Manipulating LTL formulas in Spot 1.0

Alexandre Duret-Lutz

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

ATVA'13, 2013-10-16

http://spot.lip6.fr/

Spot, as a library

- Started in 2003, as a library for explicit model checking.
- Offers algorithms to perform the automata-theoretic approach.
- Focuses of Transition-based Generalized Büchi Automata (TGBA)
- A lot of efforts put in algorithms to translate LTL formulas into automata and to simplify those automata.
- Supports the linear fragment of PSL since Spot 0.9 (May 2012).
- Distributes a collection of command-line tools since Spot 1.0 (Oct. 2012).

Latest version is Spot 1.2 (Oct. 2013).

Command-line tools

`randltl`	Generate random LTL/PSL formulas.
`ltlfilt`	Filter LTL/PSL formulas.
`genltl`	Generate LTL formulas from scalable patterns.
`ltl2tgba`	Translate LTL/PSL formulas into Büchi automata.
`ltl2tgta`	Translate LTL/PSL formulas into Testing automata.
`ltlcross`	Cross-compare LTL/PSL-to-Büchi translators.
`dstar2tgba`	Convert deterministic Rabin or Streett automata into Büchi automata [Spot 1.2].

`randltl` — random formula generator

Build 5 unique LTL formulas...

```
$ randltl -n5 a b c
G(Gb W (Gb M c))
!(GFb -> Fa)
!c & (((c xor Xc) R c) R Gc)
X(1 U Xb) M Fb
!XFb U !(Xa W 0)
```

... in LBT's format

```
$ randltl -n5 p0 p1 p2 --lbt
G W G p1 M G p1 p2
! i G F p1 F p0
& ! p2 V V ^ p2 X p2 p2 G p2
M X U t X p1 F p1
U ! X F p1 ! W X p0 f
```

... without `W`, `M`, `xor`, but with many `X`

```
$ randltl -n5 a b c --ltl-priorities 'W=0,M=0,xor=0,X=5'
G(!a R (Xb U c))
!XX!G(b | (b <-> c))
0
XGXX(c <-> !c) R Gb
1
```

`ltlfilt` — a swiss-army knife for LTL files

syntax conversion

formula transformations

formula filtering

`ltlfilt` — a swiss-army knife for LTL files

syntax conversion

syntax	input	output	example
Spot	(default)	(default)	`G(p0 -> (p1 R !p2))`
UTF-8	(default)	`-8`	`□(p0 → (p1 R ¬p2))`
Spin	(default)	`-s`	`[](p0 -> (p1 V !p2))`
Wring	(default)	`--wring`	`G(p0=1 -> (p1=1 R p2=0))`
Goal	(default)	n/a	`G(p0 --> (p1 R ~p2))`
LBT	`--lbt-input`	`-l`	`G i p0 V p1 !  p2`
LaTeX	n/a	`--latex`	`\G(p_0\limplies(p_1\R\lnot p_2))`

formula transformations

formula filtering

`ltlfilt` — a swiss-army knife for LTL files

syntax conversion

syntax	input	output	example
Spot	(default)	(default)	`G(p0 -> (p1 R !p2))`
UTF-8	(default)	`-8`	`□(p0 → (p1 R ¬p2))`
Spin	(default)	`-s`	`[](p0 -> (p1 V !p2))`
Wring	(default)	`--wring`	`G(p0=1 -> (p1=1 R p2=0))`
Goal	(default)	n/a	`G(p0 --> (p1 R ~p2))`
LBT	`--lbt-input`	`-l`	`G i p0 V p1 !  p2`
LaTeX	n/a	`--latex`	`\G(p_0\limplies(p_1\R\lnot p_2))`

formula transformations

`--boolean-to-isop`, `--negate`, `--nnf`, `--relabel`, `--relabel-bool`, `--remove-wm`, `--remove-x`, `--simplify`

formula filtering

`ltlfilt` — a swiss-army knife for LTL files

syntax conversion

syntax	input	output	example
Spot	(default)	(default)	`G(p0 -> (p1 R !p2))`
UTF-8	(default)	`-8`	`□(p0 → (p1 R ¬p2))`
Spin	(default)	`-s`	`[](p0 -> (p1 V !p2))`
Wring	(default)	`--wring`	`G(p0=1 -> (p1=1 R p2=0))`
Goal	(default)	n/a	`G(p0 --> (p1 R ~p2))`
LBT	`--lbt-input`	`-l`	`G i p0 V p1 !  p2`
LaTeX	n/a	`--latex`	`\G(p_0\limplies(p_1\R\lnot p_2))`

formula transformations

`--boolean-to-isop`, `--negate`, `--nnf`, `--relabel`, `--relabel-bool`, `--remove-wm`, `--remove-x`, `--simplify`

formula filtering

`--boolean`, `--bsize-max`, `--bsize-min`, `--equivalent-to`, `--eventual`, `--guarantee`,`--implied-by`,`--imply`, `--ltl`,`--nox`,`--obligation`, `--safety`, `--size-max`, `--size-min`, `--stutter-invariant`, `--syntactic-guarantee`, `--syntactic-obligation`, `--syntactic-persistence`, `--syntactic-recurrence`, `--syntactic-safety`, `--universal`, `--unique`, `--invert-match`

`ltlfilt` — generating specific formulas

Build 10 pathological safety formulas

```
$ randltl -n -1 --tree-size=10..15 a b |
  ltlfilt --simplify --safety --uniq |
  ltlfilt --invert-match --syntactic-safety |
  head -n 10
((!b & (b U !a)) | (b & (!b R a))) R a
(a & !b) | (!a & (F((!b & Xb) | (b & X!b)) R b))
((!b & Ga) | (b & F!a)) R a
G((!b & XGb) | (b & XF!b))
G(!a M ((!b & XGb) | (b & XF!b)))
G!a M !a
G((G(!b U a) & (GFb R !a)) | (F(b R !a) & (FG!b U a)))
F!a R Xa
G((b & GFb) | (!b & FG!b))
F(b | G!b)
```

`ltlfilt` — generating specific formulas

Build 10 pathological safety formulas

```
$ randltl -n -1 --tree-size=10..15 a b |
  ltlfilt --simplify --safety --uniq |
  ltlfilt --invert-match --syntactic-safety |
  head -n 10
((!b & (b U !a)) | (b & (!b R a))) R a
(a & !b) | (!a & (F((!b & Xb) | (b & X!b)) R b))
((!b & Ga) | (b & F!a)) R a
G((!b & XGb) | (b & XF!b))
G(!a M ((!b & XGb) | (b & XF!b)))
G!a M !a
G((G(!b U a) & (GFb R !a)) | (F(b R !a) & (FG!b U a)))
F!a R Xa
G((b & GFb) | (!b & FG!b))
F(b | G!b)
```

`ltlfilt` — generating specific formulas

Build pathological safety formulas not equivalent to ⊤ or ⊥

```
$ randltl -n -1 --tree-size=10..15 a b |
  ltlfilt --simplify --safety --uniq |
  ltlfilt --invert-match --syntactic-safety |
  head -n 10 |
  ltlfilt --invert-match --equivalent-to=1 |
  ltlfilt --invert-match --equivalent-to=0
((!b & (b U !a)) | (b & (!b R a))) R a
(a & !b) | (!a & (F((!b & Xb) | (b & X!b)) R b))
((!b & Ga) | (b & F!a)) R a
G!a M !a
F!a R Xa
G((b & GFb) | (!b & FG!b))
```

`ltlfilt` — answering simple questions

Is $a\,\mathsf{U}(b\,\mathsf{U}\,a)$ equivalent to $b\,\mathsf{U}\,a$?

```
$ ltlfilt -f 'a U (b U a)' --equivalent-to 'b U a'
a U (b U a)
```

`ltlfilt` — answering simple questions

Is $a\,\mathsf{U}(b\,\mathsf{U}\,a)$ equivalent to $b\,\mathsf{U}\,a$?

```
$ ltlfilt -f 'a U (b U a)' --equivalent-to 'b U a'
a U (b U a)
```

Which of these formulas are stutter-invariant?

```
$ ltlfilt -f 'G(a | X(a -> b))' -f 'G(a | X(a <-> b))' \
  --stutter-invariant
G(a | X(a -> b))
```

K. Etessami. A note on a question of Peled and Wilke regarding stutter-invariant LTL. *Information Processing Letters*, 75(6):261–263, 2000

`ltlfilt` — answering simple questions

Is $a\,\mathsf{U}(b\,\mathsf{U}\,a)$ equivalent to $b\,\mathsf{U}\,a$?

```
$ ltlfilt -f 'a U (b U a)' --equivalent-to 'b U a'
a U (b U a)
```

Which of these formulas are stutter-invariant?

```
$ ltlfilt -f 'G(a | X(a -> b))' -f 'G(a | X(a <-> b))' \
  --stutter-invariant
G(a | X(a -> b))
```

Give an X-free formula for $\mathsf{G}(a \vee \mathsf{X}(a \rightarrow b))$

```
$ ltlfilt -f 'G(a | X(a -> b))' --remove-x --simplify
G(a | (!a & (!a U (a & (!a | b))) & ((!b U a) | (b U a))) |
(b & (b U (!b & (!a | b))) & ((!a U !b) | (a U !b))) | ((!a |
b) & (G!a | Ga) & (G!b | Gb)) | (!b & ((!a U b) | (a U b))))
```

K. Etessami. A note on a question of Peled and Wilke regarding stutter-invariant LTL. *Information Processing Letters*, 75(6):261–263, 2000

`ltlcross` — an enhanced clone of LBTT

- The problem every author of an LTL-to-Büchi tool faces:

  How to test it?

- Spot has been using LBTT (*LTL-to-Büchi Translator Testbench*) since 2003 in its test-suite.
  - Tremendously useful, for testing **and** benchmarking.

H. Tauriainen and K. Heljanko. Testing LTL formula translation into Büchi automata. *International Journal on Software Tools for Technology Transfer*, 4(1):57–70, 2002

`ltlcross` — an enhanced clone of LBTT

- The problem every author of an LTL-to-Büchi tool faces:

  How to test it?

- Spot has been using LBTT (*LTL-to-Büchi Translator Testbench*) since 2003 in its test-suite.
  - Tremendously useful, for testing **and** benchmarking.
- However:
  - LBTT is no longuer maintained (last release in 2005).
  - LBTT is restricted to LTL (Spot now has a translator for PSL).
  - Extracting statistics from the output of LBTT is a pain.
  - The Spot library already has all the algorithms necessary to implement the same functionality.
- `ltlcross` = clone of LBTT + PSL support + additional goodies.

H. Tauriainen and K. Heljanko. Testing LTL formula translation into Büchi automata. *International Journal on Software Tools for Technology Transfer*, 4(1):57–70, 2002

`ltlcross` — basic operations

- Take a list of formulas (LTL/PSL) from file, stdin, or arguments.
- Take a list of translators T_1, T_2, ... listed as arguments.
- For any formula φ and its negation, run all translators:

$$P_i = T_i(\varphi) \qquad\qquad N_i = T_i(\neg\varphi)$$

- Perform the three checks of LBTT:
  - intersection tests: $\mathscr{L}(N_i \otimes P_j) = \emptyset$ $\qquad$ $\mathscr{L}(P_i \otimes N_j) = \emptyset$
  - cross-comparison tests (S is a random state-space)

$$\mathscr{L}(P_i \otimes S) = \emptyset \iff \mathscr{L}(P_j \otimes S) = \emptyset$$
$$\mathscr{L}(N_i \otimes S) = \emptyset \iff \mathscr{L}(N_j \otimes S) = \emptyset$$

  - consistency check: $\mathit{states}(P_i \otimes S)_{|S} \cup \mathit{states}(N_i \otimes S)_{|S} = S$

- Once all formulas have been processed, optionally output detailed statistics in a CSV file.

`ltlcross` — basic operations

- Take a list of formulas (LTL/PSL) from file, stdin, or arguments.
- Take a list of translators T_1, T_2, ... listed as arguments.
- For any formula φ and its negation, run all translators:
$$P_i = T_i(\varphi) \qquad\qquad N_i = T_i(\neg\varphi)$$
- Perform the three checks of LBTT:
  - intersection tests: $\mathscr{L}(N_i \otimes P_j) = \emptyset$ $\qquad$ $\mathscr{L}(P_i \otimes N_j) = \emptyset$
  - cross-comparison tests (S is a random state-space)
$$\mathscr{L}(P_i \otimes S) = \emptyset \iff \mathscr{L}(P_j \otimes S) = \emptyset$$
$$\mathscr{L}(N_i \otimes S) = \emptyset \iff \mathscr{L}(N_j \otimes S) = \emptyset$$
  - consistency check: $\mathit{states}(P_i \otimes S)_{|S} \cup \mathit{states}(N_i \otimes S)_{|S} = S$
- Additional intersection tests in `ltlcross` (Spot 1.2):
$$\mathscr{L}(P_i \otimes \overline{P_j}) = \emptyset \text{ if } P_j \text{ is deterministic}$$
$$\mathscr{L}(N_i \otimes \overline{N_j}) = \emptyset \text{ if } N_j \text{ is deterministic}$$
- Once all formulas have been processed, optionally output detailed statistics in a CSV file.

`ltlcross` — interface with translators

```
$ randltl -n100 a b c |
  ltlcross 'ltl3ba -f %s >%N' 'ltl2tgba --lbtt %f >%T' \
           'ltl3dra -f %s >%D' 'lbt <%L >%T' --csv=output.csv
```

Escape sequences specify how formulas are passed, and how automata should be read back.

`ltlcross` — interface with translators

```
$ randltl -n100 a b c |
  ltlcross 'ltl3ba -f %s >%N' 'ltl2tgba --lbtt %f >%T' \
           'ltl3dra -f %s >%D' 'lbt <%L >%T' --csv=output.csv
```

Escape sequences specify how formulas are passed, and how automata should be read back.

- Input formula: `%s` (Spin syntax), `%f` (Spot syntax), `%l` (LBT's syntax), or `%w` (Wring's syntax).

  Use `%S`, `%F`, `%L`, or `%W` for files instead of strings.

`ltlcross` — interface with translators

```
$ randltl -n100 a b c |
  ltlcross 'ltl3ba -f %s >%N' 'ltl2tgba --lbtt %f >%T' \
           'ltl3dra -f %s >%D' 'lbt <%L >%T' --csv=output.csv
```

Escape sequences specify how formulas are passed, and how automata should be read back.

- Input formula: `%s` (Spin syntax), `%f` (Spot syntax), `%l` (LBT's syntax), or `%w` (Wring's syntax).
  Use `%S`, `%F`, `%L`, or `%W` for files instead of strings.
- Filenames for output automata:
  `%N`: Spin neverclaim (Büchi automaton)
  `%T`: LBTT's format (BA, GBA, or TGBA)
  `%D`: `ltl2dstar`'s format (determ. Rabin or Streett) [Spot 1.2]
- Spot uses TGBA internally: Rabin and Streett automata are converted.

ltlcross – output: errors

An example of error

```
$ ltlfilt --nnf -f 'GFa xor GFb' |
  ltlcross 'modella -r12 -g -e %L %T' 'ltl2tgba --lbtt %s >%T'\
  --csv=output.csv
```

```
([](<>(p0)) && (<>([](!(p1))))) || ((<>([](!(p0)))) && ([](<>(p1))))
Running [P0]: modella -r12 -g -e 'lcr-i0-uhOnWh' 'lcr-o0-nLG9bw'
Running [P1]: ltl2tgba --lbtt '([](<>(p0)) && (<>([](!(p1))))) || ((<▶
Running [N0]: modella -r12 -g -e 'lcr-i0-GZQDSY' 'lcr-o0-h88Kfd'
Running [N1]: ltl2tgba --lbtt '(!([](<>(p0)) && (<>([](!(p1)))))) || ▶
Performing sanity checks and gathering statistics...
error: P0*N0 is nonempty; both automata accept the infinite word
       cycle{!p0 & !p1}
error: P0*N1 is nonempty; both automata accept the infinite word
       cycle{p0 & p1}
error: P1*N0 is nonempty; both automata accept the infinite word
       cycle{p0 & !p1}
```

Note: MoDeLLa 1.5.9 (March 2006) does not have a web-page anymore.

`ltlcross` – output: statistics

`output.csv` contains:

- formula φ (positive and negative on separate lines,
- translator,
- number of states, transitions, edges, acceptance sets,
- number of SCCs, nonaccepting|terminal|weak|strong SCCs,
- number of nondeterministic states,
- wether automaton is nondeterministic, terminal, weak, strong,
- time of taken by the translation,
- number of states, transitions, and SCCs in the product.

Same data can be output with `--json=output.json` for easier embedding in a web page for interactive visualization.

Conclusion

- `randltl`, `ltlfilt`, `genltl`, `ltl2tgba`, `ltl2tgta`, `ltlcross`, `dstar2tgba`
- Command-line tools are mostly about LTL (and PSL) currently.
- More automata-focused tools planned.
- Download from `http://spot.lip6.fr/`
- Send questions, suggestions, or bug reports to `spot@lrde.epita.fr`

Bonus: Simple PSL formulas that can be translated into LTL

```
$ ltlfilt -f '{c*;(a*;b*)*;c}!' -f '{a[=3]}! -> Gb' --simplify
(a | b) U c
Gb | (a R (!a | X(a R (!a | XG!a))))
```