Spot's Temporal Logic Formulas

Alexandre Duret-Lutz <adl@lrde.epita.fr>
compiled on October 16, 2026, for Spot 2.16.1

1. Reasoning with Infinite Sequences **3**
1.1. Finite and Infinite Sequences . . . 3
1.2. Usage in Model Checking . . . 3

2. Temporal Syntax & Semantics **4**
2.1. Boolean Constants . . . 4
2.1.1. Semantics . . . 4
2.2. Atomic Propositions . . . 4
2.2.1. Examples . . . 5
2.2.2. Semantics . . . 5
2.3. Boolean Operators (for Temporal Formulas) . . . 5
2.3.1. Semantics . . . 6
2.3.2. Trivial Identities (Occur Automatically) . . . 6
2.4. Temporal Operators . . . 6
2.4.1. Semantics . . . 7
2.4.2. Syntactic Sugar . . . 7
2.4.3. Trivial Identities (Occur Automatically) . . . 7
2.5. SERE Operators . . . 8
2.5.1. Semantics . . . 8
2.5.2. Syntactic Sugar . . . 9
2.5.3. Trivial Identities (Occur Automatically) . . . 10
2.6. SERE-LTL Binding Operators . . . 11
2.6.1. Semantics . . . 11
2.6.2. Syntactic Sugar . . . 12
2.6.3. Trivial Identities (Occur Automatically) . . . 12
2.7. Quantifiers . . . 12
2.7.1. Semantics . . . 12
2.7.2. Trivial Identities (Occur Automatically) . . . 13

3. Grammar **14**
3.1. Operator precedence . . . 14

4. Properties **16**
4.1. Pure Eventualities and Purely Universal Formulas . . . 16
4.2. Syntactic Hierarchy Classes . . . 17

5. Rewritings **20**
5.1. Unabbreviations . . . 20
5.2. LTL simplifier . . . 20
5.3. Negative normal form . . . 21

5.4. Simplifications . . . 21
5.4.1. Basic Simplifications . . . 22
5.4.2. Simplifications for Eventual and Universal Formulas . . . 26
5.4.3. Simplifications Based on Implications . . . 27
5.5. Unit Propagation . . . 30

A. Defining LTL with only one of `U`, `W`, `R`, or `M` **32**

B. Syntactic Implications **34**

Bibliography **34**

1. Reasoning with Infinite Sequences

1.1. Finite and Infinite Sequences

Let $\mathbb{N} = \{0, 1, 2, \ldots\}$ denote the set of natural numbers and $\omega \notin \mathbb{N}$ the first transfinite ordinal. We extend the $<$ relation from $\mathbb{N}$ to $\mathbb{N} \cup \{\omega\}$ with $\forall n \in \mathbb{N}$, $n < \omega$. Similarly let us extend the addition and subtraction with $\forall n \in \mathbb{N}$, $\omega + n = \omega - n = \omega + \omega = \omega$.

For any set A, and any number $n \in \mathbb{N} \cup \{\omega\}$, a *sequence* of length n is a function $\sigma : \{0, 1, \ldots, n-1\} \to A$ that associates each index $i < n$ to an element $\sigma(i) \in A$. The sequence of length 0 is a particular sequence called the *empty word* and denoted ε. We denote A^n the set of all sequences of length n on A (in particular A^ω is the set of infinite sequences on A), and $A^\star = \bigcup_{n \in \mathbb{N}} A^n$ denotes the set of all finite sequences. The length of any sequence σ is noted $|\sigma|$, with $|\sigma| \in \mathbb{N} \cup \{\omega\}$.

For any sequence σ, we denote $\sigma^{i..j}$ the finite subsequence built using letters from $\sigma(i)$ to $\sigma(j)$. If σ is infinite, we denote $\sigma^{i..}$ the suffix of σ starting at letter $\sigma(i)$.

1.2. Usage in Model Checking

The temporal formulas described in this document, should be interpreted on behaviors (or executions, or scenarios) of the system to verify. In model checking we want to ensure that a formula (the property to verify) holds on all possible behaviors of the system.

If we model the system as some sort of giant automaton (e.g., a Kripke structure) where each state represent a configuration of the system, a behavior of the system can be represented by an infinite sequence of configurations. Each configuration can be described by an affectation of some proposition variables that we will call *atomic propositions*. For instance $r = 1, y = 0, g = 0$ describes the configuration of a traffic light with only the red light turned on.

Let AP be a set of atomic propositions, for instance $\mathrm{AP} = \{r, y, g\}$. A configuration of the model is a function $\rho : \mathrm{AP} \to \mathbb{B}$ (or $\rho \in \mathbb{B}^{\mathrm{AP}}$) that associates a truth value ($\mathbb{B} = \{0, 1\}$) to each atomic proposition.

A behavior of the model is an infinite sequence σ of such configurations. In other words: $\sigma \in (\mathbb{B}^{\mathrm{AP}})^\omega$.

When a formula φ holds on an *infinite* sequence σ, we write $\sigma \models \varphi$ (read as σ is a model of φ).

When a formula φ holds on an *finite* sequence σ, we write $\sigma \Models \varphi$.

2. Temporal Syntax & Semantics

2.1. Boolean Constants

The two Boolean constants are '`1`' and '`0`'. They can also be input as '`true`' or '`false`' (case insensitive) for compatibility with the output of other tools, but Spot will always use '`1`' and '`0`' in its output.

2.1.1. Semantics

$$\sigma \not\models 0$$
$$\sigma \models 1$$

2.2. Atomic Propositions

Atomic propositions in Spot are strings of characters. There are no restrictions on the characters that appear in the strings, however because some of the characters may also be used to denote operators you may have to represent the strings differently if they include these characters.

1. Any string of characters represented between double quotes is an atomic proposition.
2. Any sequence of alphanumeric characters (including '_') that is not a reserved keyword and that starts with a characters that is not an uppercase '`F`', '`G`', or '`X`', is also an atomic proposition. In this case the double quotes are not necessary.
3. Any sequence of alphanumeric characters that starts with '`F`', '`G`', or '`X`', has a digit in second position, and anything afterwards, is also an atomic proposition, and the double quotes are not necessary.

Here is the list of reserved keywords:

- '`true`', '`false`' (both are case insensitive)
- '`F`', '`G`', '`M`', '`R`', '`U`', '`V`', '`W`', '`X`', '`xor`'

The only way to use an atomic proposition that has the name of a reserved keyword, or one that starts with a digit, is to use double quotes.

The reason we deal with leading '`F`', '`G`', and '`X`' specifically in rule 2 is that these are unary LTL operators and we want to be able to write compact formulas like '`GFa`' instead of the equivalent '`G(F(a))`' or '`G F a`'. If you want to name an atomic proposition '`GFa`', you will have to quote it as '`"GFa"`'.

The exception done by rule 3 when these letters are followed by a digit is meant to allow '`X0`', '`X1`', '`X2`', ... to be used as atomic propositions. With only rule 2, '`X0`' would be interpreted as '`X(0)`', that is, the LTL operator `X` applied to the constant *false*, but there is really little reason to use such a construction in a formula (the same is true for '`F`' and '`G`', and also when applied to '`1`'). On the other hand, having numbered versions of a variable is pretty common, so it makes sense to favor this interpretation.

If you are typing in formulas by hand, we suggest you name all your atomic propositions in lower case, to avoid clashes with the uppercase operators.

If you are writing a tool that produces formula that will be feed to Spot and if you cannot control the atomic propositions that will be used, we suggest that you always output atomic propositions between double quotes to avoid any unintended misinterpretation.

2.2.1. Examples

- '`"a<=b+c"`' is an atomic proposition. Double quotes can therefore be used to embed constructs specific to the underlying formalism, and still regard the resulting construction as an atomic proposition.
- '`light_on`' is an atomic proposition.
- '`Fab`' is not an atomic proposition, this is actually equivalent to the formula '`F(ab)`' where the temporal operator `F` is applied to the atomic proposition '`ab`'.
- '`FINISHED`' is not an atomic proposition for the same reason; it actually stands for '`F(INISHED)`'
- '`F100ZX`' is an atomic proposition by rule 3.
- '`FX100`' is not an atomic proposition, it is equivalent to the formula '`F(X100)`', where '`X100`' is an atomic proposition by rule 3.

2.2.2. Semantics

For any atomic proposition a, we have

$$\sigma \models a \iff \sigma(0)(a) = 1$$

In other words a holds if and only if it is true in the first configuration of σ.

2.3. Boolean Operators (for Temporal Formulas)

Two temporal formulas f and g can be combined using the following Boolean operators:

operation	preferred syntax	other supported syntaxes			UTF8 characters supported preferred	others
negation	`!` f	`~` f			¬ `U+00AC`	
disjunction	f `|` g	f `||` g	f `\/` g	f `+` g	∨ `U+2228`	∪ `U+222A`
conjunction	f `&` g	f `&&` g	f `/\` g	f `*` g[1]	∧ `U+2227`	∩ `U+2229`
implication	f `->` g	f `=>` g	f `-->` g		→ `U+2192`	→ `U+27F6`, ⇒ `U+21D2` `U+27F9`
exclusion	f `xor` g	f `^` g			⊕ `U+2295`	
equivalence	f `<->` g	f `<=>` g	f `<-->` g		↔ `U+2194`	⇔ `U+21D4`

Additionally, an atomic proposition a can be negated using the syntax 'a`=0`', which is equivalent to '`!` a'. Also 'a`=1`' is equivalent to just '`a`'. These two syntaxes help us read formulas written using Wring's syntax.

When using UTF-8 input, a '`=0`' that follow a single-letter atomic proposition may be replaced by a combining overline `U+0305` or a combining overbar `U+0304`. When instructed to emit UTF-8, Spot will output '`ā`' using a combining overline instead of '`¬a`' for any single-letter atomic proposition.

When a formula is built using only Boolean constants (section 2.1), atomic proposition (section 2.2), and the above operators, we say that the formula is a *Boolean formula*.

[1] The `*`-form of the conjunction operator (allowing better compatibility with Wring and VIS) may only used in temporal formulas. Boolean expressions that occur inside SERE (see Section 2.5) may not use this form because the `*` symbol is used as the Kleene star.

2.3.1. Semantics

$$
\begin{aligned}
\sigma \models \texttt{!}\, f &\iff (\sigma \not\models f) \\
\sigma \models f \,\texttt{\&}\, g &\iff (\sigma \models f) \wedge (\sigma \models g) \\
\sigma \models f \,\texttt{|}\, g &\iff (\sigma \models f) \vee (\sigma \models g) \\
\sigma \models f \,\texttt{->}\, g &\iff (\sigma \not\models f) \vee (\sigma \models g) \\
\sigma \models f \,\texttt{xor}\, g &\iff ((\sigma \models f) \wedge (\sigma \not\models g)) \vee ((\sigma \not\models f) \wedge (\sigma \models g)) \\
\sigma \models f \,\texttt{<->}\, g &\iff ((\sigma \models f) \wedge (\sigma \models g)) \vee ((\sigma \not\models f) \wedge (\sigma \not\models g))
\end{aligned}
$$

2.3.2. Trivial Identities (Occur Automatically)

Trivial identities are applied every time an expression is constructed. This means for instance that there is not way to construct the expression '$\texttt{!}\,\texttt{!}\,a$' in Spot, such an attempt will always yield the expression 'a'.

$\texttt{!}\,0 \equiv 1$	$1 \,\texttt{->}\, f \equiv f$	$f \,\texttt{->}\, 1 \equiv 1$
$\texttt{!}\,1 \equiv 0$	$0 \,\texttt{->}\, f \equiv 1$	$f \,\texttt{->}\, 0 \equiv \texttt{!}\, f$
$\texttt{!}\,\texttt{!}\, f \equiv f$		$f \,\texttt{->}\, f \equiv 1$

The next set of rules apply to operators that are commutative, so these identities are also valid with the two arguments swapped.

$0 \,\texttt{\&}\, f \equiv 0$	$0 \,\texttt{|}\, f \equiv f$	$0 \,\texttt{xor}\, f \equiv f$	$0 \,\texttt{<->}\, f \equiv \texttt{!}\, f$
$1 \,\texttt{\&}\, f \equiv f$	$1 \,\texttt{|}\, f \equiv 1$	$1 \,\texttt{xor}\, f \equiv \texttt{!}\, f$	$1 \,\texttt{<->}\, f \equiv f$
$f \,\texttt{\&}\, f \equiv f$	$f \,\texttt{|}\, f \equiv f$	$f \,\texttt{xor}\, f \equiv 0$	$f \,\texttt{<->}\, f \equiv 1$

The '`&`' and '`|`' operators are associative, so they are actually implemented as n-ary operators (i.e., not binary): this allows us to reorder all arguments in a unique way (e.g. alphabetically). For instance the two expressions '`a&c&b&!d`' and '`c&!d&b&a`' are actually represented as the operator '`&`' applied to the arguments $\{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{!d}\}$. Because these two expressions have the same internal representation, they are actually considered equal for the purpose of the above identities. For instance '`(a&c&b&!d)->(c&!d&b&a)`' will be rewritten to '`1`' automatically.

2.4. Temporal Operators

Given two temporal formulas f, and g, the following temporal operators can be used to construct another temporal formula.

operator	preferred syntax	other supported syntaxes	UTF8 characters supported preferred	others
(Weak) Next	$\texttt{X}\, f$	$\texttt{()}\, f$	○ U+25CB	○ U+25EF
Strong Next	$\texttt{X[!]}\, f$		ⓧ U+24CD	
Eventually	$\texttt{F}\, f$	$\texttt{<>}\, f$	◇ U+25C7	◇ U+22C4 U+2662
Always	$\texttt{G}\, f$	$\texttt{[]}\, f$	□ U+25A1	□ U+2B1C U+25FB
(Strong) Until	$f \,\texttt{U}\, g$			
Weak Until	$f \,\texttt{W}\, g$			
(Weak) Release	$f \,\texttt{R}\, g$	$f \,\texttt{V}\, g$		
Strong Release	$f \,\texttt{M}\, g$			

2.4.1. Semantics

$$
\begin{aligned}
\sigma \models \mathtt{X}\, f &\iff \sigma^{1..} \models f \\
\sigma \models \mathtt{X[!]}\, f &\iff \sigma^{1..} \models f \\
\sigma \models \mathtt{F}\, f &\iff \exists i \in \mathbb{N},\, \sigma^{i..} \models f \\
\sigma \models \mathtt{G}\, f &\iff \forall i \in \mathbb{N},\, \sigma^{i..} \models f \\
\sigma \models f\, \mathtt{U}\, g &\iff \exists j \in \mathbb{N},\, \begin{cases} \forall i < j,\, \sigma^{i..} \models f \\ \sigma^{j..} \models g \end{cases} \\
\sigma \models f\, \mathtt{W}\, g &\iff (\sigma \models f\, \mathtt{U}\, g) \vee (\sigma \models \mathtt{G}\, f) \\
\sigma \models f\, \mathtt{M}\, g &\iff \exists j \in \mathbb{N},\, \begin{cases} \forall i \leqslant j,\, \sigma^{i..} \models g \\ \sigma^{j..} \models f \end{cases} \\
\sigma \models f\, \mathtt{R}\, g &\iff (\sigma \models f\, \mathtt{M}\, g) \vee (\sigma \models \mathtt{G}\, g)
\end{aligned}
$$

Note that the semantics of `X` (weak next) and `X[!]` (strong next) are identical in LTL formulas. The two operators make sense only to build LTLf formulas (i.e., LTL with finite semantics), for which support is being progressively introduced in Spot.

Appendix A explains how to rewrite the above LTL operators using only `X` and one operator chosen among `U`, `W`, `M`,and `R`. This could be useful to understand the operators `R`, `M`, and `W` if you are only familiar with `X` and `U`.

2.4.2. Syntactic Sugar

The syntax on the left is equivalent to the syntax on the right. Some of rewritings taken from the syntax of TLSF [15] are performed from left to right when parsing a formula. They express the fact that some formula f has to be true in n steps, or at some or all times between n and m steps.

$$
\begin{aligned}
\mathtt{X}[n]\, f &\equiv \underbrace{\mathtt{X}\,\mathtt{X}\ldots\mathtt{X}}_{n \text{ occurrences of } \mathtt{X}}\, f \\
\mathtt{F}[n{:}m]\, f &\equiv \underbrace{\mathtt{X}\,\mathtt{X}\ldots\mathtt{X}}_{n \text{ occ. of } \mathtt{X}}(f \mid \underbrace{\mathtt{X}(f \mid \mathtt{X}(\ldots \mid \mathtt{X}\, f))}_{m-n \text{ occ. of } \mathtt{X}}) && \mathtt{F}[n{:}]\, f \equiv \mathtt{X}[n]\, \mathtt{F}\, f \\
\mathtt{G}[n{:}m]\, f &\equiv \underbrace{\mathtt{X}\,\mathtt{X}\ldots\mathtt{X}}_{n \text{ occ. of } \mathtt{X}}(f \,\&\, \underbrace{\mathtt{X}(f \,\&\, \mathtt{X}(\ldots \,\&\, \mathtt{X}\, f))}_{m-n \text{ occ. of } \mathtt{X}}) && \mathtt{G}[n{:}]\, f \equiv \mathtt{X}[n]\, \mathtt{G}\, f \\
\mathtt{X}[n!]\, f &\equiv \underbrace{\mathtt{X[!]X[!]}\ldots\mathtt{X[!]}}_{n \text{ occurrences of } \mathtt{X[!]}}\, f \\
\mathtt{F}[n{:}m!]\, f &\equiv \underbrace{\mathtt{X[!]X[!]}\ldots\mathtt{X[!]}}_{n \text{ occ. of } \mathtt{X[!]}}(f \mid \underbrace{\mathtt{X[!]}(f \mid \mathtt{X[!]}(\ldots \mid \mathtt{X[!]}\, f))}_{m-n \text{ occ. of } \mathtt{X[!]}}) && \mathtt{F}[n{:}!]\, f \equiv \mathtt{X}[n!]\, \mathtt{F}\, f \\
\mathtt{G}[n{:}m!]\, f &\equiv \underbrace{\mathtt{X[!]X[!]}\ldots\mathtt{X[!]}}_{n \text{ occ. of } \mathtt{X[!]}}(f \,\&\, \underbrace{\mathtt{X[!]}(f \,\&\, \mathtt{X[!]}(\ldots \,\&\, \mathtt{X[!]}\, f))}_{m-n \text{ occ. of } \mathtt{X[!]}}) && \mathtt{G}[n{:}!]\, f \equiv \mathtt{X}[n!]\, \mathtt{G}\, f
\end{aligned}
$$

2.4.3. Trivial Identities (Occur Automatically)

$$
\begin{aligned}
\mathtt{X[!]}\,0 &\equiv 0 & \mathtt{F}\,0 &\equiv 0 & \mathtt{G}\,0 &\equiv 0 \\
\mathtt{X}\,1 &\equiv 1 & \mathtt{F}\,1 &\equiv 1 & \mathtt{G}\,1 &\equiv 1 \\
& & \mathtt{F}\,\mathtt{F}\, f &\equiv \mathtt{F}\, f & \mathtt{G}\,\mathtt{G}\, f &\equiv \mathtt{G}\, f
\end{aligned}
$$

$f\,\mathtt{U}\,1 \equiv 1$	$f\,\mathtt{W}\,1 \equiv 1$	$f\,\mathtt{M}\,0 \equiv 0$	$f\,\mathtt{R}\,1 \equiv 1$
$0\,\mathtt{U}\,f \equiv f$	$0\,\mathtt{W}\,f \equiv f$	$0\,\mathtt{M}\,f \equiv 0$	$f\,\mathtt{R}\,0 \equiv 0$
$f\,\mathtt{U}\,0 \equiv 0$	$1\,\mathtt{W}\,f \equiv 1$	$1\,\mathtt{M}\,f \equiv f$	$1\,\mathtt{R}\,f \equiv f$
$f\,\mathtt{U}\,f \equiv f$	$f\,\mathtt{W}\,f \equiv f$	$f\,\mathtt{M}\,f \equiv f$	$f\,\mathtt{R}\,f \equiv f$

2.5. SERE Operators

The "SERE" acronym will be translated to different word depending on the source. It can mean either "*Sequential Extended Regular Expression*" [11, 1], "*Sugar Extended Regular Expression*" [4], or "*Semi-Extended Regular Expression*" [12]. In any case, the intent is the same: regular expressions with traditional operations (union '`|`', concatenation '`;`', Kleene star '`[*]`') are extended with operators such as intersection '`&&`', and fusion '`:`'.

Any Boolean formula (section 2.3) is a SERE. SERE can be further combined with the following operators, where f and g denote arbitrary SERE.

operation	preferred syntax	other supported syntaxes			UTF8 characters supported preferred	others
empty word	`[*0]`					
union	f `|` g	f `||` g	f `\/` g	f `+` g		∨ U+2228 ∪ U+222A
intersection	f `&&` g	f `/\` g			∩ U+2229	∧ U+2227
NLM intersection[2]	f `&` g					
concatenation	f `;` g					
fusion	f `:` g					
bounded ;-iter.	f`[*`i`..`j`]`	f`[*`i`:`j`]`	f`[*`i` to `j`]`	f`[*`i`,`j`]`		
unbounded ;-iter.	f`[*`i`..]`	f`[*`i`:]`	f`[*`i` to]`	f`[*`i`,]`		
bounded :-iter.	f`[:*`i`..`j`]`	f`[:*`i`:`j`]`	f`[:*`i` to `j`]`	f`[:*`i`,`j`]`		
unbounded :-iter.	f`[:*`i`..]`	f`[:*`i`:]`	f`[:*`i` to]`	f`[:*`i`,]`		
first match	`first_match(`f`)`					

The character '`$`' or the string '`inf`' can also be used as value for j in the above operators to denote an unbounded range.[3] For instance 'a`[*i,$]`', 'a`[*i:inf]`' and 'a`[*i..]`' all represent the same SERE.

2.5.1. Semantics

The following semantics assume that f and g are two SEREs, while a is an atomic proposition.

$$
\begin{aligned}
\sigma &\not\models \mathtt{0} \\
\sigma &\models \mathtt{1} &&\iff |\sigma| = 1 \\
\sigma &\models \mathtt{[*0]} &&\iff |\sigma| = 0 \\
\sigma &\models a &&\iff \sigma(0)(a) = 1 \wedge |\sigma| = 1 \\
\sigma &\models f \mathtt{\,|\,} g &&\iff (\sigma \models f) \vee (\sigma \models g) \\
\sigma &\models f \mathtt{\,\&\&\,} g &&\iff (\sigma \models f) \wedge (\sigma \models g) \\
\sigma &\models f \mathtt{\,\&\,} g &&\iff \exists k \in \mathbb{N}, \begin{cases} \text{either} & (\sigma \models f) \wedge (\sigma^{0..k-1} \models g) \\ \text{or} & (\sigma^{0..k-1} \models f) \wedge (\sigma \models g) \end{cases} \\
\sigma &\models f \mathtt{\,;\,} g &&\iff \exists k \in \mathbb{N}, (\sigma^{0..k-1} \models f) \wedge (\sigma^{k..} \models g)
\end{aligned}
$$

[2] *Non-Length-Matching* intersection.
[3] SVA uses '`$`' while PSL uses '`inf`'.

$$\sigma \models f : g \iff \exists k \in \mathbb{N},\ (\sigma^{0..k} \models f) \wedge (\sigma^{k..} \models g)$$

$$\sigma \models f\texttt{[*}i\texttt{..}j\texttt{]} \iff \begin{cases} \text{either} & i = 0 \wedge j = 0 \wedge \sigma = \varepsilon \\ \text{or} & i = 0 \wedge j > 0 \wedge \big((\sigma = \varepsilon) \vee (\sigma \models f\texttt{[*}1\texttt{..}j\texttt{]})\big) \\ \text{or} & i > 0 \wedge j > 0 \wedge (\exists k \in \mathbb{N},\ (\sigma^{0..k-1} \models f) \wedge (\sigma^{k..} \models f\texttt{[*}i-1\texttt{..}j-1\texttt{]})) \end{cases}$$

$$\sigma \models f\texttt{[*}i\texttt{..]} \iff \begin{cases} \text{either} & i = 0 \wedge \big((\sigma = \varepsilon) \vee (\sigma \models f\texttt{[*}1\texttt{..]})\big) \\ \text{or} & i > 0 \wedge (\exists k \in \mathbb{N},\ (\sigma^{0..k-1} \models f) \wedge (\sigma^{k..} \models f\texttt{[*}i-1\texttt{..]})) \end{cases}$$

$$\sigma \models f\texttt{[:*}i\texttt{..}j\texttt{]} \iff \begin{cases} \text{either} & i = 0 \wedge j = 0 \wedge \sigma \models \texttt{1} \\ \text{or} & i = 0 \wedge j > 0 \wedge \big((\sigma \models \texttt{1}) \vee (\sigma \models f\texttt{[:*}1\texttt{..}j\texttt{]})\big) \\ \text{or} & i > 0 \wedge j > 0 \wedge (\exists k \in \mathbb{N},\ (\sigma^{0..k} \models f) \wedge (\sigma^{k..} \models f\texttt{[:*}i-1\texttt{..}j-1\texttt{]})) \end{cases}$$

$$\sigma \models f\texttt{[:*}i\texttt{..]} \iff \begin{cases} \text{either} & i = 0 \wedge \big((\sigma \models \texttt{1}) \vee (\sigma \models f\texttt{[:*}1\texttt{..]})\big) \\ \text{or} & i > 0 \wedge (\exists k \in \mathbb{N},\ (\sigma^{0..k} \models f) \wedge (\sigma^{k..} \models f\texttt{[:*}i-1\texttt{..]})) \end{cases}$$

$$\sigma \models \texttt{first_match}(f) \iff (\sigma \models f) \wedge (\forall k < |\sigma|,\ \sigma^{0..k-1} \not\models f)$$

Notes:

- The semantics of `&&` and `&` coincide if both operands are Boolean formulas.
- The SERE $f : g$ will never hold on `[*0]`, regardless of the value of f and g. For instance $a\texttt{[*]} : b\texttt{[*]}$ is actually equivalent to $a\texttt{[*]}\ ;\ \{a\ \texttt{\&\&}\ b\}\ ;\ b\texttt{[*]}$.
- The `[:*i..]` and `[:*i..j]` operators are iterations of the `:` operator just like The `[*i..]` and `[*i..j]` are iterations of the `;` operator. More graphically:

$$f\texttt{[*}i\texttt{..}j\texttt{]} = \underbrace{f\ ;\ f\ ;\ \ldots\ ;\ f}_{\text{between } i \text{ and } j \text{ copies of } f} \qquad f\texttt{[:*}i\texttt{..}j\texttt{]} = \underbrace{f : f : \ldots : f}_{\text{between } i \text{ and } j \text{ copies of } f}$$

  with the convention that

$$f\texttt{[*0..0]} = \texttt{[*0]} \qquad f\texttt{[:*0..0]} = \texttt{1}$$

- The `[:*i..]` and `[:*i..j]` operators are not defined in PSL. While the bounded iteration can be seen as syntactic sugar on `:`, the unbounded version really is a new operator.

  `[:*1..]`, for which we define the `[:+]` syntactic sugar below, actually corresponds to the $\oplus$ operator introduced by Dax et al. [9]. With this simple addition, it is possible to define a subset of PSL that expresses exactly the stutter-invariant ω-regular languages.
- The `first_match` operator does not exist in PSL. It comes from SystemVerilog Assertions (SVA) [2]. One intuition behind `first_match(`f`)` is that the DFA for `first_match(`f`)` can be obtained from the DFA for f by removing all transitions leaving accepting states.

2.5.2. Syntactic Sugar

The syntax on the left is equivalent to the syntax on the right. These rewritings are performed from left to right when parsing a formula, and *some* are performed from right to left when writing it for output. b must be a Boolean formula.

$$b\texttt{[->}i\texttt{..}j\texttt{]} \equiv \{\{\texttt{!}\,b\}\texttt{[*0..]}\ ;\ b\}\texttt{[*}i\texttt{..}j\texttt{]} \qquad b\texttt{[=}i\texttt{..}j\texttt{]} \equiv \{\{\texttt{!}\,b\}\texttt{[*0..]}\ ;\ b\}\texttt{[*}i\texttt{..}j\texttt{]}\ ;\ \{\texttt{!}\,b\}\texttt{[*0..]}$$

$$b\texttt{[->}i\texttt{..]} \equiv \{\{\texttt{!}\,b\}\texttt{[*0..]}\ ;\ b\}\texttt{[*}i\texttt{..]} \qquad b\texttt{[=}i\texttt{..]} \equiv \{\{\texttt{!}\,b\}\texttt{[*0..]}\ ;\ b\}\texttt{[*}i\texttt{..]}\ ;\ \{\texttt{!}\,b\}\texttt{[*0..]} \text{ if } i > 0$$

$$b\texttt{[=0..]} \equiv \texttt{1[*0..]}$$

$$
\begin{array}{rlrlrlrl}
f* & \equiv f[*0..] & & & & & & \\
f[*] & \equiv f[*0..] & f[:*] & \equiv f[:*0..] & f[=] & \equiv f[=0..] & f[->] & \equiv f[->1..1] \\
f[*..] & \equiv f[*0..] & f[:*..] & \equiv f[:*0..] & f[=..] & \equiv f[=0..] & f[->..] & \equiv f[->1..] \\
f[*..j] & \equiv f[*0..j] & f[:*..j] & \equiv f[:*0..j] & f[=..j] & \equiv f[=0..j] & f[->..j] & \equiv f[->1..j] \\
f[*k] & \equiv f[*k..k] & f[:*k] & \equiv f[:*k..k] & f[=k] & \equiv f[=k..k] & f[->k] & \equiv f[->k..k] \\
f[+] & \equiv f[*1..] & f[:+] & \equiv f[:*1..] & & & &
\end{array}
$$

$$
[*k] \equiv 1[*k..k] \qquad [*] \equiv 1[*0..] \qquad [+] \equiv 1[*1..]
$$

The following adds input support for the SVA concatenation (or delay) operator [2]. The simplest equivalence are that f ##0 g, f ##1 g, f ##2 g mean respectively $f : g$, f ; g, and f ; 1 ; g, but the delay can be a range, and f can be omitted.

$$
\begin{array}{rl}
f\,\texttt{##}[i..j]\,g & \equiv f\ ;\ 1[*i-1..j-1]\ ;\ g \quad \text{if } i > 0 \\
f\,\texttt{##}[0..j]\,g & \equiv f : (1[*0..j]\ ;\ g) \quad \text{if } \varepsilon \not\models f \\
f\,\texttt{##}[0..j]\,g & \equiv (f\ ;\ 1[*0..j]) : g \quad \text{if } \varepsilon \models f \wedge \varepsilon \not\models g \\
f\,\texttt{##}[0..j]\,g & \equiv (f : g) \mid (f\ ;\ 1[*0..j-1]\ ;\ g) \quad \text{if } \varepsilon \models f \wedge \varepsilon \models g \\
f\,\texttt{##}[i..]\,g & \equiv f\ ;\ 1[*i-1..]\ ;\ g \quad \text{if } i > 0 \\
f\,\texttt{##}[0..]\,g & \equiv f : (1[*]\ ;\ g) \quad \text{if } \varepsilon \not\models f \\
f\,\texttt{##}[0..]\,g & \equiv (f\ ;\ 1[*]) : g \quad \text{if } \varepsilon \models f \wedge \varepsilon \not\models g \\
f\,\texttt{##}[0..]\,g & \equiv (f : g) \mid (f\ ;\ 1[*]\ ;\ g) \quad \text{if } \varepsilon \models f \wedge \varepsilon \models g
\end{array}
$$

$$
\begin{array}{rlrl}
\texttt{##}[i..j]g & \equiv 1[*i..j]\ ;\ g & \texttt{##}[i..]g & \equiv 1[*i..]\ ;\ g \\
f\,\texttt{##}i\,g & \equiv f\,\texttt{##}[i..i]\,g & \texttt{##}i\,g & \equiv 1[*i]\ ;\ g \\
f\,\texttt{##}[+]\,g & \equiv f\,\texttt{##}[1..]\,g & \texttt{##}[+]g & \equiv \texttt{##}[1..]g \\
f\,\texttt{##}[*]\,g & \equiv f\,\texttt{##}[0..]\,g & \texttt{##}[*]g & \equiv \texttt{##}[0..]g \\
f\,\texttt{##}[..j]\,g & \equiv f\,\texttt{##}[0..j]\,g & \texttt{##}[..j]g & \equiv 1\,\texttt{##}[0..j]\,g \\
f\,\texttt{##}[..]\,g & \equiv f\,\texttt{##}[0..]\,g & \texttt{##}[..]g & \equiv 1\,\texttt{##}[0..]\,g
\end{array}
$$

2.5.3. Trivial Identities (Occur Automatically)

The following identities also hold if j or l are missing (assuming they are then equal to ∞). f can be any SERE, while b, b_1, b_2 are assumed to be Boolean formulas.

$$
\begin{array}{rlrl}
0[*0..j] & \equiv [*0] & 0[*i..j] & \equiv 0 \text{ if } i > 0 \\
[*0][*i..j] & \equiv [*0] & f[*i..j][*k..l] & \equiv f[*ik..jl] \text{ if } k = l \vee i(k+1) \leqslant jk+1 \\
f[*0] & \equiv [*0] & f[*1] & \equiv f \\
b[:*0..j] & \equiv 1 & b[:*i..j] & \equiv b \text{ if } i > 0 \\
[*0][:*0..j] & \equiv 1 & [*0][:*i..j] & \equiv 0 \text{ if } i > 0 \\
 & & f[:*i..j][:*k..l] & \equiv f[:*ik..jl] \text{ if } k = l \vee i(k+1) \leqslant jk+1 \\
f[:*0] & \equiv 1 & f[:*1] & \equiv f \text{ if } \varepsilon \not\models f \\
\texttt{first_match}(b) & \equiv b & \texttt{first_match}(f) & \equiv [*0] \text{ if } \varepsilon \models f \\
 & & \texttt{first_match}(\texttt{first_match}(f)) & \equiv \texttt{first_match}(f)
\end{array}
$$

The following rules are all valid with the two arguments swapped.

$$
\begin{array}{lllll}
\texttt{0 \&\& } f \equiv \texttt{0} & \texttt{0 \& } f \equiv \texttt{0} & \texttt{0 | } f \equiv f & \texttt{0 : } f \equiv \texttt{0} & \texttt{0 ; } f \equiv \texttt{0} \\
\texttt{1 \&\& } b \equiv b & \texttt{1 \& } f \equiv \begin{cases} \texttt{1} \text{ if } \varepsilon \Vdash f \\ f \text{ if } \varepsilon \nVdash f \end{cases} & \texttt{1 | } b \equiv \texttt{1} & \texttt{1 : } f \equiv f \text{ if } \varepsilon \nVdash f & \\
\texttt{[*] \&\& } f \equiv f & & \texttt{[*] | } f \equiv \texttt{[*]} & & \texttt{[*] ; } f \equiv \texttt{[*]} \text{ if } \varepsilon \Vdash f \\
\texttt{[+] \&\& } f \equiv f \text{ if } \varepsilon \nVdash f & & \texttt{[+] | } f \equiv \begin{cases} \texttt{[*]} \text{ if } \varepsilon \Vdash f \\ \texttt{[+]} \text{ if } \varepsilon \nVdash f \end{cases} & & \\
\texttt{[*0] \&\& } f \equiv \begin{cases} \texttt{[*0]} \text{ if } \varepsilon \Vdash f \\ \texttt{0} \quad \text{ if } \varepsilon \nVdash f \end{cases} & \texttt{[*0] \& } f \equiv f & \texttt{[*0] | } f \equiv f \text{ if } \varepsilon \Vdash f & \texttt{[*0] : } f \equiv \texttt{0} & \texttt{[*0] ; } f \equiv f \\
f \texttt{ \&\& } f \equiv f & f \texttt{ \& } f \equiv f & f \texttt{ | } f \equiv f & f \texttt{ : } f \equiv f\texttt{[:*2]} & f \texttt{ ; } f \equiv f\texttt{[*2]} \\
b_1 \texttt{ \& } b_2 \equiv b_1 \texttt{ \&\& } b_2 & & & b_1 \texttt{ : } b_2 \equiv b_1 \texttt{ \&\& } b_2 &
\end{array}
$$

$$
\begin{array}{ll}
f\texttt{[*}i\texttt{..}j\texttt{] ; } f \equiv f\texttt{[*}i+1\texttt{..}j+1\texttt{]} & f\texttt{[*}i\texttt{..}j\texttt{] ; } f\texttt{[*}k\texttt{..}l\texttt{]} \equiv f\texttt{[*}i+k\texttt{..}j+l\texttt{]} \\
f\texttt{[:*}i\texttt{..}j\texttt{] : } f \equiv f\texttt{[:*}i+1\texttt{..}j+1\texttt{]} & f\texttt{[:*}i\texttt{..}j\texttt{] : } f\texttt{[:*}k\texttt{..}l\texttt{]} \equiv f\texttt{[:*}i+k\texttt{..}j+l\texttt{]} \\
b\texttt{[*}i\texttt{..}j\texttt{] : } b \equiv b\texttt{[*}\max(i,1)\texttt{..}j\texttt{]} & b\texttt{[*}i\texttt{..}j\texttt{] : } b\texttt{[*}k\texttt{..}l\texttt{]} \equiv b\texttt{[*}\max(i,1)+\max(k,1)-1\texttt{..}j+l-1\texttt{]}
\end{array}
$$

2.6. SERE-LTL Binding Operators

The following operators combine a SERE r with a PSL formula f to form another PSL formula.

operation	preferred syntax	other supported syntaxes	
(universal) suffix implication	$\{r\}$`[]->`f	$\{r\}$`|->`f	$\{r\}$`(`f`)`
existential suffix implication	$\{r\}$`<>->`f		
weak closure	$\{r\}$		
negated weak closure	`!`$\{r\}$		

For technical reasons, the negated weak closure is actually implemented as an operator, even if it is syntactically and semantically equal to the combination of `!` and $\{r\}$.

UTF-8 input may combine one box or diamond character from section 2.4 with one arrow character from section 2.3 to replace the operators `[]->`, `<>->`, as well as the operators `[]=>` and `<>=>` that will be defined in 2.6.2. Additionally, `|->` may be replaced by ↦ `U+21A6`, and `|=>` by ⤇ `U+2907`.

2.6.1. Semantics

The following semantics assume that r is a SERE, while f is a PSL formula.

$$
\begin{aligned}
\sigma \models \{r\}\texttt{<>->}f &\iff \exists k \geqslant 0, (\sigma^{0..k} \Vdash r) \wedge (\sigma^{k..} \models f) \\
\sigma \models \{r\}\texttt{[]->}f &\iff \forall k \geqslant 0, (\sigma^{0..k} \Vdash r) \rightarrow (\sigma^{k..} \models f) \\
\sigma \models \{r\} &\iff (\exists k \geqslant 0, \sigma^{0..k} \Vdash r) \vee (\forall k \geqslant 0, \exists \pi \in (\mathbb{B}^{\mathrm{AP}})^{\star}, (\sigma^{0..k} \prec \pi) \wedge (\pi \Vdash r)) \\
\sigma \models \texttt{!}\{r\} &\iff (\forall k \geqslant 0, \sigma^{0..k} \nVdash r) \wedge (\exists k \geqslant 0, \forall \pi \in (\mathbb{B}^{\mathrm{AP}})^{\star}, (\sigma^{0..k} \prec \pi) \rightarrow (\pi \nVdash r))
\end{aligned}
$$

The $\prec$ symbol should be read as "*is a prefix of*". So the semantic for '$\sigma \models \{r\}$' is that either there is a (non-empty) finite prefix of σ that is a model of r, or any prefix of σ can be extended into a

finite sequence π that is a model of r. An infinite sequence `a;a;a;a;a;...` is therefore a model of the formula '$\{a$`[+] ; !` $a\}$' even though it never sees '`!` a'. The same sequence is not a model of '$\{a$`[+] ; !` a `;` $(a$`[*] && (`a`[*] ; !` a `;` a`[*]))}$' because this SERE does not accept any word.

2.6.2. Syntactic Sugar

The syntax on the left is equivalent to the syntax on the right. These rewritings are performed from left to right when parsing a formula. Except the one marked with $\overset{\dagger}{\equiv}$, the opposite rewritings are also preformed on output to ease reading.

$$\{r\}\texttt{<>=>}f \equiv \{r\texttt{ ; 1}\}\texttt{<>->}f \qquad \{r\}\texttt{[]=>}f \equiv \{r\texttt{ ; 1}\}\texttt{[]->}f$$

$$\{r\}\texttt{!} \equiv \{r\}\texttt{<>->1} \qquad \{r\}\texttt{|=>}f \overset{\dagger}{\equiv} \{r\texttt{ ; 1}\}\texttt{[]->}f$$

`[]=>` and `|=>` are synonyms in the same way as `[]->` and `|->` are.
The $\{r\}$`!` operator is a *strong closure* operator.

2.6.3. Trivial Identities (Occur Automatically)

For any PSL formula f, any SERE r, and any Boolean formula b, the following rewritings are systematically performed (from left to right).

$$\begin{array}{rrrr}
\{\texttt{0}\}\texttt{[]->}f \equiv \texttt{1} & \{\texttt{0}\}\texttt{<>->}f \equiv \texttt{0} & \{\texttt{0}\} \equiv \texttt{0} & \texttt{!}\{\texttt{0}\} \equiv \texttt{1} \\
\{\texttt{1}\}\texttt{[]->}f \equiv f & \{\texttt{1}\}\texttt{<>->}f \equiv f & \{\texttt{1}\} \equiv \texttt{1} & \texttt{!}\{\texttt{1}\} \equiv \texttt{0} \\
\{\texttt{[*0]}\}\texttt{[]->}f \equiv \texttt{1} & \{\texttt{[*0]}\}\texttt{<>->}f \equiv \texttt{0} & \{\texttt{[*0]}\} \equiv \texttt{0} & \texttt{!}\{\texttt{[*0]}\} \equiv \texttt{1} \\
\{b\}\texttt{[]->}f \equiv (\texttt{!}\,b)\texttt{ | }f & \{b\}\texttt{<>->}f \equiv b\texttt{ \& }f & \{b\} \equiv b & \texttt{!}\{b\} \equiv \texttt{!}\,b \\
\{r\}\texttt{[]->1} \equiv \texttt{1} & \{r\}\texttt{<>->0} \equiv \texttt{0} & & \\
\{\texttt{[*]}\}\texttt{[]->0} \equiv \texttt{0} & \{\texttt{[*]}\}\texttt{<>->1} \equiv \texttt{1} & &
\end{array}$$

2.7. Quantifiers

Spot has some support for quantified formulas. Quantifiers my only appear in front of a formulas. Multiple atomic propositions may be quantified at once.

operation	preferred syntax	other supported syntaxes	UTF8 alternative
existential quantification	`\exists` $a : f$		$\exists$ `U+2203`
	`\exists` a_1 , a_2 , ... , $a_n : f$	`\exists` $a_1\, a_2 \ldots a_n : f$	
universal quantification	`\forall` $a : f$		$\forall$ `U+2200`
	`\forall` a_1 , a_2 , ... , $a_n : f$	`\forall` $a_1\, a_2 \ldots a_n : f$	

2.7.1. Semantics

For an infinite sequence $\sigma \in (\mathbb{B}^{\mathrm{AP}})^\omega$, and $X \subseteq \mathrm{AP}$, let $\sigma_{\setminus X} \in (\mathbb{B}^{\mathrm{AP}\setminus X})^\omega$ designate a sequence such that for all $a \in \mathrm{AP}\setminus X$ and all positions i, the two sequence agree: $\sigma(i)(a) = \sigma_{\setminus X}(i)(a)$.

In the following semantics, we assume that a, a_1, a_2, ... a_n are atomic propositions, and that f is any PSL formula where these those atomic proposition may be occurring.

$$
\begin{aligned}
\sigma \models \backslash\texttt{exists}\, a : f &\iff \exists \sigma', (\sigma' = \sigma_{\backslash\{a\}}) \wedge (\sigma' \models f) \\
\sigma \models \backslash\texttt{exists}\, a_1 \texttt{ , } a_2 \texttt{ , } \ldots \texttt{ , } a_n : f &\iff \exists \sigma', (\sigma' = \sigma_{\backslash\{a_1, a_2, \ldots, a_n\}}) \wedge (\sigma' \models f) \\
\sigma \models \backslash\texttt{forall}\, a : f &\iff \forall \sigma', (\sigma' = \sigma_{\backslash\{a\}}) \wedge (\sigma' \models f) \\
\sigma \models \backslash\texttt{forall}\, a_1 \texttt{ , } a_2 \texttt{ , } \ldots \texttt{ , } a_n : f &\iff \forall \sigma', (\sigma' = \sigma_{\backslash\{a_1, a_2, \ldots, a_n\}}) \wedge (\sigma' \models f)
\end{aligned}
$$

2.7.2. Trivial Identities (Occur Automatically)

Adjacent quantifiers of the same type are automatically fused, removing duplicate variables if any.

$$
\begin{aligned}
\backslash\texttt{exists}\, a_1 : \backslash\texttt{exists}\, a_2 : f &\equiv \backslash\texttt{exists}\, a_1 \texttt{ , } a_2 : f \\
\backslash\texttt{exists}\, \underbrace{a_1 \texttt{ , } a_2 \texttt{ , } \ldots}_{A} : \backslash\texttt{exists}\, \underbrace{b_1 \texttt{ , } \ldots}_{B} : f &\equiv \backslash\texttt{exists}\, \underbrace{a_1 \texttt{ , } a_2 \texttt{ , } \ldots \texttt{ , } b_1 \texttt{ , } \ldots}_{A \cup B} : f \\
\backslash\texttt{forall}\, a_1 : \backslash\texttt{forall}\, a_2 : f &\equiv \backslash\texttt{forall}\, a_1 \texttt{ , } a_2 : f \\
\backslash\texttt{forall}\, \underbrace{a_1 \texttt{ , } a_2 \texttt{ , } \ldots}_{A} : \backslash\texttt{forall}\, \underbrace{b_1 \texttt{ , } \ldots}_{B} : f &\equiv \backslash\texttt{forall}\, \underbrace{a_1 \texttt{ , } a_2 \texttt{ , } \ldots \texttt{ , } b_1 \texttt{ , } \ldots}_{A \cup B} : f
\end{aligned}
$$

Additionally, and even if the parser does not support this, building a quantification over an empty set of variables will simply drop the quantifier. (This comes handy when implementing some syntactic optimizations.)

3. Grammar

For simplicity, this grammar gives only one rule for each operator, even if the operator has multiple synonyms (like '`|`', '`||`', and '`\/`').

constant ::=	`0` | `1`		
atomic_prop ::=	see section 2.2		
bformula ::=	*constant*	| `(` *bformula* `)`	| *bformula* `xor` *bformula*
	| *atomic_prop*	| `!` *bformula*	| *bformula* `<->` *bformula*
	| *atomic_prop*`=0`	| *bformula* `&` *bformula*	| *bformula* `->` *bformula*
	| *atomic_prop*`=1`	| *bformula* `|` *bformula*	
sere ::=	*bformula*	| `[*`*i*`..`*j*`]`	| `##`*i* *sere*
	| `{` *sere* `}`	| `[+]`	| `##[`*i*`..`*j*`]` *sere*
	| `(` *sere* `)`	| *sere* `[*`*i*`..`*j*`]`	| *sere* `##`*i* *sere*
	| *sere* `|` *sere*	| *sere* `[+]`	| *sere* `##[`*i*`..`*j*`]` *sere*
	| *sere* `&` *sere*	| *sere* `[:*`*i*`..`*j*`]`	| `first_match(` *sere* `)`
	| *sere* `&&` *sere*	| *sere* `[:+]`	
	| *sere* `;` *sere*	| *sere* `[=`*i*`..`*j*`]`	
	| *sere* `:` *sere*	| *sere* `[->`*i*`..`*j*`]`	
tformula ::=	*bformula*	| `X` *tformula*	| `{`*sere*`}` `[]->` *tformula*
	| `(` *tformula* `)`	| `X[!]` *tformula*	| `{`*sere*`}` `[]=>` *tformula*
	| `!` *tformula*	| `X[`*i*`..`*j*`]` *tformula*	| `{`*sere*`}` `<>->` *tformula*
	| *tformula* `&` *tformula*	| `X[`*i*`..`*j*`!]` *tformula*	| `{`*sere*`}` `<>=>` *tformula*
	| *tformula* `|` *tformula*	| `F` *tformula*	| `{`*sere*`}`
	| *tformula* `->` *tformula*	| `F[`*i*`..`*j*`]` *tformula*	| `{`*sere*`}` `!`
	| *tformula* `xor` *tformula*	| `F[`*i*`..`*j*`!]` *tformula*	
	| *tformula* `<->` *tformula*	| `G` *tformula*	
	| *tformula* `U` *tformula*	| `G[`*i*`..`*j*`]` *tformula*	
	| *tformula* `W` *tformula*	| `G[`*i*`..`*j*`!]` *tformula*	
	| *tformula* `R` *tformula*		
	| *tformula* `M` *tformula*		
aplist ::=	*atomic_prop*	| *aplist* `,` *atomic_prop*	
qformula ::=	*tformula*	| `\exists` *aplist* `:` *qformula*	| `\forall` *aplist* `:` *qformula*

3.1. Operator precedence

The following operator precedence describes the current parser of Spot. It has not always been this way. Especially, all operators were left associative until version 0.9, when we changed the associativity of `->`, `<->`, `U`, `R`, `W`, and `M` to get closer to the PSL standard [1, 11].

assoc.	operators	priority
right	`[]->`, `[]=>`, `<>->`, `<>=>`	lowest
left	`;`	↓
left	`:`	
left	`##`*i*, `##[`*i*`..`*j*`]`	
right	`->`, `<->`	
left	`xor`	
left	`|`	
left	`&`, `&&`	
right	`U`, `W`, `M`, `R`	
	`F`, `G`, `F[`*i*`..`*j*`]`, `G[`*i*`..`*j*`]`	
	`X`, `X[`*i*`..`*j*`]`	
	`[*`*i*`..`*j*`]`, `[+]`, `[:*`*i*`..`*j*`]`, `[:+]`, `[=`*i*`..`*j*`]`, `[->`*i*`..`*j*`]`	
	`!`	
	`=0`, `=1`	highest

Beware that not all tools agree on the associativity of these operators. For instance Spin, ltl2ba (same parser as spin), Wring, psl2ba, Modella, and NuSMV all have `U` and `R` as left-associative, while Goal (hence Büchi store), LTL2AUT, and LTL2Büchi (from JavaPathFinder) have `U` and `R` as right-associative. Vis and LBTT have these two operators as non-associative (parentheses required). Similarly the tools do not agree on the associativity of `->` and `<->`: some tools handle both operators as left-associative, or both right-associative, other have only `->` as right-associative.

4. Properties

When Spot builds a formula (represented by an AST with shared subtrees) it computes a set of properties for each node. These properties can be queried from any `spot::formula` instance using the following methods:

`is_boolean()`	Whether the formula uses only Boolean operators.
`is_sugar_free_boolean()`	Whether the formula uses only `&`, `|`, and `!` operators. (Especially, no `->` or `<->` are allowed.)
`is_in_nenoform()`	Whether the formula is in negative normal form. See section 5.3.
`is_X_free()`	Whether the formula avoids the `X` operator.
`is_ltl_formula()`	Whether the formula uses only LTL operators. (Boolean operators are also allowed.)
`is_psl_formula()`	Whether the formula uses only PSL operators. (Boolean and LTL operators are also allowed.)
`is_sere_formula()`	Whether the formula uses only SERE operators. (Boolean operators are also allowed, provided no SERE operator is negated.)
`is_finite()`	Whether a SERE describes a finite language (no unbounded stars), or an LTL formula uses no temporal operator but `X`.
`is_eventual()`	Whether the formula is a pure eventuality.
`is_universal()`	Whether the formula is purely universal.
`is_syntactic_safety()`	Whether the formula is a syntactic safety property.
`is_syntactic_guarantee()`	Whether the formula is a syntactic guarantee property.
`is_syntactic_obligation()`	Whether the formula is a syntactic obligation property.
`is_syntactic_recurrence()`	Whether the formula is a syntactic recurrence property.
`is_syntactic_persistence()`	Whether the formula is a syntactic persistence property.
`is_syntactic_delta1()`	Whether the formula belongs to the Δ_1 class.
`is_syntactic_pi2()`	Whether the formula belongs to the Π_2 class.
`is_syntactic_sigma2()`	Whether the formula belongs to the Σ_2 class.
`is_syntactic_delta2()`	Whether the formula belongs to the Δ_2 class.
`is_marked()`	Whether the formula contains a special "marked" version of the `<>->` or `!`$\{r\}$ operators.[1]
`accepts_eword()`	Whether the formula accepts `[*0]`. (This can only be true for a SERE formula.)
`has_lbt_atomic_props()`	Whether the atomic propositions of the formula are all of the form "p*nn*" where *nn* is a string of digits. This is required when converting formula into LBT's format.[2]

4.1. Pure Eventualities and Purely Universal Formulas

These two syntactic classes of formulas were introduced by Etessami and Holzmann [14] to simplify LTL formulas. We shall present the associated simplification rules in Section 5.4.2, for now we only define these two classes.

Pure eventual formulas describe properties that are left-append closed, i.e., any accepted (infinite) sequence can be prefixed by a finite sequence and remain accepted. From an LTL standpoint, if φ is a

[1] These "marked" operators are used when translating recurring `<>->` or `!`$\{r\}$ operators. They are rendered as `<>+>` and `!+`$\{r\}$ and obey the same simplification rules and properties as their unmarked counterpart (except for the `is_marked()` property).

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

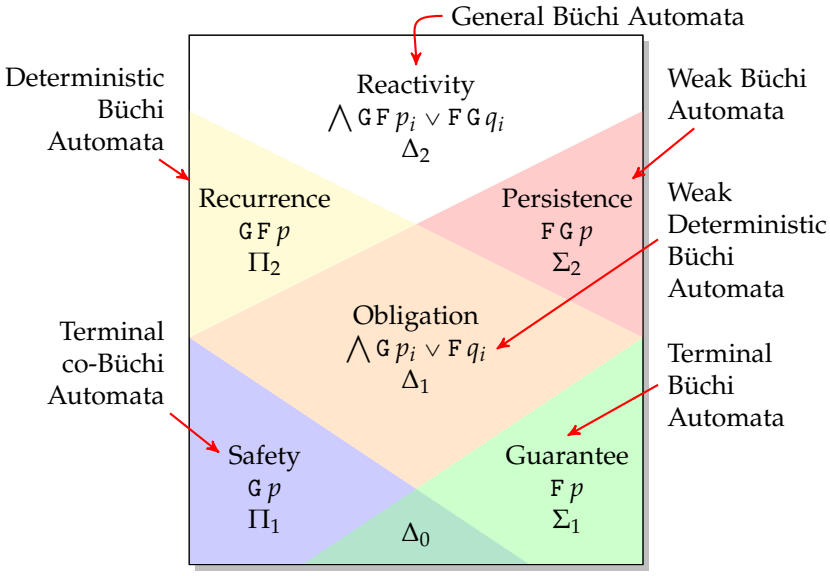

Figure 4.1.: The temporal hierarchy of Manna and Pnueli [16] with their associated classes of automata [6]. The formulas associated to each class are more than canonical examples: they show the normal forms under which any LTL formula of the class can be rewritten, assuming that p, p_i, q, q_i denote subformulas involving only Boolean operators, $\mathtt{X}$, and past temporal operators (Spot does not support the latter).

left-append closed formula, then $\mathtt{F}\,\varphi \equiv \varphi$.

Purely universal formulas describe properties that are suffix-closed, i.e., if you remove any finite prefix of an accepted (infinite) sequence, it remains accepted. From an LTL standpoint if φ is a suffix-closed formula, then $\mathtt{G}\,\varphi \equiv \varphi$.

Let φ denote any arbitrary formula and φ_E (resp. φ_U) denote any instance of a pure eventuality (resp. a purely universal) formula. We have the following grammar rules:

$$
\begin{aligned}
\varphi_E ::= {} & \mathtt{0} \mid \mathtt{1} \mid \mathtt{X}\,\varphi_E \mid \mathtt{X[!]}\,\varphi_E \mid \mathtt{F}\,\varphi \mid \mathtt{G}\,\varphi_E \mid \varphi_E \,\mathtt{\&}\, \varphi_E \mid (\varphi_E \mid \varphi_E) \mid \mathtt{!}\,\varphi_U \\
& \mid \varphi\,\mathtt{U}\,\varphi_E \mid \mathtt{1}\,\mathtt{U}\,\varphi \mid \varphi_E\,\mathtt{R}\,\varphi_E \mid \varphi_E\,\mathtt{W}\,\varphi_E \mid \varphi_E\,\mathtt{M}\,\varphi_E \mid \varphi\,\mathtt{M}\,\mathtt{1} \\
\varphi_U ::= {} & \mathtt{0} \mid \mathtt{1} \mid \mathtt{X}\,\varphi_U \mid \mathtt{X[!]}\,\varphi_U \mid \mathtt{F}\,\varphi_U \mid \mathtt{G}\,\varphi \mid \varphi_U \,\mathtt{\&}\, \varphi_U \mid (\varphi_U \mid \varphi_U) \mid \mathtt{!}\,\varphi_E \\
& \mid \varphi_U\,\mathtt{U}\,\varphi_U \mid \varphi\,\mathtt{R}\,\varphi_U \mid \mathtt{0}\,\mathtt{R}\,\varphi \mid \varphi_U\,\mathtt{W}\,\varphi_U \mid \varphi\,\mathtt{W}\,\mathtt{0} \mid \varphi_U\,\mathtt{M}\,\varphi_U
\end{aligned}
$$

Given a formula `f`, its membership to these two classes can be tested with `f.is_eventual()` and `f.is_universal()`.

4.2. Syntactic Hierarchy Classes

The hierarchy of linear temporal properties was introduced by Manna and Pnueli [16] and is illustrated on Fig. 4.1. In the case of the LTL subset of the hierarchy, a first syntactic characterization of the classes was presented by Chang et al. [7], but other presentations have been done including negation [6] and weak until [17].

Spot implements two versions of a syntactic hierarchy, and extend them to deal with PSL operators.

The first hierarchy, usually denoted with names such as Σ_i and Π_i, as shown in Fig. 4.1. Following Esparza et al. [13], we also introduce the Δ_0, Δ_1, and Δ_2 classes.

Intuitively, those classes are related to how the weak operators ($\mathtt{G}$, $\mathtt{W}$, $\mathtt{R}$) alternate with the strong operators ($\mathtt{U}$, $\mathtt{F}$, $\mathtt{M}$) in formula:

- the class Δ_0 contains all formulas that may only use $\mathtt{X}$ as temporal operator,
- formulas in Π_1 contains no strong operators,
- formulas in Σ_1 contains no weak operators,

- the class Δ_1 contains all boolean combinations of Π_1 and Σ_1,

- in each branch of a formula of Π_2 that contains both types of operator, weak operators are all above strong operators,

- in each branch of a formula of Σ_2 that contains both types of operator, strong operators are all above weak operators,

- the class Δ_2 contains all boolean combinations of Π_2 and Σ_2.

Those classes can be captured by the following grammar rules, where v denotes any variable, r any SERE, r_F any bounded SERE (no loops), and r_I any unbounded SERE.

$$
\begin{aligned}
\varphi_{\Delta_0} ::= {} & \texttt{0} \mid \texttt{1} \mid v \mid \texttt{!}\,\varphi_{\Delta_0} \mid \varphi_{\Delta_0}\,\texttt{\&}\,\varphi_{\Delta_0} \mid (\varphi_{\Delta_0}\,\texttt{|}\,\varphi_{\Delta_0}) \mid \varphi_{\Delta_0}\,\texttt{<->}\,\varphi_{\Delta_0} \mid \varphi_{\Delta_0}\,\texttt{xor}\,\varphi_{\Delta_0} \mid \varphi_{\Delta_0}\,\texttt{->}\,\varphi_{\Delta_0} \mid \texttt{X}\,\varphi_{\Delta_0} \\
& \mid \{r_F\} \mid \texttt{!}\{r_F\} \\
\varphi_{\Pi_1} ::= {} & \varphi_{\Delta_0} \mid \texttt{!}\,\varphi_S \mid \varphi_{\Pi_1}\,\texttt{\&}\,\varphi_{\Pi_1} \mid (\varphi_{\Pi_1}\,\texttt{|}\,\varphi_{\Pi_1}) \mid \varphi_S\,\texttt{->}\,\varphi_{\Pi_1} \mid \texttt{X}\,\varphi_{\Pi_1} \mid \texttt{F}\,\varphi_{\Pi_1} \mid \varphi_{\Pi_1}\,\texttt{U}\,\varphi_{\Pi_1} \mid \varphi_{\Pi_1}\,\texttt{M}\,\varphi_{\Pi_1} \\
& \mid \texttt{!}\{r\} \mid \{r\}\texttt{<>->}\varphi_{\Pi_1} \mid \{r_F\}\texttt{[]->}\varphi_{\Pi_1} \\
\varphi_{\Sigma_1} ::= {} & \varphi_{\Delta_0} \mid \texttt{!}\,\varphi_{\Pi_1} \mid \varphi_{\Sigma_1}\,\texttt{\&}\,\varphi_{\Sigma_1} \mid (\varphi_{\Sigma_1}\,\texttt{|}\,\varphi_{\Sigma_1}) \mid \varphi_{\Pi_1}\,\texttt{->}\,\varphi_{\Sigma_1} \mid \texttt{X}\,\varphi_{\Sigma_1} \mid \texttt{G}\,\varphi_{\Sigma_1} \mid \varphi_{\Sigma_1}\,\texttt{R}\,\varphi_{\Sigma_1} \mid \varphi_{\Sigma_1}\,\texttt{W}\,\varphi_{\Sigma_1} \\
& \mid \{r\} \mid \{r_F\}\texttt{<>->}\varphi_{\Sigma_1} \mid \{r\}\texttt{[]->}\varphi_{\Sigma_1} \\
\varphi_{\Delta_1} ::= {} & \varphi_{\Pi_1} \mid \varphi_{\Sigma_1} \mid \texttt{!}\,\varphi_{\Delta_1} \mid \varphi_{\Delta_1}\,\texttt{\&}\,\varphi_{\Delta_1} \mid (\varphi_{\Delta_1}\,\texttt{|}\,\varphi_{\Delta_1}) \mid \varphi_{\Delta_1}\,\texttt{<->}\,\varphi_{\Delta_1} \mid \varphi_{\Delta_1}\,\texttt{xor}\,\varphi_{\Delta_1} \mid \varphi_{\Delta_1}\,\texttt{->}\,\varphi_{\Delta_1} \\
& \mid \texttt{X}\,\varphi_{\Delta_1} \mid \{r_F\}\texttt{<>->}\varphi_{\Delta_1} \mid \{r_F\}\texttt{[]->}\varphi_{\Delta_1} \\
\varphi_{\Sigma_2} ::= {} & \varphi_{\Delta_1} \mid \texttt{!}\,\varphi_{\Pi_2} \mid \varphi_{\Sigma_2}\,\texttt{\&}\,\varphi_{\Sigma_2} \mid (\varphi_{\Sigma_2}\,\texttt{|}\,\varphi_{\Sigma_2}) \mid \varphi_{\Pi_2}\,\texttt{->}\,\varphi_{\Sigma_2} \\
& \mid \texttt{X}\,\varphi_{\Sigma_2} \mid \texttt{F}\,\varphi_{\Sigma_2} \mid \varphi_{\Sigma_2}\,\texttt{U}\,\varphi_{\Sigma_2} \mid \varphi_{\Sigma_2}\,\texttt{M}\,\varphi_{\Sigma_2} \mid \{r\}\texttt{<>->}\varphi_{\Sigma_2} \mid \{r_F\}\texttt{[]->}\varphi_{\Sigma_2} \\
\varphi_{\Pi_2} ::= {} & \varphi_{\Delta_1} \mid \texttt{!}\,\varphi_{\Sigma_2} \mid \varphi_{\Pi_2}\,\texttt{\&}\,\varphi_{\Pi_2} \mid (\varphi_{\Pi_2}\,\texttt{|}\,\varphi_{\Pi_2}) \mid \varphi_{\Sigma_2}\,\texttt{->}\,\varphi_{\Pi_2} \\
& \mid \texttt{X}\,\varphi_{\Pi_2} \mid \texttt{G}\,\varphi_{\Pi_2} \mid \varphi_{\Pi_2}\,\texttt{R}\,\varphi_{\Pi_2} \mid \varphi_{\Pi_2}\,\texttt{W}\,\varphi_{\Pi_2} \mid \{r\}\texttt{[]->}\varphi_{\Pi_2} \mid \{r_F\}\texttt{<>->}\varphi_{\Pi_2} \\
\varphi_{\Delta_2} ::= {} & \varphi_{\Pi_2} \mid \varphi_{\Sigma_2} \mid \texttt{!}\,\varphi_{\Delta_2} \mid \varphi_{\Delta_2}\,\texttt{\&}\,\varphi_{\Delta_2} \mid (\varphi_{\Delta_2}\,\texttt{|}\,\varphi_{\Delta_2}) \mid \varphi_{\Delta_2}\,\texttt{<->}\,\varphi_{\Delta_2} \mid \varphi_{\Delta_2}\,\texttt{xor}\,\varphi_{\Delta_2} \mid \varphi_{\Delta_2}\,\texttt{->}\,\varphi_{\Delta_2} \\
& \mid \texttt{X}\,\varphi_{\Delta_2} \mid \{r_F\}\texttt{<>->}\varphi_{\Delta_2} \mid \{r_F\}\texttt{[]->}\varphi_{\Delta_2}
\end{aligned}
$$

A nice property of these classes, is that they are as expressive as their corresponding automata classes. For instance any LTL/PSL property that is representable by a deterministic Büchi automaton (the recurrence class) can be represented by an LTL/PSL formula in the Π_2 fragment, even if the original formula isn't in the Π_2 fragment originally.

If the objective is to classify properties syntactically, it is useful to use some slightly more complete grammar rules. In the following list, the rules use the initial G, S, O, P, R of their corresponding property clases, as listed in Fig. 4.1 (i.e., Guarantee, Safety, Obligation, Persistence, Recurrence). Additionally, B denotes the "bottom" class (a.k.a. Δ_0). Note that φ_B, φ_G, and φ_S are rigorously equivalent to φ_{Δ_0}, φ_{Π_1}, and φ_{Σ_1}. The difference in the higher classes are highlighted. There is no generalization of φ_{Δ_2} since any LTL/PSL formula is a reactivity property.

$$
\begin{aligned}
\varphi_B ::= {} & \mathtt{0} \mid \mathtt{1} \mid v \mid \mathtt{!}\,\varphi_B \mid \varphi_B \,\mathtt{\&}\, \varphi_B \mid (\varphi_B \mid \varphi_B) \mid \varphi_B \,\mathtt{<->}\, \varphi_B \mid \varphi_B \,\mathtt{xor}\, \varphi_B \mid \varphi_B \,\mathtt{->}\, \varphi_B \mid \mathtt{X}\,\varphi_B \\
& \mid \{r_F\} \mid \mathtt{!}\{r_F\} \\
\varphi_G ::= {} & \varphi_B \mid \mathtt{!}\,\varphi_S \mid \varphi_G \,\mathtt{\&}\, \varphi_G \mid (\varphi_G \mid \varphi_G) \mid \varphi_S \,\mathtt{->}\, \varphi_G \mid \mathtt{X}\,\varphi_G \mid \mathtt{F}\,\varphi_G \mid \varphi_G \,\mathtt{U}\, \varphi_G \mid \varphi_G \,\mathtt{M}\, \varphi_G \\
& \mid \mathtt{!}\{r\} \mid \{r\}\mathtt{<>->}\varphi_G \mid \{r_F\}\mathtt{[]->}\varphi_G \\
\varphi_S ::= {} & \varphi_B \mid \mathtt{!}\,\varphi_G \mid \varphi_S \,\mathtt{\&}\, \varphi_S \mid (\varphi_S \mid \varphi_S) \mid \varphi_G \,\mathtt{->}\, \varphi_S \mid \mathtt{X}\,\varphi_S \mid \mathtt{G}\,\varphi_S \mid \varphi_S \,\mathtt{R}\, \varphi_S \mid \varphi_S \,\mathtt{W}\, \varphi_S \\
& \mid \{r\} \mid \{r_F\}\mathtt{<>->}\varphi_S \mid \{r\}\mathtt{[]->}\varphi_S \\
\varphi_O ::= {} & \varphi_G \mid \varphi_S \mid \mathtt{!}\,\varphi_O \mid \varphi_O \,\mathtt{\&}\, \varphi_O \mid (\varphi_O \mid \varphi_O) \mid \varphi_O \,\mathtt{<->}\, \varphi_O \mid \varphi_O \,\mathtt{xor}\, \varphi_O \mid \varphi_O \,\mathtt{->}\, \varphi_O \\
& \mid \mathtt{X}\,\varphi_O \mid \varphi_O \,\mathtt{U}\, \varphi_G \mid \varphi_O \,\mathtt{R}\, \varphi_S \mid \varphi_S \,\mathtt{W}\, \varphi_O \mid \varphi_G \,\mathtt{M}\, \varphi_O \\
& \mid \{r_F\}\mathtt{<>->}\varphi_O \mid \{r_I\}\mathtt{<>->}\varphi_G \mid \{r_F\}\mathtt{[]->}\varphi_O \mid \{r_I\}\mathtt{[]->}\varphi_S \\
\varphi_P ::= {} & \varphi_O \mid \mathtt{!}\,\varphi_R \mid \varphi_P \,\mathtt{\&}\, \varphi_P \mid (\varphi_P \mid \varphi_P) \mid \varphi_R \,\mathtt{->}\, \varphi_P \\
& \mid \mathtt{X}\,\varphi_P \mid \mathtt{F}\,\varphi_P \mid \varphi_P \,\mathtt{U}\, \varphi_P \mid \varphi_P \,\mathtt{R}\, \varphi_S \mid \varphi_S \,\mathtt{W}\, \varphi_P \mid \varphi_P \,\mathtt{M}\, \varphi_P \\
& \mid \{r\}\mathtt{<>->}\varphi_P \mid \{r_F\}\mathtt{[]->}\varphi_P \mid \{r_I\}\mathtt{[]->}\varphi_S \\
\varphi_R ::= {} & \varphi_O \mid \mathtt{!}\,\varphi_P \mid \varphi_R \,\mathtt{\&}\, \varphi_R \mid (\varphi_R \mid \varphi_R) \mid \varphi_P \,\mathtt{->}\, \varphi_R \\
& \mid \mathtt{X}\,\varphi_R \mid \mathtt{G}\,\varphi_R \mid \varphi_R \,\mathtt{U}\, \varphi_G \mid \varphi_R \,\mathtt{R}\, \varphi_R \mid \varphi_R \,\mathtt{W}\, \varphi_R \mid \varphi_G \,\mathtt{M}\, \varphi_R \\
& \mid \{r\}\mathtt{[]->}\varphi_R \mid \{r_F\}\mathtt{<>->}\varphi_R \mid \{r_I\}\mathtt{<>->}\varphi_G
\end{aligned}
$$

It should be noted that a formula can belong to a class of the temporal hierarchy even if it does not syntactically appears so. For instance the formula $(\mathtt{G}(q \mid \mathtt{F}\,\mathtt{G}\,p) \,\mathtt{\&}\, \mathtt{G}(r \mid \mathtt{F}\,\mathtt{G}\,\mathtt{!}\,p)) \mid \mathtt{G}\,q \mid \mathtt{G}\,r$ is not syntactically safe (and isn't even in Δ_2), yet it is a safety formula equivalent to $\mathtt{G}\,q \mid \mathtt{G}\,r$ (which is in Π_1, the syntactical class of safety formulas). Such a formula is usually said to be a *pathological safety* formula.

To illustrate the difference in the grammar for the higher classes, consider the formula $\mathtt{G}((\mathtt{G}\,a)\,\mathtt{U}\,b)$. This formula can be converted to a deterministic Büchi automaton, so it specifies a recurrence property. It is captured by the grammar rule for φ_R above, yet it does not belong to the Π_2 class because of the alternation between weak ($\mathtt{G}$), strong ($\mathtt{U}$), and weak ($\mathtt{G}$) operators. However the equivalent formula $\mathtt{G}((\mathtt{G}\,a)\,\mathtt{W}\,b)) \wedge \mathtt{G}\,\mathtt{F}\,b$ belongs to Π_2.

Spot computes the membership to each of those class whenever a formula f is constructed. Here is how the membership to each of those class can be tested:

$f \in \Delta_0, f \in B$	`f.is_syntactic_safety() && f.is_syntactic_guarantee()`
$f \in \Pi_1, f \in S$	`f.is_syntactic_safety()`
$f \in \Sigma_1, f \in G$	`f.is_syntactic_guarantee()`
$f \in \Delta_1$	`f.is_delta1()`
$f \in O$	`f.is_syntactic_obligation()`
$f \in \Pi_2$	`f.is_pi2()`
$f \in R$	`f.is_syntactic_recurrence()`
$f \in \Sigma_2$	`f.is_sigma2()`
$f \in P$	`f.is_syntactic_persistence()`
$f \in \Delta_2$	`f.is_delta2()`

5. Rewritings

5.1. Unabbreviations

The 'unabbreviate()' function can apply the following rewriting rules when passed a string denoting the list of rules to apply. For instance passing the string "^ei" will rewrite all occurrences of xor, <-> and ->.

"i"	$f \texttt{->} g \equiv (\texttt{!}\, f) \texttt{ | } g$	
"e"	$f \texttt{<->} g \equiv (f \texttt{ \& } g) \texttt{ | } ((\texttt{!}\, g) \texttt{ \& } (\texttt{!}\, f))$	
"^e"	$f \texttt{ xor } g \equiv (f \texttt{ \& !} g) \texttt{ | } (g \texttt{ \& !} f)$	
"^" without "e"	$f \texttt{ xor } g \equiv \texttt{!}(f \texttt{<->} g)$	
"F"	$\texttt{F}\, e \equiv e$	when e is a pure eventuality
"F"	$\texttt{F}\, f \equiv \texttt{1 U}\, f$	
"G"	$\texttt{G}\, u \equiv u$	when u is purely universal
"G" without "R"	$\texttt{G}\, f \equiv \texttt{0 R}\, f$	
"GR" without "W"	$\texttt{G}\, f \equiv f \texttt{ W 0}$	
"GRW"	$\texttt{G}\, f \equiv \texttt{! F !}\, f$	
"M"	$f \texttt{ M}\, e \equiv \texttt{F}(f \texttt{ \& } e)$	when e is a pure eventuality
"M"	$f \texttt{ M}\, g \equiv g \texttt{ U} (g \texttt{ \& } f)$	
"R"	$f \texttt{ R}\, u \equiv u$	when u is purely universal
"R" without "W"	$f \texttt{ R}\, g \equiv g \texttt{ W} (f \texttt{ \& } g)$	
"RW"	$f \texttt{ R}\, g \equiv g \texttt{ U} ((f \texttt{ \& } g) \texttt{ | } \texttt{G}\, g)$	
"W"	$f \texttt{ W}\, u \equiv \texttt{G}(f \texttt{ | } u)$	when u is purely universal
"W" without "R"	$f \texttt{ W}\, g \equiv g \texttt{ R} (g \texttt{ | } f)$	
"WR"	$f \texttt{ W}\, g \equiv f \texttt{ U} (g \texttt{ | } \texttt{G}\, f)$	

Among all the possible rewritings (see Appendix A) the default rules for R, W and M, those were chosen because they are easier to translate in a tableau construction [10, Fig. 11].

Besides the 'unabbreviate()' function, there is also a class 'unabbreviator() that implements the same functionality, but maintains a cache of abbreviated subformulas. This is preferable if you plan to abbreviate many formulas sharing identical subformulas.

5.2. LTL simplifier

The LTL rewritings described in the next three sections are all implemented in the 'tl_simplifier' class defined in spot/tl/simplify.hh. This class implements several caches in order to quickly rewrite formulas that have already been rewritten previously. For this reason, it is suggested that you reuse your instance of 'tl_simplifier' as much as possible. If you write an algorithm that will simplify LTL formulas, we suggest you accept an optional 'tl_simplifier' argument, so that you can benefit from an existing instance.

The 'tl_simplifier' takes an optional 'tl_simplifier_options' argument, making it possible to tune the various rewritings that can be performed by this class. These options cannot be changed afterwards (because changing these options would invalidate the results stored in the caches).

5.3. Negative normal form

This is implemented by the '`tl_simplifier::negative_normal_form`' method.

A formula in negative normal form can only have negation operators (`!`) in front of atomic properties, and does not use any of the `xor`, `->` and `<->` operators. The following rewriting arrange any PSL formula into negative normal form.

$$
\begin{array}{rcl}
!\,\mathtt{X}\,f \equiv \mathtt{X}\,!\,f & !(f\,\mathtt{U}\,g) \equiv (!\,f)\,\mathtt{R}\,(!\,g) & !(f\,\&\,g) \equiv (!\,f)\,|\,(!\,g) \\
!\,\mathtt{F}\,f \equiv \mathtt{G}\,!\,f & !(f\,\mathtt{R}\,g) \equiv (!\,f)\,\mathtt{U}\,(!\,g) & !(f\,|\,g) \equiv (!\,f)\,\&\,(!\,g) \\
!\,\mathtt{G}\,f \equiv \mathtt{F}\,!\,f & !(f\,\mathtt{W}\,g) \equiv (!\,f)\,\mathtt{M}\,(!\,g) & !(\{r\}\,\mathtt{[]->}\,f) \equiv \{r\}\mathtt{<>->}\,!\,f \\
!(\{r\}) \equiv !\{r\} & !(f\,\mathtt{M}\,g) \equiv (!\,f)\,\mathtt{W}\,(!\,g) & !(\{r\}\mathtt{<>->}\,f) \equiv \{r\}\,\mathtt{[]->}\,!\,f
\end{array}
$$

Recall that the negated weak closure $!\{r\}$ is actually implemented as a specific operator, so it is not actually prefixed by the `!` operator.

$$
\begin{array}{lll}
f\,\mathtt{xor}\,g \equiv ((!\,f)\,\&\,g)\,|\,(f\,\&\,!\,g) & !(f\,\mathtt{xor}\,g) \equiv ((!\,f)\,\&\,(!\,g))\,|\,(f\,\&\,g) & !(f\,\&\,g) \equiv (!\,f)\,|\,(!\,g) \\
f\,\mathtt{<->}\,g \equiv ((!\,f)\,\&\,(!\,g))\,|\,(f\,\&\,g) & !(f\,\mathtt{<->}\,g) \equiv ((!\,f)\,\&\,g)\,|\,(f\,\&\,!\,g) & !(f\,|\,g) \equiv (!\,f)\,\&\,(!\,g) \\
f\,\mathtt{->}\,g \equiv (!\,f)\,|\,g & !(f\,\mathtt{->}\,g) \equiv f\,\&\,!\,g &
\end{array}
$$

Note that the above rules include the "unabbreviation" of operators "`<->`", "`->`", and "`xor`", corresponding to the rules `"ei^` of function '`unabbreviate()` as described in Section 5.1. Therefore it is never necessary to apply these abbreviations before or after '`tl_simplifier::negative_normal_form`'.

If the option '`nenoform_stop_on_boolean`' is set, the above recursive rewritings are not applied to Boolean subformulas. For instance calling '`tl_simplifier::negative_normal_form`' on $!\,\mathtt{F}\,\mathtt{G}(a\,\mathtt{xor}\,b)$ will produce $\mathtt{G}\,\mathtt{F}(((!\,a)\,\&\,(!\,b))\,|\,(a\,\&\,b))$ if '`nenoform_stop_on_boolean`' is unset, while it will produce $\mathtt{G}\,\mathtt{F}(!(a\,\mathtt{xor}\,b))$ if '`nenoform_stop_on_boolean`' is set.

If the option '`keep_top_xor`' is set, the `xor` and `<->` operators are kept as-is at the top level of the formula during negative normal form conversion (i.e., when not under a temporal operator such as `X`, `F`, `G`, `U`, `W`, `R`, or `M`). Under temporal operators they are expanded according to the above rules, i.e., `xor` and `<->` are rewritten into `&` and `|` as usual. For instance calling '`tl_simplifier::negative_normal_form`' on $\mathtt{X}(a\,\mathtt{<->}\,b)$ will always produce $\mathtt{X}(((!\,a)\,\&\,(!\,b))\,|\,(a\,\&\,b))$ regardless of '`keep_top_xor`', but calling it on $(\mathtt{X}\,a)\,\mathtt{<->}\,(\mathtt{X}\,b)$ will keep the `<->` operator when '`keep_top_xor`' is set, and expand it otherwise.

5.4. Simplifications

The '`tl_simplifier::simplify`' method performs several kinds of simplifications, depending on which '`tl_simplifier_options`' was set.

The goals in most of these simplification are to:

- remove useless terms and operator.
- move the `X` operators to the front of the formula (e.g., $\mathtt{X}\,\mathtt{G}\,f$ is better than the equivalent $\mathtt{G}\,\mathtt{X}\,f$). This is because LTL translators will usually want to rewrite LTL formulas in a kind of disjunctive form: $\bigvee_i (\beta_i \wedge \mathtt{X}\,\psi_i)$ where β_is are Boolean formulas and ψ_is are LTL formulas. Moving `X` to the front therefore simplifies the translation.
- move the `F` operators to the front of the formula (e.g., $\mathtt{F}(f\,|\,g)$ is better than the equivalent $(\mathtt{F}\,f)\,|\,(\mathtt{F}\,g)$), but not before `X` ($\mathtt{X}\,\mathtt{F}\,f$ is better than $\mathtt{F}\,\mathtt{X}\,f$). Because $\mathtt{F}\,f$ incurs some indeterminism, it is best to factorize these terms to limit the sources of indeterminism.

Rewritings defined with $\overset{\blacktriangle}{\equiv}$ are applied only when '`tl_simplifier_options::favor_event_univ`' is `true`: they try to lift subformulas that are both eventual and universal *higher* in the syntax tree. Conversely, rules defined with $\overset{\triangledown}{\equiv}$ are applied only when '`favor_event_univ`' is `false`: they try to *lower* subformulas that are both eventual and universal.

Currently all these simplifications assume LTL semantics, so they make no differences between `X` and `X[!]`. For simplicity, they are only listed with `X`.

5.4.1. Basic Simplifications

These simplifications are enabled with '`tl_simplifier_options::reduce_basics`'. A couple of them may enlarge the size of the formula: they are denoted using $\overset{\star}{\equiv}$ instead of $\equiv$, and they can be disabled by setting the '`tl_simplifier_options::reduce_size_strictly`' option to `true`.

Basic Simplifications for Temporal Operators

The following are simplification rules for unary operators (applied from left to right, as usual). The terms $\mathsf{dnf}(f)$ and $\mathsf{cnf}(f)$ denote respectively the disjunctive and conjunctive normal forms if f, handling non-Boolean sub-formulas as if they were atomic propositions.

$$
\begin{array}{rcl}
\mathtt{X}\,\mathtt{F}\,\mathtt{G}\,f \equiv \mathtt{F}\,\mathtt{G}\,f & \mathtt{F}(f\,\mathtt{U}\,g) \equiv \mathtt{F}\,g & \mathtt{G}(f\,\mathtt{R}\,g) \equiv \mathtt{G}\,g \\
\mathtt{X}\,\mathtt{G}\,\mathtt{F}\,f \equiv \mathtt{G}\,\mathtt{F}\,f & \mathtt{F}(f\,\mathtt{M}\,g) \equiv \mathtt{F}(f\,\&\,g) & \mathtt{G}(f\,\mathtt{W}\,g) \equiv \mathtt{G}(f\mid g) \\
\mathtt{F}\,\mathtt{X}\,f \equiv \mathtt{X}\,\mathtt{F}\,f & \mathtt{F}\,\mathtt{G}(f\,\&\,\mathtt{X}\,g) \equiv \mathtt{F}\,\mathtt{G}(f\,\&\,g) & \mathtt{G}\,\mathtt{F}(f\mid \mathtt{X}\,g) \equiv \mathtt{G}\,\mathtt{F}(f\mid g) \\
\mathtt{G}\,\mathtt{X}\,f \equiv \mathtt{X}\,\mathtt{G}\,f & \mathtt{F}\,\mathtt{G}(f\,\&\,\mathtt{G}\,g) \equiv \mathtt{F}\,\mathtt{G}(f\,\&\,g) & \mathtt{G}\,\mathtt{F}(f\mid \mathtt{F}\,g) \equiv \mathtt{G}\,\mathtt{F}(f\mid g) \\
\mathtt{X}\,0 \equiv 0 & \mathtt{F}\,\mathtt{G}(f\mid \mathtt{G}\,g) \equiv \mathtt{F}(\mathtt{G}\,f\mid \mathtt{G}\,g) & \mathtt{G}\,\mathtt{F}(f\,\&\,\mathtt{F}\,g) \equiv \mathtt{G}(\mathtt{F}\,f\,\&\,\mathtt{F}\,g) \\
\mathtt{G}\,\mathtt{F}\,f \overset{\star}{\equiv} \mathtt{G}\,\mathtt{F}(\mathsf{dnf}(f)) & \mathtt{F}\,\mathtt{G}(f\,\&\,\mathtt{F}\,g) \overset{\star}{\equiv} \mathtt{F}\,\mathtt{G}\,f\,\&\,\mathtt{G}\,\mathtt{F}\,g & \mathtt{G}\,\mathtt{F}(f\,\&\,\mathtt{G}\,g) \overset{\star}{\equiv} \mathtt{G}\,\mathtt{F}\,f\,\&\,\mathtt{F}\,\mathtt{G}\,g \\
\mathtt{F}\,\mathtt{G}\,f \overset{\star}{\equiv} \mathtt{F}\,\mathtt{G}(\mathsf{cnf}(f)) & \mathtt{F}\,\mathtt{G}(f\mid \mathtt{F}\,g) \overset{\blacktriangle}{\equiv} \mathtt{F}\,\mathtt{G}\,f\mid \mathtt{G}\,\mathtt{F}\,g & \mathtt{G}\,\mathtt{F}(f\mid \mathtt{G}\,g) \overset{\blacktriangle}{\equiv} \mathtt{G}\,\mathtt{F}\,f\mid \mathtt{F}\,\mathtt{G}\,g
\end{array}
$$

$$
\mathtt{G}(f_1\mid\ldots\mid f_n\mid \mathtt{G}\,\mathtt{F}(g_1)\mid\ldots\mid \mathtt{G}\,\mathtt{F}(g_m)) \equiv \mathtt{G}(f_1\mid\ldots\mid f_n)\mid \mathtt{G}\,\mathtt{F}(g_1\mid\ldots\mid g_m)
$$

Here are the basic rewriting rules for binary operators (excluding `|` and `&` which are considered in Spot as n-ary operators). b denotes any Boolean formula.

$$
\begin{array}{rcl}
1\,\mathtt{U}\,f \equiv \mathtt{F}\,f & f\,\mathtt{W}\,0 \equiv \mathtt{G}\,f \\
f\,\mathtt{M}\,1 \equiv \mathtt{F}\,f & 0\,\mathtt{R}\,f \equiv \mathtt{G}\,f \\
(\mathtt{X}\,f)\,\mathtt{U}\,(\mathtt{X}\,g) \equiv \mathtt{X}(f\,\mathtt{U}\,g) & (\mathtt{X}\,f)\,\mathtt{W}\,(\mathtt{X}\,g) \equiv \mathtt{X}(f\,\mathtt{W}\,g) \\
(\mathtt{X}\,f)\,\mathtt{M}\,(\mathtt{X}\,g) \equiv \mathtt{X}(f\,\mathtt{M}\,g) & (\mathtt{X}\,f)\,\mathtt{R}\,(\mathtt{X}\,g) \equiv \mathtt{X}(f\,\mathtt{R}\,g) \\
(\mathtt{X}\,f)\,\mathtt{U}\,b \overset{\star}{\equiv} b\mid \mathtt{X}(b\,\mathtt{M}\,f) & (\mathtt{X}\,f)\,\mathtt{W}\,b \overset{\star}{\equiv} b\mid \mathtt{X}(f\,\mathtt{R}\,b) \\
(\mathtt{X}\,f)\,\mathtt{M}\,b \overset{\star}{\equiv} b\,\&\,\mathtt{X}(b\,\mathtt{U}\,f) & (\mathtt{X}\,f)\,\mathtt{R}\,b \overset{\star}{\equiv} b\,\&\,\mathtt{X}(f\,\mathtt{W}\,b) \\
f\,\mathtt{U}\,(\mathtt{G}\,f) \equiv \mathtt{G}\,f & f\,\mathtt{W}\,(\mathtt{G}\,f) \equiv \mathtt{G}\,f \\
f\,\mathtt{M}\,(\mathtt{F}\,f) \equiv \mathtt{F}\,f & f\,\mathtt{R}\,(\mathtt{F}\,f) \equiv \mathtt{F}\,f \\
f\,\mathtt{U}\,(g\mid \mathtt{G}(f)) \equiv f\,\mathtt{W}\,g & f\,\mathtt{W}\,(g\mid \mathtt{G}(f)) \equiv f\,\mathtt{W}\,g \\
f\,\mathtt{M}\,(g\,\&\,\mathtt{F}(f)) \equiv f\,\mathtt{M}\,g & f\,\mathtt{R}\,(g\,\&\,\mathtt{F}(f)) \equiv f\,\mathtt{M}\,g \\
f\,\mathtt{U}\,(g\,\&\,f) \equiv g\,\mathtt{M}\,f & f\,\mathtt{W}\,(g\,\&\,f) \equiv g\,\mathtt{R}\,f \\
f\,\mathtt{M}\,(g\mid f) \equiv g\,\mathtt{U}\,f & f\,\mathtt{R}\,(g\mid f) \equiv g\,\mathtt{W}\,f
\end{array}
$$

Here are the basic rewriting rules for n-ary operators (& and |):

$$
\begin{array}{rcl@{\qquad}rcl}
(\mathtt{F}\,\mathtt{G}\,f)\,\&\,(\mathtt{F}\,\mathtt{G}\,g) & \equiv & \mathtt{F}\,\mathtt{G}(f\,\&\,g) & (\mathtt{G}\,\mathtt{F}\,f)\mid(\mathtt{G}\,\mathtt{F}\,g) & \equiv & \mathtt{G}\,\mathtt{F}(f\mid g)\\
(\mathtt{X}\,f)\,\&\,(\mathtt{X}\,g) & \equiv & \mathtt{X}(f\,\&\,g) & (\mathtt{X}\,f)\mid(\mathtt{X}\,g) & \equiv & \mathtt{X}(f\mid g)\\
(\mathtt{X}\,f)\,\&\,(\mathtt{F}\,\mathtt{G}\,g) & \overset{\triangledown}{\equiv} & \mathtt{X}(f\,\&\,\mathtt{F}\,\mathtt{G}\,g) & (\mathtt{X}\,f)\mid(\mathtt{G}\,\mathtt{F}\,g) & \overset{\triangledown}{\equiv} & \mathtt{X}(f\mid\mathtt{G}\,\mathtt{F}\,g)\\
(\mathtt{G}\,f)\,\&\,(\mathtt{G}\,g) & \overset{\triangledown}{\equiv} & \mathtt{G}(f\,\&\,g) & (\mathtt{F}\,f)\mid(\mathtt{F}\,g) & \overset{\triangledown}{\equiv} & \mathtt{F}(f\mid g)\\
(f_1\,\mathtt{U}\,f_2)\,\&\,(f_3\,\mathtt{U}\,f_2) & \equiv & (f_1\,\&\,f_3)\,\mathtt{U}\,f_2 & (f_1\,\mathtt{U}\,f_2)\mid(f_1\,\mathtt{U}\,f_3) & \equiv & f_1\,\mathtt{U}\,(f_2\mid f_3)\\
(f_1\,\mathtt{U}\,f_2)\,\&\,(f_3\,\mathtt{W}\,f_2) & \equiv & (f_1\,\&\,f_3)\,\mathtt{U}\,f_2 & (f_1\,\mathtt{U}\,f_2)\mid(f_1\,\mathtt{W}\,f_3) & \equiv & f_1\,\mathtt{W}\,(f_2\mid f_3)\\
(f_1\,\mathtt{W}\,f_2)\,\&\,(f_3\,\mathtt{W}\,f_2) & \equiv & (f_1\,\&\,f_3)\,\mathtt{W}\,f_2 & (f_1\,\mathtt{W}\,f_2)\mid(f_1\,\mathtt{W}\,f_3) & \equiv & f_1\,\mathtt{W}\,(f_2\mid f_3)\\
(f_1\,\mathtt{R}\,f_2)\,\&\,(f_1\,\mathtt{R}\,f_3) & \equiv & f_1\,\mathtt{R}\,(f_2\,\&\,f_3) & (f_1\,\mathtt{R}\,f_2)\mid(f_3\,\mathtt{R}\,f_2) & \equiv & (f_1\mid f_3)\,\mathtt{R}\,f_2\\
(f_1\,\mathtt{R}\,f_2)\,\&\,(f_1\,\mathtt{M}\,f_3) & \equiv & f_1\,\mathtt{M}\,(f_2\,\&\,f_3) & (f_1\,\mathtt{R}\,f_2)\mid(f_3\,\mathtt{M}\,f_2) & \equiv & (f_1\mid f_3)\,\mathtt{R}\,f_2\\
(f_1\,\mathtt{M}\,f_2)\,\&\,(f_1\,\mathtt{M}\,f_3) & \equiv & f_1\,\mathtt{M}\,(f_2\,\&\,f_3) & (f_1\,\mathtt{M}\,f_2)\mid(f_3\,\mathtt{M}\,f_2) & \equiv & (f_1\mid f_3)\,\mathtt{M}\,f_2\\
(\mathtt{F}\,g)\,\&\,(f\,\mathtt{U}\,g) & \equiv & f\,\mathtt{U}\,g & (\mathtt{G}\,f)\mid(f\,\mathtt{U}\,g) & \equiv & f\,\mathtt{W}\,g\\
(\mathtt{F}\,g)\,\&\,(f\,\mathtt{W}\,g) & \equiv & f\,\mathtt{U}\,g & (\mathtt{G}\,f)\mid(f\,\mathtt{W}\,g) & \equiv & f\,\mathtt{W}\,g\\
(\mathtt{F}\,f)\,\&\,(f\,\mathtt{R}\,g) & \equiv & f\,\mathtt{M}\,g & (\mathtt{G}\,g)\mid(f\,\mathtt{R}\,g) & \equiv & f\,\mathtt{R}\,g\\
(\mathtt{F}\,f)\,\&\,(f\,\mathtt{M}\,g) & \equiv & f\,\mathtt{M}\,g & (\mathtt{G}\,g)\mid(f\,\mathtt{M}\,g) & \equiv & f\,\mathtt{R}\,g\\
f\,\&\,((\mathtt{X}\,f)\,\mathtt{W}\,g) & \equiv & g\,\mathtt{R}\,f & f\mid((\mathtt{X}\,f)\,\mathtt{R}\,g) & \equiv & g\,\mathtt{W}\,f\\
f\,\&\,((\mathtt{X}\,f)\,\mathtt{U}\,g) & \equiv & g\,\mathtt{M}\,f & f\mid((\mathtt{X}\,f)\,\mathtt{M}\,g) & \equiv & g\,\mathtt{U}\,f\\
f\,\&\,(g\mid\mathtt{X}(g\,\mathtt{R}\,f)) & \equiv & g\,\mathtt{R}\,f & f\mid(g\,\&\,\mathtt{X}(g\,\mathtt{W}\,f)) & \equiv & g\,\mathtt{W}\,f\\
f\,\&\,(g\mid\mathtt{X}(g\,\mathtt{M}\,f)) & \equiv & g\,\mathtt{M}\,f & f\mid(g\,\&\,\mathtt{X}(g\,\mathtt{U}\,f)) & \equiv & g\,\mathtt{U}\,f
\end{array}
$$

The above rules are applied even if more terms are presents in the operator's arguments. For instance $\mathtt{F}\,\mathtt{G}(a)\,\&\,\mathtt{G}(b)\,\&\,\mathtt{F}\,\mathtt{G}(c)\,\&\,\mathtt{X}(d)$ will be rewritten as $\mathtt{X}(d\,\&\,\mathtt{F}\,\mathtt{G}(a\,\&\,c))\,\&\,\mathtt{G}(b)$.

The following more complicated rules are generalizations of $f\,\&\,\mathtt{X}\,\mathtt{G}\,f \equiv \mathtt{G}\,f$ and $f\mid\mathtt{X}\,\mathtt{F}\,f \equiv \mathtt{F}\,f$:

$$
\begin{array}{rcl}
f\,\&\,\mathtt{X}(\mathtt{G}(f\,\&\,g\,\&\ldots)\,\&\,h\,\&\ldots) & \equiv & \mathtt{G}(f)\,\&\,\mathtt{X}(\mathtt{G}(g\,\&\ldots)\,\&\,h\,\&\ldots)\\
f\mid\mathtt{X}(\mathtt{F}(f)\mid h\mid\ldots) & \equiv & \mathtt{F}(f)\mid\mathtt{X}(h\mid\ldots)
\end{array}
$$

The latter rule for $f\mid\mathtt{X}(\mathtt{F}(f)\mid h\ldots)$ is only applied if all F-formulas can be removed from the argument of X with the rewriting. For instance $a\mid b\mid c\mid\mathtt{X}(\mathtt{F}(a\mid b)\mid\mathtt{F}(c)\mid\mathtt{G}\,d)$ will be rewritten to $\mathtt{F}(a\mid b\mid c)\mid\mathtt{X}\,\mathtt{G}\,d$ but $b\mid c\mid\mathtt{X}(\mathtt{F}(a\mid b)\mid\mathtt{F}(c)\mid\mathtt{G}\,d)$ will only become $b\mid c\mid\mathtt{X}(\mathtt{F}(a\mid b\mid c)\mid\mathtt{G}\,d)$.

Finally the following rule is applied only when no other terms are present in the OR arguments:

$$
\mathtt{F}(f_1)\mid\ldots\mid\mathtt{F}(f_n)\mid\mathtt{G}\,\mathtt{F}(g)\overset{\triangledown}{\equiv}\mathtt{F}(f_1\mid\ldots\mid f_n\mid\mathtt{G}\,\mathtt{F}(g))
$$

Basic Simplifications for SERE Operators

The following rules, mostly taken from Cimatti et al. [8] are not complete yet. We only show those that are implemented.

The following simplification rules are used for the n-ary operators &&, &, and |. The patterns are of course commutative. b or b_i denote any Boolean formula while r or r_i denote any SERE.

$$b \,\&\&\, r\,[*i..j] \equiv \begin{cases} b \,\&\&\, r & \text{if } i \leqslant 1 \leqslant j \\ 0 & \text{else} \end{cases} \qquad b \,\&\, r \overset{\star}{\equiv} \begin{cases} b \mid \{b : r\} & \text{if } \varepsilon \models r_i \\ b : r & \text{if } \varepsilon \not\models r_i \end{cases}$$

$$b \,\&\&\, \{r_1 : \ldots : r_n\} \equiv b \,\&\&\, r_1 \,\&\&\, \ldots \,\&\&\, r_n$$

$$b \,\&\&\, \{r_1 \,;\, \ldots \,;\, r_n\} \equiv \begin{cases} b \,\&\&\, r_i & \text{if } \exists! i,\ \varepsilon \not\models r_i \\ b \,\&\&\, (r_1 \mid \ldots \mid r_n) & \text{if } \forall i,\ \varepsilon \models r_i \\ 0 & \text{else} \end{cases}$$

$$\{b_1 \,;\, r_1\} \,\&\&\, \{b_2 \,;\, r_2\} \equiv \{b_1 \,\&\&\, b_2\} \,;\, \{r_1 \,\&\&\, r_2\} \qquad \{r_1 \,;\, b_1\} \,\&\&\, \{r_2 \,;\, b_2\} \equiv \{r_1 \,\&\&\, r_2\} \,;\, \{b_1 \,\&\&\, b_2\}$$

$$\{b_1 : r_1\} \,\&\&\, \{b_2 : r_2\} \equiv \{b_1 \,\&\&\, b_2\} : \{r_1 \,\&\&\, r_2\} \qquad \{r_1 : b_1\} \,\&\&\, \{r_2 : b_2\} \equiv \{r_1 \,\&\&\, r_2\} : \{b_1 \,\&\&\, b_2\}$$

$$\{b_1 \,;\, r_1\} \,\&\, \{b_2 \,;\, r_2\} \equiv \{b_1 \,\&\&\, b_2\} \,;\, \{r_1 \,\&\, r_2\}$$

$$\{b_1 : r_1\} \,\&\, \{b_2 : r_2\} \equiv \{b_1 \,\&\&\, b_2\} : \{r_1 \,\&\, r_2\} \quad \text{if } \varepsilon \not\models r_1 \wedge \varepsilon \not\models r_2$$

$$\begin{aligned}
\texttt{first_match}(b\,[*i..j]) &\equiv b\,[*i] \\
\texttt{first_match}(r\,[*i..j]) &\equiv \texttt{first_match}(r\,[*i]) \\
\texttt{first_match}(r\,[:*i..j]) &\equiv \texttt{first_match}(r\,[:*i]) \\
\texttt{first_match}(r_1 \,;\, r_2\,[*i..j]) &\equiv \texttt{first_match}(r_1 \,;\, r_2\,[*i]) \\
\texttt{first_match}(r_1 \,;\, r_2\,[:*i..j]) &\equiv \texttt{first_match}(r_1 \,;\, r_2\,[:*i]) \\
\texttt{first_match}(r_1 : r_2\,[*i..j]) &\equiv \texttt{first_match}(r_1 : r_2\,[*\max(i,1)]) \\
\texttt{first_match}(r_1 : r_2\,[:*i..j]) &\equiv \texttt{first_match}(r_1 : r_2\,[:*i]) \\
\texttt{first_match}(b \,;\, r) &\equiv b \,;\, \texttt{first_match}(r) \\
\texttt{first_match}(b\,[*i..j] \,;\, r) &\equiv b\,[*i] \,;\, \texttt{first_match}(b\,[*0..j-i] \,;\, r) \\
\texttt{first_match}(b\,[*i..j] : r) &\equiv b\,[*i-1] \,;\, \texttt{first_match}(b\,[*1..j-i+1] : r) \quad \text{if } i > 1 \\
\texttt{first_match}(r_1 \,;\, r_2) &\equiv \texttt{first_match}(r_1) \quad \text{if } \varepsilon \models r_2 \\
\texttt{first_match}(\texttt{first_match}(r_1) \,;\, r_2) &\equiv \texttt{first_match}(r_1) \,;\, \texttt{first_match}(r_2) \\
\texttt{first_match}(\texttt{first_match}(r_1) : r_2) &\equiv \texttt{first_match}(r_1) : \texttt{first_match}(1 : r_2)
\end{aligned}$$

Starred subformulas are rewritten in Star Normal Form [5] with:

$$r\,[*] \equiv r^{\circ}\,[*]$$

where r° is recursively defined as follows:

$$\begin{aligned}
r^{\circ} &= r \text{ if } \varepsilon \not\models r & & \\
[*0]^{\circ} &= 0 & (r_1 \,;\, r_2)^{\circ} &= r_1^{\circ} \mid r_2^{\circ} \text{ if } \varepsilon \models r_1 \text{ and } \varepsilon \models r_2 \\
r\,[*i..j]^{\circ} &= r^{\circ} \text{ if } i = 0 \text{ or } \varepsilon \models r & (r_1 \,\&\, r_2)^{\circ} &= r_1^{\circ} \mid r_2^{\circ} \text{ if } \varepsilon \models r_1 \text{ and } \varepsilon \models r_2 \\
(r_1 \mid r_2)^{\circ} &= r_1^{\circ} \mid r_2^{\circ} & (r_1 \,\&\&\, r_2)^{\circ} &= r_1 \,\&\&\, r_2
\end{aligned}$$

Note: the original SNF definition [5] does not include '&' and '&&' operators, and it guarantees that $\forall r,\ \varepsilon \not\models r^{\circ}$ because r° is stripping all the stars and empty words that occur in r. For instance $\{a\,[*] \,;\, b\,[*] \,;\, \{[*0] \mid c\}\}^{\circ}\,[*] = \{a \mid b \mid c\}\,[*]$. Our extended definition still respects this property in presence of '&' operators, but unfortunately not when the '&&' operator is used.

We extend the above definition to bounded repetitions with:

$$\begin{aligned}
r\,[*i..j] &\equiv r^{\square}\,[*0..j] \quad \text{if} \quad \varepsilon \models r\,[*i..j],\ \varepsilon \not\models r^{\square}, \text{ and } j > 1 \\
r\,[*i..j] &\equiv r^{\square}\,[*1..j] \quad \text{if} \quad \varepsilon \models r\,[*i..j],\ \varepsilon \models r^{\square} \text{ and } j > 1 \\
r\,[*i..j] &\equiv r \quad \text{if} \quad \varepsilon \models r \text{ and } j = 1
\end{aligned}$$

where $r^\square$ is recursively defined as follows:

$$
\begin{aligned}
r^\square &= r \text{ if } \varepsilon \not\models r & & \\
[*0]^\square &= 0 & (r_1 \,;\, r_2)^\square &= r_1 \,;\, r_2 \\
r[*i..j]^\square &= r^\square[*\max(1,i)..j] \text{ if } i = 0 \text{ or } \varepsilon \models r & (r_1 \,\&\, r_2)^\square &= r_1^\square \mid r_2^\square \text{ if } \varepsilon \models r_1 \text{ and } \varepsilon \models r_2 \\
(r_1 \mid r_2)^\square &= r_1^\square \mid r_2^\square & (r_1 \,\&\&\, r_2)^\square &= r_1 \,\&\&\, r_2
\end{aligned}
$$

The differences between $^\square$ and $^\circ$ are in the handling of $r[*i..j]$ and in the handling of $r_1 \,;\, r_2$.

Basic Simplifications SERE-LTL Binding Operators

The following rewritings are applied to the operators `[]->` and `<>->`. They assume that b, denote a Boolean formula.

As noted at the beginning for section 5.4.1, rewritings denoted with $\overset{\star}{\equiv}$ can be disabled by setting the '`tl_simplifier_options::reduce_size_strictly`' option to `true`.

$$
\begin{aligned}
\{[*]\}[]\text{->}f &\equiv \mathtt{G}\, f \\
\{b[*]\}[]\text{->}f &\equiv f\, \mathtt{W}\, !\, b \\
\{b[+]\}[]\text{->}f &\equiv f\, \mathtt{W}\, !\, b \\
\{r[*0..j]\}[]\text{->}f &\equiv \{r[*1..j]\}[]\text{->}f \\
\{r[*i..j]\}[]\text{->}f &\overset{\star}{\equiv} \{r\}[]\text{->}\mathtt{X}(\{r\}[]\text{->}\mathtt{X}(\ldots[]\text{->}\mathtt{X}(r[*1..j-i+1]))) \text{ if } i \geqslant 1 \text{ and } \varepsilon \not\models r \\
\{r \,;\, [*]\}[]\text{->}f &\equiv \{r\}[]\text{->}\mathtt{G}\, f \\
\{r \,;\, b[*]\}[]\text{->}f &\overset{\star}{\equiv} \{r\}[]\text{->}(f \,\&\, \mathtt{X}(f\, \mathtt{W}\, !\, b)) \text{ if } \varepsilon \not\models r \\
\{[*] \,;\, r\}[]\text{->}f &\overset{\star}{\equiv} \mathtt{G}(\{r\}[]\text{->}f) \\
\{b[*] \,;\, r\}[]\text{->}f &\overset{\star}{\equiv} (!\, b)\, \mathtt{R}\, (\{r\}[]\text{->}f) \text{ if } \varepsilon \not\models r \\
\{r_1 \,;\, r_2\}[]\text{->}f &\overset{\star}{\equiv} \{r_1\}[]\text{->}\mathtt{X}(\{r_2\}[]\text{->}f) \text{ if } \varepsilon \not\models r_1 \text{ and } \varepsilon \not\models r_2 \\
\{r_1 : r_2\}[]\text{->}f &\overset{\star}{\equiv} \{r_1\}[]\text{->}(\{r_2\}[]\text{->}f) \\
\{r_1 \mid r_2\}[]\text{->}f &\overset{\star}{\equiv} (\{r_1\}[]\text{->}f) \,\&\, (\{r_2\}[]\text{->}f) \\
\{[*]\}\text{<>->}f &\equiv \mathtt{F}\, f \\
\{b[*]\}\text{<>->}f &\equiv f\, \mathtt{M}\, b \\
\{b[+]\}\text{<>->}f &\equiv f\, \mathtt{M}\, b \\
\{r[*0..j]\}\text{<>->}f &\equiv \{r[*1..j]\}\text{<>->}f \\
\{r[*i..j]\}\text{<>->}f &\overset{\star}{\equiv} \{r\}\text{<>->}\mathtt{X}(\{r\}\text{<>->}\mathtt{X}(\ldots\text{<>->}\mathtt{X}(r[*1..j-i+1]))) \text{ if } i \geqslant 1 \text{ and } \varepsilon \not\models r \\
\{r \,;\, [*]\}\text{<>->}f &\equiv \{r\}\text{<>->}\mathtt{F}\, f \\
\{r \,;\, b[*]\}\text{<>->}f &\overset{\star}{\equiv} \{r\}\text{<>->}(f \mid \mathtt{X}(f\, \mathtt{M}\, b)) \text{ if } \varepsilon \not\models r \\
\{[*] \,;\, r\}\text{<>->}f &\overset{\star}{\equiv} \mathtt{F}(\{r\}\text{<>->}f) \\
\{b[*] \,;\, r\}\text{<>->}f &\overset{\star}{\equiv} b\, \mathtt{U}\, (\{r\}\text{<>->}f) \text{ if } \varepsilon \not\models r \\
\{r_1 \,;\, r_2\}\text{<>->}f &\overset{\star}{\equiv} \{r_1\}\text{<>->}\mathtt{X}(\{r_2\}\text{<>->}f) \text{ if } \varepsilon \not\models r_1 \text{ and } \varepsilon \not\models r_2 \\
\{r_1 : r_2\}\text{<>->}f &\overset{\star}{\equiv} \{r_1\}\text{<>->}(\{r_2\}\text{<>->}f) \\
\{r_1 \mid r_2\}\text{<>->}f &\overset{\star}{\equiv} (\{r_1\}\text{<>->}f) \mid (\{r_2\}\text{<>->}f)
\end{aligned}
$$

Here are the basic rewritings for the weak closure and its negation:

$$
\begin{aligned}
\{r\,\texttt{[*]}\} &\equiv \{r\} & \qquad !\{r\,\texttt{[*]}\} &\equiv !\{r\} \\
\{r;\texttt{1}\} &\equiv \{r\} \quad \text{if } \varepsilon \not\models r & !\{r;\texttt{1}\} &\equiv !\{r\} \quad \text{if } \varepsilon \not\models r \\
\{r;\texttt{1}\} &\equiv \texttt{1} \quad \text{if } \varepsilon \models r & !\{r;\texttt{1}\} &\equiv \texttt{0} \quad \text{if } \varepsilon \models r \\
\{r_1;r_2\} &\equiv \{r_1\} \qquad \text{if } \varepsilon \not\models r_1 \wedge \varepsilon \models r_2 & !\{r_1;r_2\} &\equiv !\{r_1\} \qquad \text{if } \varepsilon \not\models r_1 \wedge \varepsilon \models r_2 \\
\{r_1;r_2\} &\equiv \{r_1\} \mid \{r_2\} \quad \text{if } \varepsilon \models r_1 \wedge \varepsilon \models r_2 & !\{r_1;r_2\} &\equiv !\{r_1\} \,\&\, !\{r_2\} \quad \text{if } \varepsilon \models r_1 \wedge \varepsilon \models r_2 \\
\{b;r\} &\overset{\star}{\equiv} b \,\&\, \texttt{X}\,\{r\} & !\{b;r\} &\overset{\star}{\equiv} (!\,b) \mid \texttt{X}\,!\{r\} \\
\{b\,\texttt{[*}i\,\texttt{..}\,j\texttt{]};r\} &\overset{\star}{\equiv} \underbrace{b \,\&\, \texttt{X}(b\ldots}_{i \text{ occurences of } b} \,\&\, \texttt{X}\,\{b\,\texttt{[*}0\,\texttt{..}\,j-i\texttt{]}\ \texttt{;}\ r\}) & !\{b\,\texttt{[*}i\,\texttt{..}\,j\texttt{]};r\} &\overset{\star}{\equiv} \underbrace{(!\,b) \mid \texttt{X}((!\,b)\ldots}_{i \text{ occurences of } !\,b} \mid \texttt{X}\,!\{b\,\texttt{[*}0\,\texttt{..}\,j-i\texttt{]}\ \texttt{;}\ r\}) \\
\{b\,\texttt{[*}i\,\texttt{..}\,j\texttt{]}\} &\overset{\star}{\equiv} \underbrace{b \,\&\, \texttt{X}(b \,\&\, \texttt{X}(\ldots b))}_{i \text{ occurences of } b} & !\{b\,\texttt{[*}i\,\texttt{..}\,j\texttt{]}\} &\overset{\star}{\equiv} \underbrace{(!\,b) \mid \texttt{X}((!\,b) \mid \texttt{X}(\ldots(!\,b)))}_{i \text{ occurences of } !\,b} \\
\{r_1 \mid r_2\} &\overset{\star}{\equiv} \{r_1\} \mid \{r_2\} & !\{r_1 \mid r_2\} &\overset{\star}{\equiv} !\{r_1\} \,\&\, !\{r_2\}
\end{aligned}
$$

Basic simplifications for quantified formulas

Let φ be a temporal formula, let φ' be its negative normal form constructed as in section Section 5.3, and let $a \in \mathrm{AP}$ be a proposition.

We define the polarity of a in φ as:

missing if a does not appear in φ, or if it is already quantified in φ

positive if a appears in φ without being quantified, and never under a negation,

negative if a appears in φ without being quantified, and always under a negation,

both if a appears in φ without being quantified, both under a negation and not under a negation.

For instance in the formula $(a \texttt{ -> } b) \,\&\, (c \texttt{ xor } d)$, the atomic proposition a has negative polarity, b has positive polarity, c and d have both polarities, and e has missing polarity.

In what follow we assume that $f(m,p,n,b)$ is a temporal formulas over atomic propositions m, p, n, and b, and that this atomic propositions respectively have missing, positive, negative, and both polarity.

The following basic simplifications are preformed:

$$
\begin{aligned}
\texttt{\textbackslash exists}\, m\ \texttt{,}\ p\ \texttt{,}\ n\ \texttt{,}\ b : f(m,p,n,b) &\equiv \texttt{\textbackslash exists}\, b : f(m,\texttt{1},\texttt{0},b) \\
\texttt{\textbackslash forall}\, m\ \texttt{,}\ p\ \texttt{,}\ n\ \texttt{,}\ b : f(m,p,n,b) &\equiv \texttt{\textbackslash forall}\, b : f(m,\texttt{0},\texttt{1},b)
\end{aligned}
$$

In other words, atomic proposition that are either positive or negative can be replaced by a constant, missing atomic proposition can be dropped from the quantification list which should only retain atomic proposition with mixed polarity.

Removing quantified atomic propositions like this might empty set of quantified propositions; in that case the quantified is simply dropped, as mentioned in Section 2.7.2.

For instance $\texttt{\textbackslash forall}\, a\ \texttt{,}\ e : \texttt{\textbackslash exists}\, b, c : (a \texttt{ -> } b) \,\&\, (c \texttt{ xor } d)$ would be simplified to $\texttt{\textbackslash exists}\, c : c \texttt{ xor } d$.

5.4.2. Simplifications for Eventual and Universal Formulas

The class of *pure eventuality* and *purely universal* formulas are described in section 4.1.

In the following rewritings, we use the following notation to distinguish the class of subformulas:

f, f_i, g, g_i	any PSL formula
e, e_i	a pure eventuality
u, u_i	a purely universal formula
q, q_i	a pure eventuality that is also purely universal

$$\mathtt{F}\,e \equiv e \qquad f\,\mathtt{U}\,e \equiv e \qquad e\,\mathtt{M}\,g \equiv e \,\&\, g \qquad u_1\,\mathtt{M}\,u_2 \overset{\star}{\equiv} (\mathtt{F}\,u_1) \,\&\, u_2$$

$$\mathtt{F}(u) \mid q \overset{\triangledown}{\equiv} \mathtt{F}(u \mid q) \qquad f\,\mathtt{U}\,(g \mid e) \overset{\blacktriangle}{\equiv} (f\,\mathtt{U}\,g) \mid e \qquad f\,\mathtt{M}\,(g \,\&\, u) \overset{\blacktriangle}{\equiv} (f\,\mathtt{M}\,g) \,\&\, u \qquad q\,\mathtt{U}\,\mathtt{X}\,f \equiv \mathtt{X}(q\,\mathtt{U}\,f)$$

$$f\,\mathtt{U}\,(g \,\&\, q) \overset{\blacktriangle}{\equiv} (f\,\mathtt{U}\,g) \,\&\, q \qquad (f \,\&\, q)\,\mathtt{M}\,g \overset{\blacktriangle}{\equiv} (f\,\mathtt{M}\,g) \,\&\, q$$

$$\mathtt{G}\,u \equiv u \qquad u\,\mathtt{W}\,g \equiv u \mid g \qquad f\,\mathtt{R}\,u \equiv u \qquad e_1\,\mathtt{W}\,e_2 \overset{\star}{\equiv} (\mathtt{G}\,e_1) \mid e_2$$

$$\mathtt{G}(e) \,\&\, q \equiv \mathtt{G}(e \,\&\, q) \qquad f\,\mathtt{W}\,(g \mid e) \overset{\blacktriangle}{\equiv} (f\,\mathtt{W}\,g) \mid e \qquad f\,\mathtt{R}\,(g \,\&\, u) \overset{\blacktriangle}{\equiv} (f\,\mathtt{R}\,g) \,\&\, u \qquad q\,\mathtt{R}\,\mathtt{X}\,f \equiv \mathtt{X}(q\,\mathtt{R}\,f)$$

$$\mathtt{X}\,q \equiv q \qquad q \,\&\, \mathtt{X}\,f \overset{\triangledown}{\equiv} \mathtt{X}(q \,\&\, f) \qquad q \mid \mathtt{X}\,f \overset{\triangledown}{\equiv} \mathtt{X}(q \mid f)$$

$$\mathtt{X}(q \,\&\, f) \overset{\blacktriangle}{\equiv} q \,\&\, \mathtt{X}\,f \qquad \mathtt{X}(q \mid f) \overset{\blacktriangle}{\equiv} q \mid \mathtt{X}\,f$$

$$\mathtt{G}(f_1 \,\&\, \ldots \,\&\, f_n \,\&\, \mathtt{X}\,e_1 \,\&\, \ldots \,\&\, \mathtt{X}\,e_p) \equiv \mathtt{G}(f_1 \,\&\, \ldots \,\&\, f_n \,\&\, e_1 \,\&\, \ldots \,\&\, e_p)$$

$$\mathtt{G}(f_1 \,\&\, \ldots \,\&\, f_n \,\&\, \mathtt{F}(g_1 \,\&\, \ldots \,\&\, g_p \,\&\, \mathtt{X}\,e_1 \,\&\, \ldots \,\&\, \mathtt{X}\,e_m)) \equiv \mathtt{G}(f_1 \,\&\, \ldots \,\&\, f_n \,\&\, \mathtt{F}(g_1 \,\&\, \ldots \,\&\, g_p) \,\&\, e_1 \,\&\, \ldots \,\&\, e_m)$$

$$\mathtt{F}(f_1 \mid \ldots \mid f_n \mid \mathtt{X}\,u_1 \mid \ldots \mid \mathtt{X}\,u_p) \equiv \mathtt{F}(f_1 \mid \ldots \mid f_n \mid u_1 \mid \ldots \mid u_p)$$

$$\mathtt{F}(f_1 \mid \ldots \mid f_n \mid \mathtt{G}(g_1 \mid \ldots \mid g_p \mid \mathtt{X}\,u_1 \mid \ldots \mid \mathtt{X}\,u_m)) \equiv \mathtt{F}(f_1 \mid \ldots \mid f_n \mid \mathtt{G}(g_1 \mid \ldots \mid g_p) \mid u_1 \mid \ldots \mid u_m)$$

$$\mathtt{G}(f_1 \mid \ldots \mid f_n \mid q_1 \mid \ldots \mid q_p) \equiv \mathtt{G}(f_1 \mid \ldots \mid f_n) \mid q_1 \mid \ldots \mid q_p$$

$$\mathtt{F}(f_1 \,\&\, \ldots \,\&\, f_n \,\&\, q_1 \,\&\, \ldots \,\&\, q_p) \overset{\blacktriangle}{\equiv} \mathtt{F}(f_1 \,\&\, \ldots \,\&\, f_n) \,\&\, q_1 \,\&\, \ldots \,\&\, q_p$$

$$\mathtt{G}(f_1 \,\&\, \ldots \,\&\, f_n \,\&\, q_1 \,\&\, \ldots \,\&\, q_p) \overset{\blacktriangle}{\equiv} \mathtt{G}(f_1 \,\&\, \ldots \,\&\, f_n) \,\&\, q_1 \,\&\, \ldots \,\&\, q_p$$

$$\mathtt{G}\,\mathtt{F}(f_1 \,\&\, \ldots \,\&\, f_n \,\&\, q_1 \,\&\, \ldots \,\&\, q_p) \equiv \mathtt{G}(\mathtt{F}(f_1 \,\&\, \ldots \,\&\, f_n) \,\&\, q_1 \,\&\, \ldots \,\&\, q_p)$$

$$\mathtt{G}(f_1 \,\&\, \ldots \,\&\, f_n \,\&\, e_1 \,\&\, \ldots \,\&\, e_m \,\&\, \mathtt{G}(e_{m+1}) \,\&\, \ldots \,\&\, \mathtt{G}(e_p)) \overset{\blacktriangle}{\equiv} \mathtt{G}(f_1 \,\&\, \ldots \,\&\, f_n) \,\&\, \mathtt{G}(e_1 \,\&\, \ldots \,\&\, e_p)$$

$$\mathtt{G}(f_1 \,\&\, \ldots \,\&\, f_n \,\&\, \mathtt{G}(g_1) \,\&\, \ldots \,\&\, \mathtt{G}(g_m)) \equiv \mathtt{G}(f_1 \,\&\, \ldots \,\&\, f_n \,\&\, g_1 \,\&\, \ldots \,\&\, g_m)$$

$$\mathtt{F}(f_1 \mid \ldots \mid f_n \mid u_1 \mid \ldots \mid u_m \mid \mathtt{F}(u_{m+1}) \mid \ldots \mid \mathtt{F}(u_p)) \overset{\blacktriangle}{\equiv} \mathtt{F}(f_1 \mid \ldots \mid f_n) \mid \mathtt{F}(u_1 \mid \ldots \mid u_p)$$

$$\mathtt{F}(f_1 \mid \ldots \mid f_n \mid \mathtt{F}(g_1) \mid \ldots \mid \mathtt{G}(g_m)) \equiv \mathtt{F}(f_1 \mid \ldots \mid f_n \mid g_1 \mid \ldots \mid g_m)$$

$$\mathtt{G}(f_1) \,\&\, \ldots \,\&\, \mathtt{G}(f_n) \,\&\, \mathtt{G}(e_1) \,\&\, \ldots \,\&\, \mathtt{G}(e_p) \overset{\blacktriangle}{\equiv} \mathtt{G}(f_1 \,\&\, \ldots \,\&\, f_n) \,\&\, \mathtt{G}(e_1 \,\&\, \ldots \,\&\, e_p)$$

$$\mathtt{F}(f_1) \mid \ldots \mid \mathtt{F}(f_n) \mid \mathtt{F}(u_1) \mid \ldots \mid \mathtt{F}(u_p) \overset{\blacktriangle}{\equiv} \mathtt{F}(f_1 \mid \ldots \mid f_n) \mid \mathtt{F}(u_1 \mid \ldots \mid u_p)$$

Finally the following rule is applied only when no other terms are present in the OR arguments:

$$\mathtt{F}(f_1) \mid \ldots \mid \mathtt{F}(f_n) \mid q_1 \mid \ldots \mid q_p \overset{\triangledown}{\equiv} \mathtt{F}(f_1 \mid \ldots \mid f_n \mid q_1 \mid \ldots q_p)$$

5.4.3. Simplifications Based on Implications

The following rewriting rules are performed only when we can prove that some subformula f implies another subformula g. Showing such implication can be done in two ways:

Syntactic Implication Checks were initially proposed by Somenzi and Bloem [18]. This detection is enabled by the '`tl_simplifier_options::synt_impl`' option. This is a cheap way to detect implications, but it may miss some. The rules we implement are described in Appendix B.

Language Containment Checks were initially proposed by Tauriainen [19]. This detection is enabled by the '`tl_simplifier_options::containment_checks`' option.

In the following rewritings rules, $f \Rightarrow g$ means that g was proved to be implied by f using either of the above two methods. Additionally, implications denoted by $f \overset{+}{\Rightarrow} g$ are only checked if the '`tl_simplifier_options::containment_checks_stronger`' option is set (otherwise the rewriting rule is not applied). We write $f \rightleftharpoons g$ iff $f \Rightarrow g$ and $g \Rightarrow f$.

As in the previous section, formulas e and u represent respectively pure eventualities and purely universal formulas.

Finally $|f|_b$ denote the length of f were all Boolean subformulas are counted as one.

if $f \Rightarrow \texttt{!}\,g$	then	$f \texttt{ | } g \equiv \texttt{1}$
if $f \Rightarrow \texttt{!}\,g$	then	$f \texttt{\&} g \equiv \texttt{0}$
if $(f \rightleftharpoons g) \wedge (\lvert f\rvert_b < \lvert g\rvert_b)$	then	$f \texttt{ | } g \equiv f$
if $f \Rightarrow g$	then	$f \texttt{ | } g \equiv g$
if $(f \rightleftharpoons g) \wedge (\lvert g\rvert_b < \lvert f\rvert_b)$	then	$f \texttt{\&} g \equiv g$
if $f \Rightarrow g$	then	$f \texttt{\&} g \equiv f$
if $f \Rightarrow g$	then	$f \texttt{<->} g \equiv g \texttt{->} f$
if $f \Rightarrow g$	then	$f \texttt{->} g \equiv \texttt{1}$
if $(\texttt{!}\,f) \Rightarrow g$	then	$f \texttt{xor} g \equiv g \texttt{->} \texttt{!}\,f$
if $f \Rightarrow \texttt{!}\,g$	then	$f \texttt{xor} g \equiv (\texttt{!}\,g) \texttt{->} f$
if $(f \rightleftharpoons g) \wedge (\lvert f\rvert_b < \lvert g\rvert_b)$	then	$f \texttt{U} g \equiv f$
if $f \Rightarrow g$	then	$f \texttt{U} g \equiv g$
if $(f \texttt{U} g) \overset{+}{\Rightarrow} g$	then	$f \texttt{U} g \equiv g$
if $(\texttt{!}\,f) \Rightarrow g$	then	$f \texttt{U} g \equiv \texttt{F}\,g$
if $g \Rightarrow e$	then	$e \texttt{U} g \equiv \texttt{F}\,g$
if $f \Rightarrow g$	then	$f \texttt{U} (g \texttt{U} h) \equiv g \texttt{U} h$
if $f \Rightarrow g$	then	$f \texttt{U} (g \texttt{W} h) \equiv g \texttt{W} h$
if $g \Rightarrow f$	then	$f \texttt{U} (g \texttt{U} h) \equiv f \texttt{U} h$
if $f \Rightarrow h$	then	$f \texttt{U} (g \texttt{R} (h \texttt{U} k)) \equiv g \texttt{R} (h \texttt{U} k)$
if $f \Rightarrow h$	then	$f \texttt{U} (g \texttt{R} (h \texttt{W} k)) \equiv g \texttt{R} (h \texttt{W} k)$
if $f \Rightarrow h$	then	$f \texttt{U} (g \texttt{M} (h \texttt{U} k)) \equiv g \texttt{M} (h \texttt{U} k)$
if $f \Rightarrow h$	then	$f \texttt{U} (g \texttt{M} (h \texttt{W} k)) \equiv g \texttt{M} (h \texttt{W} k)$
if $f \Rightarrow h$	then	$(f \texttt{U} g) \texttt{U} h \equiv g \texttt{U} h$
if $f \Rightarrow h$	then	$(f \texttt{W} g) \texttt{U} h \equiv g \texttt{U} h$
if $g \Rightarrow h$	then	$(f \texttt{U} g) \texttt{U} h \equiv (f \texttt{U} g) \texttt{ | } h$
if $g \Rightarrow h$	then	$(f \texttt{ | } g) \texttt{U} h \equiv f \texttt{U} h$
if $(\texttt{!}\,f) \Rightarrow g$	then	$f \texttt{W} g \equiv \texttt{1}$
if $(f \rightleftharpoons g) \wedge (\lvert f\rvert_b < \lvert g\rvert_b)$	then	$f \texttt{W} g \equiv f$
if $f \Rightarrow g$	then	$f \texttt{W} g \equiv g$
if $(f \texttt{W} g) \overset{+}{\Rightarrow} g$	then	$f \texttt{W} g \equiv g$
if $f \Rightarrow g$	then	$f \texttt{W} (g \texttt{W} h) \equiv g \texttt{W} h$
if $g \Rightarrow f$	then	$f \texttt{W} (g \texttt{U} h) \equiv f \texttt{W} h$
if $g \Rightarrow f$	then	$f \texttt{W} (g \texttt{W} h) \equiv f \texttt{W} h$

$$
\begin{array}{lll}
\text{if } f \Rightarrow h & \text{then} & (f\,\mathtt{U}\,g)\,\mathtt{W}\,h \equiv g\,\mathtt{W}\,h \\
\text{if } f \Rightarrow h & \text{then} & (f\,\mathtt{W}\,g)\,\mathtt{W}\,h \equiv g\,\mathtt{W}\,h \\
\text{if } g \Rightarrow h & \text{then} & (f\,\mathtt{W}\,g)\,\mathtt{W}\,h \equiv (f\,\mathtt{W}\,g)\,\mathtt{|}\,h \\
\text{if } g \Rightarrow h & \text{then} & (f\,\mathtt{U}\,g)\,\mathtt{W}\,h \equiv (f\,\mathtt{U}\,g)\,\mathtt{|}\,h \\
\text{if } g \Rightarrow h & \text{then} & (f\,\mathtt{|}\,g)\,\mathtt{W}\,h \equiv f\,\mathtt{W}\,h \\
\text{if } (f \rightleftharpoons g) \wedge (|f|_b < |g|_b) & \text{then} & f\,\mathtt{R}\,g \equiv f \\
\text{if } g \Rightarrow f & \text{then} & f\,\mathtt{R}\,g \equiv g \\
\text{if } g \Rightarrow \mathtt{!}\,f & \text{then} & f\,\mathtt{R}\,g \equiv \mathtt{G}\,g \\
\text{if } u \Rightarrow g & \text{then} & u\,\mathtt{R}\,g \equiv \mathtt{G}\,g \\
\text{if } g \Rightarrow f & \text{then} & f\,\mathtt{R}\,(g\,\mathtt{R}\,h) \equiv g\,\mathtt{R}\,h \\
\text{if } g \Rightarrow f & \text{then} & f\,\mathtt{R}\,(g\,\mathtt{M}\,h) \equiv g\,\mathtt{M}\,h \\
\text{if } f \Rightarrow g & \text{then} & f\,\mathtt{R}\,(g\,\mathtt{R}\,h) \equiv f\,\mathtt{R}\,h \\
\text{if } h \Rightarrow f & \text{then} & (f\,\mathtt{R}\,g)\,\mathtt{R}\,h \equiv g\,\mathtt{R}\,h \\
\text{if } h \Rightarrow f & \text{then} & (f\,\mathtt{M}\,g)\,\mathtt{R}\,h \equiv g\,\mathtt{R}\,h \\
\text{if } g \Rightarrow h & \text{then} & (f\,\mathtt{R}\,g)\,\mathtt{R}\,h \equiv (f\,\mathtt{\&}\,g)\,\mathtt{R}\,h \\
\text{if } g \Rightarrow h & \text{then} & (f\,\mathtt{M}\,g)\,\mathtt{R}\,h \equiv (f\,\mathtt{\&}\,g)\,\mathtt{R}\,h \\
\text{if } h \Rightarrow g & \text{then} & (f\,\mathtt{\&}\,g)\,\mathtt{R}\,h \equiv f\,\mathtt{R}\,h \\
\text{if } (f \rightleftharpoons g) \wedge (|f|_b < |g|_b) & \text{then} & f\,\mathtt{M}\,g \equiv f \\
\text{if } g \Rightarrow f & \text{then} & f\,\mathtt{M}\,g \equiv g \\
\text{if } g \Rightarrow \mathtt{!}\,f & \text{then} & f\,\mathtt{M}\,g \equiv \mathtt{0} \\
\text{if } g \Rightarrow f & \text{then} & f\,\mathtt{M}\,(g\,\mathtt{M}\,h) \equiv g\,\mathtt{M}\,h \\
\text{if } f \Rightarrow g & \text{then} & f\,\mathtt{M}\,(g\,\mathtt{M}\,h) \equiv f\,\mathtt{M}\,h \\
\text{if } f \Rightarrow g & \text{then} & f\,\mathtt{M}\,(g\,\mathtt{R}\,h) \equiv f\,\mathtt{M}\,h \\
\text{if } h \Rightarrow f & \text{then} & (f\,\mathtt{M}\,g)\,\mathtt{M}\,h \equiv g\,\mathtt{M}\,h \\
\text{if } h \Rightarrow f & \text{then} & (f\,\mathtt{R}\,g)\,\mathtt{M}\,h \equiv g\,\mathtt{M}\,h \\
\text{if } g \Rightarrow h & \text{then} & (f\,\mathtt{M}\,g)\,\mathtt{M}\,h \equiv (f\,\mathtt{\&}\,g)\,\mathtt{M}\,h \\
\text{if } h \Rightarrow g & \text{then} & (f\,\mathtt{\&}\,g)\,\mathtt{M}\,h \equiv f\,\mathtt{M}\,h
\end{array}
$$

Many of the above rules were collected from the literature [18, 19, 3] and sometimes generalized to support operators such as `M` and `W`.

The first six rules, about n-ary operators `&` and `|`, are implemented for n operands by testing each operand against all other. To prevent the complexity to escalate, this is only performed with up to 16 operands. That value can be changed in '`tl_simplifier_options::containment_max_ops`'.

The following rules mix implication-based checks with formulas that are pure eventualities (e) or that are purely universal (u).

$$
\begin{array}{lll}
\text{if } (\mathtt{!}\,f) \Rightarrow g & \text{then} & f\,\mathtt{U}\,(g\,\mathtt{\&}\,e) \equiv \mathtt{F}(g\,\mathtt{\&}\,e) \\
\text{if } f \Rightarrow \mathtt{!}\,g & \text{then} & f\,\mathtt{R}\,(g\,\mathtt{|}\,u) \equiv \mathtt{G}(g\,\mathtt{|}\,e) \\
\text{if } (\mathtt{!}\,f) \Rightarrow g & \text{then} & \mathtt{G}(f)\,\mathtt{|}\,\mathtt{F}(g\,\mathtt{\&}\,e) \equiv f\,\mathtt{W}\,(g\,\mathtt{\&}\,e) \\
\text{if } f \Rightarrow \mathtt{!}\,g & \text{then} & \mathtt{F}(f)\,\mathtt{\&}\,\mathtt{G}(g\,\mathtt{|}\,e) \equiv f\,\mathtt{M}\,(g\,\mathtt{|}\,e)
\end{array}
$$

5.5. Unit Propagation

This simplification is implemented by `unit_propagate()` in `spot/tl/unitprop.hh`. It is vaguely inspired by unit-propagation in SAT solvers, hence its name. The main ideas come from the following four rules:

- in a formula $\mathtt{G}(\alpha) \wedge \varphi$, any occurrence of α in φ can be replaced by $\mathtt{1}$,
- in a formula $\mathtt{F}(\alpha) \vee \varphi$, any occurrence of α in φ can be replaced by $\mathtt{0}$,
- in a formula $\alpha \wedge \varphi$, any occurrence of α in φ *that still happens in the current step* can be replaced by $\mathtt{1}$,
- in a formula $\alpha \vee \varphi$, any occurrence of α in φ *that still happens in the current step* can be replaced by $\mathtt{0}$

Here are examples of simplifications illustrating the above four rules:

$$\begin{aligned}
\mathtt{G}(a\,\mathtt{U}\,b)\,\&\,\mathtt{G}\,\mathtt{F}(c\,\mathtt{xor}\,(a\,\mathtt{U}\,b)) &\equiv \mathtt{G}(a\,\mathtt{U}\,b)\,\&\,\mathtt{G}\,\mathtt{F}(\,!\,c)\\
\mathtt{F}(a)\,\mid\,(aUb) &\equiv \mathtt{F}(a)\,\mid\,b\\
a\,\&\,(a\,\mid\,b) &\equiv a\\
(a\,\mathtt{U}\,b)\,\mid\,b &\equiv a\,\mathtt{U}\,b
\end{aligned}$$

To understand the fourth example, consider that $(a\,\mathtt{U}\,b)$ is equivalent to $b \mid (a\,\&\,\mathtt{X}(a\,\mathtt{U}\,b))$ so it can contribute to the fourth rule with $\alpha = (a\,\mathtt{U}\,b)$ or with $\alpha = b$.

Data structures. The algorithm maintains four multisets of subformulas (with integer multiplicities) during a single top-down traversal:

$$\mathcal{T}_G, \mathcal{T}_L, \mathcal{F}_G, \mathcal{F}_L : \text{Formula} \to \mathbb{N}$$

$\mathcal{T}_G$ and $\mathcal{T}_L$ store subformulas known to be true (respectively globally and locally); $\mathcal{F}_G$ and $\mathcal{F}_L$ store formulas known to be false. A formula φ is *known true* if $\mathcal{T}_G(\varphi) > 0$ or $\mathcal{T}_L(\varphi) > 0$, and *known false* if $\mathcal{F}_G(\varphi) > 0$ or $\mathcal{F}_L(\varphi) > 0$. At each node, if the formula itself is a known fact it is replaced by the corresponding Boolean constant ($\mathtt{1}$ or $\mathtt{0}$).

Fact extraction. Two mutually recursive functions $T(\varphi, \ell)$ and $F(\varphi, \ell)$, with $\ell \in \{L, G\}$, return multisets of *tagged facts* (ψ, S) where $S \in \{\mathcal{T}_G, \mathcal{T}_L, \mathcal{F}_G, \mathcal{F}_L\}$ is the destination multiset:

$$\begin{aligned}
T(\mathtt{1}, \ell) &= \varnothing & F(\mathtt{1}, \ell) &= \varnothing\\
T(\mathtt{0}, \ell) &= \varnothing & F(\mathtt{0}, \ell) &= \varnothing\\
T(a, \ell) &= \{(a, \mathcal{T}_\ell)\} \quad \text{(for atomic } a\text{)} & F(a, \ell) &= \{(a, \mathcal{F}_\ell)\}\\
T(\,!\,\psi, \ell) &= \{(\,!\,\psi, \mathcal{T}_\ell)\} \cup F(\psi, \ell) & F(\,!\,\psi, \ell) &= \{(\,!\,\psi, \mathcal{F}_\ell)\} \cup T(\psi, \ell)\\
T(\psi_1\,\&\cdots\&\,\psi_n, \ell) &= \{(\psi_1\,\&\cdots\&\,\psi_n, \mathcal{T}_\ell)\} \cup \bigcup_i T(\psi_i, \ell) & F(\psi_1\,\mid\cdots\mid\,\psi_n, \ell) &= \{(\psi_1\,\mid\cdots\mid\,\psi_n, \mathcal{F}_\ell)\} \cup \bigcup_i F(\psi_i, \ell)\\
T(\mathtt{G}\,\psi, \ell) &= \{(\mathtt{G}\,\psi, \mathcal{T}_\ell)\} \cup T(\psi, G) & F(\mathtt{F}\,\psi, \ell) &= \{(\mathtt{F}\,\psi, \mathcal{F}_\ell)\} \cup F(\psi, G)\\
T(\alpha\,\mathtt{R}\,\beta, \ell) &= \{(\alpha\,\mathtt{R}\,\beta, \mathcal{T}_\ell)\} \cup T(\beta, \ell) & F(\alpha\,\mathtt{U}\,\beta, \ell) &= \{(\alpha\,\mathtt{U}\,\beta, \mathcal{F}_\ell)\} \cup F(\beta, \ell)\\
T(\alpha\,\mathtt{M}\,\beta, \ell) &= \{(\alpha\,\mathtt{M}\,\beta, \mathcal{T}_\ell)\} \cup T(\beta, \ell) & F(\alpha\,\mathtt{W}\,\beta, \ell) &= \{(\alpha\,\mathtt{W}\,\beta, \mathcal{F}_\ell)\} \cup F(\beta, \ell)\\
T(\varphi, \ell) &= \{(\varphi, \mathcal{T}_\ell)\} \quad \text{(otherwise)} & F(\varphi, \ell) &= \{(\varphi, \mathcal{F}_\ell)\} \quad \text{(otherwise)}
\end{aligned}$$

A tagged fact (ψ, S) is applied to the context by incrementing $S(\psi)$ (and removed by decrementing); when the multiplicity reaches 0 the fact is erased.

Processing & and |. For $\varphi_1\,\&\cdots\&\,\varphi_n$:

1. Compute $C_i = T(\varphi_i, L)$ for each child. Add all C_i to the context so that siblings see each other's facts.
2. For each i from 1 to n:
   - Remove C_i from the context (to prevent φ_i from simplifying itself).
   - Recursively simplify φ_i to ψ_i.
   - If $\psi_i = \mathtt{1}$, drop C_i permanently (do not re-add).
   - If $\psi_i = \mathtt{0}$, return `0` (short-circuit).
   - Otherwise keep ψ_i and re-add C_i.
3. Remove any remaining contributions from the context.

For `|`, the dual: use $F(\varphi_i, L)$, drop on `0`, short-circuit on `1`. If no children change, the original is returned. If all are dropped, the result is `1` (`&`) or `0` (`|`).

Implications. α `->` β uses a two-step sequential simplification:

1. Collect $F(\beta, L)$; apply to context. Simplify α; remove the facts.
2. Collect $T(\alpha', L)$ where α' is the result of step 1. Apply to context; simplify β; remove.

Temporal operators. Subformulas whose top-level operator is `X`, `F`, `G`, `U`, `W`, `R`, or `M`, are simplified with the following rules:

$$
\begin{aligned}
\mathtt{X}\,\varphi &\equiv \mathtt{1} \text{ if } \varphi \in \mathcal{T}_G & \mathtt{X}\,\varphi &\equiv \mathtt{0} \text{ if } \varphi \in \mathcal{F}_G \\
\mathtt{F}\,\varphi &\equiv \mathtt{1} \text{ if } \varphi \in \mathcal{T}_G \cup \mathcal{T}_L & \mathtt{F}\,\varphi &\equiv \mathtt{0} \text{ if } \varphi \in \mathcal{F}_G \\
\mathtt{G}\,\varphi &\equiv \mathtt{0} \text{ if } \varphi \in \mathcal{F}_G \cup \mathcal{F}_L & \mathtt{G}\,\varphi &\equiv \mathtt{1} \text{ if } \varphi \in \mathcal{T}_G \\
\alpha\,\mathtt{U}\,\beta &\equiv \mathtt{1} \text{ if } \beta \in \mathcal{T}_G \cup \mathcal{T}_L & \alpha\,\mathtt{U}\,\beta &\equiv \mathtt{0} \text{ if } \beta \in \mathcal{F}_G \\
\alpha\,\mathtt{W}\,\beta &\equiv \mathtt{1} \text{ if } \beta \in \mathcal{T}_G \cup \mathcal{T}_L & \alpha\,\mathtt{W}\,\beta &\equiv \mathtt{1} \text{ if } \alpha \in \mathcal{T}_G \\
\alpha\,\mathtt{R}\,\beta &\equiv \mathtt{0} \text{ if } \beta \in \mathcal{F}_G \cup \mathcal{F}_L & \alpha\,\mathtt{R}\,\beta &\equiv \mathtt{1} \text{ if } \beta \in \mathcal{T}_G \\
\alpha\,\mathtt{M}\,\beta &\equiv \mathtt{0} \text{ if } \beta \in \mathcal{F}_G \cup \mathcal{F}_L & \alpha\,\mathtt{M}\,\beta &\equiv \mathtt{0} \text{ if } \alpha \in \mathcal{F}_G
\end{aligned}
$$

The rules $\mathtt{F}\,\varphi \equiv \mathtt{0}$ when $\varphi \in \mathcal{F}_G$ and $\mathtt{G}\,\varphi \equiv \mathtt{1}$ when $\varphi \in \mathcal{T}_G$ are not implemented as explicit shortcuts: they occur automatically because the top-level check in the recursive simplification replaces any formula that belongs to $\mathcal{T}_G \cup \mathcal{T}_L$ (resp. $\mathcal{F}_G \cup \mathcal{F}_L$) by `1` (resp. `0`), and the formula constructors handle $\mathtt{F}\,\mathtt{0} = \mathtt{0}$ and $\mathtt{G}\,\mathtt{1} = \mathtt{1}$. Similarly the `X` rules are not implemented explicitly; they also follow from the top-level check plus the constructors $\mathtt{X}\,\mathtt{1} = \mathtt{1}$, $\mathtt{X}\,\mathtt{0} = \mathtt{0}$.

If the above rules do not apply, their operands are simplified recursively, but with local sets $\mathcal{T}_L$ and $\mathcal{F}_L$ cleared (saved before the recursive call and restored after). Global facts $\mathcal{T}_G$ and $\mathcal{F}_G$ persist.

For `xor` and `<->`, local facts are also cleared before processing the operands, since these operators mix polarities.

A. Defining LTL with only one of U, W, R, or M

The operators 'F', 'G', 'U', 'R', 'M', and 'W' can all be defined using only Boolean operators, 'X', and one of 'U', 'W', 'R', or 'M'. This property is usually used to simplify proofs. These equivalences can also help to understand the semantics of section 2.4.1 if you are only familiar with some of the operators.

Equivalences using U:

$$
\begin{aligned}
\mathtt{F}\,f &\equiv \mathtt{1}\,\mathtt{U}\,f \\
\mathtt{G}\,f &\equiv \mathtt{!}\,\mathtt{F}\,\mathtt{!}\,f \equiv \mathtt{!}(\mathtt{1}\,\mathtt{U}\,\mathtt{!}\,f) \\
f\,\mathtt{W}\,g &\equiv (f\,\mathtt{U}\,g) \mid \mathtt{G}\,f \equiv (f\,\mathtt{U}\,g) \mid \mathtt{!}(\mathtt{1}\,\mathtt{U}\,\mathtt{!}\,f) \\
&\equiv f\,\mathtt{U}\,(g \mid \mathtt{G}\,f) \equiv f\,\mathtt{U}\,(g \mid \mathtt{!}(\mathtt{1}\,\mathtt{U}\,\mathtt{!}\,f)) \\
f\,\mathtt{M}\,g &\equiv g\,\mathtt{U}\,(f \,\&\, g) \\
f\,\mathtt{R}\,g &\equiv g\,\mathtt{W}\,(f \,\&\, g) \equiv (g\,\mathtt{U}\,(f \,\&\, g)) \mid \mathtt{!}(\mathtt{1}\,\mathtt{U}\,\mathtt{!}\,g) \\
&\equiv g\,\mathtt{U}\,((f \,\&\, g) \mid \mathtt{!}(\mathtt{1}\,\mathtt{U}\,\mathtt{!}\,g))
\end{aligned}
$$

Equivalences using W:

$$
\begin{aligned}
\mathtt{F}\,f &\equiv \mathtt{!}\,\mathtt{G}\,\mathtt{!}\,f \equiv \mathtt{!}((\mathtt{!}\,f)\,\mathtt{W}\,\mathtt{0}) \\
\mathtt{G}\,f &\equiv \mathtt{0}\,\mathtt{R}\,f \equiv f\,\mathtt{W}\,\mathtt{0} \\
f\,\mathtt{U}\,g &\equiv (f\,\mathtt{W}\,g) \,\&\, (\mathtt{F}\,g) \equiv (f\,\mathtt{W}\,g) \,\&\, \mathtt{!}((\mathtt{!}\,g)\,\mathtt{W}\,\mathtt{0}) \\
f\,\mathtt{M}\,g &\equiv (g\,\mathtt{W}\,(f \,\&\, g)) \,\&\, \mathtt{F}(f \,\&\, g) \equiv (g\,\mathtt{W}\,(f \,\&\, g)) \,\&\, \mathtt{!}((\mathtt{!}(f \,\&\, g))\,\mathtt{W}\,\mathtt{0}) \\
f\,\mathtt{R}\,g &\equiv g\,\mathtt{W}\,(f \,\&\, g)
\end{aligned}
$$

Equivalences using R:

$$
\begin{aligned}
\mathtt{F}\,f &\equiv \mathtt{!}\,\mathtt{G}\,\mathtt{!}\,f \equiv \mathtt{!}(\mathtt{0}\,\mathtt{R}\,\mathtt{!}\,f) \\
\mathtt{G}\,f &\equiv \mathtt{0}\,\mathtt{R}\,f \\
f\,\mathtt{U}\,g &\equiv (((\mathtt{X}\,g)\,\mathtt{R}\,f) \,\&\, \mathtt{F}\,g) \mid g \equiv (((\mathtt{X}\,g)\,\mathtt{R}\,f) \,\&\, (\mathtt{!}(\mathtt{0}\,\mathtt{R}\,\mathtt{!}\,g))) \mid g \\
f\,\mathtt{W}\,g &\equiv g\,\mathtt{R}\,(f \mid g) \\
&\equiv ((\mathtt{X}\,g)\,\mathtt{R}\,f) \mid g \\
f\,\mathtt{M}\,g &\equiv (f\,\mathtt{R}\,g) \,\&\, \mathtt{F}\,f \equiv (f\,\mathtt{R}\,g) \,\&\, \mathtt{!}(\mathtt{0}\,\mathtt{R}\,\mathtt{!}\,f) \\
&\equiv f\,\mathtt{R}\,(g \,\&\, \mathtt{F}\,g) \equiv f\,\mathtt{R}\,(g \,\&\, \mathtt{!}(\mathtt{0}\,\mathtt{R}\,\mathtt{!}\,f))
\end{aligned}
$$

Equivalences using M:

$$
\begin{aligned}
\mathtt{F}\,f &\equiv f\,\mathtt{M}\,\mathtt{1} \\
\mathtt{G}\,f &\equiv \mathtt{!}\,\mathtt{F}\,\mathtt{!}\,f \equiv \mathtt{!}((\mathtt{!}\,f)\,\mathtt{M}\,\mathtt{1}) \\
f\,\mathtt{U}\,g &\equiv g\,\mathtt{M}\,(f \mid g) \\
&\equiv ((\mathtt{X}\,g)\,\mathtt{M}\,f) \mid g \\
f\,\mathtt{W}\,g &\equiv (f\,\mathtt{U}\,g) \mid \mathtt{G}\,f \equiv ((\mathtt{X}\,g)\,\mathtt{M}\,f) \mid g \mid \mathtt{!}((\mathtt{!}\,f)\,\mathtt{M}\,\mathtt{1}) \\
f\,\mathtt{R}\,g &\equiv (f\,\mathtt{M}\,g) \mid \mathtt{G}\,g \equiv (f\,\mathtt{M}\,g) \mid \mathtt{!}((\mathtt{!}\,g)\,\mathtt{M}\,\mathtt{1})
\end{aligned}
$$

These equivalences make it possible to build artificially complex formulas. For instance by applying the above rules to successively rewrite $\mathtt{U} \to \mathtt{M} \to \mathtt{R} \to \mathtt{U}$ we get

$$
\begin{aligned}
f\,\mathtt{U}\,g &\equiv ((\mathtt{X}\,g)\,\mathtt{M}\,f)\,\mathtt{|}\,g \\
&\equiv (((\mathtt{X}\,g)\,\mathtt{R}\,f)\,\mathtt{\&}\,\mathtt{!}(\mathtt{0}\,\mathtt{R}\,\mathtt{!}\,\mathtt{X}\,g))\,\mathtt{|}\,g \\
&\equiv (((f\,\mathtt{U}\,(\mathtt{X}\,g\,\mathtt{\&}\,f))\,\mathtt{|}\,\mathtt{!}(\mathtt{1}\,\mathtt{U}\,\mathtt{!}\,f))\,\mathtt{\&}\,\mathtt{!}(\underbrace{((\mathtt{!}\,\mathtt{X}\,g)\,\mathtt{U}\,(\mathtt{0}\,\mathtt{\&}\,\mathtt{!}\,\mathtt{X}\,g))}_{\text{trivially false}}\,\mathtt{|}\,\mathtt{!}(\mathtt{1}\,\mathtt{U}\,\mathtt{!}\,\mathtt{!}\,\mathtt{X}\,g)))\,\mathtt{|}\,g \\
&\equiv (((f\,\mathtt{U}\,(\mathtt{X}\,g\,\mathtt{\&}\,f))\,\mathtt{|}\,\mathtt{!}(\mathtt{1}\,\mathtt{U}\,\mathtt{!}\,f))\,\mathtt{\&}\,(\mathtt{1}\,\mathtt{U}\,\mathtt{X}\,g))\,\mathtt{|}\,g
\end{aligned}
$$

Spot is able to simplify most of the above equivalences, but it starts to have trouble when the $\mathtt{X}$ operator is involved. For instance $(f\,\mathtt{W}\,g)\,\mathtt{\&}\,\mathtt{F}\,g \equiv f\,\mathtt{U}\,g$ is one of the rewriting rules from 5.4.1. But the formula $(f\,\mathtt{W}\,\mathtt{X}\,g)\,\mathtt{\&}\,\mathtt{F}\,\mathtt{X}\,g$, which looks like it should be reduced similarly to $f\,\mathtt{U}\,\mathtt{X}\,g$, will be rewritten instead to $(f\,\mathtt{W}\,\mathtt{X}\,g)\,\mathtt{\&}\,\mathtt{X}\,\mathtt{F}\,g$, because $\mathtt{X}\,\mathtt{F}\,g \equiv \mathtt{F}\,\mathtt{X}\,g$ is another rule that gets applied first during the bottom-up rewriting.

B. Syntactic Implications

Syntactic implications are used for the rewriting of Section 5.4.3. The rules presented bellow extend those first presented by Somenzi and Bloem [18].

A few words about implication first. For two PSL formulas f and g, we say that $f \implies g$ if $\forall\sigma, (\sigma \models f) \implies (\sigma \models g)$. For two SERE f and g, we say that $f \implies g$ if $\forall\pi, (\pi \Vdash f) \implies (\pi \Vdash g)$.

The recursive rules for syntactic implication are rules are described in table B.1, in which $\Rightarrow$ denotes the syntactic implication, f, f_1, f_2, g, g_1 and g_2 denote any PSL formula in negative normal form, and f_U and g_E denote a purely universal formula and a pure eventuality.

The form on the left of the table syntactically implies the form on the top of the table if the condition in the corresponding cell holds.

Note that a given formula may match several forms, and require multiple recursive tests. To limit the number of recursive calls, some rules have been removed when they are already implied by other rules.

For instance it one would legitimately expect the rule "$\mathtt{F}\, f \Rightarrow \mathtt{F}\, g$ *if* $f \Rightarrow g$" to appear in the table. However in that case $\mathtt{F}\, g$ is pure eventuality, so we can reach the same conclusion by chaining two rules: "$\mathtt{F}\, f \Rightarrow \underbrace{\mathtt{F}\, g}_{g_E}$ *if* $f \Rightarrow \underbrace{\mathtt{F}\, g}_{g_E}$" and then "$f \Rightarrow \mathtt{F}\, g$ *if* $f \Rightarrow g$".

The rules from table B.1 should be completed by the following cases, where f_b and g_b denote Boolean formulas:

we have	if
$f \Rightarrow 1$	always
$0 \Rightarrow g$	always
$f_b \Rightarrow g_b$	$\mathrm{BDD}(f_b) \wedge \mathrm{BDD}(g_b) = \mathrm{BDD}(f_b)$

$\Rightarrow$	g	g_E	$\mathtt{X}\,g$	$g_1 \,\mathtt{U}\, g_2$	$g_1 \,\mathtt{W}\, g_2$	$g_1 \,\mathtt{R}\, g_2$	$g_1 \,\mathtt{M}\, g_2$	$\mathtt{F}\,g$	$\mathtt{G}\,g$	$g_1 \mid g_2$	$g_1 \,\&\, g_2$
f	$f = g$			$f \Rightarrow g_2$	$f \Rightarrow g_2$	$f \Rightarrow g_1$ $\wedge\, f \Rightarrow g_2$	$f \Rightarrow g_1$ $\wedge\, f \Rightarrow g_2$	$f \Rightarrow g$		$f \Rightarrow g_1$ $\vee\, f \Rightarrow g_2$	$f \Rightarrow g_1$ $\wedge\, f \Rightarrow g_2$
f_U			$f_U \Rightarrow g$						$f_U \Rightarrow g$		
$\mathtt{X}\,f$		$f \Rightarrow g_E$	$f \Rightarrow g$								
$f_1 \,\mathtt{U}\, f_2$	$f_1 \Rightarrow g$ $\wedge\, f_2 \Rightarrow g$			$f_1 \Rightarrow g_1$ $\wedge\, f_2 \Rightarrow g_2$	$f_1 \Rightarrow g_1$ $\wedge\, f_2 \Rightarrow g_2$	$f_1 \Rightarrow g_2$ $\wedge\, f_2 \Rightarrow g_1$ $\wedge\, f_2 \Rightarrow g_2$	$f_1 \Rightarrow g_2$ $\wedge\, f_2 \Rightarrow g_1$ $\wedge\, f_2 \Rightarrow g_2$	$f_2 \Rightarrow g$			
$f_1 \,\mathtt{W}\, f_2$	$f_1 \Rightarrow g$ $\wedge\, f_2 \Rightarrow g$			$f_1 \Rightarrow g_2$ $\wedge\, f_2 \Rightarrow g_2$	$f_1 \Rightarrow g_1$ $\wedge\, f_2 \Rightarrow g_2$	$f_1 \Rightarrow g_2$ $\wedge\, f_2 \Rightarrow g_1$ $\wedge\, f_2 \Rightarrow g_2$		$f_1 \Rightarrow g$ $\wedge\, f_2 \Rightarrow g$			
$f_1 \,\mathtt{R}\, f_2$	$f_2 \Rightarrow g$				$f_1 \Rightarrow g_2$ $\wedge\, f_2 \Rightarrow g_1$	$f_1 \Rightarrow g_1$ $\wedge\, f_2 \Rightarrow g_2$	$f_2 \Rightarrow g_1$ $\wedge\, f_2 \Rightarrow g_2$	$f_2 \Rightarrow g$			
$f_1 \,\mathtt{M}\, f_2$	$f_2 \Rightarrow g$			$f_1 \Rightarrow g_2$ $\wedge\, f_2 \Rightarrow g_1$	$f_1 \Rightarrow g_2$ $\wedge\, f_2 \Rightarrow g_1$	$f_1 \Rightarrow g_1$ $\wedge\, f_2 \Rightarrow g_2$	$f_1 \Rightarrow g_1$ $\wedge\, f_2 \Rightarrow g_2$	$f_1 \Rightarrow g$ $\vee\, f_2 \Rightarrow g$			
$\mathtt{F}\,f$		$f \Rightarrow g_E$									
$\mathtt{G}\,f$	$f \Rightarrow g$			$f \Rightarrow g_2$	$f \Rightarrow g_1$ $\vee\, f \Rightarrow g_2$	$f \Rightarrow g_2$	$f \Rightarrow g_1$ $\wedge\, f \Rightarrow g_2$				
$f_1 \mid f_2$	$f_1 \Rightarrow g$ $\wedge\, f_2 \Rightarrow g$										
$f_1 \,\&\, f_2$	$f_1 \Rightarrow g$ $\vee\, f_2 \Rightarrow g$										

Table B.1.: Recursive rules for syntactic implication.

Bibliography

[1] *Property Specification Language Reference Manual v1.1*. Accellera, June 2004. URL `http://www.eda.org/vfv/`.

[2] *1800-2017 - IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language*. IEEE, February 2018. doi:10.1109/IEEESTD.2018.8299595.

[3] Tomáš Babiak, Mojmír Křetínský, Vojtěch Řehák, and Jan Strejček. LTL to Büchi automata translation: Fast and more deterministic. In *Proceedings of the 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'12)*, volume 7214 of *Lecture Notes in Computer Science*, pages 95–109. Springer, 2012. doi:10.1007/978-3-642-28756-5_8.

[4] Ilan Beer, Shoham Ben-David, Cindy Eisner, Dana Fisman, Anna Gringauze, and Yoav Rodeh. The temporal logic Sugar. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Proceedings of the 13th international conferance on Computer Aided Verification (CAV'01)*, volume 2102 of *Lecture Notes in Computer Science*, pages 363–367. Springer, July 2001. ISBN 978-3-540-42345-4. doi:10.1007/3-540-44585-4_33.

[5] Anne Brüggemann-Klein. Regular expressions into finite automata. *Theoretical Computer Science*, 120:87–98, 1996. doi:10.1007/BFb0023820.

[6] Ivana Černá and Radek Pelánek. Relating hierarchy of temporal properties to model checking. In Branislav Rovan and Peter Vojtáǎ, editors, *Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science (MFCS'03)*, volume 2747 of *Lecture Notes in Computer Science*, pages 318–327, Bratislava, Slovak Republic, August 2003. Springer-Verlag. doi:10.1007/978-3-540-45138-9_26.

[7] Edward Y. Chang, Zohar Manna, and Amir Pnueli. Characterization of temporal property classes. In *Proceedings of the 19th International Colloquium on Automata, Languages and Programming (ICALP'92)*, pages 474–486, London, UK, 1992. Springer-Verlag. doi:10.1007/3-540-55719-9_97.

[8] Alessandro Cimatti, Marco Roveri, and Stefano Tonetta. Symbolic compilation of PSL. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 27(10):1737–1750, 2008. doi:10.1109/TCAD.2008.2003303. URL `https://es.fbk.eu/people/tonetta/tests/tcad07/`.

[9] Christian Dax, Felix Klaedtke, and Stefan Leue. Specification languages for stutter-invariant regular properties. In *Proceedings of the 7th International Symposium on Automated Technology for Verification and Analysis (ATVA'09)*, volume 5799 of *Lecture Notes in Computer Science*, pages 244–254. Springer-Verlag, 2009. doi:10.1007/978-3-642-04761-9_19.

[10] Alexandre Duret-Lutz. LTL translation improvements in Spot. In *Proceedings of the 5th International Workshop on Verification and Evaluation of Computer and Communication Systems (VECoS'11)*, Electronic Workshops in Computing, Tunis, Tunisia, September 2011. British Computer Society. URL `http://ewic.bcs.org/category/15853`.

[11] Cindy Eisner and Dana Fisman. *A Practical Introduction to PSL*. Series on Integrated Circuits and Systems. Springer, 2006. doi:10.1007/978-0-387-36123-9.

[12] Cindy Eisner and Dana Fisman. Structural contradictions. In Hana Chockler and Alan Hu, editors, *Proceedings of the 4th International Haifa Verification Conference (HVC'2008)*, volume 5394 of *Lecture Notes in Computer Science*, pages 164–178. Springer, October 2009. ISBN 978-3-642-01701-8. doi:10.1007/978-3-642-01702-5_17.

[13] Javier Esparza, Rubén Rubio, and Salomon Sickert. Efficient normalization of linear temporal logic. *Journal of the ACM*, 71(2), April 2024. ISSN 0004-5411. doi:10.1145/3651152.

[14] Kousha Etessami and Gerard J. Holzmann. Optimizing Büchi automata. In C. Palamidessi, editor, *Proceedings of the 11th International Conference on Concurrency Theory (Concur'00)*, volume 1877 of *Lecture Notes in Computer Science*, pages 153–167, Pennsylvania, USA, 2000. Springer-Verlag. doi:10.1007/3-540-44618-4_13. Beware of a typo in the version from the proceedings: $f \mathtt{U} g$ is purely eventual if both operands are purely eventual. The revision of the paper available at `http://www.bell-labs.com/project/TMP/` is fixed. We fixed the bug in Spot in 2005, thanks to LBTT. See also `http://arxiv.org/abs/1011.4214v2` for a discussion about this problem.

[15] Swen Jacobs, Felix Klein, and Sebastian Schirmer. A high-level LTL synthesis format: TLSF v1.1. In *Proceedings of the Fifth Workshop on Synthesis (SYNT'16)*, volume 229 of *Electronic Proceedings in Theoretical Computer Science*, pages 112–132, 2016. doi:10.4204/EPTCS.229.10.

[16] Zohar Manna and Amir Pnueli. A hierarchy of temporal properties. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing (PODC'90)*, pages 377–410, New York, NY, USA, 1990. ACM. doi:10.1145/93385.93442.

[17] Klaus Schneider. Improving automata generation for linear temporal logic by considering the automaton hierarchy. In *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 2250 of *Lecture Notes in Artificial Intelligence*, pages 39–54, Havana, Cuba, 2001. Springer-Verlag. doi:10.1007/3-540-45653-8_3.

[18] Fabio Somenzi and Roderick Bloem. Efficient Büchi automata for LTL formulae. In *Proceedings of the 12th International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *Lecture Notes in Computer Science*, pages 247–263, Chicago, Illinois, USA, 2000. Springer-Verlag. doi:10.1007/10722167_21.

[19] Heikki Tauriainen. On translating linear temporal logic into alternating and nondeterministic automata. Research Report A83, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, December 2003. URL `http://www.tcs.hut.fi/Publications/A83.shtml`. Reprint of Licentiate's thesis.