Designing Robot Metamorphosis

Anne C. van Rossum [a,b]   H. Jaap van den Herik [a]

[a] *Tilburg Centre for Cognition and Communication, Tilburg University, The Netherlands*
[b] *Almende B.V., Rotterdam, The Netherlands*

Abstract

In the recent past, modular robot assembly and metamorphosis has been *evolved* using gene regulatory networks. However, until now, no methodology exists to *engineer* such a regulatory network. Three existing representations will be employed to describe robot metamorphosis. A *graph rewriting grammar* describes *state and connectivity* transitions between robot organisms at the most abstract level. A *communicating finite state machine* introduces *messages* at an intermediate level. A *regulatory network* presents the process of metamorphosis at its least abstract level. In short, we present a *design methodology* for metamorphosis for which, as yet, only evolutionary methods did exist.

1 Introduction

Classical artificial intelligence tends to place unbridled emphasis on pattern recognition, keeping *pattern formation* on a leash. Patterns are formed in the case of snow flakes, spots on a leopard skin, and bodies growing from fertilized eggs to adult body forms. The field of *artificial morphogenesis* studies the development of a single (robot) cell toward an artificial organism.

The field of artificial morphogenesis converges upon a technique called *indirect encoding* to implement morphogenesis in a decentralized manner. Indirect encoding requires a *developmental process* from genotype to phenotype to grow a structure or topology. It would lead us too far to elaborate on the benefits of an indirect encoding scheme. We briefly summarize potential advantages: *compressibility* [7], exploitation of *output geometry* [5], robustness against *phenotypic "injuries"* [11], the ability to encode for *phenotypic plasticity*, and allowing for *epigenetic* factors [13] (the influence of the environment).

Hitherto, indirect encoding schemes have always been *evolved*. There have been no attempts to create a genome given an adult body form. We will describe such a *reverse path* from a *dynamic* body configuration toward a set of regulating entities — the genes. If an explicit design trajectory is available, it will be possible to (1) create benchmarks, (2) compare hand-coded regulatory networks with evolutionary search methods, and (3) bootstrap evolutionary search.

1.1 Replicator robots

The domain to which we will apply artificial morphogenesis or morphodynamics[1] is modular robotics. Modular robots are able to connect to each other to form large robot organisms. Imagine a bag full of modular robots, emptied in a collapsed mine. The robots assemble into larger organisms to navigate through the mine, move obstacles and find survivors. The diversity of challenges that the robots encounter asks for a diversity of body forms. Robot assembly and metamorphosis is part of a robotic challenge defined in the FP7 project Replicator [8]. In Fig. 1 the scenario is sketched. Different robot morphs are needed to overcome obstacles or climb on obstacles to obtain energy from power outlets. Our problem statement reads: *How to* ***design*** *a* ***gene regulatory network*** *for metamorphic robots?*

1.2 Overview

[1] The term morphodynamics originates from beach morphodynamics: sediments relentlessly changing sea-floor morphology. Here it denotes robots that do not just grow towards a *static* body shape, but that have a dynamic topology.

Robot metamorphosis is presented at the most abstract level as a *reconfiguration* problem in Section 2 to which effect a graph rewriting grammar is employed. At an intermediate level of abstraction, *communication* required to perform reconfiguration is solidified in Section 3. To this end communicating finite state machines are put forward. In Section 4, at the lowest level of abstraction, the content of communication is examined in the form of *regulatory* elements and circuits. Section 5 contains our conclusion.

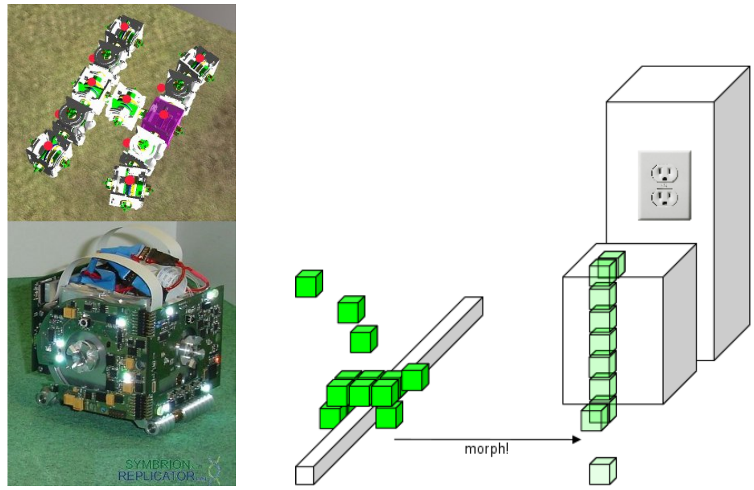

Figure 1: Left: the first prototype of the Replicator/Symbrion robot as well as a simulated organism. The robot contains four docking sites. This allows the robots to create 3D body forms. Right: a scenario for metamorphosis is sketched. A robot needs to overcome an obstacle by assembling into a certain body form. It subsequently needs to morph to another body form to reach a power outlet.

2 Reconfiguration

At the highest level of abstraction, a robot is represented by an undirected graph (Fig. 2). Transitions from one body form to the next are described in Eq. 1 by a *graph rewriting grammar*. Graph rewriting grammar has been used before to describe robot *assembly* (see Klavins in [9]). Here we (1) describe robot *metamorphosis*, (2) add a *connection matrix* (Subsection 2.1), and (3) establish additional *terminology* (Subsection 2.2) for the metamorphic case.

$$\Phi_1 = \begin{cases} r_0 : b \text{---} c & \Longrightarrow & b \quad e \\ r_1 : e \quad e & \Longrightarrow & e \overset{C_{00}}{\text{---}} f \\ r_2 : a \text{---} f & \Longrightarrow & f \text{---} b \\ r_3 : a \text{---} e & \Longrightarrow & d \text{---} b \\ r_4 : b \quad f & \Longrightarrow & b \overset{C_{02}}{\text{---}} b \end{cases} \qquad \Phi_2 = \begin{cases} u_0 : b \text{---} c & \Longrightarrow & b \quad e \\ u_1 : e \quad b & \Longrightarrow & b \overset{C_{00}}{\text{---}} b \\ u_2 : a \text{---} b & \Longrightarrow & f \text{---} b \\ u_3 : e \quad f & \Longrightarrow & d \overset{C_{02}}{\text{---}} b \\ u_4 : a \text{---} d & \Longrightarrow & d \text{---} b \end{cases} \tag{1}$$

Equation 1 shows two possible sets of rules to transition from the H-form of Fig. 2 to the snake form: Φ_1 and Φ_2. The rule sets are different, but lead to an equivalent result in the form of the snake. The rules are executed in a random order and possibly in parallel. Each label a through f stands for a robot module in a certain state. The grammar rules have a right-hand and a left-hand side. Both hands are limited to *two* robots to enforce pair-wise interactions. With a maximum of two elements per side, no coordination among three or more robot modules is needed to reorganize the robot topology. In other words, the reconfiguration problem can be solved locally.

Rule r_0 in Eq. 1 has a left-hand side which represents a module b connected to a module c. The execution of this rule causes those pairs to *disconnect* (no line is drawn at the right-hand side) and transforms c into e. This will not be a physical change, but a *state* change[2]. Starting with the H-topology in Fig. 2, the rule results in a collection of $b-b-b-b$ and $a-e$ chains, as well as intermediate forms derived from the original H-topology. In explicit terms, with this one rule, r_0, we can remove all the $a-c$ "legs" from the H-topology. Contrary to rule r_0, the rule r_1 in Eq. 1 has no line drawn at the left-hand side of the rule: $e \quad e$. Here we note in anticipation of Section 3 that such a rule involves *wireless* communication before a physical connection is made. Moreover, rule r_1 introduces an *indeterministic* element. One of the modules in state e turns into an f while the other remains the same. Using this feature, directionality can be introduced to the $a-e-f-a$ chains (the f "moves" to the extremity and binds to the $b-b-b-b$ chain).

[2]Observe that a state can also be used functionally. Modules in state b can have a different set of sensors turned on and off compared to modules in state a. For example, only the modules d in the snake in Fig. 2 might have their camera turned on.

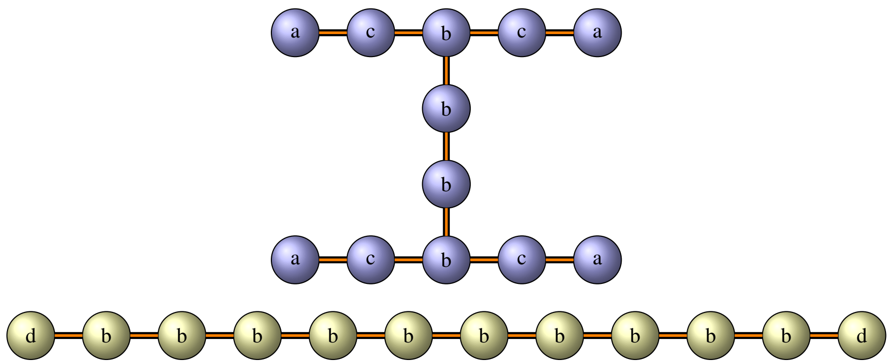

Figure 2: An H-shaped and a snake-like robot topology. Each robot is represented by a vertex with a maximum of four outgoing edges. Each edge corresponds to a connection. Labels on each vertex represent internal states. Graph rewriting rules define state transitions. The transition from the H-shaped topology to the snake topology is described by a set of graph rewriting rules.

2.1 Connection matrix

The ordinary graph grammar [9] is extended with an additional connection matrix. According to rule r_1, the connection matrix C (see [4]) defines how e and f are connected. There is only one connection possible between two modules, $\sum_{side=(i,j)=0,0}^{3,3} C_{i,j} = 1$, so we can represent this single connection by its row and column index C_{ij}. The matrix C_{00} means that both modules connect to each other with their locally defined coordinate system (for instance, both to the "north" side, from the four cardinal directions). Further details on matrix representations of robot configurations can be found in [4].

2.2 Reciprocity

In Eq. 2 a "reciprocal" set of both instances Φ_1 and Φ_2, called Υ_1, is described. This is the *reverse transition* from the snake to the H-figure in Fig. 2.

$$\Upsilon_1 = \begin{cases} s_0 : b \text{——} d & \Longrightarrow \quad a \text{——} c \\ s_1 : a \text{——} b & \Longrightarrow \quad a \quad\;\; e \\ s_2 : b \text{——} e & \Longrightarrow \quad f \text{——} e \\ s_3 : b \text{——} f & \Longrightarrow \quad g \quad\;\; e \\ s_4 : e \text{——} e & \Longrightarrow \quad a \text{——} c \\ s_5 : g \quad\;\; c & \Longrightarrow \quad g \xrightarrow{C_{10}} c \\ s_6 : g \quad\;\; c & \Longrightarrow \quad b \xrightarrow{C_{30}} c \end{cases} \tag{2}$$

A *reciprocal metamorphic pair* obeys $C = C\Phi\Upsilon$. The consecutive execution of Φ and Υ results in the same robot body form C.[3] The reciprocal set Υ_1 describes the transition from the snake form to the H-form. A robot that has the ability to morph from one form to the other needs to have (equivalents of) both sets in its repertoire. We note that the mere combination of Φ_1 or Φ_2 with Υ_1 will not necessarily lead to a *stable* outcome. The population of robots will also consist of all intermediate forms of robots.

It is possible to prevent the proliferation of intermediate body forms by introducing the concept of a *dual rule*. On inspection of Φ_1 and Υ_1 a dual rule can be found: the left-hand side of r_0 is equal to the right-hand side of s_6. In Fig. 2 rule r_0 corresponds to the removal of the $a - c$ legs from the H-figure, while the s_6 rule causes them to attach to the (shortened) snake body. The uncontrolled case executes all rules in Φ and Υ randomly. To reach a desired body shape, the dual rules need not to be executed by *chance*, but by external *control*. Control overhead can be reduced by designing such a rule set such that only *one* dual rule exist.

[3]This can be seen by performing the graph rewriting rules. A general proof of this property might be an interesting follow-up study.

3 Communication

At an intermediate level of abstraction we specify how to represent *communication* between robot modules. Biological cell communication can be seen, admittedly caricatured, as the exchange of protein vectors over time. Evolving an efficient communication scheme between entities is called language grounding [12]. In other words, inter-cell communication can be described as *cellular language grounding*. We represent the cell's interactions in the form of a *communication finite state machine* in Subsection 3.1. Moreover, a distinction is made between *wired* and *wireless* messages in Subsection 3.2 to be able to deduct the required number of message types from such a communicating state machine.

3.1 Communicating finite state machines

Each robot module can be represented by a finite state machine (FSM). Interactions between FSMs can be modeled by a *communicating* finite state machine [3]. The edges in a communicating FSM carry labels and, like in ordinary FSMs, denote state transitions. An edge can only be traversed when two communicating FSMs are in a synchronized state. Synchronization is defined by one FSM being in a state preceding an edge with label $r_i!$, while the other is in a state with an outgoing edge labeled $r_i?$. The r_i label stands for a communication channel. As soon as the FSM *writes* on $r_i!$, the other FSM *reads* on $r_i?$ and both undergo a transition to the next state. We introduce here the communicating FSM approach to model communication between robot modules.

In Fig. 4 the r_i labels correspond to the rule index in Φ_1. The !-tokens indicate modules that broadcast the message r_i. The $?$-tokens are robot modules that are listening to a certain message r_i. (A robot module cannot receive its own message.) For instance, a robot in state a broadcasts message $r_3!$ (see Φ_1 in Eq. 1). Another robot in state e receives this message. Subsequently robot a switches to d and e switches to b (see again Eq. 1). Observe that a module in state a can actually send a *composed* message $[r_2, r_3]$. This message can have two mutually exclusive effects. A module in state f might change to b, or a module in e might change to b (this corresponds to rule r_2 and r_3 in Φ_1).

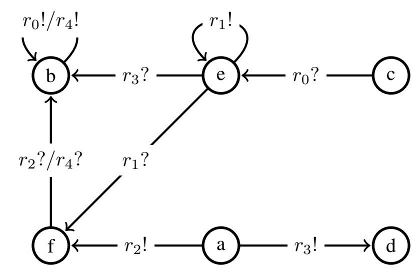

Figure 3: A communicating Finite State Machine (FSM) representation can be used to represent Φ_1. Each robot module contains a communicating FSM (so there are Z FSMs in total).

3.2 Wired and wireless communication

Figure 3 simplifies matters too much for a decision to be made on how many message types are required between communicating machines. First of all, a module in state a needs to send a composed message $[r_2, r_3]$. But, more importantly, the module in state b has to send a *wired* message r_0 in its connected b' state, and a *wireless* message r_4 in its disconnected b state (see Fig. 4).

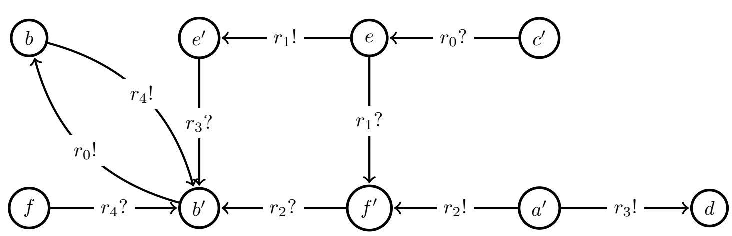

Figure 4: A communicating FSM with a distinction between modules in disconnected states $s \in S$ and connected states $s' \in S$. The number of messages needed to implement this scheme corresponds to the number of vertices with outgoing edges with !-tokens.

The number of messages required to implement the rewriting scheme Φ_1 corresponds to the number of vertices with outgoing edges with !-labels in Fig. 4. In this particular example, the vertex set is (a', b, b', e), so *four* messages are sufficient to implement the given instance of metamorphosis.

4 Regulation

This section describes the metamorphic process at the least abstract level: the *regulatory network* description. Regulatory networks have been used before for robot assembly in the form of gene regulatory networks [2]. We elaborate on this work by (1) introducing an *abstract representation* of a regulatory network in Subsection 4.1 and 4.2, (2) making explicit the relation with communicating FSMs by *coupled regulatory networks* in Subsection 4.3, and (3) opening up the possibility to design rather than evolve such a regulatory network by introducing two *elementary entities* (a) A NAND port, (b) a flip-flop in Subsection 4.4. Using those entities a regulatory network can be constructed that corresponds to the rule sets in Section 2. Due to its central role of being able to implement an indeterministic rule, a flip-flop is implemented in Subsection 4.5.

4.1 Regulatory network

For an introduction to gene regulatory networks, see [2]. Here we will briefly recapitulate its properties. A regulatory network is a tuple $\{G, P\}$. It consists out of a set of regulatory entities $(g_0, \dots, g_L) \in G$, and a set of regulated elements, artificial proteins, $(p_0, \dots, p_M) \in P$ with $p_i \in \mathbb{R}_{\geq 0}$. The proteins are updated asynchronously according to Eq. 3. We will use the shorthand p_i to indicate $p[t]_i$ and $\tilde{p}_i$ for $p[t + \Delta t]_i$ (with varying Δt). $\{G, P\}$ can be depicted as a graph, with P as the vertices and G denoting the edges.

$$U(\vec{g}_{ij}) : \tilde{p}_i = \Theta(p_i - \kappa + \sum_{j \in N_i} \Psi_{ij}) \quad with \quad \Psi_{ij} = \begin{cases} \gamma_{ij} & if \quad \alpha_{ij} < p_j < \beta_{ij} \\ -\gamma_{ij} & if \quad \beta_{ij} < p_j < \alpha_{ij} \\ 0 & else \end{cases} \tag{3}$$

The regulatory update rule U for a probabilistic chosen entity $\vec{g}_{ij}$ describes the ingoing edges from the vertices p_j in neighborhood N_i to vertex p_i. To calculate the new quantity of $\tilde{p}_i$ a non-linearity Ψ_{ij} is used. The edge $\vec{g}_{ij}$ is not a scalar, but a vector $[\alpha_{ij}, \beta_{ij}, \gamma_{ij}]$, with $\alpha_{ij}, \beta_{ij}, \gamma_{ij} \in \mathbb{R}_{\geq 0}$. The protein p_j can either up-regulate or down-regulate p_i. If $\alpha_{ij} < \beta_{ij}$ the protein p_i is up-regulated, if $\alpha_{ij} > \beta_{ij}$ it is down-regulated. The rate of regulation is defined by γ_{ij}. If the regulating entity does not fall in between the limits defined by α_{ij} and β_{ij} nothing happens. That is, the protein p_i is only decayed with a rate defined by κ. The proteins are capped by a sigmoid function, $\Theta(x)$ which enforces the protein quantity to be between θ_{min} and θ_{max}.

4.2 Types of regulators

We simplify the regulatory network by only admitting “low-pass” ($p < \alpha$) and “high-pass” ($p > \alpha$) protein quantity filters, rather than the “band-pass” filters in Ψ_{ij} in Eq. 3.

$$\Psi_{ij} = \begin{cases} \gamma_{ij}\epsilon_{ij}\dfrac{1 - \delta_{ij}}{2} & if \quad p_j > \alpha_{ij} \\ \gamma_{ij}\epsilon_{ij}\dfrac{1 + \delta_{ij}}{2} & if \quad p_j < \alpha_{ij} \\ 0 & else \end{cases} \tag{4}$$

Equation 4 only needs one threshold parameter, α_{ij}. The change in regulated protein is defined by $\gamma \in \mathbb{R}_{>0}$. There are four flavors of regulator types. The low-pass filter is defined by $\delta_{ij} = 1$, which returns a positive result for $p_j < \alpha_{ij}$. The high-pass filter is defined by $\delta_{ij} = -1$. Both filters can either increase or decrease the target product p_i. This is defined by $\epsilon_{ij} \in \{1, -1\}$. Let us introduce the following notation for the regulating entities: $R^{+\gamma}_{\top\alpha}$ is turned on at low protein quantities and is driving its target up ($\delta_{ij} = \epsilon_{ij} = 1$); $R^{-\gamma}_{\top\alpha}$ is turned on likewise, but inhibits its target ($\delta_{ij} = -\epsilon_{ij} = 1$); $R^{+\gamma}_{\perp\alpha}$ is turned on at high protein quantities and is driving its target up ($\delta_{ij} = -\epsilon_{ij} = -1$); $R^{-\gamma}_{\perp\alpha}$ is turned on likewise and inhibits its target ($\delta_{ij} = \epsilon_{ij} = -1$).

4.3 Coupled regulatory networks

The regulatory network abstraction allows us to describe one robot module. Now we will need to describe inter-module communication as explicated in Section 3. If each robot module is a dynamic system, this problem can be formulated as a synchronization problem for *coupled extended dynamical systems* [15]. The representation in Fig. 5 is different from coupled random boolean networks or Kauffman networks by the following five properties. (1) A node represents a protein p_i, an edge represents a gene (rather than a node representing a gene). (2) More than two states (or spins) per node. (3) A neighbor graph N_i per node, rather than fixing the number of neighbors to R for each node. (4) Incremental cross-coupling between networks ($\Delta p_i(M_1) = c(p_i(M_2))$ rather than $p_i(M_1) = p_i(M_2)$ with M_i a network, and c a coupling function). (5) An asynchronous updating rule for the edges.

The rational behind the listed properties is as follows. (1) Cells communicate by protein vectors. We are interested in *protein quantities* rather than in *gene activity*. (2) Protein quantities are real-valued, not binary. (3) A regulatory network needs to be *designed* and enforcing the same number of genes between two proteins introduces an unnecessary design constraint. (4) The same regulatory network exists in each robot module. If protein quantities are set to the same value in both modules, synchronization occurs and all robot modules will *converge to the same state*. Hence, the coupling is incremental (the protein quantity is increased or decreased) and cross-coupled ($r_x!$ in robot i influences $r_x?$ in robot j). (5) A synchronous updating rule — a global clock — seems to be *biased* to create rhythmic patterns [6] and such a bias we want to avoid.

Fig. 5 shows the representation of the rule r_3 from Eq. 1. To substantiate our case that it is possible to represent all rules with a regulating network, we present building blocks in the next subsection.

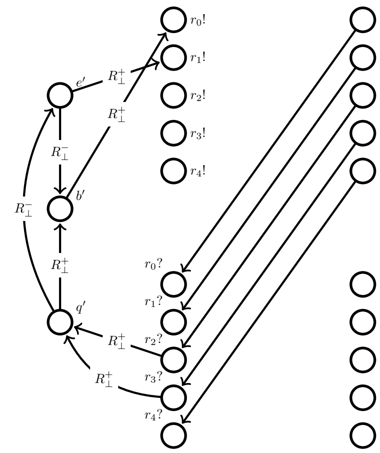

Figure 5: A regulating network corresponding to r_3. Two robot modules communicate with each other. The module on the left emits a signal belonging to state e. It is received by the module sketched on the right, which responds by sending a composed message $[r_2, r_3]$. From the picture it can be seen that this message will activate q', which will deactivate e'. The deactivation of e' has a dual effect. First the outgoing message $r_1!$ is prevented, and next the inhibition of b by e' stops. Together with the excitation from q' itself, b' will be activated, and when b' is activated it starts sending $r_0!$.

4.4 Network motifs

In [1] several *network motifs* are described, such as negative auto-regulation, positive auto-regulation, feed-forward loops and single-input modules. Here we will introduce two "network motifs" originating from digital circuit logic. (1) A logical NAND port and (2) a flip-flop. Those elements are created by the $R^+_\top$, $R^+_\perp$, $R^-_\top$ and $R^-_\perp$ regulators from Subsection 4.2.

First, we show that it is possible to implement a *logical NAND port*. A NAND port, also called the Sheffer stroke, is functionally complete and can be used to constitute a *logical formal system*. In other words, all the other boolean gates can be created by a concatenation of NAND operations.

In Fig. 6 a NAND port is visualized. Likewise, we can for example create a logical AND port can be created by the combination of an $R^{+2}_{\perp 80}$ edge, and an $R^{-2}_{\top 80}$ edge. Only when both quantities are above an upper threshold ($\alpha = 80$) the dis-inhibition of the latter edge stops.

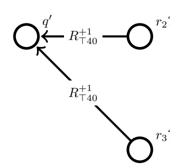

Figure 6: A NAND gate created by regulating network. Observe that it is essential that there is decay in the system. When $r_2?$ and $r_3?$ are both above a predefined threshold (here $\alpha = 40$ on a scale of $\theta_{max} = 100$) both edges will be disabled. Now q' will decay with a rate defined by κ in Eq. 3.

Second, we describe the *flip-flop*. The flip-flop is a bistable circuit with two states and a switching mechanism. It stores one bit of memory. A rule like r_1 in Eq. 1 requires such a bistable circuit. As stated before, this rule is indeterministic and can either cause a transition of e toward e' or respectively toward f'. Moreover, the flip-flop deserves some emphasis because bistable mechanisms might very well be involved in (robot) *cell differentiation* [10].

The flip-flop in Fig. 7 implements rule r_1 in Φ_1 (see Eq. 1). Exactly the same regulatory network exist on both robot modules (left and right). By asynchronous updating, it happens that either the left or the right module will increase protein x beyond its threshold ($\alpha = 80$ on a $\theta_{max} = 100$). As soon as this takes place, one of the inhibiting connections between x_i and x_j becomes active. Then the number of proteins of its "competitor" will be reduced. Although in the beginning the winners might alternate, in the end — if γ is large enough in $R^{-\gamma}_{\perp\epsilon}$ — there will be a single winner.

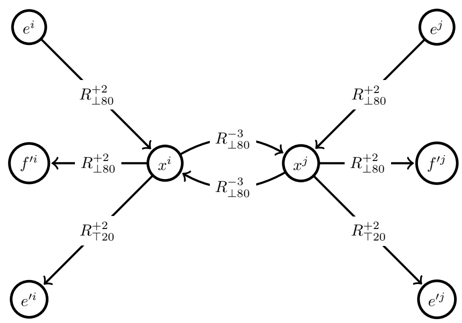

Figure 7: A flip-flop is defined across two robot modules, denoted by i (left) and j (right). Rule r_1 in Φ_1 (see Eq. 1) transitions to either e' or f'.

4.5 Bistable Circuit Implementation

We would like to reinstate that our main goal is a design methodology for metamorphosis. However, due to its pivotal role in the describe a graph rewriting rule set as Φ_1 in the form of a regulatory network, the bistable circuit is implemented. It allows describing indeterministic rules like $r_1 \in \Phi_1$ (in Eq. 1). Fig. 8 shows the results of a run with a C implementation of the flip-flop as defined in Fig. 7.

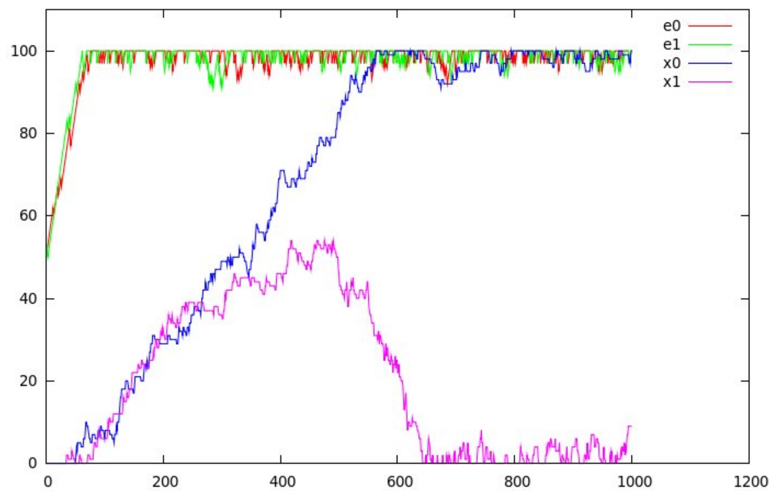

Figure 8: A run of the bistable circuit with parameters set as in Fig. 7. Time is on the horizontal axis, protein quantities on the vertical axis. Edges (with decay as self-inhibiting edge $R^{-1}_{\perp 0}$) are randomly chosen. Bistable behaviour emerges: a high concentration x_0 and a low concentration x_1.

The existence of network motifs like the logical NAND port and the bistable circuit allows us to design a regulatory circuit corresponding to the metamorphic rule sets defined in Section 2.

5 Conclusion

The design trajectory is described step-by-step. To understand a regulatory network for metamorphosis three levels of abstraction are discussed. It starts at the most abstract level with a graph rewriting grammar. At an intermediate level, it continues with a communicating finite state machine description. At the lowest level of abstraction, a regulatory network implements the state transitions on each robot module by (mutually interacting) artificial proteins.

As stated in the introduction, a top-down, formal, and explicit representation provides clear benefits over yet another ad-hoc evolutionary technique, most of all, it allows for benchmarking. In future research, the top-down approach will be compared with, among others, evolved robotic metamorphic gliders [14].

Acknowledgments

The authors would like to thank the anonymous referees for their constructive comments. The "REPLICATOR" project is funded by the European Commission within the work programme "Cognitive Systems, Interaction, Robotics" under grant no. 216240.

References

[1] U. Alon. Network motifs: theory and experimental approaches. *Nature Reviews Genetics*, 8(6):450–461, 2007.

[2] J.C. Bongard. Evolving modular genetic regulatory networks. In *Proceedings of the 2002 congress on evolutionary computation*, pages 17–21. Citeseer, 2002.

[3] D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM (JACM)*, 30(2):342, 1983.

[4] A. Castano and P. Will. Representing and discovering the configuration of conro robots. In *IEEE International Conference on Robotics and Automation*, volume 4, pages 3503–3509. IEEE; 1999, 2001.

[5] D.B. D'Ambrosio and K.O. Stanley. A novel generative encoding for exploiting neural network sensor and output geometry. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, page 981. ACM, 2007.

[6] E.A. Di Paolo. Rhythmic and non-rhythmic attractors in asynchronous random boolean networks. *BioSystems*, 59(3):185–195, 2001.

[7] S. Harding and J.F. Miller. The dead state: A comparison between developmental and direct encodings. In *Proc. GECCO Workshop on Complexity Through Development and Self-Organizing Representations*, 2006.

[8] S. Kernbach, H. Hamann, J. Stradner, R. Thenius, T. Schmickl, A. C. van Rossum, M. Sebag, N. Bredeche, Y. Yao, G. Baele, Y. Van de Peer, J. Timmis, M. Mohktar, A. Tyrrell, A. E. Eiben, S. P. McKibbin, W. Liu, and A. F. T. Winfield. On adaptive self-organization in artificial robot organisms. In *Proc. of the First IEEE International Conference on Adaptive and Self-adaptive Systems and Applications (IEEE ADAPTIVE 2009)*, Athens/Glyfada, Greece, 2009.

[9] E. Klavins. Directed self-assembly using graph grammars. *Foundations of nanoscience: self assembled architectures and devices, Snowbird, UT*, 2004.

[10] A.S. Ribeiro and S.A. Kauffman. Noisy attractors and ergodic sets in models of gene regulatory networks. *Journal of theoretical biology*, 247(4):743–755, 2007.

[11] D. Roggen and D. Federici. Multi-cellular development: is there scalability and robustness to gain? In *Parallel Problem Solving from Nature-PPSN VIII*, pages 391–400. Springer, 2004.

[12] L. Steels and P. Vogt. Grounding adaptive language games in robotic agents. In *Proceedings of the fourth european conference on artificial life*, pages 474–482. The MIT Press, 1997.

[13] S.E. Sultan. Plant developmental responses to the environment: eco-devo insights. *Current Opinion in Plant Biology*, 2009.

[14] A.C. van Rossum. Robot metamorphosis: How to build a glider with a genetic regulatory network. In *Proceedings of Seventh International Conference on Swarm Intelligence*, Brussels, 2010. submitted.

[15] D.H. Zanette and L.G. Morelli. Synchronization of coupled extended dynamical systems: a short review. *International Journal of Bifurcation and Chaos*, 13(4):781–796, 2003.