Evaluating Pre-Processing Techniques for the Separated Normal Form for Temporal Logics*

Ullrich Hustadt[1], Cláudia Nalon[2], and Clare Dixon[1]

[1] Department of Computer Science, University of Liverpool
Liverpool, L69 3BX – United Kingdom
U.Hustadt, CLDixon@liverpool.ac.uk
[2] Department of Computer Science, University of Brasília
C.P. 4466 – CEP:70.910-090 – Brasília – DF – Brazil
nalon@unb.br

Abstract We consider the transformation of propositional linear time temporal logic formulae into a clause normal form, called *Separated Normal Form*, suitable for resolution calculi. In particular, we investigate the effect of applying various pre-processing techniques on characteristics of the normal form and determine the best combination of techniques on a large collection of benchmark formulae.

1 Introduction

Clause normal forms are the foundation of most resolution-based calculi and such calculi exist for a wide range of logics, including propositional, first-order, modal and temporal logics. Given a formula φ, the clause normal form of φ is typically computed using a combination of equivalence or satisfiability preserving rewrite steps, including simplification as a special case, and renaming, which replaces complex subformulae by new propositional variables and adds definitional clauses for the new propositional variables. Variations in the normal form can greatly influence the performance of resolution-based reasoning systems and transformation procedures that compute clause normal forms typically aim to produce fewer and/or shorter clauses for a given formula. For propositional and first-order logic the computation of such 'small' clause normal forms is well-studied [16,1,18]. However, the problem has been not been investigated to the same extent for non-classical logics, one exception being the work by Dixon and Nalon on prenexing versus anti-prenexing for modal logics [14].

In this paper we consider a clausal normal form for propositional linear time temporal logic (PLTL), called *Separated Normal Form* (SNF). This normal form was originally devised as the basis of a clausal resolution calculus and decision

* The second and third authors were partially supported by the EPSRC funded RAI Hub FAIR-SPACE (EP/R026092/1) and the EPSRC funded programme grant S4 (EP/N007565/1). The third author was also partially supported by the EPSRC funded RAI Hub RAIN (EP/R026084/1).

procedure for PLTL [6]. More recently, a decision procedure for PLTL using labelled superposition has also used SNF as a starting point [22]. There are several theorem provers for PLTL that use SNF as their input language, including **TRP** [9], **TRP++** [10,11] and LS4 [22,21].

In following we revisit the problem of computing the SNF of a given PLTL formula. In particular, we determine the effect of applying various pre-processing techniques during the computation.

The paper is organised as follows. We present the language of PLTL in Section 2. The Separated Normal Form is given in Section 3, where the techniques used for producing the normal form are also presented. Experimental evaluation is discussed in Section 4.

2 The Language of PLTL

We consider a particular variety of temporal logic, which is based on a linear, discrete model of time with finite past and infinite future [7,13]. This logic can be seen as a multi-modal language with two modalities, one to represent the 'next' moment in time, the other representing all future moments in time.

The temporal operators supplied in the language operate over a sequence of distinct 'moments' in time. In this version, only future-time operators are used. It is possible to include past-time operators in the definition of the logic, as in [4], but such operators add no extra expressive power [7].

Formulae are constructed from a denumerable set $P = \{p, q, p', q', p_1, q_1, \ldots\}$ of *propositional variables* and a set of connectives[3]. In addition to the standard propositional connectives $(\neg, \vee, \wedge, \rightarrow, \leftrightarrow)$ we use a set of temporal operators consisting of '$\Diamond$' (*sometime in the future*), '$\Box$' (*always in the future*), '$\bigcirc$' (*in the next moment in the future*), '$\mathcal{U}$' (*until*), and '$\mathcal{W}$' (*unless* or *weak until*). The *set of well-formed formulae* of PLTL, denoted by $\mathsf{WFF}_{\mathsf{PLTL}}$, is inductively defined as the smallest set satisfying:

- the propositional variables are in $\mathsf{WFF}_{\mathsf{PLTL}}$;
- $\top$ and $\bot$ are in $\mathsf{WFF}_{\mathsf{PLTL}}$;
- if φ and ψ are in $\mathsf{WFF}_{\mathsf{PLTL}}$, then so are $\neg\varphi$, $(\varphi \rightarrow \psi)$, $(\varphi \leftrightarrow \psi)$, $\Diamond\varphi$, $\Box\varphi$, $\bigcirc\varphi$, $(\varphi\,\mathcal{U}\,\psi)$, $(\varphi\,\mathcal{W}\,\psi)$;
- if $\varphi_1, \ldots, \varphi_n$, $n \geq 1$, are in $\mathsf{WFF}_{\mathsf{PLTL}}$, then so are $(\varphi_1 \wedge \cdots \wedge \varphi_n)$ and $(\varphi_1 \vee \cdots \vee \varphi_n)$.

A *literal* is a propositional variable or its negation. An *eventuality* is a formula of the form $\Diamond\varphi$, for a well-formed formula φ. An *elementary formula* is one of the logical constants $\top$, $\bot$, a propositional literal or a formula of the form $\bigcirc\varphi$, for an arbitrary formula φ. A *position* is a word over the natural numbers. For a formula φ, the set $\mathrm{pos}(\varphi)$ of positions of φ is defined as follows:

[3] We consider the connectives that reflect the input language of our tool `ltl2snf` and that occur in 'real-world' or benchmark formulae, instead of restricting ourselves to a minimal, expressively complete set of connectives.

- the empty word $\epsilon \in \mathrm{pos}(\varphi)$;
- if φ is of the form $\neg\varphi_1$, $\bigcirc\varphi_1$, $\Box\varphi_1$, or $\Diamond\varphi_1$ for some formula φ_1 and $\pi \in \mathrm{pos}(\varphi_1)$ then $1.\pi \in \mathrm{pos}(\varphi)$;
- if φ is of the form $(\varphi_1 \rightarrow \varphi_2)$, $(\varphi_1 \leftrightarrow \varphi_2)$, $(\varphi_1 \,\mathcal{U}\, \varphi_2)$, or $(\varphi_1 \,\mathcal{W}\, \varphi_1)$ and $\pi \in \mathrm{pos}(\varphi_1)$ for $i \in \{1, 2\}$, then $i.\pi \in \mathrm{pos}(\varphi)$; and
- if φ is of the form $(\varphi_1 \wedge \cdots \wedge \varphi_n)$ or $(\varphi_1 \vee \cdots \vee \varphi_n)$ and $\pi \in \mathrm{pos}(\varphi_i)$ for $i \in \{1, \ldots, n\}$, then $i.\pi \in \mathrm{pos}(\varphi)$.

For a formula φ and position $\pi \in \mathrm{pos}(\varphi)$, we define the *subformula* $\varphi|_\pi$ *of* φ *at position* π as follows:

- $\varphi|_\epsilon = \varphi$;
- if φ is of the form $\neg\varphi_1$, $\bigcirc\varphi_1$, $\Box\varphi_1$, or $\Diamond\varphi_1$ for some formula φ_1 and $\pi = 1.\tau$, then $\varphi|_\pi = \varphi_1|_\tau$;
- if φ is of the form $(\varphi_1 \rightarrow \varphi_2)$, $(\varphi_1 \leftrightarrow \varphi_2)$, $(\varphi_1 \,\mathcal{U}\, \varphi_2)$, or $(\varphi_1 \,\mathcal{W}\, \varphi_1)$ and $\pi = i.\tau$ for $i \in \{1, 2\}$, then $\varphi|_\pi = \varphi_i|_\tau$; and
- if φ is of the form $(\varphi_1 \wedge \cdots \wedge \varphi_n)$ or $(\varphi_1 \vee \cdots \vee \varphi_n)$ and $\pi = i.\tau$ for $i \in \{1, \ldots, n\}$, then $\varphi|_\pi = \varphi_i|_\tau$.

The polarity $\mathrm{pol}(\varphi, \pi)$ of a subformula occurring at position π in a formula φ is defined as follows:

- $\mathrm{pol}(\varphi, \epsilon) = 1$;
- if $\pi = \tau.1$ and $\varphi|_\tau$ is of the form $\bigcirc\varphi_1$, $\Box\varphi_1$, or $\Diamond\varphi_1$, then $\mathrm{pol}(\varphi, \pi) = \mathrm{pol}(\varphi, \tau)$;
- if $\pi = \tau.1$ and $\varphi|_\tau$ is of the form $\neg\varphi_1$ or $(\varphi_1 \rightarrow \varphi_2)$, then $\mathrm{pol}(\varphi, \pi) = -\mathrm{pol}(\varphi, \tau)$;
- if $\pi = \tau.2$ and $\varphi|_\tau$ is of the form $(\varphi_1 \rightarrow \varphi_2)$, then $\mathrm{pol}(\varphi, \pi) = \mathrm{pol}(\varphi, \tau)$;
- if $\pi = \tau.i$ with $i \in \{1, 2\}$ and $\varphi|_\tau$ is of the form $(\varphi_1 \,\mathcal{U}\, \varphi_2)$ or $(\varphi_1 \,\mathcal{W}\, \varphi_2)$ then $\mathrm{pol}(\varphi, \pi) = \mathrm{pol}(\varphi, \tau)$;
- if $\pi = \tau.i$ with $i \in \{1, \ldots, n\}$ and $\varphi|_\tau$ is of the form $(\varphi_1 \wedge \cdots \wedge \varphi_n)$ or $(\varphi_1 \wedge \cdots \wedge \varphi_n)$, then $\mathrm{pol}(\varphi, \pi) = \mathrm{pol}(\varphi, \tau)$;
- if $\pi = \tau.i$ with $i \in \{1, 2\}$ and $\varphi|_\tau$ is of the form $(\varphi_1 \leftrightarrow \varphi_2)$ then $\mathrm{pol}(\varphi, \pi) = 0$.

A formula φ at position π is of *positive polarity* (resp. *negative polarity*) if $\mathrm{pol}(\varphi, \pi) = 1$ (resp. $\mathrm{pol}(\varphi, \pi) = -1$). A formula φ is *pure* if, for all positions π and π', $\mathrm{pol}(\varphi, \pi) = \mathrm{pol}(\varphi, \pi') \neq 0$.

PLTL-formulae are interpreted over infinite sequences of states $\sigma = (s_i)_{i \in \mathbb{N}}$ such that each s_i, $0 \leq i$, is a set of propositional variables. The notation $(\sigma, i) \models \varphi$ denotes the truth of a formula φ in the model σ at the state index i, $i \in \mathbb{N}$. For any formula φ, model σ, and state index i, $i \in \mathbb{N}$, then either $(\sigma, i) \models \varphi$ holds or $(\sigma, i) \models \varphi$ does not hold, where the latter is denoted by $(\sigma, i) \not\models \varphi$. The semantics of $\mathsf{WFF}_{\mathsf{PLTL}}$ can now be given as follows:

Definition 1. *Let* φ *and* ψ *be formulae in* $\mathsf{WFF}_{\mathsf{PLTL}}$*,* σ *a model, and* $i \in \mathbb{N}$ *the index of a state in* σ*.*

- $(\sigma, i) \models \top$

- $(\sigma, i) \not\models \bot$
- $(\sigma, i) \models p$ *if, and only if,* $p \in s_i$*, where* $p \in P$
- $(\sigma, i) \models \neg\varphi$ *if, and only if,* $(\sigma, i) \not\models \varphi$
- $(\sigma, i) \models (\varphi_1 \wedge \cdots \wedge \varphi_n)$ *if, and only if, for every* i*,* $1 \leq i \leq n$*,* $(\sigma, i) \models \varphi$
- $(\sigma, i) \models (\varphi_1 \vee \cdots \vee \varphi_n)$ *if, and only if, for some* i*,* $1 \leq i \leq n$*,* $(\sigma, i) \models \varphi_i$
- $(\sigma, i) \models (\varphi \rightarrow \psi)$ *if, and only if,* $(\sigma, i) \models \neg\varphi$ *or* $(\sigma, i) \models \psi$
- $(\sigma, i) \models (\varphi \leftrightarrow \psi)$ *if, and only if,* $(\sigma, i) \models (\varphi \rightarrow \psi)$ *and* $(\sigma, i) \models (\psi \rightarrow \varphi)$
- $(\sigma, i) \models \bigcirc\varphi$ *if, and only if,* $(\sigma, i+1) \models \varphi$
- $(\sigma, i) \models \Diamond\varphi$ *if, and only if,* $\exists k, k \in \mathbb{N}, k \geq i, (\sigma, k) \models \varphi$
- $(\sigma, i) \models \Box\varphi$ *if, and only if,* $\forall k, k \in \mathbb{N}$*, if* $k \geq i$*, then* $(\sigma, k) \models \varphi$
- $(\sigma, i) \models (\varphi\, \mathcal{U}\, \psi)$ *if, and only if,* $\exists k, k \in \mathbb{N}, k \geq i, (\sigma, k) \models \psi$ *and* $\forall j, j \in \mathbb{N}$*, if* $i \leq j < k$*, then* $(\sigma, j) \models \varphi$*.*
- $(\sigma, i) \models (\varphi\, \mathcal{W}\, \psi)$ *if, and only if, either* $(\sigma, i) \models \varphi\, \mathcal{U}\, \psi$ *or* $(\sigma, i) \models \Box\varphi$*.*

A formula φ is *satisfiable* if there is a model σ such that $(\sigma, 0) \models \varphi$. If $(\sigma, 0) \models \varphi$ for all models σ, then φ is said to be *valid*, denoted by $\models \varphi$. Two PLTL-formulae φ and ψ are *equi-satisfiable* if, and only if, φ is satisfiable if, and only if, ψ is satisfiable. Two PLTL-formulae φ and ψ are *equivalent* if, and only if, for every model σ and every state index i, $i \in \mathbb{N}$, $(\sigma, i) \models \varphi$ if and only if $(\sigma, i) \models \psi$

3 Normal Form

Formulae in the language of PLTL can be transformed into a normal form called Separated Normal Form (SNF) [5]. This normal form was inspired by (but it is independent of) Gabbay's separation result [8], which states that temporal formulae can be transformed into their past, present and future-time components. In the version of SNF that we present here, formulae are represented by a conjunction of clauses,

$$\bigwedge_{1 \leq i \leq n} C_i$$

where each clause C_i, $1 \leq i \leq n$, $n \in \mathbb{N}$, is in one of the following three forms.

$$\bigvee_{i=1}^{m} l_i \qquad \text{(initial clause)}$$

$$\Box(\bigvee_{i=1}^{m} l_i \vee \bigvee_{j=1}^{n} \bigcirc l'_j) \qquad \text{(global clause)}$$

$$\Box(\bigvee_{i=1}^{m} l_i \vee \Diamond l'_1) \qquad \text{(eventuality clause)}$$

where $n, m \geq 0$ and for every i, $1 \leq i \leq m$, and j, $1 \leq j \leq n$, l_i and l'_j are literals.

Note that SNF clauses do not contain occurrences of the operators $\mathcal{U}$ and $\mathcal{W}$, the operator $\Box$ only occurs as principal operator of a clause, and nesting temporal operators is limited to the combinations $\Box$ and $\bigcirc$; and $\Box$ and $\Diamond$.

Every PLTL-formula φ can be transformed into an equi-satisfiable formula in SNF. Fisher, Dixon and Peim [6] describe functions τ_0 and τ_1 with $\tau_0(\varphi) = (q_1 \wedge \tau_1(\Box(\neg q_1 \vee \varphi)))$, where q_1 is a fresh propositional variable not occurring in φ, such that $\tau_0(\varphi)$ is in SNF and equi-satisfiable to φ. The function τ_1 proceeds top-down and uses *renaming* to deal with subformulae that are not yet in normal form [17]. The inductive definition of τ_1 is shown in Figure 1, where φ, φ_i, $0 \leq i \leq n$, and ψ are PLTL-formulae, l, l_1, l_2 are literals, q is a propositional variable, and q', q'', and q''' are fresh propositional variables. Theorems 7.1.1 and 7.1.2 in [6] show that the computation of $\tau_0(\varphi)$ always terminates, that the number of SNF clauses in $\tau_0(\varphi)$ is bounded by $1 + 4 \times \text{size}(\varphi)$, and that the number of fresh propositional variables in $\tau_0(\varphi)$ is bounded by $1 + 11 \times \text{size}(\varphi)$, where $\text{size}(\varphi)$ is the size of φ. UH1: New

It is easy to see that τ_1 will not always produce a normal form with the smallest number of clauses. For example,

$$\varphi_1 = \Box(\neg q \vee \neg \bigcirc \neg p)$$

is equivalent to $\Box(\neg q \vee \bigcirc p)$ which is in normal form, but, according to Equation (8) in Figure 1, $\tau_1(\varphi_1)$ would produce a conjunction of two clauses. This could be avoided by converting formulae to *negation normal form* (NNF) before applying τ_1, using the rewrite rules for temporal formulae given in Figure 2 and the usual equivalences for classical formulae. The formula

$$\varphi_2 = \Box(\neg q \vee (p \,\mathcal{U}\, p))$$

is equivalent to $\Box(\neg q \vee p)$ which again is in normal form, but, according to Equation (16) in Figure 1, $\tau_1(\varphi_2)$ would produce a conjunction of five clauses. This could be avoided by *simplification* using well-known equivalences among temporal formulae, which are given in Figure 3, together with the well-known corresponding equivalences among Boolean formulae.

Using such equivalences we can also extend or reduce the scope of temporal operators. *Prenexing* corresponds to moving modal operators outwards a formula. Analogously, *anti-prenexing* corresponds to moving modal operators inwards a formula. For example, anti-prenexing would replace $\Diamond(p \vee q)$ by the equivalent $(\Diamond p \vee \Diamond q)$ while prenexing would do the opposite. Here we only discuss the anti-prenexing technique. In first-order logic, it has been shown [3] that the transformation of a given problem into anti-prenex normal form may result in a better set of clauses. Similar results for normal modal logics which allow the simplification of nested operators can be found in [14]. For temporal logics, anti-prenexing together with simplification may help reducing the size of a formula and, consequently, the size of the normal form. For instance, the anti-prenex normal form of $\varphi_3 = \Box(p \wedge \Box(p \wedge \Box\, p))$ is

$$\Box\, p \wedge \Box\,\Box\, p \wedge \Box\,\Box\,\Box\, p$$

$$\tau_1(\Box(\neg q \vee (\varphi_1 \wedge \cdots \wedge \varphi_n))) = \tau_1(\Box(\neg q \vee \varphi_1)) \wedge \cdots \wedge \tau_1(\Box(\neg q \vee \varphi_n)) \quad (1)$$

$$\tau_1(\Box(\neg q \vee \neg(\varphi_1 \wedge \cdots \wedge \varphi_n))) = \tau_1(\Box(\neg q \vee \neg\varphi_1 \vee \cdots \vee \neg\varphi_n)) \quad (2)$$

$$\tau_1(\Box(\neg q \vee \varphi_1 \vee \cdots \vee \varphi_n)) = \Box(\neg q \vee \varphi_1 \vee \cdots \vee q' \vee \cdots \vee \varphi_n) \wedge \Box(\neg q' \vee \varphi_i) \quad (3)$$
if $\varphi_i, 1 \leq i \leq n,$ is not an elementary formula

$$\tau_1(\Box(\neg q \vee \varphi \vee \bigcirc(\varphi_1 \vee \cdots \vee \varphi_n))) = \tau_1(\Box(\neg q \vee \varphi \vee \bigcirc\varphi_1 \vee \cdots \vee \bigcirc\varphi_n)) \quad (4)$$

$$\tau_1(\Box(\neg q \vee \varphi \vee \bigcirc\psi)) = \tau_1(\Box(\neg q \vee \varphi \vee \bigcirc q')) \wedge \tau_1(\Box(\neg q' \vee \psi)) \quad (5)$$
if ψ is not a disjunction of literals

$$\tau_1(\Box(\neg q \vee (\varphi \rightarrow \psi))) = \tau_1(\Box(\neg q \vee \neg\varphi \vee \psi)) \quad (6)$$

$$\tau_1(\Box(\neg q \vee \neg(\varphi \rightarrow \psi))) = \tau_1(\Box(\neg q \vee \varphi)) \wedge \tau_1(\Box(\neg q \vee \neg\psi)) \quad (7)$$

$$\tau_1(\Box(\neg q \vee \neg\bigcirc\varphi)) = \tau_1(\Box(\neg q \vee \bigcirc q')) \wedge \tau_1(\Box(\neg q' \vee \neg\varphi)) \quad (8)$$

$$\tau_1(\Box(\neg q \vee \Box\varphi)) = \tau_1(\Box(\neg q \vee \Box q')) \wedge \tau_1(\Box(\neg q' \vee \varphi)) \quad (9)$$
if φ is neither a literal nor a constant

$$\tau_1(\Box(\neg q \vee \Box l)) = \Box(\neg q \vee l) \wedge \Box(\neg q \vee q') \wedge \Box(\neg q' \vee \bigcirc l) \wedge \Box(\neg q' \vee \bigcirc q') \quad (10)$$

$$\tau_1(\Box(\neg q \vee \neg\Box\varphi)) = \tau_1(\Box(\neg q \vee \Diamond\neg\varphi)) \quad (11)$$

$$\tau_1(\Box(\neg q \vee \Diamond\varphi)) = \tau_1(\Box(\neg q \vee \Diamond q')) \wedge \tau_1(\Box(\neg q' \vee \varphi)) \quad (12)$$
if φ is neither a literal nor a constant

$$\tau_1(\Box(\neg q \vee \neg\Diamond\varphi)) = \tau_1(\Box(\neg q \vee \Box\neg\varphi)) \quad (13)$$

$$\tau_1(\Box(\neg q \vee (\varphi\,\mathcal{U}\,\psi))) = \tau_1(\Box(\neg q \vee (q'\,\mathcal{U}\,\psi))) \wedge \tau_1(\Box(\neg q' \vee \varphi)) \quad (14)$$
if φ is neither a literal nor a constant

$$\tau_1(\Box(\neg q \vee (\varphi\,\mathcal{U}\,\psi))) = \tau_1(\Box(\neg q \vee (\varphi\,\mathcal{U}\,q'))) \wedge \tau_1(\Box(\neg q' \vee \psi)) \quad (15)$$
if ψ is neither a literal nor a constant

$$\tau_1(\Box(\neg q \vee (l_1\,\mathcal{U}\,l_2))) = \Box(\neg q \vee \Diamond l_2) \wedge \Box(\neg q \vee l_1 \vee l_2) \wedge \Box(\neg q \vee q' \vee l_2) \wedge \Box(\neg q' \vee \bigcirc l_1 \vee \bigcirc l_2) \wedge \Box(\neg q' \vee \bigcirc q' \vee \bigcirc l_2) \quad (16)$$

$$\tau_1(\Box(\neg q \vee \neg(\varphi\,\mathcal{U}\,\psi))) = \tau_1(\Box(\neg q \vee (q'\,\mathcal{W}\,q''))) \wedge \Box(\neg q'' \vee q') \wedge \Box(\neg q'' \vee q''') \wedge \tau_1(\Box(\neg q' \vee \neg\psi)) \wedge \tau_1(\Box(\neg q''' \vee \neg\varphi)) \quad (17)$$

$$\tau_1(\Box(\neg q \vee (\varphi\,\mathcal{W}\,\psi))) = \tau_1(\Box(\neg q \vee (q'\,\mathcal{W}\,\psi))) \wedge \tau_1(\Box(\neg q' \vee \varphi)) \quad (18)$$
if φ is neither a literal nor a constant

$$\tau_1(\Box(\neg q \vee (\varphi\,\mathcal{W}\,\psi))) = \tau_1(\Box(\neg q \vee (\varphi\,\mathcal{W}\,q'))) \wedge \tau_1(\Box(\neg q' \vee \psi)) \quad (19)$$
if ψ is neither a literal nor a constant

$$\tau_1(\Box(\neg q \vee (l_1\,\mathcal{W}\,l_2))) = \Box(\neg q \vee l_1 \vee l_2) \wedge \Box(\neg q \vee q' \vee l_2) \wedge \Box(\neg q' \vee \bigcirc l_1 \vee \bigcirc l_2) \wedge \Box(\neg q' \vee \bigcirc q' \vee \bigcirc l_2) \quad (20)$$

$$\tau_1(\Box(\neg q \vee \neg(\varphi\,\mathcal{W}\,\psi))) = \tau_1(\Box(\neg q \vee (q'\,\mathcal{U}\,q''))) \wedge \Box(\neg q'' \vee q') \wedge \Box(\neg q'' \vee q''') \wedge \tau_1(\Box(\neg q' \vee \neg\psi)) \wedge \tau_1(\Box(\neg q''' \vee \neg\varphi)) \quad (21)$$

$$\tau_1(\varphi) = \varphi, \text{ if no other rule applies} \quad (22)$$

Figure 1. Definition of Transformation Function τ_1

$$\mathrm{nnf}(\neg \bigcirc \varphi) = \bigcirc \mathrm{nnf}(\neg\varphi) \quad (23)$$
$$\mathrm{nnf}(\neg \Box \varphi) = \Diamond \mathrm{nnf}(\neg\varphi) \quad (24)$$
$$\mathrm{nnf}(\neg \Diamond \varphi) = \Box \mathrm{nnf}(\neg\varphi) \quad (25)$$
$$\mathrm{nnf}(\bigcirc \varphi) = \bigcirc \mathrm{nnf}(\varphi) \quad (26)$$
$$\mathrm{nnf}(\Diamond \varphi) = \Diamond \mathrm{nnf}(\varphi) \quad (27)$$
$$\mathrm{nnf}(\Box \varphi) = \Box \mathrm{nnf}(\varphi) \quad (28)$$
$$\mathrm{nnf}(\neg(\varphi \,\mathcal{U}\, \psi)) = (\mathrm{nnf}(\neg\psi) \,\mathcal{W}\, \mathrm{nnf}(\neg\varphi \wedge \neg\psi)) \quad (29)$$
$$\mathrm{nnf}(\neg(\varphi \,\mathcal{W}\, \psi)) = (\mathrm{nnf}(\neg\psi) \,\mathcal{U}\, \mathrm{nnf}(\neg\varphi \wedge \neg\psi)) \quad (30)$$
$$\mathrm{nnf}((\varphi \,\mathcal{U}\, \psi) = (\mathrm{nnf}(\varphi) \,\mathcal{U}\, \mathrm{nnf}(\psi)) \quad (31)$$
$$\mathrm{nnf}((\varphi \,\mathcal{W}\, \psi) = (\mathrm{nnf}(\varphi) \,\mathcal{W}\, \mathrm{nnf}(\psi)) \quad (32)$$

Figure 2. Definition of the Negation Normal Form for Temporal Formulae

$$\bigcirc \top \leftrightarrow \top \quad (33)$$
$$\bigcirc \bot \leftrightarrow \bot \quad (34)$$
$$\Diamond \bot \leftrightarrow \bot \quad (35)$$
$$\Diamond \top \leftrightarrow \top \quad (36)$$
$$\Box \bot \leftrightarrow \bot \quad (37)$$
$$\Box \top \leftrightarrow \top \quad (38)$$
$$\Diamond \Diamond \varphi \leftrightarrow \Diamond \varphi \quad (39)$$
$$\Box \Box \varphi \leftrightarrow \Box \varphi \quad (40)$$
$$\Diamond \Box \Diamond \varphi \leftrightarrow \Box \Diamond \varphi \quad (41)$$
$$\Box \Diamond \Box \varphi \leftrightarrow \Diamond \Box \varphi \quad (42)$$
$$\Diamond(\varphi \,\mathcal{U}\, \psi) \leftrightarrow \Diamond \psi \quad (43)$$
$$\Box(\varphi \,\mathcal{W}\, \psi) \leftrightarrow \Box(\varphi \vee \psi) \quad (44)$$
$$(\varphi \,\mathcal{U}\, \Diamond \psi) \leftrightarrow \Diamond \psi \quad (45)$$
$$(\varphi \,\mathcal{U}\, \neg\varphi) \leftrightarrow \Diamond \neg\varphi \quad (46)$$
$$(\neg\varphi \,\mathcal{U}\, \varphi) \leftrightarrow \Diamond \varphi \quad (47)$$
$$(\varphi \,\mathcal{U}\, \varphi) \leftrightarrow \varphi \quad (48)$$
$$(\Box \varphi \,\mathcal{W}\, \psi) \leftrightarrow (\Box \varphi \vee \psi) \quad (49)$$
$$(\varphi \,\mathcal{W}\, \neg\varphi) \leftrightarrow \top \quad (50)$$
$$(\neg\varphi \,\mathcal{W}\, \varphi) \leftrightarrow \top \quad (51)$$
$$(\varphi \,\mathcal{W}\, \varphi) \leftrightarrow \varphi \quad (52)$$
$$(\varphi \,\mathcal{U}\, \top) \leftrightarrow \top \quad (53)$$
$$(\varphi \,\mathcal{U}\, \bot) \leftrightarrow \bot \quad (54)$$
$$(\top \,\mathcal{U}\, \varphi) \leftrightarrow \Diamond \varphi \quad (55)$$
$$(\bot \,\mathcal{U}\, \varphi) \leftrightarrow \varphi \quad (56)$$
$$(\varphi \,\mathcal{W}\, \top) \leftrightarrow \top \quad (57)$$
$$(\varphi \,\mathcal{W}\, \bot) \leftrightarrow \Box \varphi \quad (58)$$
$$(\top \,\mathcal{W}\, \varphi) \leftrightarrow \top \quad (59)$$
$$(\bot \,\mathcal{W}\, \varphi) \leftrightarrow \varphi \quad (60)$$

Figure 3. Temporal Equivalences for Simplification

As $\Box\Box\psi$ is equivalent to $\Box\psi$, for any formula ψ, after simplification, we obtain the formula $\Box p$. Figure 4 shows the equivalences that can be used to either extend or reduce the scope of temporal operators. For anti-prenexing the equivalences are used as rewrite-rules from left to right. For instance, the anti-prenex normal form of the formula $\bigcirc\Box\bigcirc\Box\bigcirc\Box p$ is $\Box\Box\Box\bigcirc\bigcirc\bigcirc p$, which can then be simplified to $\Box\bigcirc\bigcirc\bigcirc p$.

Since τ_1 only preserves satisfiability, we can go even further for the formulae φ_1, φ_2 and φ_3, given before. In all these formulae the propositional variable p only occurs with positive polarity. In analogy to propositional logic, we can apply *pure literal elimination*, that is, we replace variables that only occur positively (resp. negatively) by $\top$ (resp. $\bot$), and then simplify. For φ_1, φ_2, and φ_3 we obtain $\top$ as result.

Finally, a peculiarity of the normal form transformation by τ_0 and τ_1 is that for a formula φ in normal form, $\tau_0(\varphi)$ will not be the same as φ. Say, φ_4 is $(q_2 \wedge \Box(\neg q_2 \vee p))$. Then $\tau_0(\varphi_4) = (q_1 \wedge \tau_1(\Box(\neg q_1 \vee ((q_2 \wedge \Box(\neg q_2 \vee p))))))$. Computing τ_1 will involve Equation (10) in Figure 1, creating four additional clauses. We can ameliorate this problem by modifying τ_1 so that it treats $\Box(\neg q_1 \vee \Box\varphi)$ like $\Box(\neg q_1 \vee \varphi)$ for the specific propositional variable q_1 used by τ_0. The next lemma

UH1: Is this true for 65 and 66? CD1: It is. I have added a variation of the reviewer's example.

$$\bigcirc(\varphi_1 \vee \cdots \vee \varphi_n) \leftrightarrow (\bigcirc\varphi_1 \vee \cdots \vee \bigcirc\varphi_n) \tag{61}$$
$$\Diamond(\varphi_1 \vee \cdots \vee \varphi_n) \leftrightarrow (\Diamond\varphi_1 \vee \cdots \vee \Diamond\varphi_n) \tag{62}$$
$$\bigcirc(\varphi_1 \wedge \cdots \wedge \varphi_n) \leftrightarrow (\bigcirc\varphi_1 \wedge \cdots \wedge \bigcirc\varphi_n) \tag{63}$$
$$\Box(\varphi_1 \wedge \cdots \wedge \varphi_n) \leftrightarrow (\Box\varphi_1 \wedge \cdots \wedge \Box\varphi_n) \tag{64}$$
$$\bigcirc\Box\varphi \leftrightarrow \Box\bigcirc\varphi \tag{65}$$
$$\bigcirc\Diamond\varphi \leftrightarrow \Diamond\bigcirc\varphi \tag{66}$$
$$\bigcirc(\varphi\,\mathcal{U}\,\psi) \leftrightarrow ((\bigcirc\varphi)\,\mathcal{U}\,(\bigcirc\psi)) \tag{67}$$
$$\bigcirc(\varphi\,\mathcal{W}\,\psi) \leftrightarrow ((\bigcirc\varphi)\,\mathcal{W}\,(\bigcirc\psi)) \tag{68}$$
$$((\varphi \wedge \psi)\,\mathcal{U}\,\vartheta) \leftrightarrow ((\varphi\,\mathcal{U}\,\vartheta) \wedge (\psi\,\mathcal{U}\,\vartheta)) \tag{69}$$
$$(\varphi\,\mathcal{U}\,(\psi \vee \vartheta)) \leftrightarrow ((\varphi\,\mathcal{U}\,\psi) \vee (\varphi\,\mathcal{U}\,\vartheta)) \tag{70}$$
$$((\varphi \wedge \psi)\,\mathcal{W}\,\vartheta) \leftrightarrow ((\varphi\,\mathcal{W}\,\vartheta) \wedge (\psi\,\mathcal{U}\,\vartheta)) \tag{71}$$
$$(\varphi\,\mathcal{W}\,(\psi \vee \vartheta)) \leftrightarrow ((\varphi\,\mathcal{W}\,\psi) \vee (\varphi\,\mathcal{W}\,\vartheta)) \tag{72}$$

Figure 4. Temporal Equivalences for Prenexing and Anti-Prenexing

shows that this simplification step in the transformation is correct, that is, that satisfiability is preserved.

Lemma 1. *Let φ be a* PLTL*-formula and q_1 a propositional variable not occurring in φ. Then, $\Box\varphi$ is satisfiable if, and only if, $q_1 \wedge \tau_1(\Box\varphi)$ is satisfiable.*

Proof. ($\Rightarrow$) By the results in [6], $\Box\varphi$ is satisfiable if, and only if, $q_1 \wedge \Box(\neg q_1 \vee \tau_1(\Box\varphi))$ is satisfiable. From the definition of satisfiability, there is a model σ such that $(\sigma, 0) \models q_1 \wedge \Box(\neg q_1 \vee \tau_1(\Box\varphi))$. It follows that (1) $(\sigma, 0) \models q_1$ and $(\sigma, 0) \models \Box(\neg q_1 \vee \tau_1(\Box\varphi))$. From the definition of satisfiability of the temporal operator $\Box$, for all i, $i \in \mathbb{N}$, we have that $(\sigma, i) \models \neg q_1 \vee \tau_1(\Box\varphi)$. In particular, $(\sigma, 0) \models \neg q_1 \vee \tau_1(\Box\varphi)$. As $(\sigma, 0) \models q_1$, it follows from the definition of satisfiability of disjunctions that (2) $(\sigma, 0) \models \tau_1(\Box\varphi)$. From (1) and (2), $q_1 \wedge \tau_1(\Box\varphi)$ is satisfiable.

($\Leftarrow$) If $q_1 \wedge \tau_1(\Box\varphi)$ is satisfiable, then there is a model σ such that $(\sigma, 0) \models q_1 \wedge \tau_1(\Box\varphi)$. We construct a model σ' such that $s'_0 = s_0 \cup \{q_1\}$ and, for all $i > 0$, $s'_i = s_i \setminus \{q_1\}$. It follows, by construction, that (3) $(\sigma', 0) \models q_1$. As $(\sigma, 0) \models \tau_1(\Box\varphi)$ and the evaluation of $\tau_1(\Box\varphi)$ does not depend on the evaluation of q_1, we have that $(\sigma', 0) \models \tau_1(\Box\varphi)$ and, from the definition of satisfiability of disjunctions, (4) $(\sigma', 0) \models \neg q_1 \vee \tau_1(\Box\varphi)$. Also, by construction, for all $i > 0$, $(\sigma', i) \models \neg q_1$. Thus, for all i, $i \in \mathbb{N}$, (5) $(\sigma', i) \models \neg q_1 \vee \tau_1(\Box\varphi)$. From (4) and (5), it follows that, for all $i \geq 0$, $(\sigma', i) \models \neg q_1 \vee \tau_1(\Box\varphi)$. From the definition of satisfiability of the $\Box$ operator, we obtain that (6) $(\sigma', 0) \models \Box(\neg q_1 \vee \tau_1(\Box\varphi))$. From (3) and (6), we have that $(\sigma', 0) \models q_1 \wedge \Box(\neg q_1 \vee \tau_1(\Box\varphi))$. By the results in [6], $\Box\varphi$ is satisfiable.

```
1  Algorithm: ltl2snf(φ)
2  repeat
3  |  ⟨numsimp, φ⟩ ← input_simplification(φ);
4  until (numsimp = 0);
5  φ ← nnf_transformation(φ);
6  repeat
7  |  ⟨φ, numaprenex⟩ ← aprenex_transformation(φ);
8  |  ⟨numsimp, φ⟩ ← input_simplification(φ);
9  until (numsimp = 0 and numaprenex = 0);
10 return snf_transformation(φ);
```

Figure 5. Main Loop

4 Implementation and Evaluation

We have implemented τ_0, τ_1 and the techniques described in Section 3. The tool `ltl2snf` is a transformer written in C, which takes a formula in the language of PLTL and produces a set of SNF clauses in the syntax used by **TRP++** [10] and `LS4` [22,21]. The source code of `ltl2snf`, together with installation and usage instructions, is available in [15].

The techniques for pre-processing the input are coded as independently as possible in order to allow for the easy addition and testing of new features. By default, for a given PLTL-formula φ, `ltl2snf` just computes $\tau_0(\text{nnf}(\varphi))$. The resulting formula is in simplified form: repeated literals, the constant $\bot$, and the formulae $\bigcirc\bot$ and $\Diamond\bot$ within a clause are deleted; and clauses containing the constant $\top$, and the formulae $\bigcirc\top$ or $\Diamond\top$ are removed. Once such a set of SNF clauses has been computed, no further simplification is performed. In particular, clause subsumption or variable elimination techniques such as those described in [10,20] are not applied. We consider this to be a task for theorem provers that take SNF clauses as input.

UH2: Shouldn't that be 'clauses containing the constant $\top$ are removed'? Changed.

The main loop of `ltl2snf` is schematically presented in Figure 5, where the input_simplication function returns the number of transformation steps given by the optional techniques enable by the user, namely:

- `-ple` for pure literal elimination together with constant propagation, and
- `-simp` for simplification.

Two more processing techniques can be enabled by the options:

- `-aprenex` for the anti-prenexing transformation, which is performed by the function aprenex_transformation in Figure 5; and
- `-isnf` for the modified version of τ_1, which is implemented as part of the transformation into the normal form (snf_transformation).

UH1: We should state which function implement these. Also, aprenex_transformation does not return a count, so how do we know a fixpoint has been reached? I have added the counter for aprenex in the algorithm. I have also added the functions in the text.

Both the simplification procedure and the transformation into anti-prenex normal form are performed until a fixed-point is reached, that is, no further transformation/simplification is possible. We also note that simplification and pure literal elimination are performed before the transformation into Negation Normal Form, as this allows for better memory use and performance. For instance, the NNF of

$$\neg(p_1 \,\mathcal{U}\, (p_2 \,\mathcal{U}\, p_3))$$

results in

$$((\neg p_3 \,\mathcal{W}\, (\neg p_2 \wedge \neg p_3)) \,\mathcal{W}\, (\neg p_1 \wedge (\neg p_3 \,\mathcal{W}\, (\neg p_2 \wedge \neg p_3))))$$

where all literals are pure. It is easy to see that for formulae with similar structure, the result of the translation into NNF is exponential in the size of the original formula. Applying pure literal elimination together with constant propagation to the original formula avoids this problem. For this particular example, the resulting formula is $\top$.

To evaluate the effectiveness of each technique and their combinations we have used a set of PLTL-formulae collected by Schuppan and Darwiche [19]. The collection consists of 7450 formulae, half of these are taken from the literature or previous collections of benchmark formulae, the other half is obtained by negating these formulae. Since we also compared `ltl2snf` with an earlier implementation of the SNF transformation in the tool `translate` [12], we have only 6135 of these formulae, on the remaining formulae, `translate` does not produce a normal form within a time limit of 1000 CPU seconds. The collection is divided into seven classes: acacia, alaska, anzu, forobots, rozier, schuppan, trp. Most classes consist of several families of formulae, sometimes with quite different characteristics (for a detailed description of each class and each family see [19]). We have therefore re-categorised the formulae as follows:

- *application* category: consists of the formulae in the classes acacia, alaska, anzu, and forobots, all of which relate to 'real world' applications, as well as the counter family of the rozier class, containing formulae that specify serial counters, and the phltl family of the schuppan class, containing formulae that specify temporalised instances of the pigeon hole problem;
- *pattern* category: consists of the pattern family of the rozier class and the O1formula and O2formula families of the schuppan class; these are series of scalable temporal formulae that follow simple patterns;
- *random* category: consists of the formulas family of the rozier class, containing random temporal formulae;
- *semi-random* category: consists of the trp class, containing formulae that each follow a pattern but also have a random part. Half of the formulae in this category (the unnegated formulae) are already in SNF.

Table 1 shows syntactic properties of these categories. The random category contains more formulae than all other categories combined. However, the combined size of its formulae is smaller than that of the application and semi-random

Category	#For-mulae	#Temporal Operators Occurrences	Avg Temporal Operators	#Boolean Operator Occurrences	Boolean Variables	Total Size	Avg Size
application	542	65733	121.3	194029	6772	381525	703.9
pattern	407	30703	75.4	15051	24068	72739	178.7
random	4000	87734	21.9	141048	11542	311636	77.9
semi-random	1187	128146	108.0	234298	19163	523890	441.4
total	6136	312316	50.9	584426	61545	1289790	210.2

Table 1. Properties of the benchmark formulae

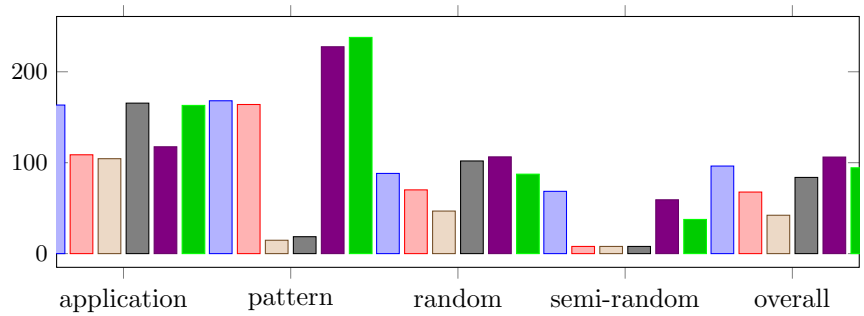
categories. The pattern category is the smallest, both in terms of number of formulae and their combined size. It is the only category in which the number of temporal operator occurrences is larger, and significantly so, than the number of occurrences of Boolean operators. On the other hand, the average number of temporal operators per formula is considerably higher for the application and semi-random categories than the others. The application category is the only one where formulae contain the Boolean constants $\top$ and $\bot$, on average 13 occurrences in each formula.

Besides considering the effect of various pre-processing techniques implemented in `ltl2snf` on the normal form, we have also compared `ltl2snf` with the latest version of `translate` [12], an earlier implementation of the transformation of PLTL to SNF. `translate` which is written in OCaml, implements the same set of simplification rules used by `ltl2snf`, the main differences being that conjunctions and disjunctions are taken as binary operators and that some prenexing is also applied (specifically, Equations (61) to (64), (67) and (68) in Figure 4). The option `-s` activates simplification. With the option `-r`, during the normal form transformation `translate` will replace $(\varphi \vee \psi)$ by $(p \vee \psi)$ together with the definition $\Box(\neg p \vee \varphi)$, if both φ and ψ are temporal formulae. This is assumed to result in a smaller normal form in most cases, although it not always does.

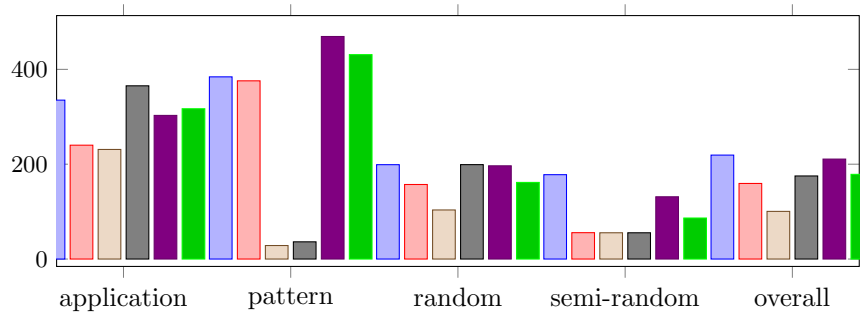
Figure 6 shows for particular combinations of `ltl2snf` and `translate` options for each category (a) the average number of fresh propositional variables introduced in the transformation, (b) the average number of clauses produced, (c) the average size of the normal form and (d) the average computation time. Note that `ltl2snf` always computes the NNF of the input formula. We therefore cannot investigate whether this in itself has a positive or negative effect.

Table 1 together with Figures 6(a) and 6(b) show that on average the number of fresh propositional variables introduced in the SNF transformation as well as the number of clauses produced is much lower than the worst case upper bound established in [6]. Instead of four times the number of clauses and eleven times the number fresh variables in the size of formulae, even without any pre-processing we see that on everage we get a number of clauses linear in the size of a formulae and a number of fresh propositional variables at half the size of a formula. With pre-processing both can further be reduced by half.

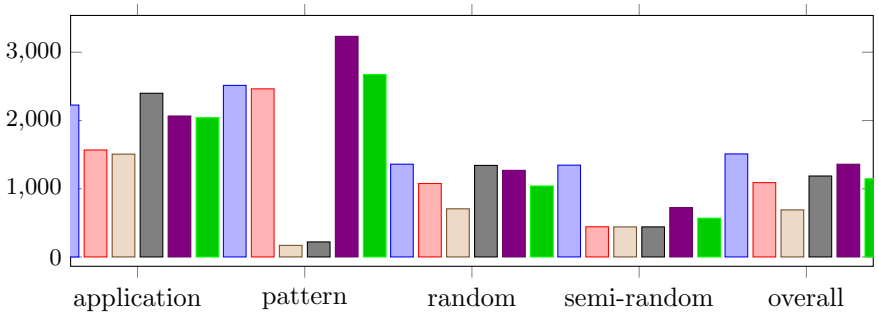
UH1: New. What actually happens with negated until formulae? CN1: It is rewritten using unless: $\neg(p \ \mathcal{U} \ q) = (\neg q \ \mathcal{W} \ (\neg p \wedge \neg q))$. Maybe we should say here that applying renaming during the transformation into the normal form could produce smaller normal forms? I'm not sure about that, though. UH2: I think we should use the release operator instead.



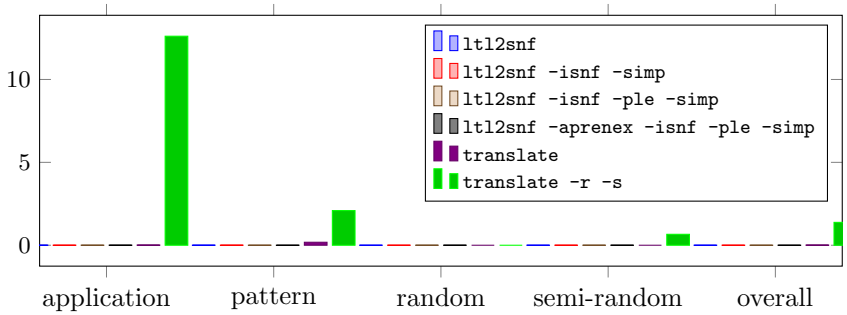
(a) Average number of fresh variables introduced for each problem during transformation



(b) Average number of SNF clauses for each problem



(c) Average size of SNF for each problem



(d) Average time for transformation of each problem (in CPU seconds)

Figure 6. Evaluation of `ltl2snf` and `translate`

Overall, the combination of `-isnf`, `-simp`, and `-ple` offers the best result. The option `-isnf` offer the greatest improvement on the semi-random category, as half its formulae are already in normal form. The option `-ple`, pure literal elimination, shows the greatest improvements on the pattern and random categories. For the pattern category this is the case because most formulae only contain positive propositional literals. It seems to have been overlooked in their construction that the formulae consequently have trivial models. For the random category, again a lot of them contain pure literals as an artefact of the particular way they were randomly generated. On the other hand, on the application and semi-random categories, pure literal elimination has almost no effect. For the application category, one can take this as an indication that in the formalisation of 'real world' applications, pure literals are rare. For the semi-random category, the lack of pure literals is an artefact of their construction. Option `-aprenex`, anti-prenexing, appears to have a detrimental effect. This is in contrast to the results in [14] for basic modal logic, where anti-prenexing was found to be beneficial. This indicates that the assumption that anti-prenexing leads to chains of temporal operators that can be collapsed, is not true for any of the benchmark categories.

Regarding the time it takes to compute the normal form, we can see in Figure 6(d) that `ltl2snf` typically takes less than 1ms to compute the SNF of a formula, independent of the pre-processing techniques that are applied. In contrast, for `translate` the use of simplification increases the computation time dramatically, in particular, for the application category where it increases from an average of 0.01 CPU seconds to 12.61 CPU seconds. Also remember that we have already excluded over 1300 formulae for which `translate` does not complete the transformation within 1000 CPU seconds, in particular, it does not do so with simplification enabled. One possible explanation for the gap between the time spent by `translate` and that spent by `ltl2snf` is that the former does not flatten conjunctions and disjunctions, but a polynomial, thus non-optimal, sorting algorithm is applied to conjuncts and disjuncts before applying simplification.

UH3: Move the sentence about the 1300 excluded formulae in front of the explanation.

We have also started to evaluate how the provers `LS4` and **TRP++** perform on the various sets of SNF clauses that `ltl2snf` can produce for each of the benchmark formulae. Initial results suggest that at least on average, the smallest normal form indeed leads to the best performance by the two provers.

UH3: Correction

5 Conclusion

Overall, the results show that the application of pre-processing techniques significantly reduced the size of the normal form and that, if implemented well, as in `ltl2snf`, this application comes at negligible computational cost. We are currently implementing the prenex transformations given in Figure 4 and different techniques for renaming in order to reduce further the size of the generated set of clauses. Finally, although 'small' normal forms seem to be a good measure for determining the quality of the translation into the normal form, we are planning

for a more systematic evaluation of the impact they have in the efficiency of the provers. As part of our investigation, we are planning to implement different variants of the separated normal form, for instance a variant introduced in [2] where only a single eventuality clause is allowed. While this leads to a bigger normal form when we need to reduce several eventualities in the input to just one, proof search by existing PLTL decision procedures may become easier and therefore result in better overall performance.

References

1. Azmy, N., Weidenbach, C.: Computing tiny clause normal forms. In: Proc. CADE-24. Lecture Notes in Computer Science, vol. 7898, pp. 109–125. Springer (2013)
2. Degtyarev, A., Fisher, M., Konev, B.: A simplified clausal resolution procedure for propositional linear-time temporal logic. In: Proc. TABLEAUX 2002. Lecture Notes in Computer Science, vol. 2381, pp. 85–99. Springer (2002)
3. Egly, U.: On the value of antiprenexing. In: Proc. LPAR 1994. Lecture Notes in Artificial Intelligence, vol. 822, pp. 69–83. Springer (1994)
4. Fisher, M.: A Resolution Method for Temporal Logic. In: Proc. IJCAI 1991. pp. 99–104. Morgan Kaufman (1991)
5. Fisher, M.: A Normal Form for Temporal Logic and its Application in Theorem-Proving and Execution. Journal of Logic and Computation 7(4), 429–456 (Aug 1997)
6. Fisher, M., Dixon, C., Peim, M.: Clausal temporal resolution. ACM Transactions on Computational Logic 2(1), 12–56 (2001)
7. Gabbay, D., Pnueli, A., Shelah, S., Stavi, J.: The Temporal Analysis of Fairness. In: Proc. POPL 1980. pp. 163–173. ACM (1980)
8. Gabbay, D.M.: Declarative Past and Imperative Future: Executable Temporal Logic for Interactive Systems. In: Proc. Colloquium on Temporal Logic in Specification. Lecture Notes in Computer Science, vol. 398, pp. 402–450. Springer (1987)
9. Hustadt, U.: `TRP` 1.4 [online] (2008), `http://cgi.csc.liv.ac.uk/~ullrich/TRP/`
10. Hustadt, U., Konev, B.: **TRP++**: A temporal resolution prover. In: Baaz, M., Makowsky, J., Voronkov, A. (eds.) Collegium Logicum, pp. 65–79. Kurt Gödel Society (2004), `http://www.csc.liv.ac.uk/~ullrich/publications/HK_KGS.pdf`
11. Konev, B.: **TRP++** 2.2 [online] (2011), `http://cgi.csc.liv.ac.uk/~konev/software/trp++/`
12. Konev, B.: Translate [online] (2016), `http://cgi.csc.liv.ac.uk/~konev/software/trp++/translator/`
13. Lichtenstein, O., Pnueli, A., Zuck, L.: The Glory of the Past. In: Proc. Logics of Programs 1985, Lecture Notes in Computer Science, vol. 193, pp. 196–218. Springer (1985)
14. Nalon, C., Dixon, C.: Anti-prenexing and prenexing for modal logics. In: Proc. JELIA 2006. Lecture Notes in Computer Science, vol. 4160, pp. 333–345. Springer (2006)
15. Nalon, C., Hustadt, U., Dixon, C.: `ltl2snf`: a translator for LTL formulae into SNF clauses [online] (2018), `http://www.cic.unb.br/~nalon/software/ltl2snf-0.1.0.tar.gz`
16. Nonnengart, A., Weidenbach, C.: Computing small clause normal forms. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, chap. 6, pp. 335–367. Elsevier (2001)

17. Plaisted, D.A., Greenbaum, S.A.: A Structure-Preserving Clause Form Translation. Journal of Logic and Computation 2, 293–304 (1986)
18. Reger, G., Suda, M., Voronkov, A.: New techniques in clausal form generation. In: Proc. GCAI 2016. EPiC Series in Computing, vol. 41, pp. 11–23. EasyChair (2016)
19. Schuppan, V., Darmawan, L.: Evaluating LTL satisfiability solvers. In: Proc. ATVA 2011. Lecture Notes in Computer Science, vol. 6996, pp. 397–413. Springer (2011)
20. Suda, M.: Variable and clause elimination for LTL satisfiability checking. Mathematics in Computer Science 9(3), 327–344 (2015)
21. Suda, M.: `LS4` [online] (2018), `https://github.com/quickbeam123/ls4`
22. Suda, M., Weidenbach, C.: A PLTL-prover based on labelled superposition with partial model guidance. In: Proc. IJCAR 2012. pp. 537–543. Springer (2012)