Standard Grammars for Temporal Logics

This manuscript (permalink) was automatically generated from marcofavorito/tl-grammars@b3e0368 on May 9, 2021.

https://arxiv.org/abs/2012.13638
marcofavorito/tl-grammars

Document version: v0.1.1

WARNING: this version v0.1.1 is a draft. You are encouraged to email the contact author for any comment or suggestion.

Authors

- **Marco Favorito**
  0000-0001-9566-3576 · marcofavorito · https://marcofavorito.me
  Department of Computer, Control and Management Engineering - Sapienza University of Rome

Abstract

The heterogeneity of tools that support temporal logic formulae poses several challenges in terms of interoperability. This document proposes standard grammars for Linear Temporal Logic (LTL) [1] and Linear Dynamic Logic [2,3]

Introduction

This section explains the motivations behind the existence of this standard, states the goals of the standard, describes the notation conventions used thorough the document, and lists the normative references[1].

Motivation

Temporal logics have a long history [4]. One of the most influential formalisms is Linear Temporal Logic (LTL) [1], which has been applied for program specification and verification. The variant over finite traces has been introduced in [3]. Linear Dynamic Logic (LDL) [2,3] is the extension of LTL with regular expressions (RE). The idea behind LDL is to have a formalism that merges the declarativeness and convenience of LTL, as expressive as star-free RE, with the expressive power of RE. The finite trace setting has been explored by [3] for LTL/LDL and [5] for PLTL/PLDL. The syntax that naturally supports empty traces has been employed in [6] for LTL/LDL and [5] for PLTL/PLDL.

The topic has gained more and more attention both in academia and industry, also because such logics have been considered compelling also from a practical point of view. Among areas of Computer Science and Artificial Intelligence, we encounter reactive synthesis [7], model checking [8], planning with temporal goal [9], theory of Markov Decision Process with non-Markovian rewards [10], business processes specification [11], just to name a few. For what concerns industry applications, Intel proposed the industrial linear time specification language `ForSpec` [12], and the IEEE association standardized the *Property Specification Language* (PSL) [13]. Both standards witness the need of specifications based on LTL and regular expressions. Also, the research community has proposed a plethora of software tools and libraries to handle LTL and/or LDL formulas for a variety of purposes: `Spot` [14,15], `Owl` [16], `SPIN` [17], `Syft` [18], `Lisa` [19], `FLLOAT` [20,21], `LTLf2DFA` [22], and more. Another related work is represented by TLSF v1.1 [23], although its focus is on a format for LTL synthesis problems.

All these tools and formats assume the input formulae to be written in a certain grammar. Unfortunately, as often happens when dealing with parser implementations with lack of coordination, the grammars to represent the formulae have some form of discrepancies; e.g. different alternative ways to denote boolean conjunctions or temporal operators, different lexical rules to describe the allowed atomic propositions or boolean constants, underspecifications on how to handle special characters (linefeed, tab, newline, etc.), how to handle associativity of the operators.

Goals

To enhance interoperability between the aforementioned tools, this document proposes a standard grammar for writing temporal logic formulae. In particular, we specify grammars for:

- Linear Temporal Logic (LTL)
- Linear Dynamic Logic (LDL)

In future versions of this standard, we would like to provide grammars for:

- Past Linear Temporal Logic (PLTL)
- Past Linear Dynamic Logic (PLDL)

We would like this standard to be:

- An *open* standard, fostering collaboration and contributions from the research community;
- As much compliant as possible to existing and widely used tools;
- Written by researchers, for researchers. In other words, this is not strictly tight to industrial needs; for instance, we deliberately dropped the modeling of multiple clock and reset signals of `ForSpec` and `PSL`, as these are constructs not relevant for domains outside formal verification.
- Tool-agnostic. Often, grammars are reported alongside software manuals and descriptions. Instead, our aim is to propose a common denominator for all the grammars in use.

Notation

We describe the syntax in Extended Backus-Naur Form (EBNF) [24]. We follow the notation used for the specification of XML [25]; we discarded the EBNF standard version ISO/IEC 14977 [26], as it has been often rejected by the community of those who write language specifications for a variety of reasons [27,28].

Normative

We refer to [29] for requirement level key words. We also refer to Unicode standard [30,31] to define legal characters. For versioning this standard, we use SemVerDocs [32], inspired by SemVer [33].

Common definitions

In this section, we describe syntactic rules shared across every logic formalism.

Characters

Parsers MUST be able to accept sequence of *characters* (see definition below) which represent temporal logic formulae. A *character* is an atomic unit of text as specified by ISO/IEC 10646:2020 [30]. Legal characters are tab, carriage return, line feed, and the ASCII characters of Unicode and ISO/IEC 10646.

The range of characters to be supported is defined as:

```
Char ::= #x9 | #xA | #xD | [#x20-#x7e]
```

That is, the character tabulation, line feed, carriage return, and all the printable ASCII characters.

Boolean constants

For LTL and PLTL, we use `true` and `false` to denote boolean constants. For LDL and PLDL, we make a further distinction between *propositional* booleans, denoted by `true` and `false`, and *logical* booleans, denoted by `tt` and `ff`.

```
True  ::= "true"
False ::= "false"
TT    ::= "tt"
FF    ::= "ff"
PropBooleans  ::= TRUE | FALSE
LogicBooleans ::= TT | FF
```

Atomic Propositions

An atomic proposition is a string of characters. In particular, it can be:

- any string of printable characters, excepted the quotation character used (see `QuotedName`)
- any string of at least one character that starts with `[A-Za-z_]` and continues with `[A-Za-z0-9_]`.

```
NameStartChar ::= [A-Z] | [a-z] | "_"
NameChar ::= NameStartChar | [0-9]
Name ::= NameStartChar (NameChar)*
QuotedName ::= ('"' [^"\n\t\r]* '"') | ("'" [^'\n\t\r]* "'")
Atom ::= Name | QuotedName
```

Boolean operators

The supported boolean operations are: negation, conjunction, disjunction, implication, equivalence and exclusion.

Follows the list of characters used for each operator:

- negation: `!`, `~`;
- conjunction: `&`, `&&`;
- disjunction: `|`, `||`;
- implication: `->`, `=>`;
- equivalence: `<->`, `<=>`;
- exclusive disjunction: `^`;

```
Non   ::= "!" | "~"
And   ::= "&" | "&&"
Or    ::= "|" | "||"
Impl  ::= "->" | "=>"
Equiv ::= "<->" | "<=>"
Xor   ::= "^"
```

Parenthesis

We use `(` and `)` for parenthesis.

```
LeftParen  ::= "("
RightParen ::= ")"
```

White Spaces

It is often convenient to use “white spaces” (spaces, tabs, and blank lines) to set apart the formulae for greater readability. These characters MUST be ignored when processing the text input.

LTL

In this section, we specify a grammar for LTL.

Atoms

An LTL formula is defined over a set of *atoms*. In this context, an atom formula is defined by using the `Atom` regular language defined above:

```
LTLAtom ::= Atom
```

Temporal operators

Here we specify the regular languages for the temporal operators.

- (Weak) Next: `X`;
- Strong Next: `X[!]`;
- (Strong) Until: `U`;
- Weak Until: `W`;
- (Weak) Release: `R`, `V`;
- Strong Release: `M`;
- Eventually: `F`;
- Always: `G`;

In EBNF format:

```
WeakNext      ::= "X"
Next          ::= "X[!]"
Until         ::= "U"
WeakUntil     ::= "W"
Release       ::= "R" | "V"
StrongRelease ::= "M"
Eventually    ::= "F"
Always        ::= "G"
```

Grammar

```
ltl_formula ::= LTLAtom
                | True
                | False
                | LeftParen ltl_formula RightParen
                | Not ltl_formula
                | ltl_formula And ltl_formula
                | ltl_formula Or ltl_formula
                | ltl_formula Impl ltl_formula
                | ltl_formula Equiv ltl_formula
                | ltl_formula Xor ltl_formula
                | ltl_formula Until ltl_formula
                | ltl_formula WeakUntil ltl_formula
                | ltl_formula Release ltl_formula
                | ltl_formula StrongRelease ltl_formula
                | Eventually ltl_formula
                | Always ltl_formula
                | WeakNext ltl_formula
                | Next ltl_formula
```

For the semantics of these operators, we refer to [1] for the infinite setting, and [3] for the finite setting.

Precedence and associativity of operators

The precedence and associativity of the LTL operators are described by the following table (priorities from lowest to highest). For brevity, aliases for boolean operators are omitted.

associativity	operators
right	`->`, `<->`
left	`^`
left	`|`
left	`&`
right	`U`, `W`, `M`, `R`
right	`F`, `G`
right	`X`, `X[!]`
right	`!`

LDL

In this section, we specify a grammar for LDL.

Temporal operators

LDL supports two temporal operators:

- *Diamond* operator: `<regex>ldl_formula`;

- *Box* operator: `[regex]ldl_formula`;

`regex` will be presented in the next paragraph.

```
LeftDiam  ::= "<"
RightDiam ::= ">"
LeftBox   ::= "["
RightBox  ::= "]"
```

In EBNF format, an LDL formula is defined as follows:

```
ldl_formula ::= TT
              | FF
              | LeftParen ldl_formula RightParen
              | Not ldl_formula
              | ldl_formula And ldl_formula
              | ldl_formula Or ldl_formula
              | ldl_formula Impl ldl_formula
              | ldl_formula Equiv ldl_formula
              | LeftDiam regex RightDiam ldl_formula
              | LeftBox regex RightBox ldl_formula
```

Regular Expressions

In this section, we define the regular expression used by Diamond and Box operators.

A regular expression is defined inductively as:

- a *propositonal formula* over as set of propositional atoms.
- a *test expression*: `ldl_formula?
- a *concatenation* between two regular expressions: `regex_1 ; regex_2`
- a *union* between two regular expressions: `regex_1 + regex_2`
- a *star* operator over a regular expression: `regex*`

The symbols are listed below:

```
Test   ::= "?"
Concat ::= ";"
Union  ::= "+"
Star   ::= "*"
```

The EBNF grammar for a regular expression is:

```
propositional ::= Atom
                | True
                | False
                | LeftParen propositional RightParen
                | Not propositional
                | propositional And propositional
                | propositional Or propositional
                | propositional Impl propositional
                | propositional Equiv propositional
                | propositional Xor propositional

regex ::= propositional
        | LeftParen regex RightParen
        | regex Test
        | regex Concat regex
        | regex Union regex
        | regex Star
```

For the semantics of the operators, we refer to [3].

Precedence and associativity of operators

The precedence and associativity of the LDL operators are described by the following table (priorities from lowest to highest). For brevity, aliases for boolean operators are omitted.

associativity	operators
right	`->`, `<->`
left	`^`
left	`|`
left	`&`
N/A	`<>`, `[]`
left	`;`
left	`+`
left	`*`
left	`?`
right	`!`

Future work

In future versions of this standard, we would like to add:

- `Spot`-like syntactic sugars for regular expressions (SERE) and temporal operators [14,23];
- Compatibility with the PSL standard [13];
- Support full Unicode characters, so to use UTF-8 characters like ○ (U+25CB) for the Next operator and ◇ (U+25C7) for the Eventually operator etc. as alternative symbols.

- Grammars for PLTL and PLDL.

References

1. **The temporal logic of programs**
   Amir Pnueli
   18th Annual Symposium on Foundations of Computer Science (sfcs 1977) (1977-10)
   http://dx.doi.org/10.1109/sfcs.1977.32
   DOI: 10.1109/sfcs.1977.32

2. **The rise and fall of LTL**
   Moshe Y Vardi
   GandALF (2011)

3. **Linear Temporal Logic and Linear Dynamic Logic on Finite Traces**
   Giuseppe De Giacomo, Moshe Y. Vardi
   Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (2013)
   http://dl.acm.org/citation.cfm?id=2540128.2540252
   ISBN: 978-1-57735-633-2

4. **A Survey on Temporal Logics**
   Savas Konur
   (2010-05) http://arxiv.org/abs/1005.3199v3

5. **Pure-Past Linear Temporal and Dynamic Logic on Finite Traces**
   Giuseppe De Giacomo, Antonio Di Stasio, Francesco Fuggitti, Sasha Rubin
   Twenty-Ninth International Joint Conference on Artificial Intelligence and Seventeenth Pacific Rim International Conference on Artificial Intelligence IJCAI-PRICAI-20 (2020-07)
   http://dx.doi.org/10.24963/ijcai.2020/690
   DOI: 10.24963/ijcai.2020/690 · ISBN: ['9780999241165']

6. **LTLf/LDLf Non-Markovian Rewards**
   Ronen Brafman, Giuseppe De Giacomo, Fabio Patrizi
   (2018) https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17342

7. **Synthesis for LTL and LDL on Finite Traces**
   Giuseppe De Giacomo, Moshe Y. Vardi
   Proceedings of the 24th International Conference on Artificial Intelligence (2015)
   http://dl.acm.org/citation.cfm?id=2832415.2832466
   ISBN: 978-1-57735-738-4

8. **Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications**
   E. M. Clarke, E. A. Emerson, A. P. Sistla
   ACM Trans. Program. Lang. Syst. (1986) https://doi.org/10.1145/5397.5399
   DOI: 10.1145/5397.5399

9. **Planning for temporally extended goals**
   Fahiem Bacchus, Froduald Kabanza
   Annals of Mathematics and Artificial Intelligence (1998)

10. **Rewarding behaviors**
    Fahiem Bacchus, Craig Boutilier, Adam Grove

Proceedings of the National Conference on Artificial Intelligence (1996)

11. **Enacting declarative languages using LTL: avoiding errors and improving performance**
    Maja Pešić, Dragan Bošnački, Wil MP van der Aalst
    International SPIN Workshop on Model Checking of Software (2010)

12. **The ForSpec temporal logic: A new temporal property-specification language**
    Roy Armoni, Limor Fix, Alon Flaisher, Rob Gerth, Boris Ginsburg, Tomer Kanza, Avner Landver, Sela Mador-Haim, Eli Singerman, Andreas Tiemeyer, others
    International Conference on Tools and Algorithms for the Construction and Analysis of Systems (2002)

13. **IEEE Standard for Property Specification Language (PSL)**
    IEEE
    IEEE (2010) http://dx.doi.org/10.1109/ieeestd.2010.5446004
    DOI: 10.1109/ieeestd.2010.5446004 · ISBN: ['9780738162553']

14. **Spot's Temporal Logic Formulas**
    Alexandre Duret-Lutz
    Tech. rep. Available online: https://spot.lrde.epita.fr/tl.pdf (2016)

15. **Spot 2.0A Framework for LTL and 19970\\backslash 19970-Automata Manipulation**
    Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, Laurent Xu
    International Symposium on Automated Technology for Verification and Analysis (2016)

16. **Owl: A Library for omega-Words, Automata, and LTL**
    Jan Kretinsky, Tobias Meggendorfer, Salomon Sickert
    Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings (2018) https://doi.org/10.1007/978-3-030-01090-4\textbackslash{}_34
    DOI: 10.1007/978-3-030-01090-4\textbackslash{}_34

17. **The SPIN Model Checker: Primer and Reference Manual**
    Gerard Holzmann
    Addison-Wesley Professional (2011)
    ISBN: 0321773713

18. **Symbolic LTLf Synthesis**
    Shufang Zhu, Lucas M. Tabajara, Jianwen Li, Geguang Pu, Moshe Y. Vardi
    Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (2017-08)
    http://dx.doi.org/10.24963/ijcai.2017/189
    DOI: 10.24963/ijcai.2017/189 · ISBN: 9780999241103

19. **Hybrid Compositional Reasoning for Reactive Synthesis from Finite-Horizon Specifications**
    Suguman Bansal, Yong Li, Lucas M. Tabajara, Moshe Y. Vardi
    The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020 (2020) https://aaai.org/ojs/index.php/AAAI/article/view/6528

20. **RiccardoDeMasellis/FLLOAT**
    Riccardo De Masellis

(2015) https://github.com/RiccardoDeMasellis/FLLOAT

21. **Reinforcement learning for LTLf/LDLf goals: Theory and implementation**
    Marco Favorito
    Master's thesis. DIAG, Sapienza Univ. Rome (2018)

22. **LTLf2DFA**
    Francesco Fuggitti
    WhiteMech (2018) https://github.com/whitemech/LTLf2DFA

23. **A high-level LTL synthesis format: TLSF v1. 1**
    Swen Jacobs, Felix Klein, Sebastian Schirmer
    arXiv preprint arXiv:1604.02284 (2016)

24. **The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM conference**
    John W Backus
    Proceedings of the International Comference on Information Processing, 1959 (1959)

25. **Extensible Markup Language (XML) 1.0 (Fifth Edition)**
    W3C
    (2008) https://www.w3.org/TR/xml/

26. **ISO/IEC 14977: 1996 (E)**
    Extended BNF ISO
    ISO: Geneva (1996)

27. **Don't Use ISO/IEC 14977 Extended Backus-Naur Form (EBNF)**
    David Wheeler
    (2020) https://dwheeler.com/essays/dont-use-iso-14977-ebnf.html

28. **BNF was here: what have we done about the unnecessary diversity of notation for syntactic definitions**
    Vadim Zaytsev
    Proceedings of the 27th Annual ACM Symposium on Applied Computing (2012)

29. **Key words for use in RFCs to Indicate Requirement Levels**
    Scott Bradner
    RFC2119 (1997)

30. **Information technology—Universal coded character set**
    ISO/IEC
    International Organization for Standardization (2020)

31. **The Unicode Standard, Version 13.0.0**
    The Unicode Consortium
    (2020) https://www.unicode.org/versions/Unicode13.0.0/
    ISBN: 978-1-936213-26-9

32. **Semantic Versioning for Documents 1.0.0**
    Nils Tekampe
    SemVerDoc (2018) https://semverdoc.org/semverdoc.html

33. **Semantic Versioning 2.0.0**
Tom Preston-Werner
Semantic Versioning (2011) https://semver.org/

---

1. You can get the sources of this document at this repository: https://github.com/marcofavorito/tl-grammars↩