and hence assemblies correspond to graph processes. However, meaningful algorithms do not consist only of aggregation steps, but also of global steps in which assemblies are modified. A theoretical algorithm is presented in which the two kinds of steps are combined: on the one hand aggregation steps that build assemblies, and on the other hand global steps which act on the assemblies.

1 Introduction

The study of self-assembly has been an interesting and promising part of the fascinating area of natural computing for several years [WLWS98,KR08,Cas06]. The phenomenon is an important aspect of biological systems [ETP+04] and has potential applications in nanotechnology, chemistry and material sciences [GJC91]. The basic idea is that components such as molecules or proteins aggregate to form assemblies that have interesting emerging properties which are not present in the original components. It is obviously important to control this aggregation process, i.e. we want to be able to design the building blocks in such a way that certain a priori known structures emerge as a result of spontaneous aggregation. These structures may in their turn interact in a meaningful way with other components.

The basic step in an assembly process is sketched in Figure 1: two components (left) aggregate to form an assembly (right). It is assumed that this happens because there is a particular relationship between their surface structures: these contain active parts (bold segments) that spontaneously stick together; one may think of atoms or molecules that form bonds between them, like in the case of Watson-Crick complementarity. Components will be called assemblies whenever we want to stress that they are built by self-assembly.

The use of graph rewriting [EKMR97,EEPT] as a tool for studying natural computing and self-assembly has been explored before [HLP08,KGL04]. However there are a lot of possible directions to follow because of the variety of processes that need to be described as well as the variety of graph rewriting mechanisms. The aim of this paper is to explore, in a very preliminary and perhaps naive way, how the work on graph rewriting with embedding and the corresponding theory of graph processes from, e.g., [VJ02] can be used in this context. It turns out that components, surface structure and assemblies correspond to rules, graphs and graph processes, respectively.

F. Drewes, A. Habel, B. Hoffmann, D. Plump (Eds.): Manipulation of Graphs, Algebras and Pictures. Essays Dedicated to Hans-Jörg Kreowski on the Occasion of His 60th Birthday, pp. 191–203, 2009.

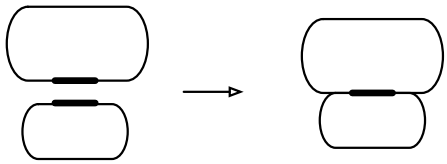

Fig. 1. Aggregation

The fact that both the surface structure and the assemblies are described within the same formal framework enables one to switch between the graph/surface structure and the process/assembly view: the former allows one to describe the aggregation steps (combination of various assemblies based on their surface structure) whereas the latter allows one to describe global steps, caused by external manipulation (heating, extraction, ...). The first kind of step occurs when a large amount of simple building blocks (molecules) is put into a solution and allowed to form assemblies; the latter kind happens in response to other manipulations acting in a uniform way on the assemblies as a whole (e.g. partially breaking them down). We present an algorithm in which the two kinds of steps are combined; it (theoretically, at least) allows one to recognize the difference between two solutions, a "pure" one and one that is contaminated with one or more extra (but unknown) components, by building assemblies that encode the contaminating components (but only those). These assemblies can then be extracted and used to mark the contaminating components, so that they can be removed from the contaminated solution.

In Section 2 the basic assumptions underlying this work are given, and the relationship is discussed between components, surface structure and assemblies on the one hand and rules, graphs and graph processes on the other hand. The example algorithm is presented in Section 3 and the paper ends by a brief discussion section.

2 Components, rules and processes

In this section the basic assumptions of the approach are given, and the necessary elements of graph rewriting, embedding mechanism and processes are briefly sketched. A formal treatment can be found in, e.g., [VJ02].

2.1 Basic assumptions

A graph transformation system consist essentially of a set of *rules* that describe local changes applied to graphs. Traditionally, a rule has a *left-hand side* and a *right-hand side*, which are both also graphs. A rule is applied to a graph g by matching its left-hand side with a subgraph of g. That subgraph is then removed and replaced by the right-hand side. In the approach used in this paper the rules

are equipped with additional information, called "embedding mechanism", which is used to determine the edges of the resulting graph.

The way graph rewriting is related to aggregation is the following. Consider an assembly step, like the one depicted in the upper part of Figure 2: an existing assembly (top) gets larger by aggregating with a building block (bottom). The surface structure of the former is represented by a graph g_1 with nodes a, b, c, d, e. The aggregation leads to a larger assembly with a modified surface structure, represented by a graph g_2 with nodes c, d, e, f, g, h. Thus the effect of adding the new building block on the surface structures is that g_1 is changed into g_2, in a way that can be captured by graph rewriting: the building block is viewed as a graph rewriting rule and the aggregation step corresponds to its application, removing the nodes a, b and replacing them by f, g, h. Evidently one also has to deal with the edges, which represent relationships between surface elements. We come back to this in subsection 2.2. The approach entails the following three assumptions concerning aggregation.

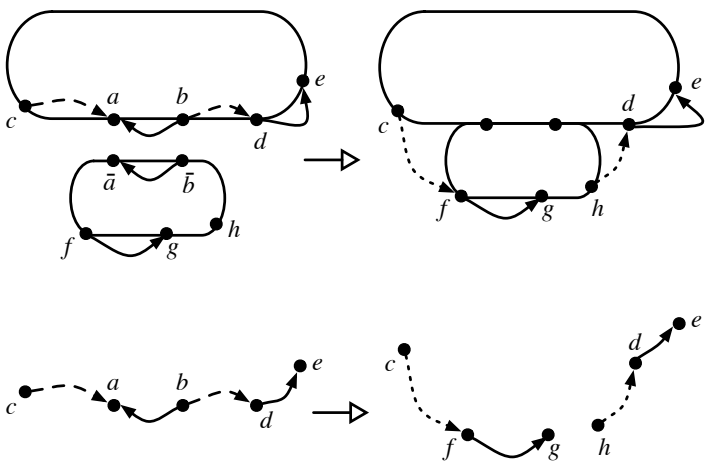

Fig. 2. Aggregation and graph transformation

The first working hypothesis is that the surface structure of components, which governs the way in which they aggregate, can be adequately described as a labeled graph. The nodes represent atoms, molecules, ... that are present at specific locations on the surface, the node labels distinguish between various kinds of such surface elements, and the edges describe relationships between these elements that are important for determining whether a group of nodes is active in the sense that it causes aggregation. One may think of spatial relationships or vicinity, but there may be others. In Figure 2 the symbols $\overline{a}$ and $\overline{b}$ are used to indicate the fact that binding or aggregation between physical components is

caused by elements that are "complementary" in some sense (e.g. having opposite polarities, Watson-Crick complements, ...). In our approach this information is implicitly represented by the fact that building blocks are described by graph rewriting rules which have a designated left-hand side. This is why we will not use complementary labels such as a and $\overline{a}$ in the next section; the usual notion of matching suffices.

The second hypothesis is that the relevant relationships between the elements of the new surface (i.e., the new edges) can be determined from (1) the surface of the existing component and (2) the new component. Thus the components, such as the ones depicted in the left part of Figure 1 will not be treated equally: one of them (the upper one in the figures) may be thought of as a large assembly that grows by aggregating with the other one, which is small and simple. Hence the large assembly "grows" by adding a new building block. As a result of this, the building blocks of an assembly are partially ordered, making them similar to graph processes. The surface of the assembly after the aggregation step consists of most of the "old" surface combined with a small, new part that belongs to the building block. It is assumed that in determining the new surface structure, one does not need the entire internal structure of the large assembly. The upper half of Figure 2 depicts an aggregation step where the surface structures are graphs. Technically the letters $a, b, \ldots$ are node labels, but throughout this section we need not to distinguish between nodes and their label. The lower half of Figure 2 depicts the transformation of the surface structure, which is now a graph transformation.

A last assumption is that the effect of an aggregation step is *local*: a surface element that is irrelevant for a location does not suddenly become relevant when an aggregation takes place involving that location: e.g. in Figure 2, e is not relevant to the locations a and b involved in the aggregation – and thus e is not connected to either of them. In terms of graph rewriting, the assumption means that the newly introduced nodes can only be connected to those existing nodes that are neighbors of the nodes removed by the rewriting. Thus the neighbors of the new nodes f, g, h are either also new or chosen among the "old" neighbors c, d of a and b.

2.2 The embedding mechanism

The lower half of Figure 2 depicts the change in surface structure that corresponds to the aggregation step in the upper half of the figure. This change can be described by the application of a graph rewriting rule to the graph on the left: the rule removes nodes a, b and creates f, g, h. The edges of the new surface are either edges of the old surface, such as (d, e), or edges of the surface of the newly added building block, such a (f, g), or edges that connect the new nodes with the old ones, such as (c, f) or (h, d). The mechanism for establishing the latter kind of edges is known as an *embedding mechanism*. Here the embedding mechanism is very simple: each of the new nodes may take over the incoming and/or outgoing edges of one or more of the nodes that have been removed. In

Figure 2, f takes over the incoming edges from a and h takes over the outgoing edges of b. The rule applied is depicted in Figure 3: it consists of two graphs (the left-hand side and the right-hand side) and two binary relations *in* and *out* that express the way edges are established. In Figure 3 both relations contain only one pair; in general $in, out \subseteq N_L \times N_R$ where N_L and N_R are the sets of nodes of the left-hand side and the right-hand side, respectively.

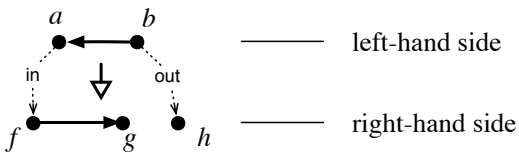

Fig. 3. A rule

2.3 Processes

In [CMR96,VJ02] graph processes are proposed as a way to describe "runs" of a graph rewriting system. Informally, a graph process is a structure obtained by gluing together the rule occurrences of a run, where the gluing is consistent with the way the rules are applied. Thus a graph process is essentially a directed acyclic graph where the nodes are those that occur in the run and where the edges represent the direct causal dependency relation: whenever a rule occurrence removes a node a and introduces a new node b, then b is directly causally dependent on a. This DAG is further decorated with extra information: the initial graph of the run is given as well as the rule occurrences. However there is no information on the order in which the rules are applied other than the causality relation. Figure 4 depicts a process: the initial graph is the linear structure at the top and there are three occurrences of the rule depicted at the left. There is no information about the relative order of rule occurrences 1 and 2, and so the process describes in fact three possible runs: the three rule occurrences may happen either in the order 1,2,3, or 2,1,3, or 1 and 2 may happen simultaneously, followed by 3. The dotted edges are established according to the embedding mechanism.

Using this notion one has three ways to view the components that act as building blocks in aggregation steps such as the one considered in Figure 2: a component with a surface structure, a graph rewriting rule, and a process. Since such components are not composed of smaller ones they are called "atomic". Similarly, the processes that represent a single rule are called atomic processes. Figure 5 depicts the three views; the arrows/lines labeled *in* and *out* represent the embedding mechanism.

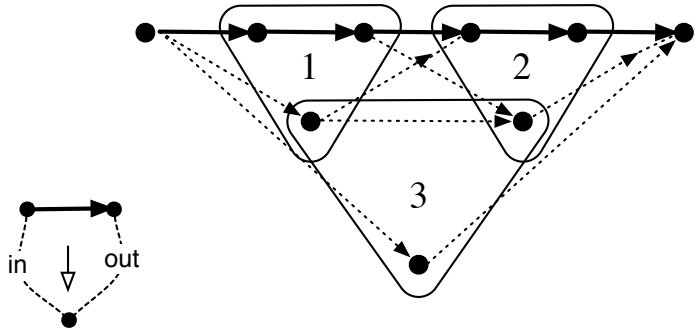

Fig. 4. Process

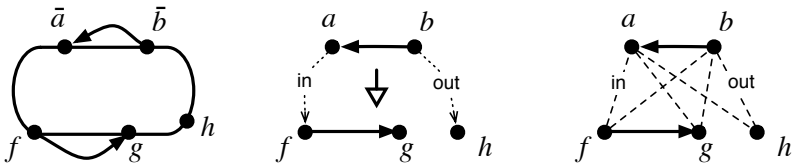

Fig. 5. Atomic component, rule and atomic process

The relationship between processes and assemblies is illustrated in Figure 6 (in the process, on the right, only the direct causality relation is represented). An important property of processes is that, when the embedding satisfies certain conditions, each slice (maximal set of causally unrelated nodes) *uniquely* determines a graph corresponding to that slice: the graph obtained by applying the rule occurrences that precede the slice in the causality relation to the initial graph – in any order consistent with the causality relation. In particular, the graph resulting from the process is uniquely determined; it is the configuration corresponding to the set of maximal nodes of the causality relation. It has been proven earlier [VJ02] that the embedding mechanism used in this paper satisfies the necessary condition. In Figure 6, one may e.g. consider the situation of the assembly after building blocks 1 and 2 are added. The corresponding slice consists of the square nodes. The property then means that the surface structure at this point of the aggregation process does not depend on the order in which blocks 1 and 2 were added; a posteriori inspection of an assembly (which blocks are present and how are they glued together) suffices to determine its surface structure. Since the order in which the aggregation takes place would probably be very hard to control, this property is of crucial importance.

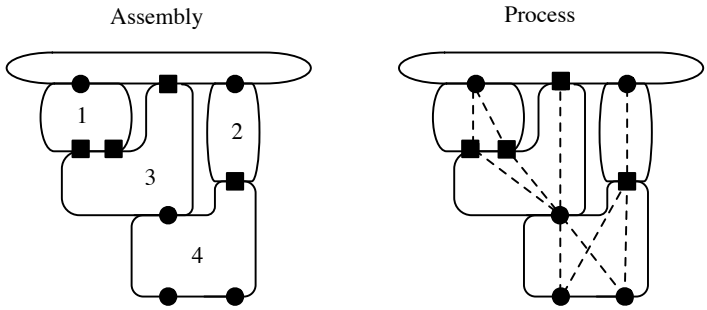

Fig. 6. Assembly and process

3 The algorithm

The aim of this section is to sketch how aggregation steps (building assemblies) and global steps (acting in a uniform way on all components) may be combined into a meaningful algorithm. In the context of this paper an algorithm describes a sequence of steps in which test tubes containing a solution are manipulated in order to obtain a solution with certain desired properties. One important way to manipulate a test tube is to cause a self-assembly process in it: atomic components (encoding graph transformation rules) are added to the test tube and self-assembly is allowed to happen.

The design of aggregation steps is now viewed as the design of a suitable set of graph transformation rules, and thus it is implicitly assumed that for each of those rules a component can be constructed which has the right surface structure, and that this component interacts in the right way with the other components. It has to be noted that the latter assumption is not obvious; however there is evidence that unintended interactions can be made improbable by a clever design of the components. The problem is similar to the DNA code word problem, which is an interesting research topic on its own.

The global steps, where all components in a solution are modified in a uniform way, are not local changes based on the surface structure of components, and thus the rewriting of graphs describing their surface structure is not a natural way to formalize them. However the more complete description of an assembly by a graph process provides information that is sufficient to express the global steps: the atomic components it is built from and the way they are combined. Two kinds of global steps are needed.

1. *extract*(m), where the symbol m represents a marker, i.e. a part of a component that can easily be detected by its physical properties. The operation removes all components in which the marker occurs from the test tube.
2. *disassemble*(R), where R is a set of rules used for aggregation. The operation removes all atomic components corresponding to rules of R from the assem-

blies in the test tube. Hence this operation may cause assemblies to fall apart into smaller pieces. The physical implementation would be a manipulation that breaks down the components corresponding to R, and only those. This could be achieved by designing these components in such a way that they are less stable or less resistant to heat than the other components.

The problem we focus on is the following. Consider two test tubes X and Y; Y contains the same components as X but also some additional ones; these are viewed as a contamination that has to be removed. The algorithm has to recognize the contaminating components, by yielding a test tube containing assemblies that encode the latter, where "encoding" means that their surface structure contains a copy, up to a relabeling, of that of the encoded components. The relabeling is needed to distinguish the encoding from the original.

It is assumed that only some of the components are relevant; in the example these have the surface structure depicted in Figure 7, where the x_i belong to the set $\{a, b\}$. The symbols a, b, l, r represent certain kinds of surface elements and n is even. The symbols a, b, l, r are relabeled $\tilde{a}, \tilde{b}, \tilde{l}, \tilde{r}$ in the encoding.

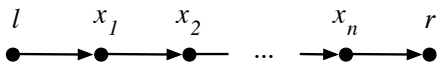

Fig. 7. Initial structure

For our purposes X and Y are sets of concrete structures, And obviously both of them will in general contain many isomorphic copies of each of their elements.

The algorithm consists of the following steps:

1. Recognize the relevant components of Y: prepare their encoding by forming a suitable assembly around each of them.
2. Extract these assemblies from Y.
3. Add the result of this to X and form assemblies that mark the encodings of components that occur in X.
4. Extract the marked encodings; the remaining ones are the desired ones.

To realize step 1, first ignore the relabeling. Then the step can be carried out by adding to Y the rules (i.e. components realizing the rules) of Figure 8: these form an assembly that is essentially a binary tree where the leaves are labeled a or b and each two consecutive leaves have the same parent.

The graph process in the upper part of Figure 9 represents an intermediate stage in the formation of such an assembly: one more step (applied to the two square nodes) will complete the tree. Only if the component is of the right form

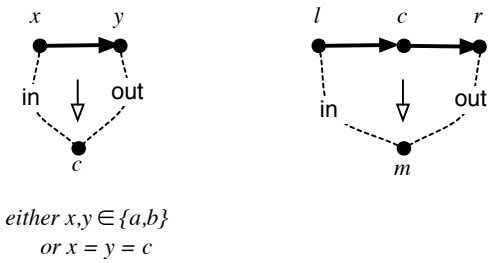

Fig. 8. Rules for assembly in step 1

the process can be completed by an application of the rule at the right of Figure 8 which attaches the marker m to the assembly (lower left part of Figure 9). The relabeling of a, b, l, r into $\tilde{a}, \tilde{b}, \tilde{l}, \tilde{r}$ can be taken into account by modifying the rules in the way depicted in Figure 10: a relabeled copy of the encoded structure is produced.

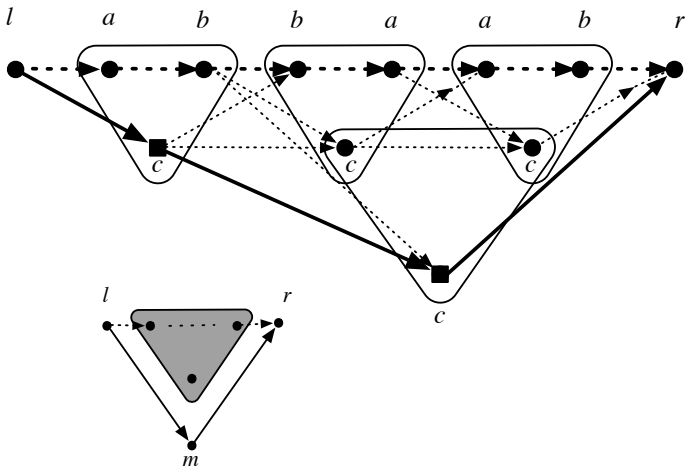

Fig. 9. Assembly in step 1

Step 2 can be realized by an *extract*(m) operation. A potential problem is that the rules of step 1 can be applied to a component of the right form in such a way that the assembly obtained is a forest, but not a tree; then that component is not encoded. However, one may improve the result by using an

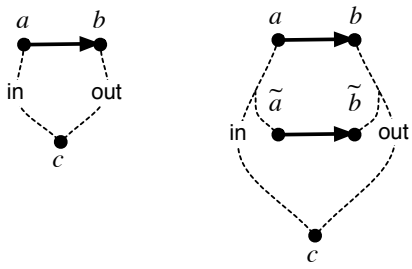

Fig. 10. Modified rule

iterative procedure: repeat the sequence (step 1, *extract*(m), *disassemble*(R_1)), where R_1 is the set of rules of step 1, until nothing is extracted.

Step 3 can be realized by adding the assemblies extracted in step 2 and the rules of Figure 11 to X. The effect is depicted in Figure 12: an encoding (starting with $\tilde{l}$) and a component (starting with l) are traversed, until L becomes adjacent to r and $\tilde{r}$. In that case the encoding is marked, using the rule at the right of Figure 11. The dotted edges in Figure 12 are established by the embedding mechanism. If the encoding and the component do not correspond, then L does not become adjacent to both r and $\tilde{r}$ and the marking does not occur. Step 4 can be done by an *extract* operation. Again, there is a potential problem because the assemblies with the encodings may get neutralized by trying to combine with the wrong component: e.g. when an assembly encoding *aaba* combines with component *aabba* the aggregation process of Figure 12 gets stuck after 4 steps. Again, an iterative procedure is needed: repeat the sequence (step 3, step 4, *disassemble*(R_3)) where R_3 is the set of rules of step 3.

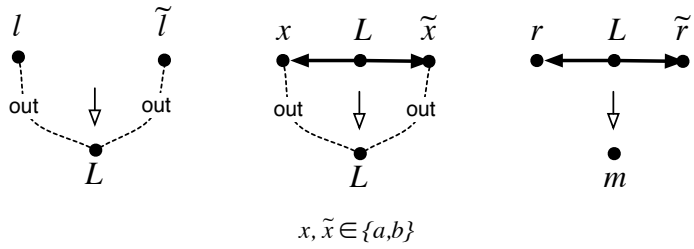

Fig. 11. Rules for traversal

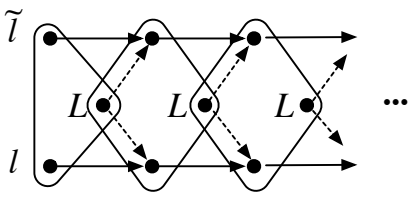

Fig. 12. Traversal - assembly view

The result of the algorithm is a test tube containing assemblies which encode the contaminants, i.e. the components of Y that do not occur in X. Using an iterative procedure similar to the one combining steps 3 and 4 above, it is in principle possible to use this to remove the contaminants from Y.

4 Discussion

The aim of the paper is to explore the use of graph rewriting based on embedding for the understanding of self-assembly and natural computing. The basic idea is that graphs capture the active surface structure that controls the way components in a solution aggregate, and that the way in which such aggregation changes the surface structure can be captured by graph rewriting. However one may expect that most meaningful algorithms in this context do not only require the building of ever larger assemblies, but also operations that break down or modify such assemblies, and in the algorithm sketched in Section 3 a few of these operations are used: extracting certain components according to particular "marker" labels, or removing certain atomic components from the assemblies in a solution. Thus what seems to be needed is an interplay between aggregation operations, which are described by graph transformation rules, and which act on the graphs that describe the active surface of components, and global operations in which all assemblies of a given kind in a solution are modified. Since assemblies correspond to processes of the graph rewriting systems that describe their formation, the theory of graph rewriting and graph processes may provide a way to obtain a formal framework in which both kinds of operations can be combined in an elegant way.

Obviously, the material presented here is of a very speculative nature, since the implicit assumptions concerning the possible realization of the approach in the physical world may turn out to be naive or unrealistic. To mention just a few: when reducing the problem of controlling self-assemby to the problem of writing a suitable graph transformation system, it is assumed that each rule written down can be realized by a component that behaves exactly in the right way: not only does it aggregate with another component when the structure corresponding

to its left-hand side matches part of the structure of the other component, but this is also the *only* way it interacts with other components. Another thorny issue is the assumption about the information to be encoded into the edges, information that is handled by the embedding mechanism: what are exactly the relationships between locations on a component that are relevant? How to encode spatial information into those edges? Also for the the global operations many questions remain: on the one hand they may seem rather ad-hoc, but on the other hand they are quite simple. In spite of these reservations, however, the correspondence between graph rewriting and graph processes on the one hand and aggregation and assemblies on the other hand seems simple and natural enough to deserve further attention.

References

[Cas06] Leandro N. De Castro. *Fundamentals of natural computing: basic concepts, algorithms, and applications.* Published by CRC Press, 2006.

[CMR96] A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundam. Inf.*, 26(3-4):241–265, 1996.

[EEPT] H. Ehrig, K. Ehrig, U. Prange, and G. Taenzer. *Fundamentals of Algebraic Graph Transformation.* EATCS Monographs in Theoretical Computer Science.

[EKMR97] H. Ehrig, H-J. Kreowski, U. Montanari, and G. Rozenberg. *Handbook of Graph grammars and Computing by Graph Transformation.* World Scientific, 1997.

[ETP+04] A. Ehrenfeucht, T.Harju, I. Petre, D.M. Prescott, and G. Rozenberg. *Computation in Living Cells - Gene Assembly in Ciliates.* Natural Computing Series. Springer Verlag, 2004.

[GJC91] GM.Whitesides, JP.Mathias, and CT.Seto. Molecular self-assembly and nanochemistry - a chemical strategy for the synthesis of nanotructures. *Science*, 254:1312–1319, 1991.

[HLP08] Tero Harju, Chang Li, and Ion Petre. Graph theoretic approach to parallel gene assembly. *Discrete Applied Mathematics*, 156(18):3416–3429, 2008.

[KGL04] Eric Klavins, Robert Ghrist, and David Lipsky. Graph grammars for self assembling robotic systems. In *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, pages 5293–5300, 2004.

[KR08] Lila Kari and Grzegorz Rozenberg. The many facets of natural computing. *Commun. ACM*, 51(10):72–83, 2008.

[VJ02] N. Verlinden and D. Janssens. Algebraic properties of processes for local action systems. *Mathematical. Structures in Comp. Sci.*, 12(4):423–448, 2002.

[WLWS98] E. Winfree, F. Liu, L.A. Wenzler, and N.C. Seeman. Design and self-assembly of two-dimensional dna crystals. *Nature*, (394):539–544, 1998.

Prof. Dr. Dirk Janssens

Department of Computer Science
University of Antwerp
2020 Antwerpen
Dirk.Janssens@ua.ac.be

Dirk Janssens and Hans-Jörg Kreowski have co-authored several papers. Moreover, Hans-Jörg Kreowski was in Dirk Janssens' Ph.D. jury in 1983. When being asked for their relation, Dirk points out that it has always been a pleasure to meet Hans-Jörg Kreowski, not only because of his inspiring scientific ideas and his way of concentrating on the right questions, but also because of his kindness and patience.