Bringing LTL Model Checking to Biologists

Zara Ahmed[1], David Benque[2], Sergey Berezin[3], Anna Caroline E. Dahl[4], Jasmin Fisher[1,5], Benjamin A. Hall[6], Samin Ishtiaq[1], Jay Nanavati[1], Nir Piterman[7], Maik Riechert[1], and Nikita Skoblov[3]

[1] Microsoft Research, Cambridge, UK
[2] Royal College of Art, London, UK
[3] Moscow State University, Moscow, Russia
[4] Center for Technology in Medicine and Health, KTH Royal Institute of Technology, Huddinge, Sweden
[5] Department of Biochemistry, University of Cambridge, Cambridge, UK
[6] MRC Cancer Unit, University of Cambridge, Cambridge, UK
[7] University of Leicester, Leicester, UK

Abstract The BioModelAnalyzer (BMA) is a web based tool for the development of discrete models of biological systems. Through a graphical user interface, it allows rapid development of complex models of gene and protein interaction networks and stability analysis without requiring users to be proficient computer programmers. Whilst stability is a useful specification for testing many systems, testing temporal specifications in BMA presently requires the user to perform simulations. Here we describe the LTL module, which includes a graphical and natural language interfaces to testing LTL queries. The graphical interface allows for graphical construction of the queries and presents results visually in keeping with the current style of BMA. The Natural language interface complements the graphical interface by allowing a gentler introduction to formal logic and exposing educational resources.

1 Introduction

Formal verification techniques offer a powerful set of approaches for analysing and understanding the behaviours of different systems. The advantages of such approaches are well understood and widely applied in the development of hardware and software systems. Outside of computing the usage of such techniques has been less widespread. Whilst standard techniques such as SAT solving and BDDs have been highly successful in individual investigations (see [1–5] for some recent examples), their broader utility has been limited by the fundamental requirement for proficiency in computing.

To address this skills gap tools such as BioModelAnalyzer (BMA, [6]) and GinSim [7] have been developed explicitly to better enable users to construct and analyse biological models. BMA presently allows users to construct models, perform simulations, and stability analysis. Stability analysis in BMA typifies the opportunities for algorithm discovery in biology. Standard approaches are insufficient in analysing many usefully large and complex biological models, so

a bespoke algorithm [8] is used to analyse the models. Users do not need an in-depth knowledge of that algorithm to apply it, and BMA enables this, by effectively encapsulating the computer science. By making model development and checking simple, this has prompted further algorithm development as models can be constructed more quickly and easily by users with deep expertise in a variety of biological systems [9]. As such, tool development has supported both biologists who wish to use powerful but inaccessible technology and computer scientists interested in addressing novel challenges that can arise from under-explored interdisciplinary areas. However, many questions that biologists wish to address require more complex tools than stability analysis. Some may want to ensure a specific dynamic behavior is observed; others may wish to explore instability in more depth. The present release of BMA only allows for such questions to be answered using simulation based approaches. Claessen et al. recently reported an LTL (bounded) model checker for biological models [10] that adapts the stability proving algorithm of Cook et al. [8] to enable faster model checking than provided by naive SAT based approaches. This had, however, not been integrated into the tool's front-end and, as such, was not available to most users.

Supporting LTL for users who do not have experience with logic and programming, poses distinct problems relative to the stability analysis. In this paper we report on our solution to make LTL more accessible for biologists. We allow users two possible approaches for constructing LTL queries. First, we present a graphical interface that abstracts the nuances of LTL by providing users with a graphical language, making use of visual cues such as icons, shapes and colour to denote operators, formulas and results respectively. Users are then able to use graphical controls and intuitive gestures such as drag-and-drop to visually construct queries as well as evaluate results. Second, we present a natural language interface (NLI), which exposes query creation and testing through a text-based *chatbot* (an interactive virtual assistant). The conversational nature of the chatbot's interface provides users with a higher level of abstraction over LTL than the graphical interface as it functions by interpreting *intents* rather than requiring explicit instructions. This means that instead of learning how to encode their queries into semantically valid LTL formulas, biologists can perform analysis by using natural language to describe queries in terms of cell states over time.

We further discuss the engineering and design challenges identified in the construction of this module, and describe how it may be adapted in the future to provide biologists with more accessible as well as scalable ways to leverage formal methods in their analysis.

BioModelAnalyzer is available at `http://biomodelanalyzer.org/`.

2 BMA basics

BMA and the motivations for developing such a dedicated tool for biologists has been described in depth previously (see, e.g., [6, 7, 11]). Briefly, users are able to develop models by "drawing" onto a blank canvas, in the same way that models

of cell signaling are communicated in scientific literature. Variables representing molecules within a cell or its environment and their relationships are depicted, and the resultant directed graph is treated as a *qualitative network* (QN) [11]. A QN $Q(V, T, N)$, of granularity $N + 1$ consists of variables: $V = (v_1, v_2, \cdots, v_n)$. The state of the system is a map $s : V \to \{0, 1, \cdots N\}$. The set of initial states is the set of all states. Each variable $v_i \in V$ has a *target function* $T_i \in T$ associated with it: $T_i : \{0, 1, \cdots, N\}^n \to \{0, 1, \cdots N\}$, describing the relationship between the variable and its inputs.

Target functions in qualitative networks direct the execution of the network from state $s = (d_1, d_2, \cdots, d_n)$. Variables in the QN update synchronously, i.e., the system is deterministic. The *next state* $s' = (d'_1, d'_2, \cdots, d'_n)$ is computed by:

$$d'_v = \begin{cases} d_v + 1 & d_v < T_v(s) \text{ and } d_v < N, \\ d_v - 1 & d_v > T_v(s) \text{ and } d_v > 0, \\ d_v & \text{otherwise.} \end{cases} \tag{1}$$

A target function of a variable v is an algebraic function over several variables $w_1, w_2, \cdots w_m$. Variables $w_1, w_2, \cdots, w_m$ are called *inputs* of v and v is an *output* of each one of $w_1, w_2, \cdots, w_m$.

3 Graphical User Interface (GUI)

LTL queries are substantially more complex than stability testing. Whereas the workflow of stability testing is simple, and common to every model (that is to say, attempt to prove stability, and then optionally search for counter examples), each step in performing an LTL query requires manual intervention. This cannot be avoided; each query represents a specification that will differ depending on the specific model being tested. In other biologist-targeted tools, this is achieved by exporting the model to SMV and expecting the user to independently use NuSMV or a similar tool [7, 12]. One of the major design principles of BMA is to avoid the requirement for use of command-line and computing proficiency, so this is not appropriate here. However, the requirement to write LTL queries poses some unique challenges. We do not expect users to be comfortable with complex operator precedence issues and balancing parentheses. Furthermore, there are some specific challenges that relate to BMA and exploring biological systems; notably models are expected to have large numbers of variables, and endpoints are of particular interest due to their role in describing cell fate and other developmental processes.

Two stage workflow

Our graphical interface addresses these issues through a two stage workflow. We separate *states*, and a *temporal and logical* layer. States are defined as a conjunction of linear constraints on variable values. Constraints can be set up

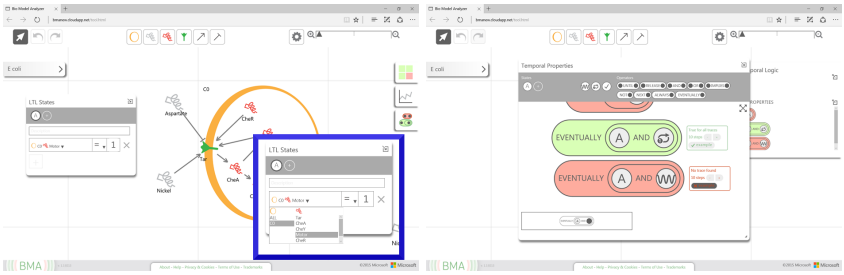

Figure 1: The LTL state editor and the LTL query editor.

by either selecting variables through a drop down menu, similar to the menus found in file browsers, or by dragging variables from the canvas onto the drop down menu (Figure 1). This makes adding linear constraints substantially less complex. LTL states also correspond closely to states that can be observed in simulations, and as such their construction and use is intuitive to non-experts (confirmed in user testing).

The second part of the workflow is to use the states within a temporal (and further logical layer) query (Figure 1). This is achieved by making a new canvas available for users, onto which they can drop logical and temporal operators (rendered as bubbles with sockets) and states. Each operator contains the appropriate number of sockets that match its expected number of operands. Operands can be dragged and dropped into sockets. Operands can be states, or other formulas constructed in the same way. As such, complex queries are created by repeatedly nesting different operators. Operators and queries can also be dragged to a *copy zone* that effectively allows for rapid copy and pasting in situations where this is desirable. Operator precedence and parenthesis checking is effectively enforced by the nested bubbles and missing elements are highlighted if the user should try to check an incomplete query. The final stage is the testing of the query, which requires the user to define the length of paths (relating to bounded model checking).

Default states

In addition to user defined states, we also include default states that can be used. We include the nullary (state) operators True, self-loop, and oscillation. The self-loop and oscillation describe states that, respectively, lie on a self loop or within a (strictly) larger loop. The description of these states through other operators would be extremely cumbersome. For example, a self loop state is characterised by the formula $\bigwedge_{v \in V} v = Xv$.[8] Self-loop and oscillation are important features in biological models, as developmental processes where end results are known require reasoning about the values of such end results.

[8] To the best of our knowledge, most LTL tools do not support such a direct comparison between the value of a variable and its value in the next state.

Non-standard temporal operators

User testing revealed that the operators *until* and *weak until* are very confusing in that they allow their first operand not to hold at all. We have supplemented these operators with the operator *Upto*, which carries similar meaning in English but can be assigned the stricter semantics without confusing users who are familiar with LTL.

Result visualisation

Finally, in response to running a query users are presented with three possible outcomes (Figure 2). The query is determined to be true for all traces, some traces or no traces. These results are determined by performing both the query as stated and the negation of the query on the back-end. Examples of traces that satisfy or fail to satisfy the query are made available, and can be visualised as graphs using the existing simulation visualisation tools. In the LTL tool, these traces are further annotated with the states that are satisfied at each time point, to aid analysis and interpretation of the results.

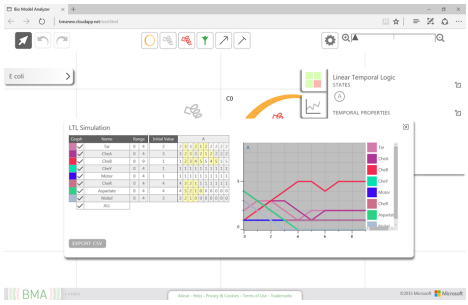

Figure 2: An example trace from an LTL query.

4 Natural Language Interface (NLI)

While the GUI makes syntactic aspects of LTL more accessible by handling operator precedence and formula parenthesisation, biologists are still required to have an understanding of LTL semantics in order to express biological concepts in the context of formal logic. Furthermore, the subtle differences between temporal operators as well as the discrete notion of time in LTL adds complexity, as biologists have to learn how to reason about biological processes (which are often stochastic and concurrent) in terms of discrete time-steps. By design, these issues cannot be addressed directly through the GUI and yet they contribute towards raising the barrier to entry to LTL-based analysis for users that have little to no experience with formal logic. In some cases, advanced users might appreciate the use of a simple text interface and prefer it to the GUI. For example, complicated and large queries may become unclear in the GUI, c.f., Figure 3. At the same

time, writing manually the Boolean part describing a "state" comprised of a large number of variables could be an issue with the NLI.

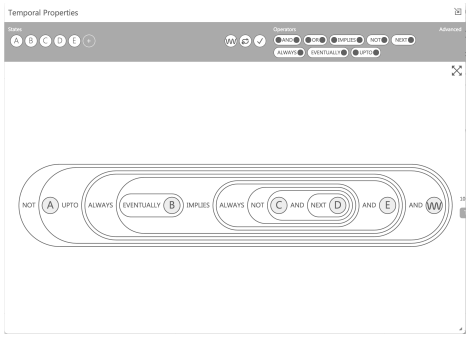

Figure 3: An example of a complex LTL query in the GUI

Knowledge base

In order to help users learn about the semantics of LTL, we developed a knowledge base that consists of definitions as well as example usages of LTL operators and developmental end states. The user can simply ask the NLI questions about a given operator in natural language, which the NLI answers using the knowledge base. The NLI supports simple questions such as: *"what is the meaning of until?"* and *"can you show me an example of always operator?"*. We believe this is an effective way of filling in knowledge gaps around the semantics of LTL on the fly. That is, the user can start constructing queries and cross check their understanding of the operators quickly along the way, reinforcing their learning at the same time.

Step-by-step tutorials

The NLI also supports step-by-step tutorials, which guide users through an example LTL querying exercise using a pre-built BMA model. At each step the NLI generates intermediate BMA model files as well as screenshots so the users can verify that they are following the tutorial correctly. The user can access these tutorials by asking questions such as: *"show me the LTL tutorials"* and selecting the complexity of the tutorial they would like to follow. This functionality is useful as it shows LTL querying in action, helping the user understand how LTL can be applied in the context of realistic biological models as well as making them familiar with LTL querying using the GUI at the same time.

Natural language understanding

In addition to understanding basic questions about LTL operators, the NLI can also interpret LTL queries from natural language. The aim of this feature is to

let biologists conceptualise their analysis as if they were discussing it with a colleague, rather than inputting instructions to a simulator in a different language. To that end, the NLI supports questions such as: *"I'd like to see a simulation where ras is 1 to begin with and sometime later proliferation is true"* or *"show me a simulation where if notch is greater than 5 then the process always results in an oscillation"*.

The queries mentioned above also show how the user can use natural language to perform inferential cell fate analysis by asking for simulations of the form *"if ϕ then ψ"*. This is effective as it provides users with a more intuitive way of expressing inferential relationships between cell states than the *implies* operator in LTL.

Finally, the NLI is able to interpret conversational temporal phrases such as: *"later"*, *"sometime later"*,*"in the future"* and *"never"* that are likely to be used by biologists when conceptualising their analysis but can be hard to encode using LTL, as multiple temporal modalities are involved. This is beneficial as it handles very subtle semantical differences that can be overlooked by users who are less familiar with LTL. An example of this is where a user might not realise that the *eventually* operator tests the current state as well as future next states and that the *next* operator is required to omit testing the current state, something that is expressed implicitly in natural language when phrases such as *sometime later* are used.

The chatbot can be effective especially when working with multiple operators where subtle differences in operator ordering can result in formulas that are semantically inconsistent with the user's intention. For example, a query that tests if a cell is *never* in the state ϕ can easily be encoded wrongly in LTL as $\neg\Box\phi$ instead of $\Box\neg\phi$, where the former actually means *"ϕ is eventually false"*. This results in a query that is inconsistent with the original intention as it will evaluate to true for traces that contain evaluations of ϕ as true, as long as ϕ evaluates to false at some time step. The same query can be performed more easily using the chatbot by describing the core intention, for example, as *"show me a simulation where it is never the case that ϕ is true"*.

Formula history

A history of user-entered formulas is maintained throughout the session to let users construct complex formulas by combining simple formulas easily. Each formula is assigned a "formula pointer", which is a generated name that the user can change to signify domain specific meaning. Users are able to access the formula list by instructing the NLI by writing phrases such as: *"show me the formula list"*. Similarly, existing formulas can be deleted and renamed. This addresses the GUI related issues identified by experienced users, who prefer a command line like interface that eliminates overheads such as scrolling and drag-drop when working with complex formulas.

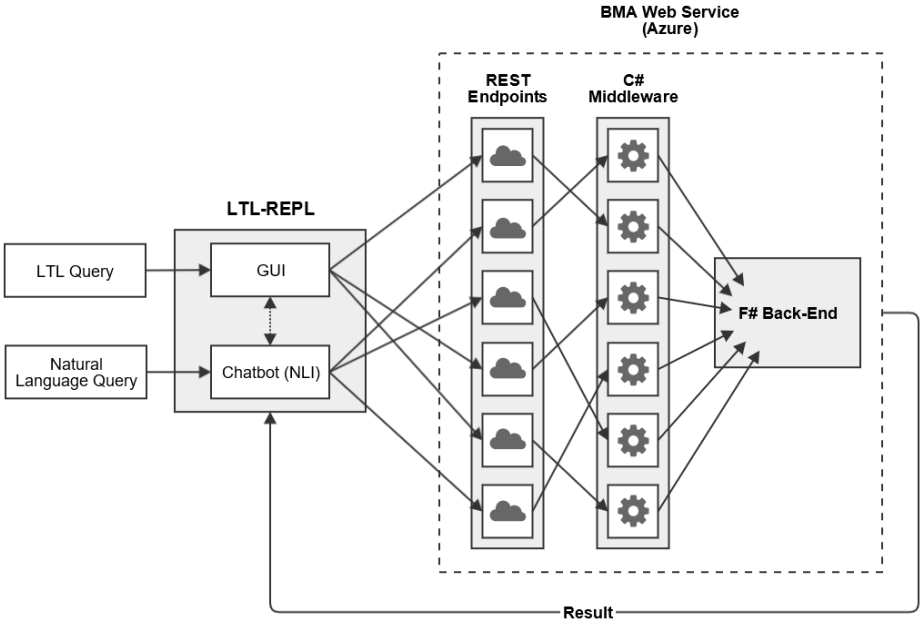

Figure 4: High level system architecture

In-line execution

The bot can also execute queries from within the context of a conversation and provide a textual summary of the results. This makes the NLI fit for exploratory research as users can leverage formula pointers to rapidly construct very complex queries and test them in place. For the full results the NLI refers the users back to the GUI.

5 System overview

The general architecture of the LTL moduele of BMA is presented in Figure 4. The *evaluation* of queries is performed in an F# back-end, which is exposed using a service-orientated architecture (SOA) through a set of REST endpoints. Taking an SOA approach allows the interfaces to be decoupled from the back-end, making the architecture more suitable for future expansion of additional back-end features.

The REST endpoints represent a Web API. This is reflected in several stages all the way to the back-end, in a traditional ASP.NET Web API 2 stack. Starting from the outside, a C# controller implements each API function. For instance, the api/Analyze API is implemented using an AnalyzeController class, and the Post method of this class takes exactly the arguments (typed — by this time — from the untyped JSON representation sent over the wire) that the front-end passes as the payload when it makes a POST call to the *api/Analyze* REST endpoint. The second stage is between this thin C# controller implementation

and the core back-end functionality. This is mediated by a C# interface that is essentially the set of functions (checkStability, findCExBifurcates, checkLTL, simulateTick, etc) that can be called in an atomic/synchronous way; the result returned immediately to the front-end. The interface has no state, and uses types that are shared between C# and F#.

The back-end runs on the Windows Azure cloud platform. LTL queries put a significant load on the back-end and we use several approaches to ensure BMA remains responsive. Firstly, front-end requests and back-end analysis run in two separate Azure roles (akin to processes in the cloud). This enables us to support interface updates instantaneously even when the system is busy. Secondly, Azure's autoscale feature automatically spawns new instances of servers if the average CPU usage increases above a given threshold. Finally, long queries are terminated after a time out period, and the user is notified.

GUI Implementation

The GUI is developed as an HTML5 front-end, written in TypeScript using the Model-View-Presenter (MVP) architectural pattern: the Model stores the objects (biological models, proof results, simulation graphs) of the current session; the View implements the graphics the user sees and interacts with. The implementation uses and extends JQueryUI HTML5 controls; the Presenter is the "middle man", which understands and performs all the logic of the front-end. The Presenter passes actions undertaken in the View by the user to the Model for the latter's state to be updated, and similarly passes updates from the Model back to the View for the canvas to be updated.

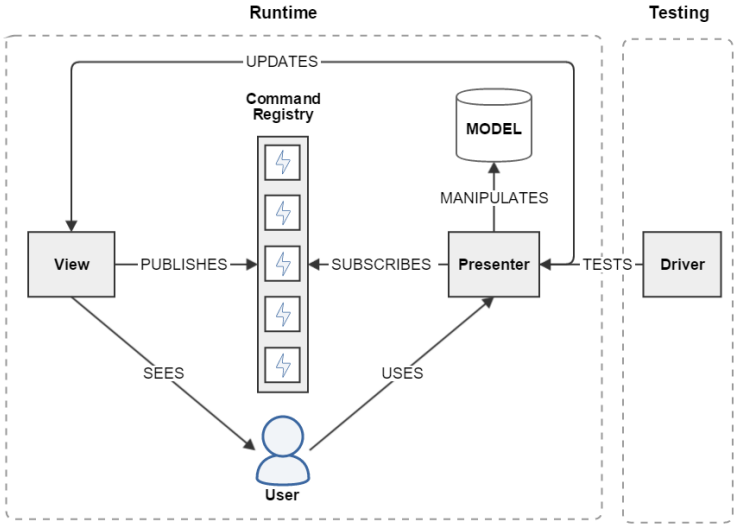

Figure 5: MVP implementation in the GUI

As seen in Figure 5, we take a Pub/Sub (publish-subscribe) approach to handle View-Presenter communication, decoupling the View logic from the Presenter logic. The Viewer publishes a set of events to a Command Registry, which the Presenter subscribes to. As the user interacts with the View, events are asynchronously fired and handled by the Presenter. This adds scalability to the GUI as additional Presenters can be added easily to handle multiple view events concurrently, to support more complex GUIs in the future. The Pub/Sub mechanism also allows us to perform functional unit testing of the Presenter logic independently of the View. We do this by using the Jasmine testing library to "mock" the View logic through a Driver object and invoke the event handlers in the Presenter directly.

The MVP implementation is a light-weight, custom-built one, not relying on any existing JavaScript frameworks. The whole implementation is bundled and minified, and then deployed on to the server.

NLI Implementation

The NLI is developed as an independent Node.js application and written in TypeScript. It uses technologies such as the Microsoft Bot Framework [13], Microsoft Cognitive Services [14] and Chevrotain [15]. As seen in Figure 6, the Bot Framework is used to expose the NLI's functionality as a chatbot. The Bot Framework is a software development kit (SDK) that allows cross platform conversational agents (i.e., chatbots) to be developed easily. We use this framework to handle the boilerplate aspects of a conversational agent. For example, client connectivity, session management and file import/export. While we currently only support the Skype chat client, the Bot Framework can allow us to support other chat clients such as Facebook messenger, Telegram and Slack easily in the future thanks to its homogeneous REST interface, which we have implemented in the BMA Bot Service.

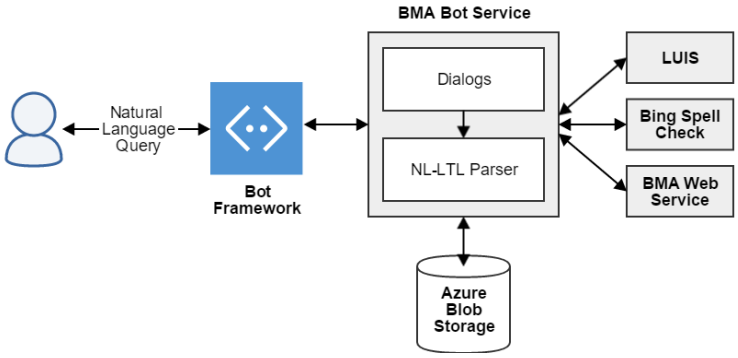

Figure 6: NLI Architecture

The BMA Bot Service forms the core of the NLI and consists of two main components: Dialogs and NL-LTL Parser. The NLI generates BMA model files

whenever a user requests an LTL query to be executed. This model can be automatically opened in BMA by following a URL sent in the conversation. This feature relies on Azure blob storage and saves the user the need to download files from Skype and upload them to BMA.

We used Microsoft's Language Understanding Intelligent Service (LUIS) to solve the problem of handling a range of query types through a single interface. That is, from simple questions about LTL operators to LTL formulas described in natural language. LUIS is a web service that allows language understanding models to be built from sample utterances, which can classify natural language queries based on their underlying *intent*. In order to build a LUIS model fit for our problem domain, we worked with biologists to identify the kind of phrases that they may utter when interacting with the NLI. We used this information to build a hierarchy of intents (Figure 7) that covered all query types supported by the NLI and used it to train our LUIS model.

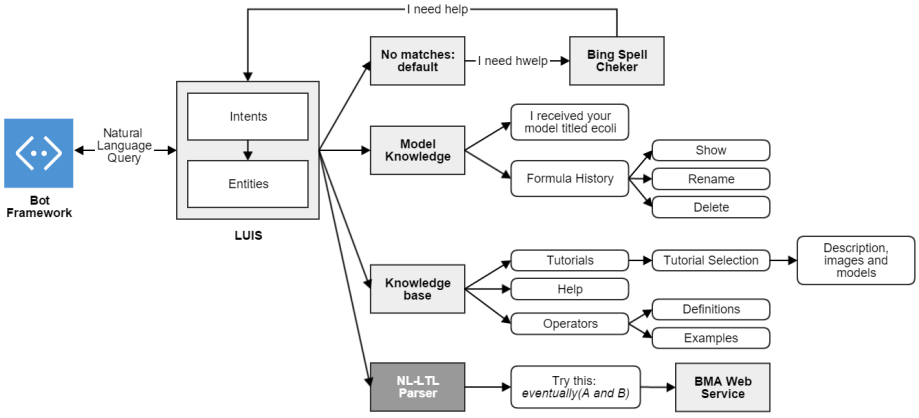

Figure 7: NLI natural language processing infrastructure

As seen in Figure 7, valid queries can be classified as belonging to either the Model Knowledge, Knowledge Base or NL-LTL Parser intent categories. Queries that cannot be assigned an intent category are passed through the Bing spell checker web service once and fed into the system as a new user input, given spelling errors were corrected in the original query. The Model Knowledge intent category captures queries that either change the state of the current BMA model. For example, uploading of a new model or contain references to aspects relating to an existing model such as formula pointers. Queries that require the NLI to use its Knowledge Base in order to provide definitions or example usage of LTL operators as well as run tutorials are labelled as having a Knowledge Base intent. Finally, queries relating to the construction and execution of LTL formulas are labelled with an NL-LTL Parser intent and are passed through the NL-LTL Parser pipeline, where a further set of NLP tasks are performed to infer logical structure from NL.

6 Conclusion and Future Work

Providing an extensive front-end for users to use LTL represents a major step forward in functionality in BMA. In addition to expanding the types of analysis that can be performed in the tool, the underlying changes, both in terms of the web app architecture and the visual technique for constructing formulas, will support the future development of new features. Future work will draw on these improvements to expand the range of analyses offered and refine existing features, such as the target function editor, by adapting the LTL query editor.

We are currently working on taking the insights from the graphical editor for LTL to offer an alternative approach to the construction of target functions. This will carry some of the advantages of the GUI editing, such as no need to consider operator precedence and parentheses balancing to another technical part of model construction. At the same time, we would like to offer an alternative textual editor interchangeably with both formula creation GUIs.

Some of our most advanced users have been using directly the back-end part of BMA to do thorough testing of their models. We are looking into better supporting these users through incorporation of some of their requirements into the NLI interface. Thus, an advanced user would be able to replace a script that automates some calls to the back-end with instructions to the NLI interface to do the same. Furthermore, expanding the natural language understanding capabilities of the NLI would increase the tool's effectiveness and improve user experience. For example, we could replace the reliance on a static dictionary by using word semantic similarity, as in [16, 17].

References

1. Bonzanni, N., Garg, A., Feenstra, K.A., SchÃijtte, J., Kinston, S., Miranda-Saavedra, D., Heringa, J., Xenarios, I., GÃűttgens, B.: Hard-wired heterogeneity in blood stem cells revealed using a dynamic regulatory network model. Bioinformatics **29** (2013) i80–i88
2. Chuang, R., Hall, B., Benque, D., Cook, B., Ishtiaq, S., Piterman, N., Taylor, A., Vardi, M., Koschmieder, S., Gottgens, B., Fisher, J.: Drug target optimization in chronic myeloid leukemia using innovative computational platform. Scientific Reports, 5:8190 (2015)
3. Moignard, V., Woodhouse, S., Haghverdi, L., Lilly, J., Tanaka, Y., Wilkinson, A., Buettner, F., Nishikawa, S., Piterman, N., Kouskoff, V., Theis, F., Fisher, J., Gottgens, B.: Decoding the regulatory network of early blood development from single cell gene expression measurements. Nature Biotechnology, 33:269âĂŞ276 (2015)
4. Remy, E., Rebouissou, S., Chaouiya, C., Zinovyev, A., Radvanyi, F., Calzone, L.: A modeling approach to explain mutually exclusive and co-occurring genetic alterations in bladder tumorigenesis. Cancer Research **75** (2015) 4042–4052
5. Terfve, C.D.A., Wilkes, E.H., Casado, P., Cutillas, P.R., Saez-Rodriguez, J.: Large-scale models of signal propagation in human cells derived from discovery phosphoproteomic data. Nature communications **6** (2015) 8033

6. Benque, D., Bourton, S., Cockerton, C., Cook, B., Fisher, J., Ishtiaq, S., Piterman, N., Taylor, A., Vardi, M.: BMA: Visual tool for modeling and analyzing biological networks. In: 24th International Conference on Computer Aided Verification. Volume 7358 of Lecture Notes in Computer Science., Springer (2012) 686–692
7. Naldi, A., Thieffry, D., Chaouiya, C.: Decision diagrams for the representation and analysis of logical models of genetic networks. In: Computational Methods in Systems Biology. Volume 4695 of Lecture Notes in Computer Science., Springer (2007) 233–247
8. Cook, B., Fisher, J., Krepska, E., Piterman, N.: Proving stabilization of biological systems. In: Verification, Model Checking, and Abstract Interpretation. Volume 6538 of Lecture Notes in Computer Science., Springer (2011) 134–149
9. Cook, B., Fisher, J., Hall, B., Ishtiaq, S., Juniwal, G., Piterman, N.: Finding instability in biological models. In: 26th International Conference on Computer Aided Verification. Volume 8559 of Lecture Notes in Computer Science., Vienna, Austria, Springer-Verlag (2014) 358–372
10. Claessen, K., Fisher, J., Ishtiaq, S., Piterman, N., Wang, Q.: Model-checking signal transduction networks through decreasing reachability sets. In: 25th International Conference on Computer Aided Verification. Volume 8044 of Lecture Notes in Computer Science., Springer (2013) 85–100
11. Schaub, M., Henzinger, T., Fisher, J.: Qualitative networks: A symbolic approach to analyze biological signaling networks. BMC Systems Biology **1** (2007)
12. Bean, D., Heimbach, J., Ficorella, L., Micklem, G., Oliver, S., Favrin, G.: esyN: Network building, sharing, and publishing. PLoS ONE **9** (2014) e106035
13. Microsoft: Microsoft Bot Framework. https://dev.botframework.com/ (2016)
14. Microsoft: Microsoft Cognitive Services. https://www.microsoft.com/cognitive-services/ (2016)
15. SAP: Chevrotain a JavaScript parsing DSL. https://github.com/SAP/chevrotain/ (2014)
16. van der Plas, L., Tiedemann, J.: Finding synonyms using automatic word alignment and measures of distributional similarity. In: Proceedings of the COLING/ACL on Main Conference Poster Sessions. COLING-ACL '06, Stroudsburg, PA, USA, Association for Computational Linguistics (2006) 866–873
17. Richardson, R., Smeaton, A., Murphy, J.: Using wordnet as a knowledge base for measuring semantic similarity between words (1994)