Standard Grammars for LTL and LDL (v0.1.0)

Marco Favorito
Department of Computer, Control and Management Engineering
Sapienza University of Rome
Via Ariosto, 25, 00185 Roma RM, Italy
favorito@diag.uniroma1.it
https://marcofavorito.me

December 29, 2020

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) (Pnueli 1977) and Linear Dynamic Logic (Vardi 2011; De Giacomo and Vardi 2013).

WARNING: this version (v0.1.0) is a draft. You are encouraged to email the contact author for any comment or suggestion.

---

1 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].

1.1 Motivation

Temporal logics have a long history (Konur 2010). One of the most influential formalisms is Linear Temporal Logic (LTL) (Pnueli 1977), which has been applied for program specification and verification. The variant over finite traces has been introduced in (De Giacomo and Vardi 2013). Linear Dynamic Logic (LDL) (Vardi 2011; De Giacomo and Vardi 2013) is the extension of LTL with regular expressions (RE). The idea behind LDL is to have a formalism that

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

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 (De Giacomo and Vardi 2013) for LTL/LDL and (De Giacomo et al. 2020) for PLTL/PLDL. The syntax that naturally supports empty traces has been employed in (Brafman, De Giacomo, and Patrizi 2018) for LTL/LDL and (De Giacomo et al. 2020) 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 (De Giacomo and Vardi 2015), model checking (Clarke, Emerson, and Sistla 1986), planning with temporal goal (Bacchus and Kabanza 1998), theory of Markov Decision Process with non-Markovian rewards (Bacchus, Boutilier, and Grove 1996), business processes specification (Pešić, Bošnački, and Aalst 2010), just to name a few. For what concerns industry applications, Intel proposed the industrial linear time specification language `ForSpec` (Armoni et al. 2002), and the IEEE association standardized the *Property Specification Language* (PSL) (IEEE 2010). 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` (Duret-Lutz 2016; Duret-Lutz et al. 2016), `Owl` (Kretinsky, Meggendorfer, and Sickert 2018), `SPIN` (Holzmann 2011), `Syft` (Zhu et al. 2017), `Lisa` (Bansal et al. 2020), `FLLOAT` (De Masellis 2015; Favorito 2018), `LTLf2DFA` (Fuggitti 2018), and more. Another related work is represented by TLSF v1.1 (Jacobs, Klein, and Schirmer 2016), 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.

1.2 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.

1.3 Notation

We describe the syntax in Extended Backus-Naur Form (EBNF) (Backus 1959). We follow the notation used for the specification of XML (W3C 2008); we discarded the EBNF standard version ISO/IEC 14977 (ISO 1996), as it has been often rejected by the community of those who write language specifications for a variety of reasons (Wheeler 2020; Zaytsev 2012).

1.4 Normative

We refer to (Bradner 1997) for requirement level key words. We also refer to Unicode standard (ISO/IEC 2020; The Unicode Consortium 2020) to define legal characters. For versioning this standard, we use SemVerDocs (Tekampe 2018), inspired by SemVer (Preston-Werner 2011).

2 Common definitions

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

2.1 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 (ISO/IEC 2020). 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.

2.2 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
```

2.3 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
```

2.4 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   ::= "^"
```

2.5 Parenthesis

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

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

2.6 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.

3 LTL

In this section, we specify a grammar for LTL.

3.1 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
```

3.2 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"
```

3.3 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 (Pnueli 1977) for the infinite setting, and (De Giacomo and Vardi 2013) for the finite setting.

3.4 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	`!`

4 LDL

In this section, we specify a grammar for LDL.

4.1 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
```

4.2 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 (De Giacomo and Vardi 2013).

4.3 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	`!`

5 Future work

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

- `Spot`-like syntactic sugars for regular expressions (SERE) and temporal operators (Duret-Lutz 2016; Jacobs, Klein, and Schirmer 2016);
- Compatibility with the PSL standard (IEEE 2010);
- Support full Unicode characters, so to use UTF-8 characters like $\circ$ (U+25CB) for the Next operator and $\diamond$ (U+25C7) for the Eventually operator etc. as alternative symbols.

- Grammars for PLTL and PLDL.

6 License

References

Armoni, Roy, Limor Fix, Alon Flaisher, Rob Gerth, Boris Ginsburg, Tomer Kanza, Avner Landver, et al. 2002. "The Forspec Temporal Logic: A New Temporal Property-Specification Language." In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 296–311. Springer.

Bacchus, Fahiem, Craig Boutilier, and Adam Grove. 1996. "Rewarding Behaviors." In *Proceedings of the National Conference on Artificial Intelligence*, 1160–7.

Bacchus, Fahiem, and Froduald Kabanza. 1998. "Planning for Temporally Extended Goals." *Annals of Mathematics and Artificial Intelligence* 22 (1-2): 5–27.

Backus, John W. 1959. "The Syntax and Semantics of the Proposed International Algebraic Language of the Zurich Acm-Gamm Conference." *Proceedings of the International Comference on Information Processing, 1959*.

Bansal, Suguman, Yong Li, Lucas M. Tabajara, and Moshe Y. Vardi. 2020. "Hybrid Compositional Reasoning for Reactive Synthesis from Finite-Horizon Specifications." In *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*, 9766–74. AAAI Press.

Bradner, Scott. 1997. "Key Words for Use in Rfcs to Indicate Requirement Levels." *RFC2119*.

Brafman, Ronen, Giuseppe De Giacomo, and Fabio Patrizi. 2018. "LTLf/Ldlf Non-Markovian Rewards."

Clarke, E. M., E. A. Emerson, and A. P. Sistla. 1986. "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications." *ACM Trans. Program. Lang. Syst.* 8 (2): 244263. https://doi.org/10.1145/5397.5399.

De Giacomo, Giuseppe, Antonio Di Stasio, Francesco Fuggitti, and Sasha Rubin. 2020. "Pure-Past Linear Temporal and Dynamic Logic on Finite Traces." In

Twenty-Ninth International Joint Conference on Artificial Intelligence and Seventeenth Pacific Rim International Conference on Artificial Intelligence Ijcai-Pricai-20. International Joint Conferences on Artificial Intelligence Organization. https://doi.org/10.24963/ijcai.2020/690.

De Giacomo, Giuseppe, and Moshe Y. Vardi. 2013. "Linear Temporal Logic and Linear Dynamic Logic on Finite Traces." In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, 854–60. IJCAI '13. Beijing, China: AAAI Press. http://dl.acm.org/citation.cfm?id=2540128.2540252.

———. 2015. "Synthesis for Ltl and Ldl on Finite Traces." In *Proceedings of the 24th International Conference on Artificial Intelligence*, 1558–64. IJCAI'15. Buenos Aires, Argentina: AAAI Press.

De Masellis, Riccardo. 2015. "Github.com/Riccardodemasellis/Flloat."

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

Duret-Lutz, Alexandre, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. 2016. "Spot 2.0: A Framework for Ltl and Omega-Automata Manipulation." In *International Symposium on Automated Technology for Verification and Analysis*, 122–29. Springer.

Favorito, Marco. 2018. "Reinforcement Learning for Ltlf/Ldlf Goals: Theory and Implementation." *Master's Thesis. DIAG, Sapienza Univ. Rome*.

Fuggitti, Francesco. 2018. "Github.com/Whitemech/Ltlf2dfa."

Holzmann, Gerard. 2011. *The Spin Model Checker: Primer and Reference Manual*. 1st ed. Addison-Wesley Professional.

IEEE. 2010. "IEEE Standard for Property Specification Language (Psl)." IEEE. https://doi.org/10.1109/ieeestd.2010.5446004.

ISO, Extended BNF. 1996. "ISO/Iec 14977: 1996 (E)." *ISO: Geneva*.

ISO/IEC. 2020. "Information Technology—Universal Coded Character Set." ISO/IEC 10646:2020. Geneva, Switzerland: International Organization for Standardization.

Jacobs, Swen, Felix Klein, and Sebastian Schirmer. 2016. "A High-Level Ltl Synthesis Format: TLSF V1. 1." *arXiv Preprint arXiv:1604.02284*.

Konur, Savas. 2010. "A Survey on Temporal Logics."

Kretinsky, Jan, Tobias Meggendorfer, and Salomon Sickert. 2018. "Owl: A Library for Omega-Words, Automata, and Ltl." In *Automated Technology for Verification and Analysis - 16th International Symposium, Atva 2018, Los Angeles, ca, Usa, October 7-10, 2018, Proceedings*, edited by Shuvendu K. Lahiri and Chao Wang, 11138:543–50. Lecture Notes in Computer Science. Springer. https://doi.org/10.1007/978-3-030-01090-4\textbackslash{}_34.

Pešić, Maja, Dragan Bošnački, and Wil MP van der Aalst. 2010. "Enacting Declarative Languages Using Ltl: Avoiding Errors and Improving Performance." In *International Spin Workshop on Model Checking of Software*, 146–61. Springer.

Pnueli, Amir. 1977. "The Temporal Logic of Programs." In *18th Annual Symposium on Foundations of Computer Science (Sfcs 1977)*. IEEE. https://doi.org/10.1109/sfcs.1977.32.

Preston-Werner, Tom. 2011. "Semantic Versioning 2.0.0." *Semantic Versioning*. https://semver.org/.

Tekampe, Nils. 2018. "Semantic Versioning for Documents 1.0.0." *SemVerDoc*. https://semverdoc.org/semverdoc.html.

The Unicode Consortium. 2020. *The Unicode Standard, Version 13.0.0*. https://www.unicode.org/versions/Unicode13.0.0/.

Vardi, Moshe Y. 2011. "The Rise and Fall of Ltl." *GandALF* 54.

W3C. 2008. "Extensible Markup Language (Xml) 1.0 (Fifth Edition)." https://www.w3.org/TR/xml/.

Wheeler, David. 2020. "Don't Use Iso/Iec 14977 Extended Backus-Naur Form (Ebnf)." https://dwheeler.com/essays/dont-use-iso-14977-ebnf.html.

Zaytsev, Vadim. 2012. "BNF Was Here: What Have We Done About the Unnecessary Diversity of Notation for Syntactic Definitions." In *Proceedings of the 27th Annual Acm Symposium on Applied Computing*, 1910–5.

Zhu, Shufang, Lucas M. Tabajara, Jianwen Li, Geguang Pu, and Moshe Y. Vardi. 2017. "Symbolic Ltlf Synthesis." *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, August. https://doi.org/10.24963/ijcai.2017/189.