PyFoReL: A Domain-Specific Language for Formal Requirements in Temporal Logic

Jacob Anderson
School of Comp. and Aug. Intelligence
Arizona State University
Tempe, AZ, USA
jwande18@asu.edu

Mohammad Hekmatnejad
School of Comp. and Aug. Intelligence
Arizona State University
Tempe, AZ, USA
mhekmatn@asu.edu

Georgios Fainekos
Future Research Department
Toyota Research Institute of North America
Ann Arbor, MI, USA
georgios.fainekos@toyota.com

***Abstract*—Temporal Logic (TL) bridges the gap between natural language and formal reasoning in the field of complex systems verification. However, in order to leverage the expressivity entailed by TL, the syntax and semantics must first be understood—a large task in itself. This significant knowledge gap leads to several issues: (1) the likelihood of adopting a TL-based verification method is decreased, and (2) the chance of poorly written and inaccurate requirements is increased. In this ongoing work, we present the Pythonic Formal Requirements Language (PyFoReL) tool: a Domain-Specific Language inspired by the programming language Python to simplify the elicitation of TL-based requirements for engineers and non-experts.**

***Index Terms*—domain-specific language, temporal logic, formal requirements, requirements-based testing**

I. Introduction

A significant number of formal reasoning frameworks exist today that aid in the testing of complex requirements for Cyber-Physical Systems (CPS) [1]–[4]. These tools rely on different branches of Temporal Logic (TL) to formalize a system's requirements against some desired specification. Consequently, this dependency limits the effective usage of these tools to experts of TL while increasing the risk of ill-formed requirements formulated by non-experts [5]. Therefore, with such obstacles, wide-scale adoption of these frameworks becomes harder.

In this work, we present a preliminary version of an ongoing effort of the Pythonic Formal Requirements Language (PyFoReL) tool: a Domain-Specific Language (DSL) inspired by Python [6] to ease the requirements elicitation process for TL-based testing frameworks and tools. Our tool semantically translates PyFoReL programs into equivalent TL formulas and reports erroneous requirements to mitigate the risk of wrongly formed specifications which can lead to wasted effort in the system testing and verification process. In addition, it supports several branches of TL including linear temporal logic, metric temporal logic, signal temporal logic, timed propositional temporal logic, timed quality temporal logic [7], and spatio-temporal perception logic [8].

II. PyFoReL

A PyFoReL program consists of a sequence of statements. Currently, there are six core statements in the PyFoReL tool as shown in Table I. The statements are categorized into two types: simple and compound. Compound statements utilize a block structure—indicated by indentations—whereas simple statements do not. For statements in sequence, the resulting translation is the conjunction of each, and nested statements semantically represent subformulas in the resulting translation. The statement syntax can be found in the footnote below.

TABLE I
PyFoReL statement types

Statement	Purpose
Declarations	Data and time variable declarations
Function Definition	Define reusable requirements
Function Call	Modularize a requirement
Verbatim	Embed TL statements
Conditional	Support for implication operators
Temporal	Support for temporal features of TL

The semantic translation scheme of a PyFoReL program is shown in Figure 1. In the following sections, the two layers and their respective components are reviewed. This includes the DSL layer which is responsible for translating PyFoReL programs to TL formulas; and, the TL layer which is responsible for validating these resulting translations.

A. DSL Layer

Before translation, all PyFoReL programs are validated syntactically and semantically. This check provides an early catch for ill-formed requirements.

There are four types of syntactical checks performed: (1) extraneous inputs, (2) mismatched inputs, (3) block indentations, and (4) reserved word usages. If any check fails, the error is reported with the associated line and column number pair within the corresponding PyFoReL program to assist in deducing the source of error.

There are two types of semantic checks performed on PyFoReL programs: (1) undefined function calls and (2) function re-declarations. Each error is thrown similarly to the syntactic errors with additional information such as offending name and location to facilitate debugging.

[†]PyFoReL: https://gitlab.com/sbtg/pyforel

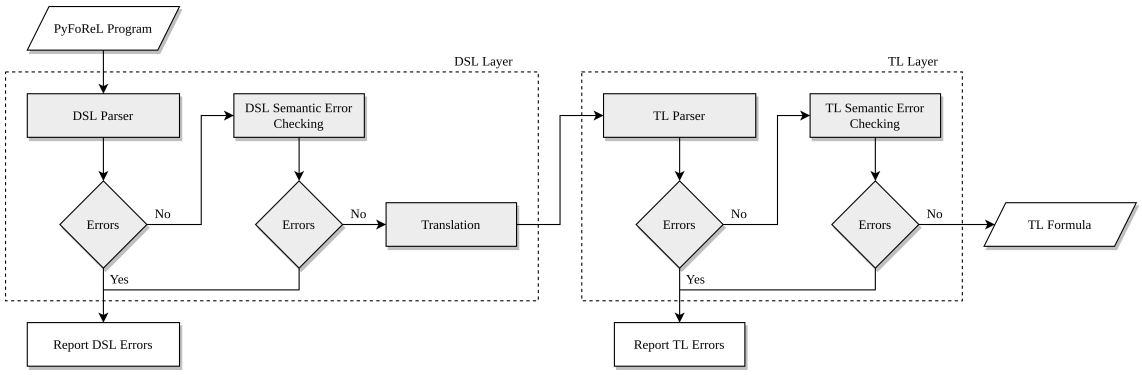

Fig. 1. The Pythonic Formal Requirements Language architecture

B. Temporal Logic Layer

When a complete PyFoReL program is translated without any errors, it is sent to the TL layer for processing. This processing aims to alleviate the necessity of understanding the complex syntax and semantics associated with TL to effectively debug formal requirements.

The set of procedures include similar steps to that of the DSL layer (i.e., syntactic and semantic checks). This includes the following syntactic errors: (1) extraneous inputs and (2) mismatched inputs; and, the following semantics errors: (1) variable re-declaration, (2) variable out-of-scope, (3) type mismatch, and (4) undeclared reference.

III. Demonstration

As a formal proof of equivalence is ongoing, we have evaluated the translation of PyFoReL programs through a number of specifications to demonstrate the equivalencies. For example, a modified version of the natural language requirement tested from [8] is showcased as follows:

There is at least one frame in which at least two unique objects are from the Car class.

where the formal requirement in spatio-temporal logic is:

$$\Diamond \exists id_1 . \exists id_2 (id_1 \neq id_2 \ \wedge \ C(id_1) = Car \ \wedge \ C(id_2) = Car)$$

The requirement is developed as a PyFoReL program with reusable logical components as shown below:

```
1  func is_car(object obj):
2      verb:
3          "class(obj) == Car"
4
5  func is_not_equal(object obj1, object obj2):
6      verb:
7          "obj1 != obj2"
8
9  eventually: # requirement starts here
10     exists obj1, obj2:
11         is_not_equal(obj1, obj2)
12         is_car(obj1)
13         is_car(obj2)
```

The resulting translated PyFoReL program produces the following equivalent spatio-temporal perception logic formula:

```
eventually (@(EXISTS, obj1, obj2) ((obj1 != obj2) &&
    (class(obj1) == Car) && (class(obj2) == Car)))
```

IV. Conclusion and Future Work

In this ongoing work, we proposed the PyFoReL tool as a simplified interface in the elicitation of formal requirements used in TL-based testing and verification frameworks for complex CPS. This tool aims to assist system model engineers and non-experts of TL to develop accurate and extensible requirements. In future work, the following is planned: (1) a requirement template system to guide non-experts to write semantically correct specifications, and (2) a user evaluation case study on the effectiveness of PyFoReL.

Acknowledgements

This work was partially supported by the NSF under grants CNS-2038666 and IIP-1361926, and the NSF I/UCRC Center for Embedded Systems.

References

[1] D. Ničković and T. Yamaguchi, "Rtamt: Online robustness monitors from stl," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2020, pp. 564–571.

[2] J. Cralley, O. Spantidi, B. Hoxha, and G. Fainekos, "Tltk: A toolbox for parallel robustness computation of temporal logic specifications," in *International Conference on Runtime Verification*. Springer, 2020, pp. 404–416.

[3] A. Donzé, "Breach, a toolbox for verification and parameter synthesis of hybrid systems," in *International Conference on Computer Aided Verification*. Springer, 2010, pp. 167–170.

[4] Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, "S-taliro: A tool for temporal logic falsification for hybrid systems," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2011, pp. 254–257.

[5] A. Dokhanchi, B. Hoxha, and G. Fainekos, "Formal requirement debugging for testing and verification of cyber-physical systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 17, no. 2, pp. 1–26, 2017.

[6] P. C. Team, *Python: A dynamic, open source programming language*, Python Software Foundation, 2021. [Online]. Available: https://www.python.org/

[7] A. Balakrishnan, A. G. Puranic, X. Qin, A. Dokhanchi, J. V. Deshmukh, H. B. Amor, and G. Fainekos, "Specifying and evaluating quality metrics for vision-based perception systems," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 1433–1438.

[8] M. Hekmatnejad, "Formalizing safety, perception, and mission requirements for testing and planning in autonomous vehicles," Ph.D. dissertation, Arizona State University, 2021.