Synthesizing Biological Theories

Hillel Kugler, Cory Plock, and Andy Roberts

Microsoft Research, Cambridge, UK
hkugler@microsoft.com, coryp@acm.org, i-anrobe@microsoft.com

Abstract. Studies of biological systems are often facilitated by diagram models that summarize the current understanding of underlying mechanisms. The increasing complexity of our understanding of biology necessitates computational models that can extend these representations to include their dynamic behavior. We present here a new tool we call *Synthesizing Biological Theories* which enables biologists and modelers to construct high-level theories and models of biological systems, capturing biological hypotheses, inferred mechanisms, and experimental results within the same framework. Among the key features of the tool are convenient ways to represent several competing theories and the interactive nature of building and running the models using an intuitive, rigorous scenario-based visual language. The definition of the modeling language is geared towards enabling formal verification and analysis.

1 Introduction

Biological systems and their ability to function robustly and perform complicated processes have been a source of fascination and scientific enquiry for many years. Recently, significant advances in biology have enabled a deeper mechanistic understanding of many fundamental underlying processes. Computational modeling is gaining prominence in the biological community as it allows us to formulate hypotheses and reason about the biological implications of these models in a precise and formal way that could lead to an improved scientific understanding. Performing certain experiments in the biological system is often not possible, or may require a lot of work to overcome technical hurdles. Yet, running a simulation and analyzing the underlying model is possible and holds a potential of guiding experimental work and thinking about the consequences of performing certain experiments even before they become feasible.

In this paper we present a new tool for biological modeling we call *Synthesizing Biological Theories* (SBT) which enables biologists and modelers to construct high-level theories and models of biological systems, capturing biological hypotheses, inferred mechanisms, and experimental results within the same framework using an intuitive, rigorous scenario-based visual language. The tool is publicly available at [SBT11] together with a user manual, documentation and sample models. In this paper we present the main challenges we tried to address while building the tool, its design principles, highlights of the main functionality and also briefly discuss several of the future directions.

G. Gopalakrishnan and S. Qadeer (Eds.): CAV 2011, LNCS 6806, pp. 579–584, 2011.

2 Main Challenges and Design Principles

We set out to build a tool that would be intuitive for experimental biologists, yet powerful to allow modeling, simulation and analysis of large and complex biological systems. We use scenarios as the basic modeling unit for specifying behavior. In SBT Version 1.0 we support a scenario-based language inspired by live sequence charts [DH01]. The reason for adopting a scenario based language is the ability to break up a complex model specifications into many charts (scenarios). Each chart can represent an explicit hypothesis about the system, a certain property that holds for the system, or a representation of an experimental result. The ability to view a chart in isolation from the entire model and examine its assumptions is important for biological modeling; the utility and impact of the model depends critically on all the biological assumptions it makes and the ability of experimental biologists to understand the representation in detail.

A user interface allows the user to build the model by interacting with a simple browser to define classes, types, objects and their properties and methods. Navigating complex models with many charts and filtering relevant objects is supported, where the object name serves as the key and all relevant items starting with a given prefix are displayed. The motivation for using this modeling scheme rather than coding custom software applications is to provide an easier entry point for experimental biologists with the goal of making such tools useful for a wider community. For this reason we use a visual representation for the scenario modeling language and provide a simple-to-use interface to construct scenarios, using the same visual representation during execution to provide meaningful feedback to the user. The tool was designed to support complex models and has been tested with models containing thousands of objects and hundreds of charts.

3 The SBT Language and Tool

We now describe the main features of our tool *Synthesizing Biological Theories* (SBT), with emphasis on the novel aspects of our work. The current version of SBT is 1.0, publicly available at [SBT11]. We also mention some extensions that we implemented in experimental versions of the tool but are not part of the first version, which we plan to make available in next versions. SBT uses a specification language inspired by live sequence charts (LSCs) [DH01].

3.1 Classes, Objects, Types

We adopt an object oriented framework, where a model is composed of a set of classes which are restricted to single inheritance. An object is an instance of a class, and a model contains a definition of static objects that are present at runtime when the model starts executing. In addition dynamic objects can be created and deleted during runtime as specified in the model scenarios.

A *class* is a structural template from which physical and conceptual entities called *objects* originate. In a biological model a class can, for example, represent a

certain type of cell, while objects of this class are the actual cells. Classes, which can be added by the user in SBT, consist of *methods*, *messages*, and *properties*. Methods are parameterless communications that may be received by the class or its objects. Messages are similar to methods, except with parameters. Properties are variables belonging to the class and collectively describe the state of the class and its objects. Properties can be modified using messages called *setters*.

Properties and message parameters have *types* which describe their domain. User-defined types can be constructed by parameterizing the SBT primitives `range` and `enumeration`, `bound`, and `boolean`.

3.2 Generalized Charts

The LSC definition [DH01] makes a distinction between two types of charts, *universal* and *existential*, but the SBT execution engine maps these to a single type of chart we call a *generalized chart*. This single chart type is made possible through the use of a third *warm* temperature [Plo08]. The intuition is that a cold LSC location is generally understood to denote that no progress is required beyond the location, whereas a hot location means that progress is required. Applying this notion to precharts, however, is not straightforward: strictly interpreted, this means that the chart would never have to progress beyond the initial locations of the chart and therefore the chart is vacuously satisfied without ever having left the initial state. What is actually intended is a similar but somewhat looser interpretation whereby progress is still not required, but advancement is required if an enabled event occurs at any prechart location. Our use of generalized charts and warm temperatures allows SBT to capture our intended semantics in an elegant way while avoiding special cases in the underlying execution engine.

3.3 Subcharts

Subcharts are charts which can be embedded in a parent chart. The execution engine internally maintains subchart executions as their own individual execution, which can satisfy or violate just as any other chart. Subcharts follow an orthogonal approach whereby any chart in the requirements may be added as a subchart to any other LSC[1].

Upon termination, subchart termination status propagates up to the parent chart, which will respond accordingly. For example, if a subchart is located within the prechart of its parent and it violates, the parent will close without violation. If the subchart is located in the main chart of the parent, however, the parent chart will also violate. When the subchart is satisfied, its parent may progress beyond the subchart.

3.4 Symbolic Instances

Symbolic instances are those that can be bound to objects of a specified class at runtime when the first visible event on the instance line occurs. A quantifier

[1] The user interface of SBT 1.0 does not fully support this currently although the execution engine does.

expression over the class properties is associated with the symbolic instance. For each object satisfying the quantifier expression at binding-time, a new chart is created in which the symbolic instance is replaced with the concrete object.

The set can be universally or existentially quantified: all executions in the set must satisfy for universal instances, whereas only one must satisfy for existential instances. The symbolic instances are numbered such that charts with multiple symbolic instance lines will be quantified in a specific order.

Symbolic instances can also be used to create new objects dynamically from a class definition using the special SBT method `create`. Once created, dynamic objects can be referenced using symbolic instances and even destroyed.

4 Execution Modes

SBT presents a set of novel language constructs and runtime execution semantics geared towards biological modeling, we highlight some of these in the following section.

4.1 Deterministic vs. Nondeterministic Execution

Scenarios consist of *visible* events which are observable and *hidden* events which are non-observable, where SBT (on behalf of the biological system) picks and executes events internally. The choice of which internal events the biological system picks can have a great impact on whether or not the chart is eventually satisfied (i.e., accepts the behaviors.) SBT supports both a deterministic and nondeterministic mode:

According *deterministic* mode, a visible event is executed and then the execution engine determines whether any hidden events are enabled. If so, it arbitrarily selects and executes one at the same instant[2] as the closest preceding visible event. This is repeated until no more hidden events are enabled.

On the other hand, *nondeterministic* mode attempts all possibilities by effectively splitting a single run into multiple threads of parallel execution, of which one must eventually be satisfied. The LSC is considered satisfied if any run in the set accepts. This mode operates across two axes of nondeterminism: it selects not only the choice of which hidden event to execute next, but also the choice of whether to proceed to then next hidden event (if one exists) or remain in the present location.

For nondeterministic mode we found that the visualization of progress along the charts is useful to follow the nondeterminism. Nondeterminism is essential for modeling of biological systems as it provides an easy way for the modeler to represent a certain assumption when the precise mechanism is still unknown. Additional experience still needs to be gained to identify the most useful way to specify this nondeterminism and to restrict it in cases where it is not needed to simplify progress visualization and improve performance.

[2] The word "instant" is used here in the sense of reactive systems operating under the synchrony hypothesis.

4.2 Stepping Mode

Stepping mode specifies how events are to be generated. There are two modes:

Interactive mode allows the user to execute any event by selecting it from a list of available events. Both *system* and *environment* events are available for selection, where system events are those controlled by the biological model and environment events are external. *Super-stepping* mode allows the user to execute a visible *environment* event, and the biological system will respond with a multi-event response called a *super-step*, which terminates when no system events are enabled or the main charts of all universal charts have closed.

In super-stepping mode, the event selection policy can either be *arbitrary* (default) or *random*, where the engine will select only non-violating events according to the chosen policy.

Violating executions are removed from the nondeterministic execution set. The set itself violates when and if the last execution in the set violates.

4.3 Configurations

A user can run simulations and investigate the system behavior at any stage during construction of the model. The user can set a configuration by selecting which charts in the model are used and which are not, allowing to ignore charts that are still "in progress." Several different configurations per model can be maintained, one of them is designated as the active configuration in use, the typical usage of multiple configurations in biological modeling is for maintaining several competing hypotheses that share most of the scenarios in the model but differ on certain specific charts thus representing different variants of the model.

4.4 Display Options

The basic visualization provided by SBT shows property values of all objects in the model (i.e., all static objects and dynamic objects that have been created but not yet deleted) and the progress of all active charts. The user can set any of these options to be active or not. Runs of simulations can be recorded and later replayed for more careful investigation. The execution engine was designed to run independently from the user interface to ensure a clean separation between the modules and allowing to connect other components to the execution engine, version 1.0 supports connecting the executed models to a processing visualization environment.

5 Biological Examples and Future Directions

A scenario-based approach using LSCs and the Play-Engine [HM01] to model vulval development in *C. elegans* was presented in [KKM08]. The model focused on six cells and how their fate is specified. This is conceptually an important process in developmental biology, where developing up-to-date realistic and predictive models for this system is a major challenge requiring development of powerful tools and methodologies [FH07].

To deal with models with large cell populations, SBT supports dynamic creation and destruction of objects. It is now being actively used to study stem cell population dynamics in the *C. elegans* germ line, embryogenesis and emergent behavior of bacteria populations [KLH10]. As these models and new ones are published, they will be made part of the tool's online samples and made available as a resource to the research community.

We hope that in the future, similar competitions to those conducted for SMT solvers in the SMT-competitions can be organized for biological models. Since the format of the models and the properties are still under investigation, we believe that making our tool and other such tools available will help allow CAV and the formal verification community to become familiar with the challenges and special characteristics of such models. SBT experimental versions support verification and analysis of models [KS09, KPP09, MK11], the inherent difficulties in scaling such methods to large-scale realistic biological models can provide challenging research opportunities.

References

[DH01] Damm, W., Harel, D.: LSCs: Breathing life into message sequence charts. Formal Methods in System Design 19(1), 45–80 (2001)

[FH07] Fisher, J., Henzinger, T.A.: Executable Cell Biology. Nature Biotechnology 25(11), 1239–1249 (2007)

[HM01] Harel, D., Marelly, R.: Specifying and executing behavioral requirements: The play-in/play-out approach. In: Software and System Modeling, SoSyM (2003)

[KKM08] Kam, N., Kugler, H., Marelly, R., Appleby, L., Fisher, J., Pnueli, A., Harel, D., Stern, M.J., Hubbard, E.J.A.: A scenario-based approach to modeling development: A prototype model of C. elegans vulval fate specification. Developmental Biology 323(1), 1–5 (2008)

[KPP09] Kugler, H., Plock, C., Pnueli, A.: Controller Synthesis from LSC Requirements. In: Chechik, M., Wirsing, M. (eds.) FASE 2009. LNCS, vol. 5503, pp. 79–93. Springer, Heidelberg (2009)

[KS09] Kugler, H., Segall, I.: Compositional Synthesis of Reactive Systems from Live Sequence Chart Specifications. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. LNCS, vol. 5505, pp. 77–91. Springer, Heidelberg (2009)

[MK11] Milicevic, A., Kugler, H.: Model Checking Using SMT and Theory of Lists. In: Bobaru, M., Havelund, K., Holzmann, G.J., Joshi, R. (eds.) NFM 2011. LNCS, vol. 6617, pp. 282–297. Springer, Heidelberg (2011)

[Plo08] Plock, C.: Synthesizing Executable Programs from Requirements. PhD thesis, New York Univ. (2008)

[SBT11] Microsoft Research Cambridge, Synthesizing Biological Theories (2011), `http://research.microsoft.com/SBT/endthebibliography`