Model Verification Framework with Application to Molecular Biology

Valeria Perelman
valeriya@technion.ac.il
Technion, Israel Institute of Technology

Judith Somekh
yuditol@technion.ac.il
Technion, Israel Institute of Technology

Dov Dori
dori@mit.edu
Technion, Israel Institute of Technology and Massachusetts Institute of Technology

Keywords: Finite-state Transition Systems, Formal Verification, Systems Biology

Abstract

A myriad of detailed pieces of knowledge regarding the structure and function of the living cell have been accumulating at an alarmingly increasing rate. Emphasis is shifting from the study of a single molecular process to cellular pathways, cycles, and the entire cell as a system.

Object-Process Methodology (OPM) is a holistic graphical modeling methodology that combines the behavioral and structural aspects of a system in a single model. The OPM methodology includes OPM-based development process, OPM Case Tool (OPCAT), and a modeling language.

A framework for supporting the biological researcher for hypotheses modeling and verification is proposed. The framework consists of (1) OPM modeling of complex molecular biological systems intuitively yet formally and (2) a set of translation rules from OPM to a finite-state transition system (FTS) to enable model verification. An example from the mRNA transcription subsystem of gene expression demonstrates OPM-based modeling and its translation into FTS.

1. INTRODUCTION

In recent years we have been witnessing an unprecedented increase in amount, variety, and complexity of information resources available to researchers in life sciences. A myriad of detailed pieces of knowledge regarding the structure and function of living cell have been accumulating at an alarmingly increasing rate. We are thus at a turning point in biological research, where emphasis is shifting from the study of a single molecular process to complete cellular pathways and the entire cell as a system. This trend calls for adopting a holistic, integrating, model-based Systems Biology paradigm that would enable making system-level sense of the countless pieces of information that have been gathered thanks to decades of meticulous laboratory research by thousands of scientists. These efforts, many of which are currently considered as contributions to Systems Biology, are aimed at understanding the underlying structure and behavior of biological systems at the molecular, cellular, organism, and habitat levels. Many formal approaches for systems design and model checking originally designed to overcome increasing complexity of software and hardware systems, nowadays have being adopted to overcome similar problems in biological systems. Model verification combined with systems design provides a tool to verify the developed design against the existing system specifications making it more accurate.

Specification and modeling of biological systems, such as metabolic pathways and regulatory networks, is currently done in various methods. A brief overview of the methods follows.

Boolean network models are used to describe and simulate gene regulatory circuits [1]. Their drawbacks are the difficulty to compose larger models from smaller building blocks and their limitation in representing only the regulation aspect of molecular systems.

Computational approaches for modeling biological systems use formal descriptions or algorithms to describe natural phenomena. Petri nets are utilized for modeling the concurrent behavior of biochemical networks representing various biological pathways [2, 3]. The application of Petri Net based model checking and validation was exemplified on various types of biological networks [4, 5].

Statecharts-based models were developed to describe the life span of various cell types [6]. Vulval cell fate determination was modeled using Statecharts and Live Sequence Charts (LSC) [7].

OPM [8, 9] was selected as the framework for modeling biological systems as it is a graphical methodology that incorporates the static-structural and dynamic-procedural aspects of a system into a unifying model, which is presented in its entirety using a single diagram type. OPM consists of two semantically equivalent modalities of the same model: graphical and textual. Since the corresponding textual model is generated in a subset of English, it is immediately understood by domain experts, who need not learn any special language. Using OPM as a modeling framework for the mRNA lifecycle, a knowledge gap was discovered and consequently established empirically that the

translation termination factor eRF3 is found in processing bodies in the cytoplasm after a starvation period 9].

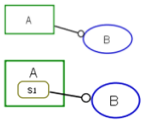
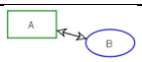
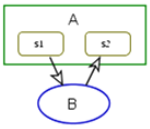
In this paper we propose a modeling and verification framework to support research in Systems Biology. The framework includes OPM-based modeling of biological systems, presented in [10], and verification techniques for model analysis and refinement. Based on Finite-state Transition Systems (FTS), we present a set of translation rules from an OPM model into a formal logical model to enable automatic model checking. The value of the system to biology researchers is exemplified on the transcription subsystem of the mRNA lifecycle within the living cell.

The paper is organized as follows. Section 2 briefly surveys OPM semantics. Section 3 presents the proposed research framework. The biological system of the mRNA lifecycle with focus on the transcription subsystems is presented in section 3.1 and its specification through OPM is presented in section 3.2. The OPM-to-FTS translation rules and possible specifications are demonstrated on the transcription subsystem in Section 4. A summary in Section 5 concludes the paper.

2. OBJECT-PROCESS METHODOLOGY

The living cell is a prime example of a highly complex system, in which the two main system aspects—structure and behavior—are highly intertwined and hard to separate. Motivated by the requirement of a single model to represent these two major system aspects, OPM is founded upon three elementary building blocks —*objects* possibly with *states* and *processes*—which represent the system's structure and dynamics. The processes transform the system's objects by creating them, consuming them, or changing their states, in a balanced way without highlighting one at the expense of the other. These building blocks are called *Entities*.

Table 1. Main OPM Elements and Their Symbols.

Category	Name	Symbol	Definition
Entity	Object		An **object** is a thing that exists.
	Process		A **process** is a thing that transforms at least one object.
	State		A **state** is situation an object can be at or a value it can assume.
Structural Link (connecting objects)	Aggregation-Participation		**Whole –Part Relation**
Procedural link (connecting objects and processes)	Instrument Link		Enabling link. **"Wait until"** semantics: Process B cannot happen if object A or state s1 does not exist
	Effect Link		Process B **changes the state** of Object A; the details of the effect may be added at a lower level.
	State-Specified Effect Link (*Input-Output Links Pair*)		Process B **changes the state** of Object A from State s1 to State s2.
	Consumption Link		Process B **consumes** Object A.
	Result Link		Process B **creates** Object A.

Examples of biological objects are Protein, Cell, and Organism, and examples of biological processes are Cleavage, Mitosis, and Birth. OPM entities are connected via *links* which can be *structural* or *procedural*. A *structural link* expresses a static, time-independent relation between pairs of *entities*.

A *procedural link* connects objects and processes to describe the behavior of a system. The behavior is manifested in three major ways: (1) a process can transform (generate by a *Result Link*, consume by *a Consumption Link*, or change the state of one or more objects by an *Effect Link*); (2) a process can be enabled by an object without transforming it (by an *Instrument Link*), in which case the object acts as an enabler, i.e., an enzyme or an inanimate instrument; and (3) a process can be invoked by events triggered by entities if some conditions are met (by an *Event Link*). A partial list of OPM elements and links with their symbols and definitions is provided in Table 1..

Two semantically equivalent modalities, one graphic and the other textual, are used to describe each OPM model. The textual representation which is built as a subset of English can ease the comprehension of the models by non expert viewers.

The complexity of systems is managed in OPM models by abstraction-refinement mechanisms, notably out- and in-zooming and folding/unfolding, which can be used to hierarchically expose or hide details of objects and processes. This way, a top-level view of the system is expanded into a set of increasingly detailed diagrams. OPM is supported by OPCAT, a software environment that is used in this work to model the presented mRNA lifecycle.

3. MODELING & VERIFICATION FRAMEWORK

We propose a framework aimed for conceptual modeling and verification of biological systems as described in Figure 1. The workflow starts with the **(1) Initial Design Modeling** process, which includes gathering information from research papers in the relevant field and specifying it through an

OPM model using the OPCAT. Then, the **(2.a) SMV Script Generating** process is executed automatically. The process is based on an automatic translation incorporated into OPCAT. The researcher then proceeds with a manual **(2.b) Specification Definition** process. This process concerns representing facts gathered from scientific papers, resulting in a set of specification rules.

The next step is the **(3) Verification** process, which is fully automatic and employs the SMV tool.

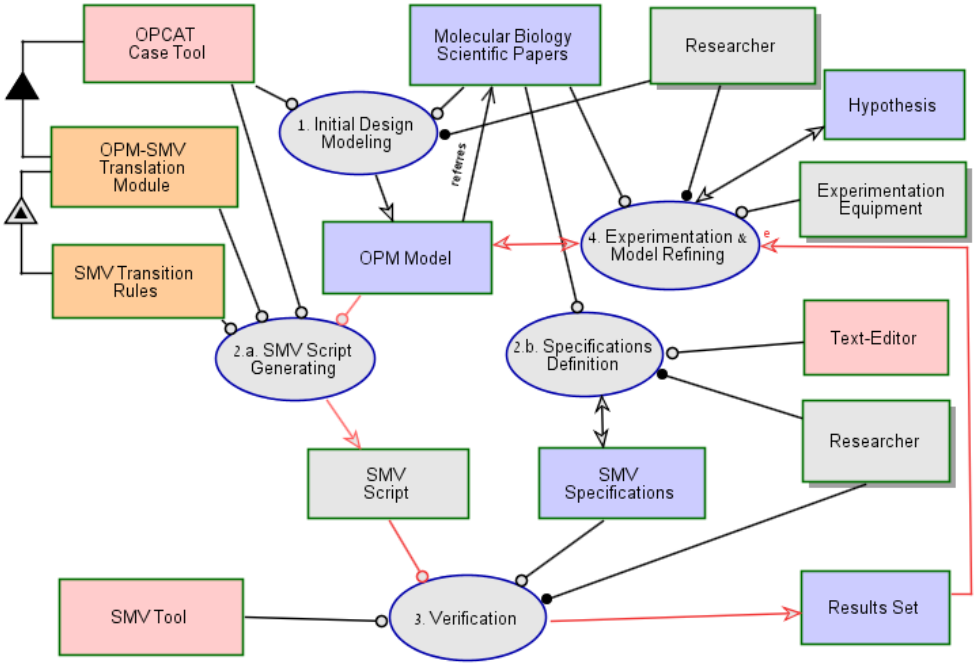


Figure 1. **The Modeling & Verification Framewor**

The process aims to detect contradictions between the SMV script, which represents the OPM model, and the set of facts gathered from the pertinent literature. Once the **Verification** succeeds, the researcher can conduct an experiment and refine the **OPM Model.** The event link from the **Results Set** object to the **Experimentation & Model Refining** process indicates that the processes of script generating, verification and experimentation are recurring. This paper describes the first two phases of the working framework, including the **Initial Design Modeling** process, the generated **OPM model**, and description of the **Translation Rules Set** from the OPM model into the TS model.

3.1. The Transcription Process

The expression of protein encoding genes is a complex process that determines which genes are expressed as proteins at any given time, as well as the relative levels of these proteins. This process includes RNA synthesis, or transcription.

Transcription by pol II, the first stage in the expression of protein-encoding genes, produces RNA—the primary transcript. Pol II requires a series of additional proteins, general transcription factors, and other proteins to initiate transcription on the DNA inside the cell.

During the elongation phase of transcription, the nascent RNA undergoes three types of processing events: a special nucleotide (m(7)GpppN), named cap, is added to its 5' end (capping), intron sequences are removed from the middle of the RNA molecule (splicing), and the 3' end (3' poly(A) tail) of the RNA is generated (cleavage and polyadenylation). Each of these processes is initiated by proteins or RNA molecules that travel along with the RNA polymerase II (pol II) by binding to its sites on its long extended C-terminal tail (i.e.,CTD).

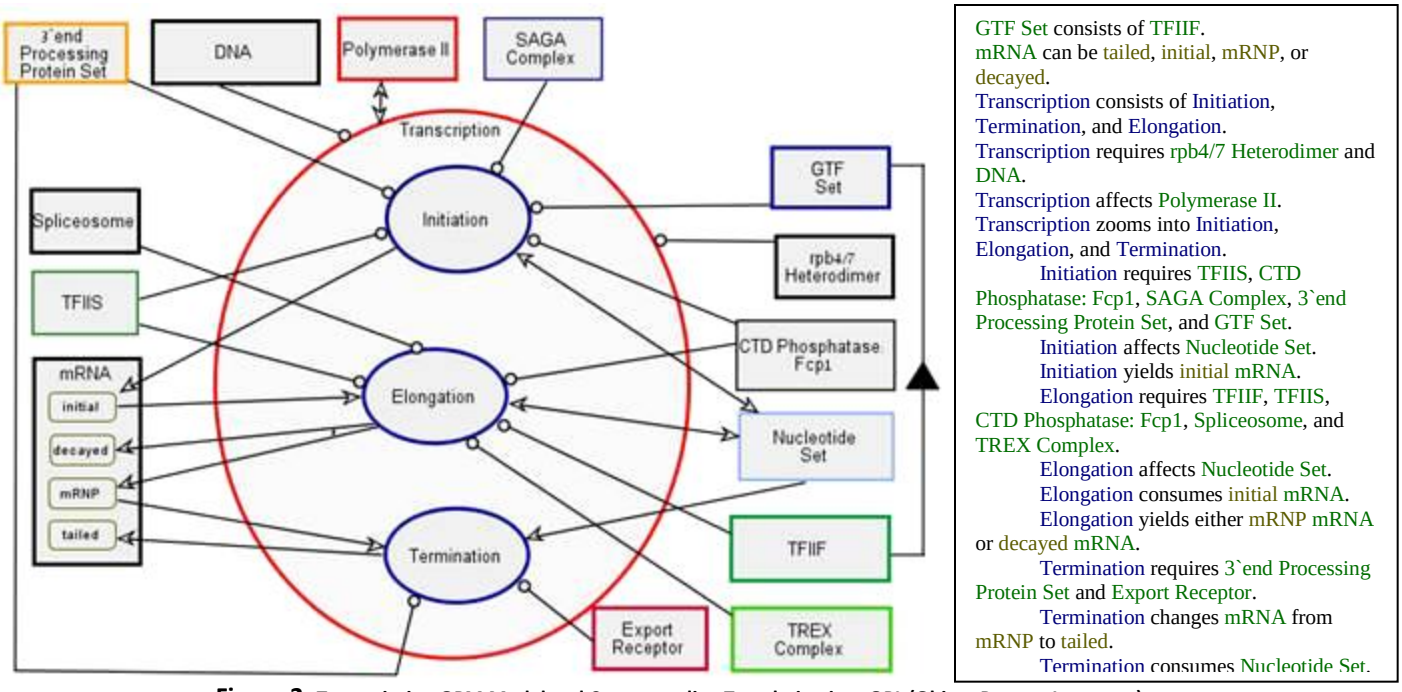


Figure 2. **Transcription OPM Model and Corresponding Translation into OPL (Object-Process Language)**

3.2. Transcription Specification with OPM

The **Transcription** process is in-zoomed in Figure 2. **Transcription** includes three subprocesses; **Initiation, Elongation** and **Termination**. The process flows from the top-most subprocess, **Initiation**, followed by **Elongation**, and ends with **Termination**. The **Initiation** process yields a new **mRNA** in an initial state. Participating biological objects are connected to the relevant process through procedural links. For example, **GTF Set** (General Transcription Factor Set) participates in and enables the Initiation process. Additionally, the **Elongation** process changes the state of **mRNA** from initial into **decayed** or **mRNP** (consumption link emerging from the **initial** state and result links connected to the **decayed** and **mRNP** states). The *XOR* link between these result links is expresses by the line connecting the links starting point.

4. TRANSLATION OF OPM TO FINITE-STATE TRANSITION SYSTEM

Finite-state Transition System (FTS) is a computational model for synchronous and asynchronous finite-state systems, however, the work was inspired also by a Clock Transition System defined in [11] and the timed automata model 12].

A finite-state transition system defines possible moves between states, for a bounded number of variables having finite number of possible states. Formally, it is a tuple $<Vars, \Theta, \Upsilon>$.Vars denotes a finite set of state variables. A state of the system is defined by a full evaluation of the variables in the set. Θ is a predicate over Vars that defines the initial condition, and Υ is a finite set of transitions. Each transition is a function $\tau : 2^{Vars} \mapsto 2^{2^{Vars}}$.

A state is denoted by σ, and $\sigma(x)$ for $x \in Vars$ denotes the value of x in state σ. For notational convenience we will write simply x to denote $\sigma(x)$, when it is clear from the context that we refer to a state σ. We denote by x' the value of x in the next configuration.

4.1. Finite-state Transition System corresponding to an OPM Model

We define finite-state transition system $\Phi_{OPM} :< Vars, \Theta, \Upsilon >$ as the FTS corresponding to an OPM model. We define $Vars \subseteq D \cup I$, where D is a set of state variables, and I is a set of Boolean input variables, which are used to represent OPM non-deterministic semantics, such as tossing of a process termination and environmental objects creation. One of the variables in D is called *SysState*, and denotes one of *{init, tick, event, stepI, stepII, end}*. All system transitions are separated into groups. State of the *SysState* variable defines which group of transitions is to be invoked in the current configuration. It should be notices that the technique that we employ in our model checking mechanism recalls a simple discrete event scheduling algorithm used in simulations.

In addition, we define a set *Ev* to hold all the currently unprocessed events. These events are a convenient notion for describing all the triggers of OPM's operational semantics. For example, a process termination activates an appropriate event in FTS. Technically, an event is defined by a Boolean assignment to the relevant event variable (i.e., activated or deactivated) and results from an evaluation of a predicate.

Examples to the possible event variables are:

- *ev_terminate* - denotes the system termination
- ev_term_{ik} - denotes a termination of an object or a process *i*, instance *k*
- $ev\_create\_o_{ik}[s]\_p_{jf}$ - denotes creation of a new object instance *i*, optionally in the state *s* by the process instance
- $ev\_invoke\_p_{ik}$ - denotes event of the process invocation for a process *i*, instance *k*.

Following list contains two basic Φ_{OPM} attributes:

Handling of events – Φ_{OPM} has a clock-asynchronous timing policy, which means that within a tick cycle internal events generated by the system transitions and all the inputs are processed. Thus, the system accepts a set of events at the beginning of each clock *tick*; takes some *step* and becomes unstable until all the enabled steps are carried out. The generation of events and their effect take no time.

Execution cycle – The OPM execution cycle includes following transitions set: {*init, tick, event, step I, step II, end*}. The *init* transition is used to set up the initial configuration. External and temporal events are obtained right after the *tick* transition via an *event* transition, and then, if the event set *Ev* is not empty, the system becomes unstable, requiring a *step I* transition. New internal events can be generated through a *step I* transition, making the system unstable again. Hence, the system makes *step II*. The activities related to the processes termination are handled through *step I*, and then all activities related to the processes initiation are taken through *step II*. This separation into two phases is done to eliminate some race conditions. For example, consuming an object instance and using it by an invoked process. Finally, the system becomes stable and a new *tick* of time occurs, beginning the next cycle.

State Variables -- let obj_i/$proc_i$ denote an object/process in the OPM model. An OPM model includes a set *Obj* of OPM objects and a set *Proc* of OPM processes. Let $O=\{o_{ik}\}_{i=1..|Objs|,\ k=1..K}$ be the set of state variables representing the *K* instances of the |*Obj*| objects, and $P=\{p_{jk}\}_{i=1..|Procs|,\ k=1..K}$ be the set of state variables representing the *K* instances of the |*Proc*| processes. $O \cup P \cup Time \cup SysState \subset D$.

For each link in the OPM model between node $obj_i/proc_i$ and $obj_j/proc_j$, *D* includes *K*x*K* Boolean variables. These variables indicate whether each pair of instances in the instance lists of o_i/p_i and o_j/p_j are linked in the current configuration. For better readability we will use $thing_{ik}$ to denote either o_{ik} or p_{ik}.

4.2. FTS –to- OPM Translation Examples

We now exemplify partial translation to FTS using the **Transcription** sub process defined in Figure 2.

State Variables: Transcription process uses **DNA**, **3'end Processing Protein Set** and **GTF Set**, **Export Receptor** and other objects. Every object that is not an instantiation of other object is translated as a set of object variables identified by the object name followed by the index of an instance. In the FTS of **Transcription**, the following variables represent the object instances:

(1) $mRNA_1..mRNA_K \in O$

//these variables represent instances of the mRNA object, their type in SMV is expressed as follows:

$mRNA_i$:{*initial, decayed, mRNP, tailed, nonExistent*}

Now we describe process variables. Indexes refer to the different process instances. Any process instance has three possible states: not existent, waiting or active. In addition, attributes specifying the current execution timeline, named *currTime,* are added to all the in-zoomed processes. In the example, in-zoomed **Transcription** process has three different timelines defining execution order of its subprocesses, starting with **Initiation** at timeline 1. Following are the variables representing **Transcription** process:

(2) $Transcription_1..Transcription_K \in P$

$Transcription_i$:{*nonExistent, isWaiting, isActive*}

$Transcription_1.currTimeline$: {1, 2, 3}

$Transcription_K.currTimeline$: {1, 2, 3}

Information regarding link instances is kept via Boolean link instance matrixes. In (3) we demonstrate the effect link between **Transcription** process and **Pol II (Polymerase II)** object:

(3) $Effect(Transcription_i, PolII_j) \in D$, where $i, j \in [1..K]$.

$Effect(Transcription_i, PolII_j) : Boolean$

Transitions Invocation: Suppose, that current configuration is the following:

$\vartheta_i : SysState = 'event' \wedge Ev = \varnothing$

- The partial object variables configuration:

  $DNA_1 \wedge PolII_1 \wedge SAGAComplex_1 \wedge GTFSet_1 \wedge$
  $TFIIF_1 \wedge TFIIF_2 ... \wedge mRNA_{i\in[1..K]} = 'notExist' ...$

- The partial configuration of the structural links (the aggregation-participation link instances between **GITSet** and **TFIIF** object instances):

  $Composition(GITSet_1, TFIIF_1) \wedge$
  $Composition(GITSet_1, TFIIF_2) ...$

- The process instances:

$Initiation_1 = 'isActive' \wedge Transcription_1 = 'isAcitve' \wedge$
$In\text{-}zooming(Transcription_1, Initiation_1) \wedge$
$Transcription_1.currTimeline = 1...$

- The procedural links:

$Effect(Transcription_1, PolII_1) \wedge$
$Effect(Initiation_1, PolII_1)...$

The *result link* between a terminated **Initiation** process and an **mRNA** object is translated during *event* transition into process termination and process invocation events generation, and also an **mRNA** object creation event:

$$\xrightarrow{event} SysState = 'stepI' \wedge ev\_terminate\_Initiation_1 \in Ev$$
$$\wedge ev\_create\_mRNA_1[initial] \wedge ev\_invoke\_Elongation_1.$$

Then a *step I* transition changes the system configuration. The **Elongation** process is invoked entering its waiting state. The **Initiation** process instance becomes non-active, and a new **mRNA** instance is created in the initial state -- all the changes during this transition are triggered by events generated in the previous transition.

$$\xrightarrow{stepI} SysState = 'stepII' \wedge Elongation_1 = 'isWaiting'$$
$$\wedge Initiation_1 = 'nonExistent' \wedge mRNA_1 = 'initial' \wedge$$
$$In\text{-}zooming(Transcription_1, Elongation_1) \wedge$$
$$Result(Transcription_1, mRNA_1) \wedge Transcription_1.currTimeline = 2.$$

Through the next transition (*step II*), preconditions of the waiting process instances are evaluated. For example, the main precondition requires that all (or part depend on the specification) of the instrument object instances should exist and are in the appropriate states. Only those object instances that belong to the in-zoomed parent process of the waiting process are evaluated in the pre-condition.

Satisfied precondition (1) changes the process state from waiting to active and (2) sets all the appropriate procedural link instances between the input object instances and the activated process instance:

$$\xrightarrow{stepII} SysState = 'tick' \wedge Elongation_1 = 'isActive' \wedge$$
$$Effect(Transcription_1, PolII_1) \wedge Effect(Elongation_1, PolII_1) \wedge$$
$$Result(Transcription_1, mRNA_1) \wedge Update(Elongation_1, mRNA_1).$$

Model Verification Specification Examples:

(1) Does the **Transcription** process represent an infinite loop?

$$AG\ (\ AF\ (\ \vee_{i\in[1..K]}\ (Transcription_i = isActive)).$$

(2) Does the **TFIIF** transcription factor is an instrument (i.e. participates) to more than one sub-processes?

$$\varphi_{i,j,f} : AG[(Initiation_j = isActive \rightarrow Instruments(TFIIF_i, Initiation_j)$$
$$\wedge (Elongation_f = isActive \rightarrow Instruments(TFIIF_i, Elongation_f)]$$
$$\wedge AF(Initiation_j = isActive) \wedge AF(Elongation_f = isActive)$$
$$\vee_{i,j,f\in[1..K]} \varphi_{i,j,f}$$

5. SUMMARY

We have presented a framework that combines OPM-based conceptual modeling with FTS-based model checking and verification mechanisms. This approach provides for (1) exploring options for updating a model after new experimental evidence becomes available, and (2) based on the rules within the framework, determining whether a given hypothesis is feasible such that it is worth checking in a wet lab experiment or not.

OPM differs from other conceptual modeling methods by its ability to show object-process connections and effects as well as object state transitions in a single type of diagram. The relative simplicity of OPM raises the likelihood of success to analyze complex systems such as the ones in the living cell.

The proposed framework aims to help in filling knowledge gaps and assessing the feasibility of given conjectures. Using rules that are part of the proposed modeling and verification framework, it is possible to define whether hypotheses that attempt to close a knowledge gap worth experimenting. Our next step will be to verify the generated model against specifications representing biological facts. We also write a software tool for automatic rules translation.

Reference

1. Shmulevich, I., Lahdesmaki, H., Dougherty, E.R., Astola, J. & Zhang, W. (2003). The role of certain Post classes in Boolean network models of genetic networks. Proc. Natl. Acad. Sci. USA 100: 10734–10739.
2. Chaouiya, C. (2007). Petri net modelling of biological networks. Brief. Bioinform. 8: 210–219.
3. Kielbassa J., Bortfeldt R., Schuster S., Koch I. (2009), Modeling of the U1 snRNP assembly pathway in alternative splicing in human cells using Petri nets, Computational Biology and Chemistry 33 (1) 46–61.
4. Peleg, M., Yeh, I., Altman, R.B. (2002). Modelling biological processes using workflow and petri net models. Bioinformatics 18: 825–837.
5. Breitling, R., Donaldson, R., Gilbert, D., & Heiner, M. (2010). Biomodel Engineering–From structure to behavior. Transactions on Computational Systems Biology XII, , 1-12.
6. Harel D., Setty Y., Efroni S., Swerdlin N., and Cohen I. R. (2008). Concurrency in biological modeling: Behavior, execution and visualization. FBTC 2007: Electronic Notes in Theoretical Computer Science, 194(3):119{131.
7. Fisher, J., Piterman, N., Hubbard, E.J., Stern, M.J. & Harel, D. (2005). Computational insights into Caenorhabditis elegans vulval development. Proc. Natl. Acad. Sci. USA 102: 1951–1956.
8. Dori D., (2002). Object-Process Methodology-A Holistic Systems Paradigm. HeidelbergNew York: Springer Verlag.
9. Dori D. and Choder M. (2007). Conceptual modeling in systems biology fosters empirical findings: the mRNA lifecycle. PLoS ONE. 2: e872.
10. J. Somekh, D. Dori and M. Choder, "Modeling Complex biological Systems with Object-Process Methodology:, *INCOSE 2011*, 2011.
11. Y. Kesten, Z. Manna, and A. Pnueli, "Verification of clocked and hybrid systems," *Acta Inf.*, vol. 36, no. 11, pp. 837-912, 2000.
12. R. Alur and D. L. Dill, "The theory of timed automata," in *Real-Time: Theory in Practice, REX Workshop, Mook, The M\Netherlands, June 3-7, 1991, Proceedings*, 1991, pp. 45-73.