NL2LTL – A Python Package for Converting Natural Language (NL) Instructions to Linear Temporal Logic (LTL) Formulas

Francesco Fuggitti[1,2], **Tathagata Chakraborti**[3]
[1] Sapienza University, Rome (Italy)
[2] York University, Toronto (Canada)
[3] IBM Research, Cambridge (USA)
fuggitti@diag.uniroma1.it, tathagata.chakraborti1@ibm.com

Abstract

This is a demonstration of our newly released Python package `NL2LTL` which leverages the latest in natural language understanding (NLU) and large language models (LLMs) to translate natural language instructions to linear temporal logic (LTL) formulas. This allows direct translation to formal languages that a reasoning system can use, while at the same time, allowing the end-user to provide inputs in natural language without having to understand any details of an underlying formal language. The package comes with support for a set of default LTL patterns, corresponding to popular DECLARE templates, but is also fully extensible to new formulas and user inputs. The package is open-source and is free to use for the AI community under the MIT license. **Open Source:** https://github.com/IBM/nl2ltl. **Video Link:** https://bit.ly/3dHW5b1

Natural Language and LTL

A host of enterprise applications revolve around the management of workflows – this includes data processing pipelines in AutoML (He, Zhao, and Chu 2021), web service composition (Lemos, Daniel, and Benatallah 2015), dialogue trees in conversational systems (Muise et al. 2020), and so on. An emerging theme in this area is the adoption of natural language as a desired input modality (Chakraborti et al. 2022), aimed at reducing the barrier of entry and expertise required for users looking to adopt workflow management tools.

One of the scientific advances towards easier authoring tools for workflow management is the notion of *"declarative specifications"*. An important specification language in this paradigm is LTL (Pnueli 1977), which allows for compilations to many well-known problems in process management e.g. conformance checking (De Giacomo et al. 2017; De Giacomo 2021), while also admitting translations to specifications of standard reasoning engines such as automated planners (more on this later).

While allowing for a declarative design paradigm, LTL also has a secondary benefit: LTL formulas can be readily described in an easy-to-understand natural language format. For example, DECLARE templates (Pesic and Van der Aalst 2006) – a very popular specification language in the world of business process management – is readily translatable to LTL formulas (De Giacomo, De Masellis, and Montali 2014); and there exists a variety of tools to generate a human-readable construct given an LTL formula (Cherukuri, Ferrari, and Spoletini 2022). In this work, we aim to make the journey in the opposite direction – from natural language to LTL. This has two main advantages: 1) unstructured inputs to a system can be translated to a form that reasoning engines can consume; while 2) the interface to the end-user remains as accessible as possible.

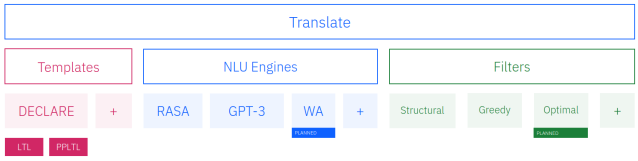

Figure 1: `NL2LTL` components

Related Work Existing works fall under two buckets: one which admits support for a range of LTL formulas but compromises on the expressiveness of the input (Hahn et al. 2022; Schmitt 2022; Narizzano et al. 2018; Narizzano and Vuotto 2017), and the other which admits natural language inputs but were built for a particular domain like robotics and are not readily useful as a general purpose package for practitioners (Wang et al. 2020; Wang 2020; Nikora and Balcom 2009; Dwyer, Avrunin, and Corbett 1998; Kim, Banks, and Shah 2017; Lignos et al. 2015). Furthermore, among these works, other than (Wang 2020; Narizzano and Vuotto 2017; Schmitt 2022), none have publicly available code and are therefore not readily usable for practitioners. On the other hand, there are a couple of code bases that have also attempted natural language to LTL translation (Head 2015; Zheng 2020) but they are at a very rudimentary stage with little to no support or documentation. To the best of our knowledge, our package is the first one going public with support for a significant breadth of LTL patterns and an extensible API to make it usable in different domains.

`NL2LTL` Overview

`NL2LTL` is built with a unique focus on extensibility: 1) The inputs and outputs are domain agnostic so it can be adopted

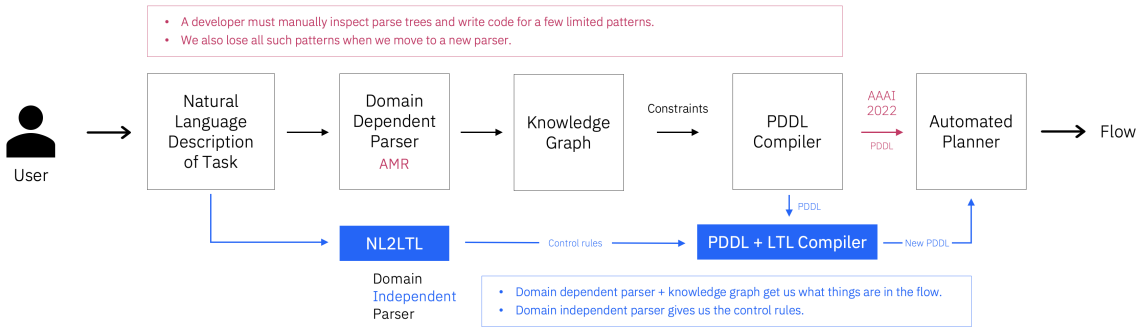

Figure 2: An application of NL2LTL to a real industrial application (Brachman et al. 2022).

```
DECLARE Template: (ChainResponse Slack Gmail)
English meaning:  Every time activity Slack happens, it must be
                  directly followed by activity Gmail.
Confidence:       0.9999997615814209

DECLARE Template: (ExistenceTwo Slack)
English meaning:  Slack will happen at least twice.
Confidence:       1.0441302578101386e-07

DECLARE Template: (RespondedExistence Slack Gmail)
English meaning:  If Slack happens at least once then Gmail has
                  to happen or happened before Slack.
Confidence:       6.342362723898987e-08
```

Figure 3: Sample output (pretty print) of NL2LTL, illustrating candidate DECLARE templates suggested by the package, using the Rasa NLU Engine, for the request: *"Send me a Slack after receiving a Gmail"*.

into any domain of choice; and 2) any and all components – be it the natural language understanding module or the scope of supported LTL formulas – are extendable or modifiable.

Sample Interactions Figure 3) illustrates the NL2LTL output – a list of translations, ranked by confidence. Each candidate translation has two interesting properties: **1) Explanations:** Each formula is translated back to English to illustrate how the formal representation interprets it. Depending on the downstream application, the developer can use this to explain to the end user to what extent the translation matches their original request; and **2) Alternative Representations:** While this example if for DECLARE templates, each candidate can also be translated to other equivalent representations, such as LTL_f (De Giacomo and Vardi 2013) and PPLTL (De Giacomo et al. 2020).

Architecture The primary function of NL2LTL is a translation function, which is assisted by three different components (Figure 1). The first component is a set of supported patterns, or "templates", to be identified e.g. DECLARE templates (Pesic and Van der Aalst 2006). While this covers a wider range of LTL formulas, a developer can also add their own templates specific to their application. Secondly, NL2LTL can be configured with different NLU engines while the API remains exactly the same. At the moment, NL2LTL comes with two engines pre-configured – one based on the intent-entity paradigm from Rasa (Bocklisch et al. 2017) that is traditionally used for natural language understanding, while the other is a language model-based extraction, tapping into the Open AI API (Brown et al. 2020). Finally, the package also implements a filter functions to post-process for better candidate sets of translations. This is because the patterns are not independent: 1) two LTL formulas can conflict with each other when both cannot be true at the same time; or 2) one LTL formula can be subsumed by another when the latter always implies the former.

Demonstration Logistics

The demonstration will consist of two live components – first the package itself, and then an illustration of how it can be employed in a real-world application.

Demonstration of `NL2LTL` For demonstration of the package itself, the AAAI audience will be able to interact with a command line interface where they can type in constraints in English, and explore the resultant LTL-translations along with the entities detected and what those translations mean when translated back to English.

Industry Application For the real-world industry-scale application, we will build on a system demonstration at AAAI 2022 (Brachman et al. 2022) on a web service composition task using natural language. Such a system was dependent on code that is specific to the parser in order to look for specific patterns requested by the user – this means that there is a significant developer overhead in manually investigating the parser (Abstract Syntax Representations (Astudillo et al. 2020)) for a set of inputs, writing pattern extraction methods for it limited to that scope, while eventually that code is not reusable once the system upgrades to a different parser.

By augmenting the processing pipeline with the NL2LTL package, we are able to remove all parser-specific code and instead generate the required patterns with zero maintenance overhead. The patterns detected using the package, parallel to the original processing pipeline, are eventually merged for a singular PDDL input to the automated planner in the end, using the compilation which receives as input the PDDL specification generated by the original pipeline along with the LTL patterns detected by the NL2LTL package, and produces a compiled PDDL for the planner where the control rules are enforced (Figure 2). For the LTL to PDDL compilation, we use (De Giacomo, Favorito, and Fuggitti 2022) but any existing approach (Bacchus and Kabanza 2000; Baier and McIlraith 2006b,a; Torres and Baier 2015) will suffice.

Acknowledgments

Francesco started and completed most of this work as an intern at IBM Research. Francesco was also partially supported by the ERC Advanced Grant WhiteMech (No. 834228), the EU ICT-48 2020 project TAILOR (No. 952215), the PRIN project RIPER (No. 20203FFYLK), and the JPMorgan AI Faculty Research Award "Resilience-based Generalized Planning and Strategic Reasoning".

References

Astudillo, R. F.; Ballesteros, M.; Naseem, T.; Blodgett, A.; and Florian, R. 2020. Transition-Based Parsing with Stack-Transformers. *arXiv:2010.10669*.

Bacchus, F.; and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Artificial intelligence*, 116(1-2): 123–191.

Baier, J. A.; and McIlraith, S. A. 2006a. Planning with First-Order Temporally Extended Goals using Heuristic Search. In *AAAI*.

Baier, J. A.; and McIlraith, S. A. 2006b. Planning with Temporally Extended Goals Using Heuristic Search. In *ICAPS*.

Bocklisch, T.; Faulkner, J.; Pawlowski, N.; and Nichol, A. 2017. Rasa: Open Source Language Understanding and Dialogue Management. *arXiv:1712.05181*.

Brachman, M.; Bygrave, C.; Chakraborti, T.; Chaudhary, A.; Ding, Z.; Dugan, C.; Gros, D.; Gschwind, T.; Johnson, J.; Laredo, J.; Czasch, C. M.; Pan, Q.; Rai, P.; Ramalingam, R.; Scotton, P.; Surabathina, N.; and Talamadupula, K. 2022. A Goal-driven Natural Language Interface for Creating Application Integration Workflows. In *AAAI Demonstration*.

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language Models are Few-Shot Learners. *NeurIPS*.

Chakraborti, T.; Rizk, Y.; Isahagian, V.; Aksar, B.; and Fuggitti, F. 2022. From Natural Language to Workflows: Towards Emergent Intelligence in Robotic Process Automation. In *BPM*.

Cherukuri, H.; Ferrari, A.; and Spoletini, P. 2022. Towards Explainable Formal Methods: From LTL to Natural Language with Neural Machine Translation. In *REFSQ*.

De Giacomo, G. 2021. Artificial Intelligence-based Declarative Process Synthesis for BPM. Invited Talk.

De Giacomo, G.; De Masellis, R.; and Montali, M. 2014. Reasoning on LTL on Finite Traces: Insensitivity to Infiniteness. In *AAAI*.

De Giacomo, G.; Di Stasio, A.; Fuggitti, F.; and Rubin, S. 2020. Pure-Past Linear Temporal and Dynamic Logic on Finite Traces. In *IJCAI*, volume 20.

De Giacomo, G.; Favorito, M.; and Fuggitti, F. 2022. Planning for Temporally Extended Goals in Pure-Past Linear Temporal Logic: A Polynomial Reduction to Standard Planning. *arXiv:2204.09960*.

De Giacomo, G.; Maggi, F.; Marrella, A.; and Patrizi, F. 2017. On the Disruptive Effectiveness of Automated Planning for LTLf-Based Trace Alignment. In *AAAI*.

De Giacomo, G.; and Vardi, M. 2013. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *IJCAI*.

Dwyer, M. B.; Avrunin, G. S.; and Corbett, J. C. 1998. Property Specification Patterns for Finite-State Verification. In *Workshop on Formal Methods in Software Practice*.

Hahn, C.; Schmitt, F.; Tillman, J. J.; Metzger, N.; Siber, J.; and Finkbeiner, B. 2022. Formal Specifications from Natural Language. *arXiv:2206.01962*.

He, X.; Zhao, K.; and Chu, X. 2021. AutoML: A Survey of the State-of-the-Art. *Knowledge-Based Systems*.

Head, A. 2015. LTLTrans. https://github.com/andrewhead/LTLTrans. Accessed: 2016-02-09.

Kim, J.; Banks, C. J.; and Shah, J. A. 2017. Collaborative Planning with Encoding of Users' High-Level Strategies. In *AAAI*.

Lemos, A. L.; Daniel, F.; and Benatallah, B. 2015. Web Service Composition: A Survey of Techniques and Tools. *ACM Computing Surveys*.

Lignos, C.; Raman, V.; Finucane, C.; Marcus, M. P.; and Kress-Gazit, H. 2015. Provably correct reactive control from natural language. *Autonomous Robots*, 38(1): 89–105.

Muise, C.; Chakraborti, T.; Agarwal, S.; Bajgar, O.; Chaudhary, A.; Lastras-Montano, L. A.; Ondrej, J.; Vodolan, M.; and Wiecha, C. 2020. Planning for Goal-Oriented Dialogue Systems. *arXiv:1910.08137*.

Narizzano, M.; Pulina, L.; Tacchella, A.; and Vuotto, S. 2018. Consistency of Property Specification Patterns with Boolean and Constrained Numerical Signals. In *NASA Formal Methods Symposium*.

Narizzano, M.; and Vuotto, S. 2017. Structured Natural Language To Formal Language. https://github.com/SAGE-Lab/snl2fl. Accessed: 2018-07-12.

Nikora, A. P.; and Balcom, G. 2009. Automated Identification of LTL Patterns in Natural Language Requirements. In *Symposium on Software Reliability Engineering*.

Pesic, M.; and Van der Aalst, W. M. 2006. A Declarative Approach for Flexible Business Processes Management. In *BPM*.

Pnueli, A. 1977. The Temporal Logic of Programs. In *Symposium on Foundations of Computer Science*.

Schmitt, F. 2022. ML2: Machine Learning for Mathematics and Logics. https://github.com/reactive-systems/ml2. Accessed: 2022-02-03.

Torres, J.; and Baier, J. A. 2015. Polynomial-Time Reformulations of LTL Temporally Extended Goals into Final-State Goals. In *IJCAI*.

Wang, C. 2020. Grounded LTL Parser. https://github.com/czlwang/grounded_LTL_parser. Accessed: 2021-04-27.

Wang, C.; Ross, C.; Kuo, Y.; Katz, B.; and Barbu, A. 2020. Learning a Natural-Language to LTL Executable Semantic Parser for Grounded Robotics. In *Proceedings of Machine Learning Research*.

Zheng, S. 2020. Translation from natural language to Linear Temporal Logic. https://github.com/suchzheng2/Lang2LTL. Accessed: 2020-12-12.