Manipulating LTL formulas using Spot 1.0

Alexandre Duret-Lutz

LRDE, EPITA, Kremlin-Bicêtre, France
`adl@lrde.epita.fr`

Abstract. We present a collection of command-line tools designed to generate, filter, convert, simplify, lists of Linear-time Temporal Logic formulas. These tools were introduced in the release 1.0 of Spot, and we believe they should be of interest to anybody who has to manipulate LTL formulas. We focus on two tools in particular: `ltlfilt`, to filter and transform formulas, and `ltlcross` to cross-check LTL-to-Büchi-Automata translators.

1 Introduction

Spot is a C++ library of model-checking algorithms that has been around for nearly 10 years [5]. It contains algorithms to perform the usual task in the automata-theoretic approach to LTL model checking [13]. So far, and because it is a library, Spot did not provide any convenient access to its features from the command-line. The adventurous user would use some of the programs built for the test-suite of Spot, but these programs were never designed to offer a user-friendly interface.

This situation has changed with the recent release of Spot 1.0: it now installs a collection of command-line tools that give access to many of Spot's features, and allows to combine them with pipes, in the purest Unix tradition. The current tool set (which we describe in this paper) is focused on the handling of linear-time temporal-logic formulae and on its conversion to Büchi automata. The library also includes many algorithms that work on automata, but which are not yet available from the command-line.

We invite the reader to download Spot from `http://spot.lip6.fr/` and install it, in order to play with the example commands provided in this paper. In addition to the man pages that are installed along with Spot, a more detailed description of the tools can be read at `http://spot.lip6.fr/userdoc/tools.html`.

2 Linear-time Temporal Logic(s)

Spot supports the usual LTL operators: X (next), F (eventually), G (globally), U (until), R (release), W (weak until), and M (strong release). These can be combined with Boolean operators, Boolean constants, and identifiers that represent atomic propositions.

Although there are many tools using LTL, there is no standard syntax for the representation of LTL formulas. For instance the formula $\mathsf{G}(\mathit{request} \rightarrow \mathsf{F}(\mathit{grant}))$ could be written as `[](request => <>(grant))` by Spin [7], `[](request --> <>(grant))` by Goal [12], `G(request=1 -> F(grant=1)` by Wring [10], `G i "request" F "grant"` by ltl2dstar [8], or even `G i p0 F p1` by tools like LBT[1] or Scheck [9] that do not accept arbitrary identifiers as propositions. Spot's tools will write `G(request -> F(grant))` by default, but they can read all the above syntaxes, and can write into most of them (the only missing output is Goal, because Goal can already read Spin's syntax).

In addition to LTL operators, we support operators from the linear fragment of the Property Specifications Language (PSL) [1]. These operators connect Semi-Extended Regular Expressions with LTL. A SERE is built using the usual three regular operators, ';' (concatenation), $\cup$ (union), and $\star$ (Kleene star), but extended with additional operators such as $\cap$ (intersection), ':' (fusion), and many other operators that are just syntactic sugar over these.[2] The main two PSL operators are:

- $\{e\} \mathrel{\Box\!\!\rightarrow} f$: *any* finite prefix matching the SERE e must trigger the verification of f (any formula using PSL or LTL operators) from the last letter of the prefix, and
- $\{e\} \mathrel{\Diamond\!\!\rightarrow} f$: f must be verified from the last letter of *some* prefix matching e.

Again more syntactic sugar exists on top of these. For instance $\{e\}!$ is syntactic sugar for $\{e\} \mathrel{\Diamond\!\!\rightarrow} \top$: some finite prefix must match the SERE e.

As an example, the PSL formula $\{(\top;\top)^\star\} \mathrel{\Box\!\!\rightarrow} p$ states that p should hold every two states, and has no equivalent LTL formula.

3 Tools

Spot installs six command-line tools: `randltl` is a random LTL/PSL formula generator; `ltlfilt` is a multi-function LTL/PSL formula filter, able to convert formulas between formats, filter formulas matching certain criteria, and perform some simple syntactic transformations; `genltl` is a formula generator for various scalable families of LTL formulas; `ltl2tgba` is a translator from LTL/PSL formulas to different kinds of Büchi automata [4]; `ltl2tgta` is a translator from LTL/PSL formulas to different kinds of testing automata [2]; and `ltlcross` is a test-bench for LTL/PSL translators. By lack of space, we only illustrate three of these commands over a few command-line examples.

3.1 `ltlfilt` and `randltl`

```
% ltlfilt --safety --relabel=abc --uniq --spin formulas.ltl
```

Reads formulas from file `formulas.ltl` (one formula per line), retains only those that represent safety properties, renames the atomic propositions occurring in each formula using the letters '`a`', '`b`', '`c`',... suppresses duplicate formulas, and outputs formulas using Spin's syntax. The safety check is automaton-based [3], so "pathological formulas" that represent safety properties without looking so syntactically are also captured.

```
% randltl -n -1 --tree-size=10..15 a b | ltlfilt --simplify --safety |
ltlfilt --invert-match --syntactic-safety --uniq | head -n 10
```

The `randltl` command generates an unbounded (`-n -1`) stream of LTL formulas with a tree size between 10 and 15, and using atomic propositions '`a`' and '`b`'. These formulas

[1] `http://www.tcs.hut.fi/Software/maria/tools/lbt/`

[2] A complete description of all the supported operators and their semantics can be found in `doc/tl/tl.pdf` inside the Spot distribution.

are then simplified (using Spot's LTL rewriting rules) and filtered to preserve only safety formulas; the result is then filtered again to remove all "syntactic safety" formulas, as well as duplicate formulas. The result of these three commands is therefore a stream of pathological safety formulas, from which we only display the first 10 using the standard `head` command from Unix.

Chaining commands this way to generate random formulas has proven to be a very useful way to generate sets of formulas matching a certain criterion. The following example generates a list of 20 PSL formulas that are not LTL formulas (i.e., they must use PSL operators) and that are equivalent to $a \mathsf{U} b$.

```
% randltl --psl -n-1 --tree-size=5..10 a b |ltlfilt --invert-match --ltl|
ltlfilt --uniq --equivalent-to 'a U b' | head -n 20
```

Simplification rules are able to transform *some* PSL formulas into LTL formulas. For instance the PSL formula $\{a^\star; b^\star; c\}!$ is equivalent to the LTL formula $a \mathsf{U}(b \mathsf{U} c)$. Similarly the PSL formulas $\{a[\to 2]\} \mathrel{\Box\!\!\to} b$, which states that b should hold every time a holds for the second time, can be transformed into $a \mathsf{R}(\bar{a} \lor \mathsf{X}(a \mathsf{R}(\bar{a} \lor b)))$.

```
% ltlfilt --simplify -f '{a*;b*;c}!' -f '{a[->2]}[]->b'
a U (b U c)
a R (!a | X(a R (!a | b)))
```

Note that PSL is more expressive than LTL, so not all PSL formulas can be converted into LTL. Currently, we only implements rewriting for *some* straightforward PSL patterns, and these rewriting rules will certainly be improved in the future.

Occasional questions such as "Is $\mathsf{F}(\bar{a} \land \mathsf{X}a \land \mathsf{X}b)$ stutter-invariant?" can also be answered by instructing `ltlfilt` to match only stutter-invariant formulas:

```
% ltlfilt --stutter-invariant -f 'F(!a & Xa & Xb)'
F(!a & Xa & Xb)
```

Since the formula was output, it is stuttering invariant. Another option, `--remove-x`, can be used to rewrite this formula without the X operator.[3] Other day-to-day questions like "Is formula φ equivalent to formula ψ?" can be answered similarly.

3.2 `ltlcross`

Spot has used LBTT, the *LTL-to-Büchi Translator Testbench* [11] in its test-suite since its early days. LBTT feeds randomly generated LTL formulae to the configured LTL-to-Büchi translators, and then cross-compares the results of all tools, using several checks to detect possible bugs in implementations, or simply to compare the results from a statistical standpoint. Unfortunately, LBTT is no longer maintained, we have found it quite hard to extend to gather new kinds of statistics, and most importantly it is restricted to LTL. We therefore introduce `ltlcross`, a reimplementation of LBTT using Spot, with support for PSL formulas.

`ltlcross` reads a list of formulas from its standard input (usually some output of `randltl`) or from a file, runs these formulas through several (PSL or) LTL-to-Büchi translators, read the output of these translators (as never claims or in LBTT's syntax) and then performs the same tests as LBTT on the resulting automata.

[3] Stutter invariance is actually asserted using automata to test the language equivalence of the input formula and its rewriting without X [6]. Currently this only works for LTL.

The output of `ltlcross` is a CSV or JSON file that contains more statistics about the produced automata. These files are easily post-processed to compute summary table or graphics. A typical invocation would look as follows:

```
% randltl -n 100 a b c | ltlfilt --remove-wm |
ltlcross --csv=out.csv 'ltl2tgba -s %f >%N' 'spin -f %s >%N' 'lbt <%L >%T'
```

Here 100 random formulas over a, b, and c are produced, the operators W and M are rewritten away by `ltlfilt` (because W and M are not supported by `spin -f` and `lbt`), and finally `ltlcross` uses the resulting formulas with 3 different translators, and gather statistics in `out.csv`.

The invocation of each tool is configured with `%`-sequences showing how the formula to translate should be passed (e.g., `%f`, `%s`, `%l` are replaced respectively by the formula is Spot's, Spin's, or LBT's syntax, while `%F`, `%S`, `%L` are replaced by the name of a file that contains the formula in these syntaxes) and how to read the result (`%T` for a filename that will contain output in LBTT's syntax, and `%N` for a filename that will contain a neverclaim). If any error is detected while running these translators, or when comparing their outputs (we perform the same checks as LBTT), `ltlcross` will report it.

References

1. Property Specification Language Reference Manual v1.1. Accellera (Jun 2004), `http://www.eda.org/vfv/`
2. Ben Salem, A.E., Duret-Lutz, A., Kordon, F.: Model checking using generalized testing automata. Transactions on Petri Nets and Other Models of Concurrency (ToPNoC), 4, 94–112, Springer (2012)
3. Dax, C., Eisinger, J., Klaedtke, F.: Mechanizing the powerset construction for restricted classes of ω-automata. In: Proc. of ATVA'07. LNCS, vol. 4762. Springer (Oct 2007)
4. Duret-Lutz, A.: LTL translation improvements in Spot. In: Proc. of VECoS'11. British Computer Society (Sep 2011), `http://ewic.bcs.org/category/15853`
5. Duret-Lutz, A., Poitrenaud, D.: SPOT: an Extensible Model Checking Library using Transition-based Generalized Büchi Automata. In: Proc. of MASCOTS'04. pp. 76–83. IEEE Computer Society Press (Oct 2004)
6. Etessami, K.: A note on a question of Peled and Wilke regarding stutter-invariant LTL. Information Processing Letters, 75(6), 261–263 (2000)
7. Holzmann, G.J.: The Spin Model Checker: Primer and Reference Manual. Addison-Wesley (2003)
8. Klein, J., Baier, C.: Experiments with deterministic ω-automata for formulas of linear temporal logic. Theoretical Computer Science, 363(2), 182–195 (2006)
9. Latvala, T.: Efficient model checking of safety properties. In: Proc. of Spin'03. LNCS, vol. 2648, pp. 74–88. Springer (2003)
10. Somenzi, F., Bloem, R.: Efficient Büchi automata for LTL formulæ. In: Proc. of CAV'00. LNCS, vol. 1855, pp. 247–263. Springer (2000)
11. Tauriainen, H., Heljanko, K.: Testing LTL formula translation into Büchi automata. International Journal on Software Tools for Technology Transfer, 4(1), 57–70, Springer (2002)
12. Tsay, Y.K., Chen, Y.F., Tsai, M.H., Wu, K.N., Chan, W.C., Luo, C.J., Chang, J.S.: Tool support for learning büchi automata and linear temporal logic. Formal Aspects of Computing, 21(3), 259–275, Springer (2009)
13. Vardi, M.Y.: An automata-theoretic approach to linear temporal logic. In: Proc. of Banff'94. LNCS, vol. 1043, pp. 238–266. Springer (1996)